Algorithmes parallèles pour le traitement rapide de géométries 3D

Hélène Legrand

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2017-ENST-0053

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Signal et Images »

présentée et soutenue publiquement par

Hélène LEGRAND

le 27 octobre 2017

Algorithmes parallèles pour le traitement rapide de géométries 3D

Directeur de thèse : **Tamy BOUBEKEUR**

Jury

M. Gilles Gesquière, Professeur, LIRIS, Université Lumière Lyon 2	Rapporteur
M. Franck Hétroy, Professeur, i3, Université de Strasbourg	Rapporteur
M. François Goulette, Professeur, CAOR, Mines ParisTech	Examinateur
Mme Pooran Memari, Chargée de Recherche CNRS, LIX, Ecole Polytechnique, Univ. Paris-Saclay	Examinatrice
M. Michel Roux, Maître de Conférences, LTCI, Télécom ParisTech, Université Paris-Saclay	Examinateur
M. Tamy Boubekeur, Professeur, LTCI, Télécom ParisTech, Université Paris-Saclay	Directeur

TELECOM ParisTech
école de l'Institut Mines-Télécom - membre de ParisTech
46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - www.telecom-paristech.fr

T H È S E

RÉSUMÉ

Au cours des vingt dernières années, les principaux concepts du traitement du signal ont trouvé leur homologue pour le cas de la géométrie numérique, et en particulier des modèles polygonaux de surfaces 3D. La simplification de surface (sous-échantillonnage), la subdivision de surface (sur-échantillonnage), le remaillage, le filtrage et divers autres opérateurs de traitement des surfaces sont aujourd'hui capables de produire des résultats de qualité élevée, y compris pour des modèles denses et volumineux (plusieurs millions de polygones). Ces traitements requièrent néanmoins un temps de calcul non négligeable lorsqu'on les applique sur des modèles de taille conséquente. Cette charge de calcul devient un frein important dans le contexte actuel, où les quantités massives de données 3D générées à chaque seconde peuvent potentiellement nécessiter l'application d'un sous-ensemble de ces opérateurs. La capacité à exécuter des opérateurs de traitement géométrique en un temps très court, tout en conservant une qualité suffisante, représente alors un verrou important pour les systèmes de conception, capture et restitution 3D dynamiques.

Cette thèse s'inscrit dans le cadre d'un nouveau paradigme d'acquisition et de traitement de données 3D. Tirant parti de l'omniprésence de capteurs (appareils photos, smartphones, caméras 3D...), le processus d'acquisition est aujourd'hui déterminé par les flux de données disponibles et non plus l'inverse. Le traitement rapide de la géométrie, si possible à la vitesse à laquelle sont générées les données, devient alors un problème clé. Dans ce contexte, l'objectif de cette thèse est d'aller vers une chaîne complète de traitement de surfaces pouvant s'exécuter instantanément sur des objets de taille non triviale. En pratique, on cherche à accélérer de plusieurs ordres de grandeur certains algorithmes de traitement géométrique actuels, et à reformuler ou approcher ces algorithmes afin de diminuer leur complexité ou de les adapter à un environnement parallèle.

Dans cette thèse, nous nous appuyons sur un objet compact et efficace permettant d'analyser les surfaces 3D à plusieurs échelles : les quadriques d'erreurs. En particulier, nous proposons de nouveaux algorithmes haute performance, maintenant à la surface des quadriques d'erreur représentatives de la géométrie. Un des principaux défis tient ici à la génération des structures adaptées en parallèle, afin d'exploiter les processeurs parallèles à grain fin que sont les GPU, la principale source de puissance disponible dans un ordinateur moderne.

TABLE DES MATIÈRES

CHAPITRE 1

INTRODUCTION

Dans ce chapitre, nous présentons de manière générale le contexte et les grands concepts sur lesquels est basée cette thèse. Nous détaillerons les objectifs visés et les contributions apportées lors de ces travaux, ainsi que l'organisation de leur présentation.

1.1 Contexte

1.1.1 Modèles géométriques 3D

L'informatique graphique repose en grande partie sur la création et la manipulation de données géométriques tridimensionnelles. Selon le contexte, ces données seront utilisées dans le domaine du cinéma, pour des effets spéciaux ou des films d'animation entièrement en images de synthèse, du jeu vidéo, de la visualisation scientifique, de la conception assistée par ordinateur, ou encore de l'impression 3D (où le rendu d'une image n'est pas la finalité). Ces domaines variés appellent chacun à des représentations spécifiques de la géométrie, qui peuvent être volumétriques ou surfaciques, continues ou discrètes.

Dans la majorité des cas, on manipule ces données sous forme de maillages polygonaux (figure 1.1), c'est à dire un ensemble de points et un ensemble de polygones indexés sur ces points (en général des triangles), qui représentent la surface de l'objet. Ces points peuvent également porter des informations supplémentaires, telles que des couleurs, des coordonnées de texture ou des normales par exemple. Il est également courant de ne pas avoir à disposition d'information de topologie, et de manipuler directement un modèle sous la forme d'un nuage de points, notamment dans le contexte de la capture 3D.

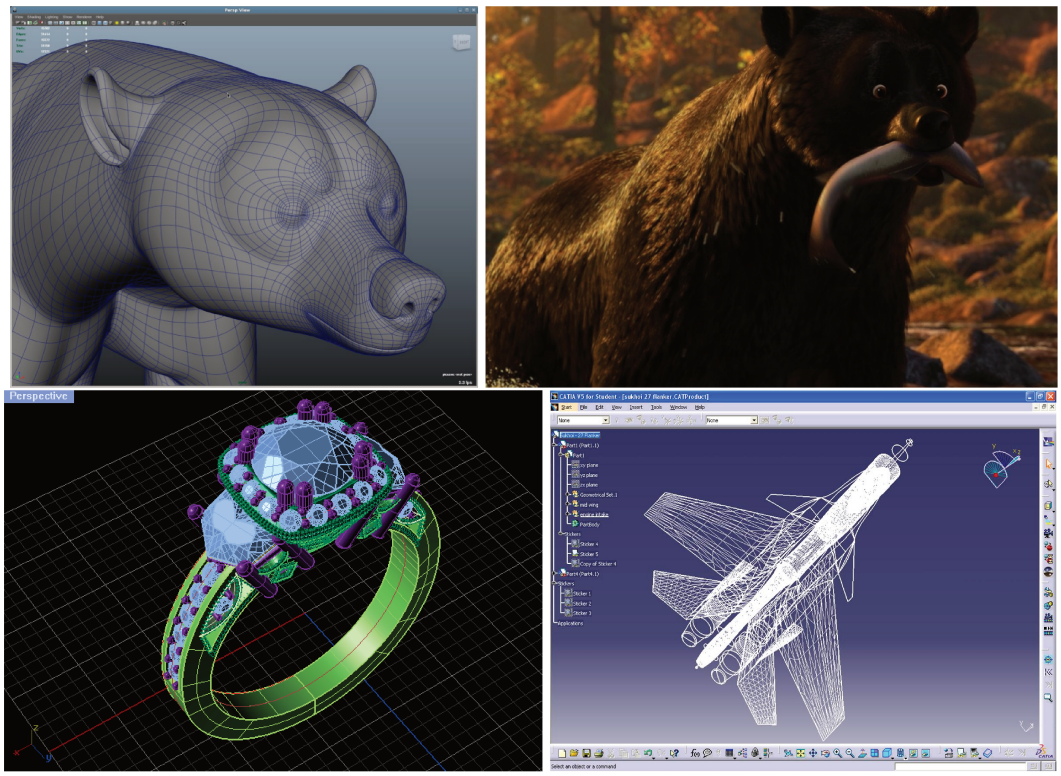


FIGURE 1.1 – *Première ligne :* Modèle d'un personnage et image extraite du film d'animation *Brave* (Pixar). *Deuxième ligne :* Exemples de modèles en Conception Assistée par Ordinateur.

1.1.2 Création

L'étape de création d'un modèle 3D peut se dérouler de plusieurs façons. L'objet peut être créé manuellement, en utilisant des logiciels de modélisation tels que Blender ou Maya. Ce processus long et méticuleux, plutôt réservé aux artistes, nécessite un savoir-faire spécifique pour obtenir des résultats convaincants (figure 1.2). L'artiste peut être assisté dans cette tâche par des techniques de capture du réel, avec par exemple la capture de mouvement, qui permet de numériser le mouvement d'un certain nombre de points repères sur un véritable acteur, et ainsi d'obtenir efficacement des animations réalistes pour des personnages préalablement modélisés.

A l'inverse, grâce aux avancées dans les technologies de capture et de reconstruction, les modèles peuvent être entièrement capturés, de manière dense, à partir d'objets réels. Les outils d'acquisition vont du scanner laser (figure 1.3), précis mais lourd et coûteux, à la photogrammétrie, qui reconstruit à partir de simples photos (2D) de l'objet, prises de différents points de vue.

FIGURE 1.2 – A gauche, Maillage réalisé manuellement par un artiste (*Toy Story*, Pixar). A droite, Modèle reconstruit à partir d'une sculpture de dragon scannée (*Stanford 3D Scanning Repository*).

On peut distinguer deux types d'approches opposés pour la capture d'objets 3D. D'une part, on peut avoir l'acquisition "délibérée" des données, c'est à dire que l'acquisition est planifiée, avec un certain contrôle sur les conditions de numérisation. Le scan est effectué soigneusement et tire parti au mieux du système de capture utilisé. C'est l'approche la plus classique, qu'on utilisera en général pour des applications telles que la numérisation de pièces mécaniques pour la rétro-ingénierie, la documentation et la réplication de sites et objets culturels, les relevés archéologiques, l'imagerie médicale 3D ou encore la création de modèles pour le cinéma ou les jeux vidéo.

D'autre part, avec l'apparition de capteurs 3D à bas coût accessibles au grand public, la démocratisation des appareils photos numériques et le développement des réseaux de communication permettant le partage instantané des données capturées, on assiste à l'émergence d'une nouvelle approche basée sur des flux de données de plus en plus massifs.

Ainsi, on cherchera par exemple à reconstruire un modèle 3D d'un lieu à partir d'une base non structurée de milliers de photos, prises dans des conditions non contrôlées (par exemple le projet *Building Rome in a day* [AFS+11], à partir d'images partagées sur Flickr), ou à partir de flux de données continus issus d'un ou plusieurs capteurs (2D ou avec des informations de profondeur). On sort alors du cadre de la capture délibérée et des avantages qu'elle apporte sur le contrôle des conditions d'acquisition, pour s'intéresser à la masse de données brutes produite actuellement. Il devient donc nécessaire de développer de nouveaux outils adaptés.

Les modèles bruts générés par les méthodes de capture 3D sont rarement utilisables tels quels, surtout dans le cas de l'approche décrite précédemment. On obtient en général plusieurs jeux de nuages de points 3D, qu'il est nécessaire de réaligner pour obtenir le modèle. En outre, les données sont bruitées, relativement peu prévisible, non structurées (absence d'informations de topologie), potentiellement incomplètes, trop volumineuses pour être exploitées facilement par les machines actuelles. Elles doivent donc être traitées afin d'être utilisables pour les applications visées.

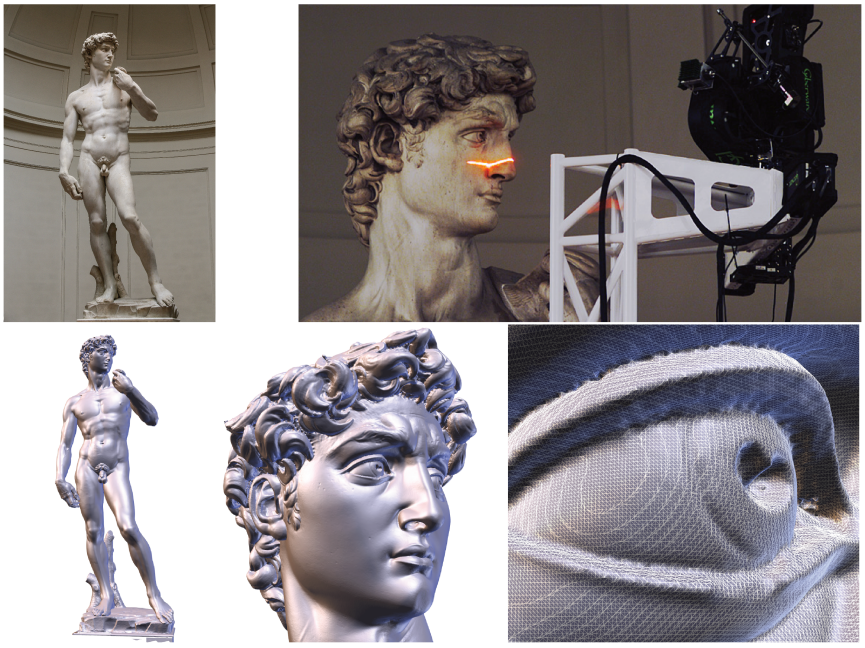


FIGURE 1.3 – **Première ligne :** Photo du David de Michel-Ange, numérisation laser (*The Digital Michelangelo Project* [LPC+00]). **Deuxième ligne :** Modèle dense numérisé et gros plans.

1.1.3 Traitements géométriques

Le traitement géométrique de modèles 3D, en particulier de maillages et nuages de points, englobe les techniques permettant d'analyser et transformer des formes 3D. Dans le cas de données scannées, la première étape de la chaîne de traitement est en général consacrée à l'alignement des différents nuages de points acquis, et éventuellement à la reconstruction à partir de ceux-ci pour obtenir un maillage.

L'étape suivante consiste bien souvent à "réparer" les modèles 3D obtenus pour les rendre utilisables, c'est à dire faire en sorte que des artefacts indésirables pour une application donnée soient corrigés. Par exemple, certains de ces algorithmes ont pour but de produire des surfaces sans trous, de modifier un modèle pour qu'il soit une 2-variété, ou encore de réorienter les normales de manière cohérente. On peut ensuite vouloir éliminer le bruit induit par l'acquisition, réduire la taille du modèle pour le rendre gérable par la machine ou réduire sa complexité géométrique, lisser le modèle, remailler la surface pour optimiser la forme des triangles et assurer la stabilité de certains algorithmes (simulations physiques par exemple), l'analyser pour en extraire des structures de contrôle permettant à un utilisateur de la manipuler facilement, etc.

Un très grand nombre de traitements sont possibles, selon l'application visée, du plus simple au plus complexe. Malgré leur diversité, ces derniers ont bien souvent en commun d'utiliser un certain nombre de "briques" de base. Ce sont des opérateurs de traitement de surface

FIGURE 1.4 – Nuage de points brut d'une statue numérisée (photogrammétrie), et résultat maillé après reconstruction et traitements.

relativement simples mais fondamentaux, tels que le filtrage, la simplification ou encore la subdivision de surface, dont l'efficacité et la robustesse sont critiques, surtout dans le cas de grandes quantités de données. C'est pourquoi la vitesse d'exécution de ces algorithmes influencera toute la chaîne de traitement.

1.1.4 Contraintes de temps

On peut distinguer plusieurs scénarios lorsqu'on traite des données géométriques. Dans certains cas, lorsque l'on a besoin de résultats irréprochables, par exemple dans l'industrie du cinéma, ou qu'on a peu de données à traiter, on se permettra d'allouer un temps relativement long au processus.

Ce n'est cependant pas toujours possible, de nombreuses applications imposent des contraintes de temps fortes. Par exemple une application comportant une interaction avec un utilisateur pourra être inutilisable si la fréquence d'affichage est trop faible. Le cas de flux de données continus impose également de fortes restrictions sur le temps disponible, de par le rythme d'arrivée des données et leur quantité. On parle alors d'applications en temps

interactif, ou en temps réel lorsqu'on vise une fréquence d'au moins 30Hz, seuil à partir duquel la sensation de mouvement est restituée correctement.

Dans ces situations où la première contrainte est le temps, il devient nécessaire de tolérer un compromis (contrôlé) sur la qualité des résultats. Cela peut se faire en effectuant diverses approximations, qu'on s'efforce de minimiser par une implémentation la plus rapide possible, tirant parti par exemple des capacités de calcul grandissantes offertes par les GPU.

1.1.5 GPGPU

Les GPU (*Graphics Processing Units*) ont historiquement été conçus pour accélérer le processus de rendu dans les applications 3D. La chaîne de rendu était alors fixe, et prenait en entrée un maillage triangulaire, auquel était appliqué un ensemble de fonctions prédéfinies, avec des unités dédiées au niveau matériel, pour aboutir à l'image affichée à l'écran. Au cours des années, les différentes étapes de cette chaîne sont devenues peu à peu programmables, avant d'évoluer vers une architecture unifiée.

On peut maintenant voir les GPU comme des processeurs massivement parallèles à visée générale (*General Purpose computation on Graphics Processing Units*, ou GPGPU). Afin de tirer parti au maximum de la flexibilité des GPU modernes, des frameworks de calcul parallèle sur GPU, comme CUDA et OpenCL, ont alors été créés, et permettent d'écrire des programmes en s'affranchissant complètement de la chaîne de rendu originale.

Les GPU sont particulièrement adaptés dans le cas d'algorithmes présentant un parallélisme à grain fin. Le cas idéal est celui des algorithmes pour lesquels il est possible de diviser le travail, afin que chaque thread puisse traiter indépendamment et en même temps une primitive donnée en entrée, en exécutant à chaque fois le même ensemble d'instructions (modèle SIMD, *Single Instruction Multiple Data*). Dans le cas du traitement géométrique, certains algorithmes remplissent immédiatement ces conditions, tandis que d'autres nécessitent un travail d'adaptation spécifique pour maximiser le parallélisme, comme par exemple lorsqu'on a besoin d'utiliser une structure hiérarchique.

1.2 Objectifs

L'objectif principal de cette thèse est d'accélérer de plusieurs ordres de grandeur des opérateurs de base du traitement géométrique, afin qu'ils puissent fonctionner à la même vitesse, sinon plus vite, que les données 3D sont générées. On vise ainsi à ce qu'ils ne représentent

plus le goulot d'étranglement de la chaîne de traitement, dans de but de consacrer plus de temps à des opérations plus complexes comme par exemple de l'analyse de données.

Pour cela, on cherche à approcher ou reformuler les algorithmes actuels qui ne sont pas adaptés au temps réel, pour diminuer leur complexité ou les adapter à un environnement parallèle, tout en gardant un résultat pratique équivalent. Cette reformulation passe par l'utilisation de structures de données spécifiques, de métriques adaptées et d'approximations contrôlées.

On se place dans le contexte du traitement temps réel / temps interactif, avec pour but d'augmenter le nombre et la complexité des opérations possibles lorsqu'on traite à la volée de gros volumes ou des flux de données.

1.3 Contributions

Cette thèse s'articule autour de trois contributions principales :

- un algorithme parallèle permettant de simplifier de manière adaptative des maillages et nuages de points de grande taille en temps réel, s'appuyant sur le nouveau concept d'*intégrale de Morton*,
- une représentation enrichie de la surface d'un modèle basée sur la *métrique d'erreur quadrique* (QEM), calculable et exploitable rapidement sur GPU, et son application à une gamme de filtres adaptatifs interactifs,
- une méthode d'extraction de modes et de partitionnement de surface rapide, s'appuyant également sur cette représentation.

1.4 Organisation de la thèse

Dans le chapitre 2, nous apporterons un contexte technique pour les notions fondamentales utilisées tout au long de cette thèse. Nous présenterons ensuite dans le chapitre 3 un algorithme de simplification adaptative parallèle temps réel et les structures de données GPU utilisées. Le chapitre 4 ensuite sera consacré au filtrage adaptatif sur GPU basé QEM. Enfin le chapitre 5 détaillera une nouvelle méthode d'analyse modale par partitionnement de surface rapide parallèle, avant de conclure dans le chapitre 6.

CHAPITRE 2

CONTEXTE TECHNIQUE

Ce chapitre vise à apporter un contexte technique pour les éléments de fond utilisés tout au long de cette thèse. Les travaux connexes spécifiques seront abordés au fur et à mesure dans les chapitres suivants, pour se concentrer ici sur les notions transverses.

2.1 Métrique d'erreur quadrique

La métrique d'erreur quadrique (*Quadric Error Metric* ou QEM) est une métrique introduite par Garland et Heckbert [GH97], qui fournit une bonne description locale de la surface, avec pour avantages d'avoir un faible coût d'évaluation, une empreinte mémoire faible et fixe quelle que soit la région considérée, et de ne pas faire de suppositions sur la structure de la géométrie (maillages, soupes de polygones, nuages de surfels, etc.).

2.1.1 Calcul de QEM

On considère une région de la surface, modélisée par un ensemble d'échantillons (triangles, sommets, points équipés d'une normale). Chaque échantillon définit un plan 3D, duquel on mesure la distance quadratique dans tout l'espace.

Pour l'ensemble t des plans associés à ces échantillons, on peut calculer en tout point v de l'espace la QEM, comme la somme des distances au carré de v à tous les plans de t.

$$E(v) = \sum_{i \in t} dist_i^2(v)$$

En 3D, les isosurfaces $E(v) = \epsilon$ sont des quadriques, d'où le nom de "métrique d'erreur quadrique". Plus précisément, il s'agit d'ellipsoïdes, qui peuvent être dégénérées si l'on considère moins de trois plans, ou si tous les plans sont parallèles à une même droite. Dans ce cas, les isosurfaces sont des cylindres. Si tous les plans sont parallèles entre eux, les isosurfaces sont des plans.

Un avantage de la QEM est de permettre une représentation pratique et compacte de l'erreur à chaque sommet. On définit une quadrique Q comme un triplet :

$$Q = (A, b, c)$$

où A est une matrice 3×3 symétrique, b un vecteur de taille 3, et c un scalaire, ou encore en représentation homogène :

$$Q = \left[\begin{array}{c|c} A & b \\ \hline b^T & c \end{array}\right]$$

Une quadrique Q associe à tout point v de l'espace une valeur $Q(v)$:

$$Q(v) = v^T A v + 2b^T v + c \qquad \text{ou encore} \qquad Q(v) = (v^T|1)Q(v^T|1)^T$$

Les isosurfaces $Q(v) = \epsilon$ sont des quadriques.

Pour obtenir A, b et c pour un sommet, on commence par calculer les *quadriques fondamentales* de ses triangles adjacents. Pour un plan définit par $n^T v + d = 0$, on définit sa quadrique fondamentale :

$$Q_f = (nn^T, dn, d^2)$$

ou en notation homogène :

$$Q_f = \left[\begin{array}{c|c} nn^T & dn \\ \hline dn^T & d^2 \end{array}\right]$$

$Q_f(v)$ donne la distance au carré de v au plan. Les isosurfaces $Q_f(v) = \epsilon$ sont des paires de plans.

Pour un ensemble de plans, on somme simplement leurs quadriques fondamentales. On peut alors définir une quadrique pour la région entière en sommant les quadriques de ses échantillons individuels. L'approximation ainsi faite de la distance quadratique à la région a plusieurs avantages. Tout d'abord, il s'agit toujours d'une quadrique, modélisée par une matrice symétrique 4×4. Cela permet d'être efficace en termes d'occupation mémoire, étant donné que pour un modèle 3D on a seulement besoin de stocker une quadrique par primitive, c'est à dire 10 coefficients (matrice symétrique).

On aura facilement accès à la quadrique d'un ensemble de primitives, en sommant leurs quadriques. Le coût de l'évaluation de l'erreur en un point est relativement faible également, car elle consiste en quelques additions et multiplications ($Q(v) = (v^T|1)Q(v^T|1)^T$).

Un défaut de cette représentation est le fait qu'elle dépende de la tesselation du modèle d'entrée. En effet, le plan d'un triangle très petit aura le même poids que celui d'un triangle très grand.

2.1.2 Normalisation

Avec la représentation décrite précédemment, la quadrique de deux régions géométriquement identiques, mais tessellées différemment, ne sera pas la même, ce qui n'est pas souhaitable pour la plupart des applications.

On définit donc une quadrique pondérée :

$$Q_w = (wA, wb, wc) \quad \text{ou encore} \quad Q_w = wQ \quad \text{avec } w \text{ un scalaire}$$

On choisit en général de pondérer par l'aire des triangles. En effet, la somme des quadriques pondérées par l'aire tend vers l'intégrale des quadriques sur la surface, quand l'aire des triangles tend vers 0.

2.1.3 Minimisation

Une fois qu'on a calculé une quadrique pour un sommet ou un ensemble de sommets, il est souvent nécessaire de trouver le point minimisant l'erreur pour cette quadrique, en particulier dans le cadre de la simplification de surface [GH97, Lin00].

Dans ce contexte on vise à remplacer plusieurs sommets par un unique point, tout en gardant une bonne approximation de la géométrie. On cherche alors le point v tel que $Q(v)$ est minimal, avec Q la somme des quadriques des sommets concernés.

$$v = -A^{-1}b$$

On peut alors voir le point résultat comme l'optimiseur de la région, que l'on peut utiliser comme sommet représentatif pour la géométrie de la région. Par la suite, on utilisera souvent les termes "minimiseur" ou "optimiseur" de Q pour désigner ce point.

Il est possible qu'une unique solution n'existe pas, si A est singulière. Cela se produit lorsque les plans considérés sont tous parallèles à une même droite ou entre eux. L'ensemble des points minimisant $Q(v)$ est alors une droite ou un plan. Il faut dans ce cas mettre en place une stratégie alternative pour choisir un point représentant au mieux la géométrie. On pourra par exemple, selon le cas, choisir d'utiliser le minimiseur de $Q(v)$ le long de la normale à la surface, ou de se rabattre sur la moyenne des points.

Minimiser la QEM permet de placer les nouveaux points de manière à bien préserver les détails et les caractéristiques importantes de la forme, comme on peut le constater dans [GH97] (contraction des sommets deux par deux) ou encore [Lin00] (contraction de clusters entiers).

2.2 Calcul GPU

Cette section apporte une vue d'ensemble des notions de calcul GPU utilisées dans cette thèse. Nous décrirons à l'aide de la terminologie CUDA l'architecture générale des GPU actuels, ainsi que les primitives algorithmiques essentielles du calcul parallèle.

2.2.1 Architecture

Les GPU sont des processeurs prévus à l'origine pour une tâche très spécifique. Leur architecture a été pensée pour permettre de synthétiser des images en temps réel de la manière la plus optimale possible, principalement pour des applications comme les jeux vidéos. Plus précisément, ils ont été créés pour prendre en charge les calculs relatifs à la chaîne de rendu classique, c'est à dire un ensemble d'opérations sur les primitives géométriques de la scène, une étape de rastérisation, puis un ensemble d'opérations sur les pixels ou fragments obtenus. Les étapes de cette chaîne de traitements, qui étaient fixes pour les premières cartes graphiques, sont devenues petit à petit programmables, au travers des *vertex shaders* et *fragment shaders*, puis des *geometry shaders* et des *compute shaders* plus généraux. Des

technologies comme CUDA et OpenCL ont également été développées, afin de permettre d'exploiter les GPU dans un contexte plus général, en s'affranchissant des API graphiques.

De par leur forte spécialisation, les GPU sont peu flexibles mais très efficaces dans des contextes particuliers. Afin de pouvoir traiter en temps réel un grand nombre de polygones ou de pixels, ils fonctionnent de manière massivement parallèle, en tirant parti du peu de dépendances entre les primitives. Contrairement aux CPU, qui peuvent gérer quelques threads complexes avec un petit nombre de cœurs, les GPU contiennent plusieurs centaines (voire milliers) de cœurs pouvant exécuter simultanément un très grand nombre de threads légers, à condition qu'ils exécutent tous le même programme. Ces cœurs GPU fonctionnent à une fréquence plus faible que les cœurs CPU, et sont ne sont capables d'effectuer qu'un nombre limité d'opérations très optimisées, notamment pour l'arithmétique de base. Par conséquent, les applications exploitant au mieux le GPU sont celles qui effectuent une quantité importante de calculs sur des ensembles de données volumineux, et qui présentent un fort parallélisme où chaque élément des données peut être traité indépendamment.

CUDA est une technologie développée par NVIDIA permettant d'effectuer des calculs généraux sur GPU. CUDA est basé sur le langage C/C++, avec quelques ajouts propres au GPU. Les calculs parallèles à exécuter sur le GPU sont spécifiés dans des fonctions particulières appelées *kernels*. Quand un kernel est lancé, il est exécuté par un ensemble de *threads*. Cet ensemble de threads est organisé en *blocs* de threads, eux même organisés en *grille* (figure 2.1). Cette représentation haut niveau reflète l'architecture bas niveau des GPU. Nous faisons le choix dans cette thèse d'exploiter CUDA pour des questions de performance, ses principes et contraintes sont similaires au standard OpenCL, disponible sur l'ensemble des processeurs graphiques modernes, bien qu'en général moins performant.

Un bloc est un ensemble de threads, au maximum 512 ou 1024 selon le matériel, pouvant coopérer via une mémoire partagée rapide, et ayant la possibilité de synchroniser leur exécution via des barrières. Chaque thread possède un identifiant unique, le situant dans son bloc et la grille.

Mémoire

Les threads peuvent accéder à différents types de mémoire pendant leur exécution. Chaque thread a sa propre mémoire locale. Il peut également accéder à une mémoire partagée rapide, visible par tous les threads du même bloc, dont la durée de vie est la même que celle du bloc. Pour finir tous les threads ont accès à une même mémoire globale, qui est elle persistante (figure 2.1). La mémoire globale représente le moyen principal de communication entre la partie GPU et la partie CPU. Elle permet les transferts de données de l'un à l'autre et est relativement lente en lecture/écriture.

Pour optimiser l'utilisation de la bande passante et de la mémoire cache, et ainsi obtenir de meilleures performances, on s'efforce au maximum d'avoir des accès cohérents : des threads contigus accèdent à des données contiguës en mémoire, dans l'ordre (figure 2.2).

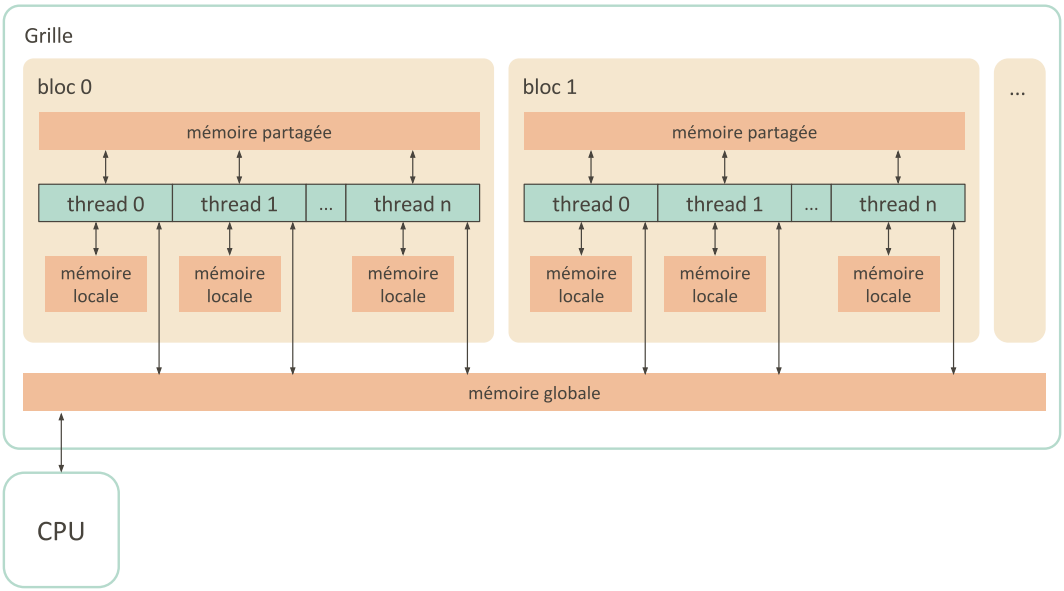


FIGURE 2.1 – Vue globale du modèle mémoire et de l'organisation des threads sur le GPU.

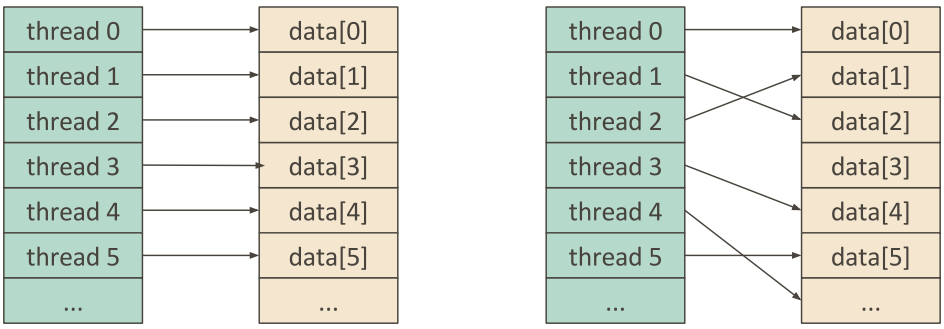


FIGURE 2.2 – A gauche : schéma d'accès optimal à la mémoire globale. A droite : accès non cohérents.

SIMT

Un GPU supportant CUDA comprend un certain nombre de multi-processeurs de flux (*streaming multiprocessor* ou SM). Quand un kernel est lancé, les blocs sont répartis sur les SM disponibles. Plusieurs blocs peuvent être gérés simultanément par un même SM.

Un SM exécute les threads par groupes de 32, appelés *warps*. Tous les threads d'un warp exécutent la même instruction en même temps. Par conséquent, en cas de branchement, l'exécution sera plus rapide si tous les threads d'un même warp empruntent la même branche. Dans le cas contraire le warp effectuera les calculs pour chaque branche séquentiellement, pour tous les threads du warp. On parle alors de *divergence*, qu'on s'efforce d'éviter au maximum.

Synchronisation intra-bloc

Les threads d'un bloc peuvent se synchroniser à l'aide de barrières. Une barrière garantit qu'aucun thread concerné ne pourra continuer au-delà de la barrière tant que tous les threads du bloc ne l'ont pas atteinte. On est alors sûrs que toutes les opérations d'écriture la précédant seront terminées, ce qui permet aux threads du bloc de communiquer en accédant à la mémoire partagée. En revanche, différents blocs n'ont aucun moyen efficace de communiquer directement sans passer par la mémoire globale. On pourra alors par exemple utiliser des opérations atomiques pour les coordonner.

Opérations atomiques

Une opération atomique est une fonction indivisible, c'est à dire dont on est sûr qu'elle ne sera pas interrompue. Il peut s'agir par exemple d'une addition, d'un maximum, d'une incrémentation ou d'un échange de valeurs. Les atomiques deviennent nécessaires quand plusieurs threads ont besoin d'accéder de manière concurrente à un même emplacement mémoire, et ont par conséquent besoin de se coordonner pour éviter les comportements indéterminés.

Les opérations réalisées par une fonction atomique suivent en général le schéma suivant : lecture de la valeur à l'emplacement mémoire, modification de la valeur, et enfin écriture de la valeur modifiée. La valeur renvoyée par la fonction est l'ancienne valeur. Cette suite d'opération ne doit pas être interrompue. En effet, si par exemple un autre thread accède à cet emplacement mémoire avant l'écriture de la valeur modifiée, une des modifications sera perdue. Pour éviter cela, l'accès à cet emplacement est verrouillé jusqu'à la complétion de la lecture-modification-écriture, et on garantit ainsi que l'on n'aura pas de comportement indéterminé. La performance d'une telle fonction est fortement réduite par rapport à son équivalent non atomique, bien que les architectures récentes les gèrent de plus en plus efficacement.

Utilisées correctement, les fonctions atomiques permettent d'adapter au GPU une large gamme de structures de données autrement inaccessibles. On peut ainsi par exemple créer des listes chaînées [YHGT10] ou accumuler facilement des valeurs dans des histogrammes [Hoe14]. Toutefois, des fonctions atomiques mal utilisées dégraderont fortement le parallélisme. La meilleure illustration de cette séquentialisation est lorsque tous les threads doivent incrémenter un même compteur : un premier thread va verrouiller l'emplacement mémoire, tandis que tous les autres devront attendre que ce verrouillage soit levé, et ainsi de suite de manière entièrement séquentielle. Il convient donc d'utiliser les fonctions atomiques avec parcimonie, et en s'assurant que les accès concurrents à un même emplacement mémoire ne soient pas trop nombreux.

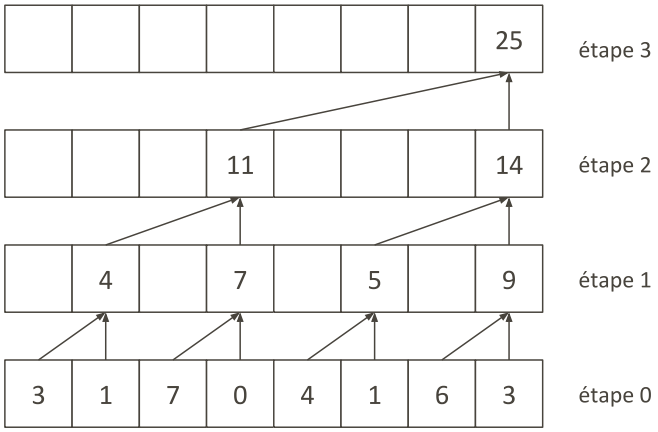


FIGURE 2.3 – Exemple de réduction parallèle, pour la somme sur un tableau d'entiers.

2.2.2 Primitives algorithmiques

Un algorithme séquentiel composé de boucles dont les itérations sont indépendantes les unes des autres aura une adaptation parallèle assez immédiate. Chaque thread se voit simplement assigné une ou plusieurs itérations de la boucle.

Tous les algorithmes ne correspondant pas à ce modèle, nous introduisons dans cette section les briques algorithmiques de base utilisées dans de nombreuses applications GPU et dans les chapitres suivants, en mettant plus particulièrement l'accent sur le *scan* et les manières de l'utiliser.

Réduction

L'opération de *réduction* est un des éléments de base du développement d'algorithmes parallèles. Cela consiste à appliquer un opérateur binaire associatif à un ensemble d'éléments pour le réduire à un seul, par exemple la somme des éléments ou leur maximum. Le caractère parallèle de cette opération est peu intuitif. Le principe est ici de calculer à chaque étape en parallèle des réductions partielles, et de réitérer sur les résultats de ces réductions jusqu'à ce qu'il n'en reste qu'un (figure 2.3). Si le nombre d'éléments du tableau à réduire n'est pas une puissance de 2, on peut simplement s'y ramener en complétant avec l'élément neutre de l'opérateur (0 pour l'addition par exemple).

Scatter et gather

Deux autres opérations fondamentales du calcul parallèle sont *gather* (collecter) et *scatter* (disperser), pour lesquelles des données seront "lues depuis" (collectées) ou "écrites vers" (dispersées) un emplacement donné. En d'autres termes, lors d'un *gather* chaque thread va aller lire des données depuis des emplacements arbitraires (figure 2.4), et pour un *scatter* écrire des données à un emplacement arbitraire (figure 2.5).

Il est important de garder à l'esprit lorsqu'on conçoit un algorithme que ces lectures, et encore plus ces écritures, en accès aléatoires à la mémoire sont beaucoup plus coûteuses que ne le seraient des accès séquentiels.

Tableau de structures ou structure de tableaux

En programmation CPU classique, des données liées sont souvent organisées en structures, stockées ensembles sous forme de tableau. Cela peut s'avérer très inefficace lorsqu'on accède aux données, particulièrement sur GPU. Par exemple, si on veut manipuler un ensemble de points (coordonnées) stockés de cette manière, lorsque chaque thread accède à la première coordonnée de chaque point, les accès à des emplacements mémoire non contigus d'un thread à l'autre entraînent une utilisation excessive de la bande passante, ce qui affecte fortement la performance (figure 2.6).

Une manière plus optimisée d'organiser les données serait de s'assurer que les threads accèdent en même temps à des données voisines en mémoire, afin d'avoir des accès cohérents. Si on reprend l'exemple précédent, on stockerait alors la première coordonnée de tous les points à la suite, puis la deuxième, puis la troisième. On passe d'un tableau de structures à une structure de tableaux (figure 2.7).

La manière la plus adaptée d'organiser les données dépend bien sur des cas et devra être adaptée selon la situation.

Scan

Le *scan*, ou encore *somme préfixe* ou *somme cumulée*, a été introduit sur GPU par Horn [Hor05] et représente une composante de base de nombreux algorithmes parallèles.

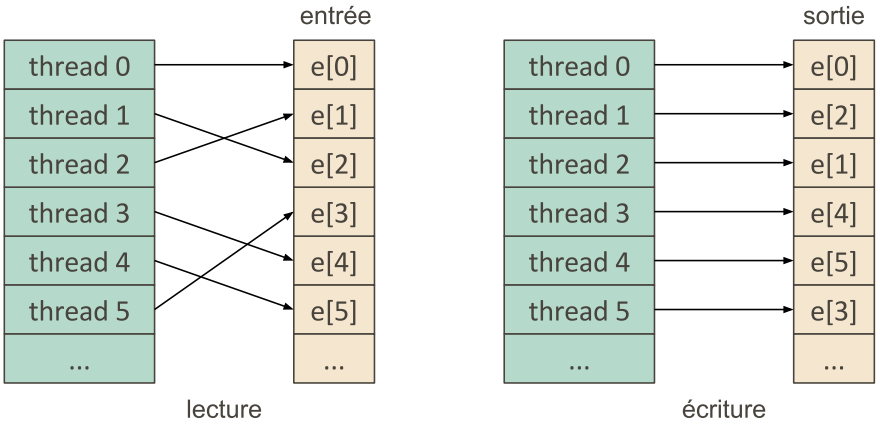


FIGURE 2.4 – **Gather**. L'étape de lecture effectue des accès aléatoire, celle d'écriture des accès cohérents.

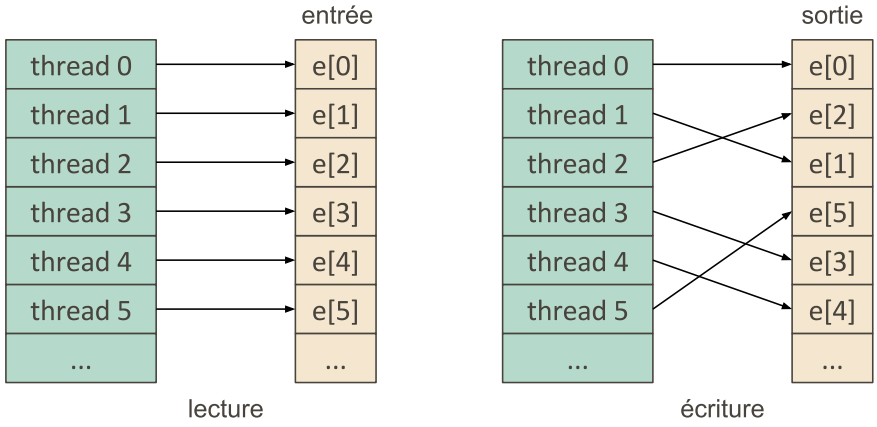


FIGURE 2.5 – **Scatter**. L'étape de lecture effectue des accès cohérents, celle d'écriture des accès aléatoires.

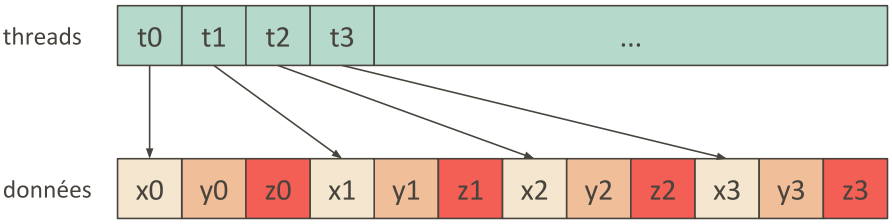


FIGURE 2.6 – **Tableau de structures.** Les accès mémoire ne sont pas cohérents.

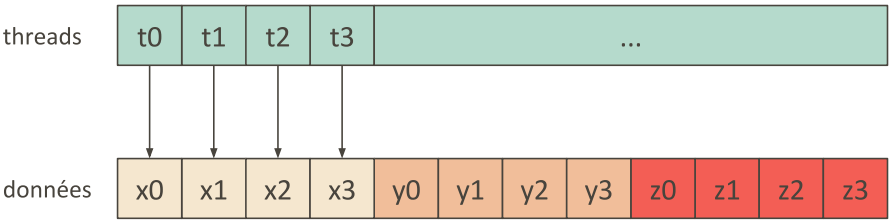


FIGURE 2.7 – **Structure de tableaux.** Les accès mémoire sont cohérents.

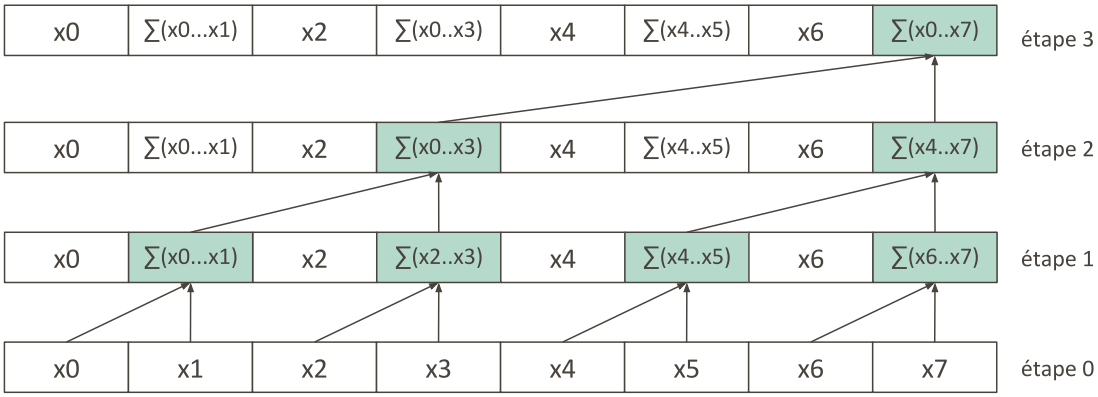


FIGURE 2.8 – Première phase de l'algorithme de calcul de scan.

Il prend en entrée un tableau de valeurs et un opérateur binaire associatif, par exemple l'addition, et produit un tableau de même taille dans lequel chaque élément est la somme de toutes les valeurs qui le précèdent dans le tableau original.

On parle de *scan exclusif* lorsque l'élément i du tableau résultat est la somme des éléments précédents du tableau d'entrée jusqu'à l'indice $i - 1$, l'élément i est exclu. Le scan est *inclusif* si l'élément i est inclus dans la somme. Par la suite, en l'absence de précision, le mot *scan* sera utilisé pour désigner un scan exclusif.

L'algorithme séquentiel est trivial, mais l'adaptation au calcul parallèle est loin d'être immédiate. Dans l'idéal, on souhaite avoir un algorithme de scan aussi efficace en termes de nombre d'opérations que l'algorithme séquentiel ($O(n)$), tout en tirant parti du parallélisme du GPU. En nous basant sur l'implémentation décrite par Harris et al. [HSO07], nous détaillons ici un algorithme permettant de calculer un scan parallèle effectuant $O(n)$ opérations. Des optimisations plus poussées existent, mais nous nous limiterons aux idées fondamentales.

Le principe ici est de construire implicitement un arbre binaire sur les données, et de le parcourir successivement des feuilles vers la racine puis de la racine vers les feuilles pour calculer la somme cumulée. Lors de la première phase, comme pour une réduction, on calcule des sommes partielles à chaque nœud interne de l'arbre (figure 2.8). Chaque nœud contient la somme des feuilles de son sous-arbre.

La deuxième phase traverse l'arbre à partir de la racine et utilise les sommes partielles calculées à la première étape pour générer toutes les sommes préfixes. On commence par insérer l'élément neutre (0 pour l'addition) à la racine de l'arbre. À chaque étape chaque nœud transmet sa valeur à son fils gauche. Il transmet à son fils droit la somme de cette même valeur et de l'ancienne valeur du fils gauche (figure 2.9).

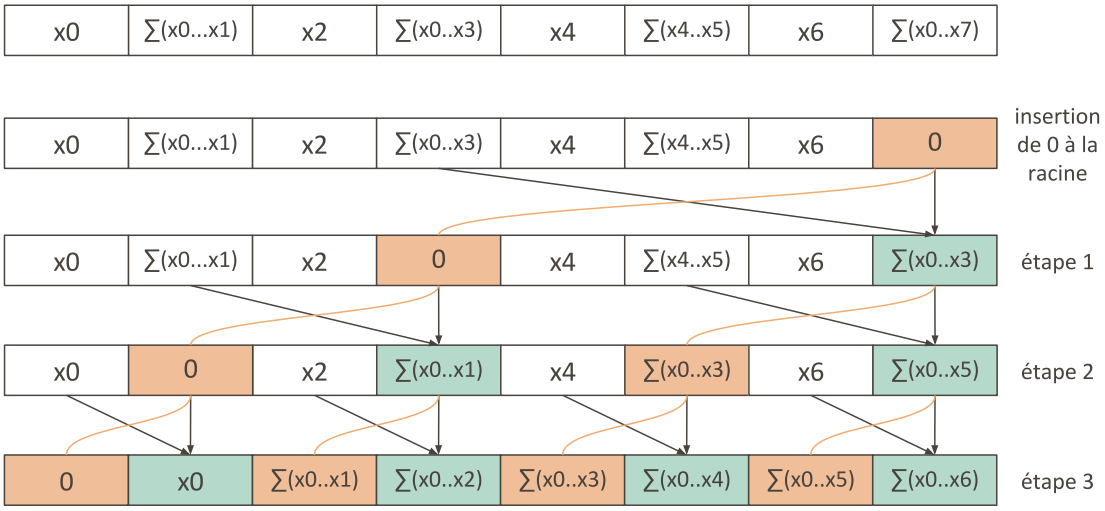


FIGURE 2.9 – Phase descendante de l'algorithme de calcul de scan.

Puisqu'il est nécessaire de synchroniser les threads à chaque étape, cet algorithme n'est valable qu'à l'intérieur d'un bloc (synchronisation). Pour l'étendre à des tableaux de grande taille, il faut en plus stocker la somme totale pour chaque bloc (avant de la mettre à 0 à l'étape 2), scanner ce tableau des sommes des blocs, et ajouter la valeur obtenue pour le bloc i à chaque élément du bloc $i + 1$. Ainsi chaque élément du tableau final contient bien la somme de tous les éléments le précédant.

Compactage de flux

Le compactage de flux est une autre primitive importante du calcul parallèle. Il permet des compacter un tableau comprenant des éléments inutiles, pour ne garder que les éléments intéressants pour la suite des calculs.

Un compactage s'effectue en deux étapes. Dans un premier temps, on "marque" dans un tableau temporaire les éléments à garder en les mettant à 1, les autres sont mis à 0. Ce tableau est ensuite scanné via l'algorithme précédent. On obtient pour chaque élément à garder son indice dans le tableau compact.

On peut connaître la taille à allouer pour le tableau compact en additionnant la dernière valeur du tableau de marquage et du tableau scanné. La dernière étape consiste à disperser (*scatter*) les éléments marqués aux adresses calculées (figure 2.10).

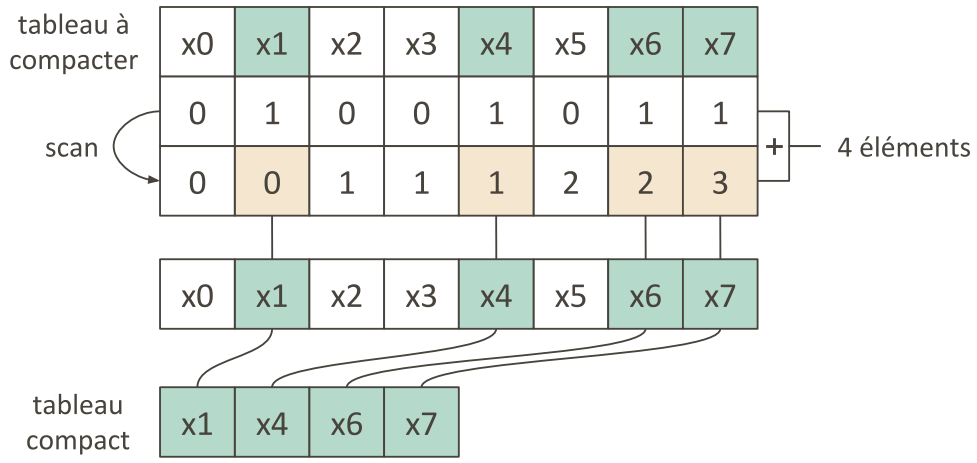


FIGURE 2.10 – Compactage de flux.

Tris

Une autre composante algorithmique de base non triviale à transposer en parallèle est le tri. Nous présentons ici deux méthodes de tri parallèles que nous utiliserons dans la suite de cette thèse : le tri par base (*radix sort*) et le tri comptage (*counting sort*).

Tri par base Le tri par base (*radix sort*) est un algorithme de tri pouvant s'adapter relativement simplement en parallèle, en tirant encore une fois parti de scans. Il est particulièrement rapide quand les clés à trier sont relativement courtes.

Globalement, le tri se déroule ainsi : on itère sur les bits des clés, en commençant par le moins significatif. À chaque itération, on "divise" les clés pour que toutes celles pour lesquelles ce bit est à 0 soient placées avant celles pour lesquelles il est à 1, tout en conservant l'ordre original dans ces deux sous-tableaux. L'opération est répétée pour tous les bits, le tableau obtenu est trié (figure 2.11).

En pratique, on peut exploiter le parallélisme du GPU en effectuant un scan pour chaque étape de division.

A chaque itération, on cherche à obtenir pour chaque clé son indice dans le nouveau tableau. Pour cela, de manière similaire à un compactage, on marque dans un tableau temporaire tous les éléments dont le bit considéré est à 0, avant de scanner ce tableau. Comme pour le compactage, on obtient pour chaque élément marqué son indice dans le nouveau tableau, et on peut récupérer le nombre t de clés dont le bit est à 0.

Pour les clés dont le bit est à 1, elles seront déplacées à l'indice $i - s + t$, où i est l'indice courant de l'élément dans le tableau, s la valeur dans le tableau scanné à l'indice i, et t le

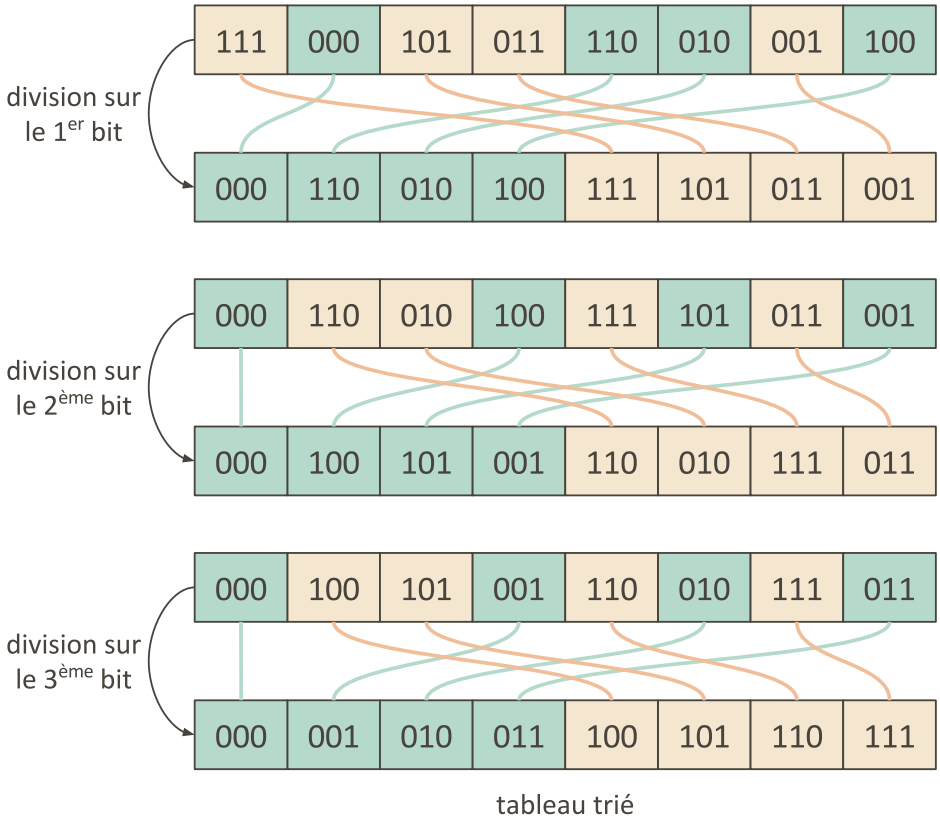


FIGURE 2.11 – Exemple de radix sort sur des clés de 3 bits.

nombre de clés dont le bit est à 0, calculé à l'étape précédente. Enfin, on écrit toutes les clés à la nouvelle adresse calculée (Figure 2.12).

Les éléments dont le bit considéré est à 0 se retrouvent placés ensembles au début du tableau dans leur ordre original, et ceux dont le bit est à 1 à la fin. On répète l'opération pour tous les bits.

Encore une fois, cet algorithme n'est valable qu'à l'intérieur d'un bloc. Pour trier des tableaux plus grands, on peut trier un sous-tableau par bloc, puis passer à un tri fusion récursif pour les combiner.

Tri comptage Le tri comptage (*counting sort*) est également spécialisé dans le tri d'entiers. Il est prévu pour trier des clés comprises dans un intervalle bien défini, dont les bornes sont relativement proches, c'est à dire que le nombre de valeurs différentes possibles n'est idéalement pas beaucoup plus élevé que le nombre d'éléments à trier.

L'idée générale de l'algorithme est la suivante. Le nombre d'occurrences de chaque clé est compté dans un tableau auxiliaire, de la taille de l'intervalle (histogramme). Dans un

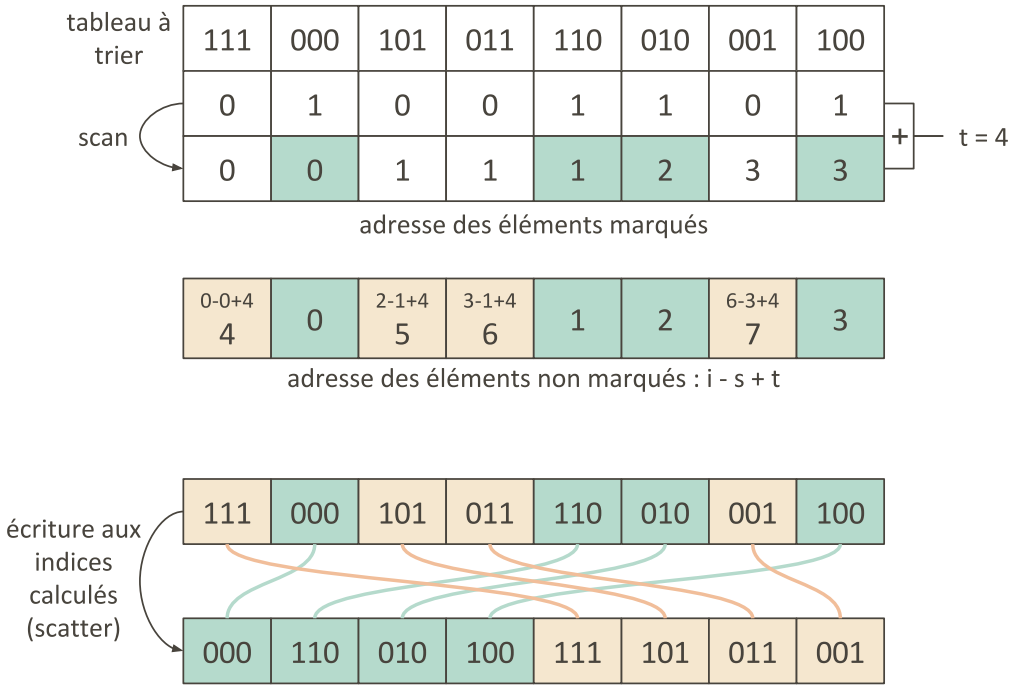


FIGURE 2.12 – Division parallèle du tableau pour le premier bit, à l'aide d'un scan.

second temps, une somme cumulée de ce tableau est calculée (via un scan) pour obtenir, pour chaque clé, la position de sa première occurrence dans le tableau trié. Cela permet dans une dernière étape de construire le tableau trié, en déplaçant le contenu du tableau initial aux adresses calculées. En parallèle, cet algorithme s'implémente à l'aide de fonctions atomiques. Nous présentons ici une version non stable du tri.

Considérons un tableau à trier tab de n éléments, dont les clés sont des entiers compris entre 0 et max. Dans la première passe de l'algorithme, deux tableaux intermédiaires sont alloués et initialisés à 0 : $compteur$, de taille $max + 1$, et $offset$, de taille n. Pour chaque élément i de tab, une unique opération atomique permet à la fois de lui attribuer un offset, et d'incrémenter le compteur des éléments ayant cette clé.

$$offset[\,i\,] \leftarrow \textbf{\textit{atomicInc}}(compteur[\,clé[\,i\,]\,])$$

où la fonction $atomicInc$ incrémente la valeur à l'adresse spécifiée, et renvoie l'ancienne valeur. À la fin de cette passe, le tableau $compteur$ contient, pour chaque clé, son nombre d'occurrences dans tab.

La deuxième étape consiste à faire un scan exclusif du tableau $compteur$. Le tableau $compteurScan$ résultant contient, pour chaque clé, l'adresse de sa première occurrence dans le tableau trié final. Enfin, il reste dans la dernière étape à copier les données du tableau ini-

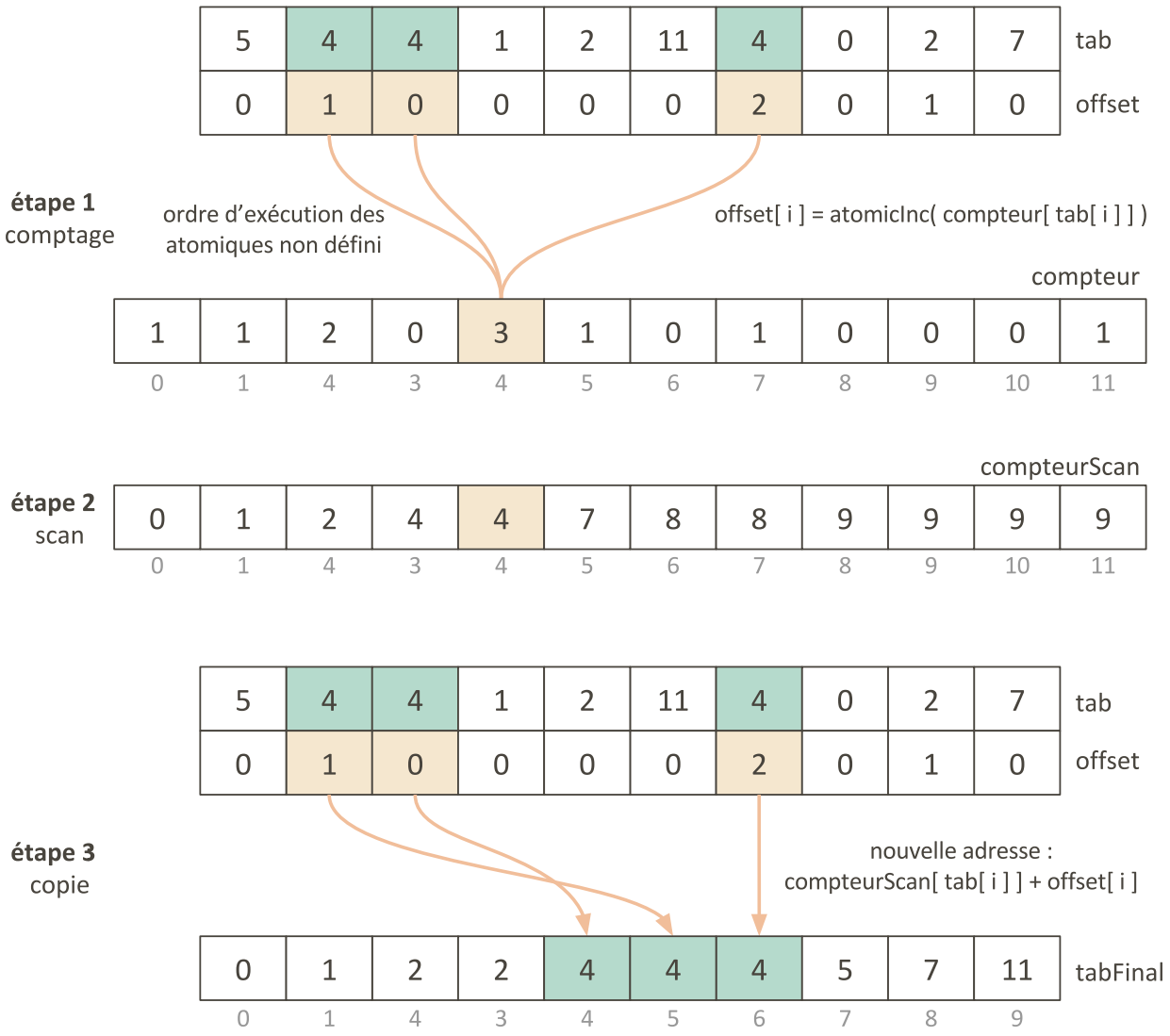


FIGURE 2.13 – Exemple de tri comptage parallèle. Les clés sont dans l'intervalle $[\,0, 11\,]$.

tial dans le tableau trié $tabFinal$. On obtient une adresse unique pour chaque élément, à l'aide des adresses calculées dans $compteurScan$ et des offsets. On note que, du fait de l'exécution en concurrence des incrémentations atomiques, on n'a pas d'information sur l'ordre d'attribution des offsets, et donc que des éléments ayant la même clé peuvent être dans un ordre différent dans le tableau final : le tri n'est pas stable. Pour chaque élément i de tab :

$$tabFinal\,[\,compteurScan[\,clé[\,i\,]\,] + offset[\,i\,]\,] = i$$

La figure 2.13 montre le déroulement de cet algorithme sur un exemple.

CHAPITRE 3

SIMPLIFICATION ADAPTATIVE RAPIDE DE GÉOMÉTRIE 3D

Dans ce chapitre, nous présentons un algorithme parallèle permettant de simplifier, en temps réel et de manière adaptative, des maillages et nuages de points de grande taille, en s'appuyant sur le nouveau concept d'*intégrale de Morton*.

3.1 Contexte

Les pipelines d'acquisition 3D actuels permettent de capturer la géométrie d'objets du monde réel rapidement et avec précision. La vitesse croissante à laquelle sont générées les données a été le point de départ d'un grand nombre de projets de recherche, visant à développer des systèmes capables de traiter ces données en temps réel.

Dans le cadre de ces scénarios contraints, il est souvent préférable d'utiliser des approximations, avec un contrôle sur la perte de qualité, plutôt que de dépasser les limites de temps de traitement imposées. Cette tendance dans le traitement géométrique haute performance a donné lieu à un certain nombre de résultats, avec désormais des solutions pour effectuer certaines des étapes critiques de la chaîne de traitements à la volée, pour des entrées de taille non triviale et sans nécessiter de matériel coûteux.

Le cas de la simplification adaptative de maillages (et le sous-échantillonnage de nuages de points) reste cependant difficile. Le principal problème vient du mécanisme de compression des données, qui ne s'adapte pas naturellement au calcul parallèle à grain fin. Bien que deux décennies de recherche aient progressivement mené à des algorithmes permettant de contrôler le compromis entre la qualité de la surface et l'effort de calcul, une simplification

de qualité en temps réel de modèles de plusieurs millions de primitives reste difficile, et les méthodes actuelles de simplification agressive ont un champ d'application limité.

Une simplification de meilleure qualité serait souhaitable pour les applications interactives et temps réel que nous visons, telles que l'affichage interactif sur appareil mobile, le rendu multi-vues dynamique ou la transmission à la volée de données capturées (caméra 3D par exemple). La géométrie simplifiée n'est généralement conservée en mémoire qu'un court instant, et est requise immédiatement après la génération des données en pleine résolution, ou à la demande par la suite. Un grand nombre de niveaux de détails différents est souvent nécessaire. Dans ce contexte, il est crucial de fournir instantanément des simplifications visuellement satisfaisantes.

Un aspect clé dans le traitement de maillages haute performance, largement exploité pour le filtrage et la tesselation, est la parallélisation des opérateurs et leur capacité à s'exécuter sur un processeur graphique (GPU).

Dans le contexte de la simplification adaptative, nous faisons deux observations. Tout d'abord, un échantillonnage adaptatif rapide repose souvent sur une structure de données hiérarchique sous-jacente, qui est coûteuse à construire et à maintenir dans un environnement parallèle. Ensuite, lorsqu'on dispose d'une telle hiérarchie, simplifier le maillage revient souvent à extraire une coupe particulière, basée sur l'erreur commise à chaque nœud de l'arbre. Cette opération ne s'adapte pas directement à une architecture parallèle à grain fin.

Nous proposons une méthode de simplification rapide, produisant des sous-échantillonnages adaptatifs basés sur une métrique d'erreur, tout en restant dans des temps d'exécution de l'ordre du temps réel pour des tailles de modèles en entrée/sortie typiques des applications visées.

Nous apportons des éléments de réponse au problème de la génération de la hiérarchie, grâce à l'utilisation de l'ordre de Morton sur les primitives du modèle en entrée, puis le calcul d'une somme cumulée 1D du coût géométrique associé à chaque échantillon. Cette représentation intermédiaire permet de construire efficacement un kd-tree sur les données, et dans un second temps d'évaluer de manière concurrente les erreurs géométriques associées à tous les nœuds de l'arbre.

Nous utilisons la QEM [GH97] comme mesure d'erreur. En effet, il s'agit de la métrique de référence dans le cadre de la simplification. De plus, les quadriques ont la propriété intéressante de rester des quadriques lorsqu'on les somme, ce qui nous permet de traiter le problème du calcul d'erreur aux différents niveaux de la hiérarchie en utilisant le même principe que les images intégrales [Cro84, VJ01]. La hiérarchie définie par le code de Morton des échantillons ne nécessite qu'un unique tri, réalisé au début de l'algorithme.

Bien que moins précis qu'une hiérarchie se raffinant purement suivant l'erreur (BSP-tree par exemple), le partitionnement spatial obtenu donne des résultats significativement supérieurs aux techniques GPU utilisant une partition uniforme.

Par conséquent, notre algorithme est capable de simplifier des maillages, soupes de polygones et nuages de points de grande taille en temps réel, tout en prenant en compte les détails et structures géométriques, ainsi que d'autres attributs éventuellement présents sur la surface.

3.2 Travaux Antérieurs

La plupart des méthodes de simplification définissent un critère d'optimisation, avec une métrique qui mesure l'erreur causée par la simplification, en général sous la forme d'une distance entre l'objet original et la version simplifiée. Les algorithmes de simplification peuvent habituellement être classés dans deux catégories : la simplification itérative et le partitionnement des sommets.

Les méthodes itératives [HDD+93, GH97] réduisent progressivement le nombre de primitives du maillage en effectuant, à chaque étape, une opération de simplification locale causant la plus petite erreur possible pour la métrique choisie. Ces méthodes donnent en général des maillages de haute qualité mais sont difficiles à paralléliser efficacement en raison de leur nature séquentielle. Elles nécessitent souvent une bonne connectivité qui, dans le contexte de la capture et du traitement instantanés, ne peut pas être garantie. Bien que leur méthode n'atteigne pas le temps réel, Grund et al. [GDG11] proposent un algorithme de simplification parallèle basé sur cette approche.

Nous nous focalisons ici sur les méthodes basées partitionnement, qui définissent une partition sur le maillage en groupant les primitives, calculent un sommet ou un polygone représentatif pour chaque groupe, et créent un maillage à partir de ce nouvel ensemble de primitives.

Le choix de la structure de partitionnement a un fort impact sur la performance globale du processus, avec des solutions incluant des grilles simples [RB93, Lin00], des octrees [SW03, Lin03, SG05], des BSP-trees [SG01] ou encore des partitions par k-means [CSAD04]. L'élément représentatif est encore une fois choisi de manière à minimiser une certaine métrique, parmi lesquelles on trouve en particulier la QEM [GH97], qui modélise le coût de simplification comme la somme des distances au carré des points représentatifs aux plans définis par les triangles du groupe (section 2.1). On peut citer également la métrique $L^{2,1}$ [CSAD04], qui utilise l'information donnée par les normales pour créer autant que possible de grandes régions planes.

L'étape du maillage final peut s'effectuer par une réindexation des triangles du modèle d'entrée : ceux ayant leurs sommets dans trois groupes différents sont réindexés sur les nouveaux points représentatifs calculés [RB93, BA09]. Une autre méthode consiste à générer des polygones à partir des frontières des groupes [CSAD04].

Étant donné que chaque groupe de primitives est en grande partie traité indépendamment des autres, les méthodes basées partitionnement sont plus adaptées au calcul parallèle, notamment sur GPU. En particulier, Decoro et Tatarchuk [DT07] ont proposé une implémentation GPU de la simplification basée QEM avec une partition en grille [Lin00], ainsi qu'une structure d'octree probabiliste pour commencer à introduire de l'adaptativité.

En ce qui concerne le type de simplification obtenue, notre méthode s'inscrit dans la même catégorie que celles de Schaefer et al. [SW03], Lindstrom et al. [Lin03] et Shaffer et al. [SG05], avec une hiérarchie alignée sur les axes fondée sur un tri des primitives spatialement cohérent. L'essentiel de notre contribution consiste à produire des résultats très similaires en termes de qualité, tout en réalisant une simplification entièrement parallèle sur GPU en temps interactif voire en temps réel.

Lors de la création de la partition initiale, les structures hiérarchiques de partitionnement de l'espace permettent un bon compromis entre l'adaptativité totale mais coûteuse, et une partition uniforme de type grille. La création efficace de telles structures a principalement été étudiée dans le contexte du rendu. En particulier, l'ordre de Morton (ou ordre-z) donne une paramétrisation de l'espace très adaptée à la construction de hiérarchies [LGS$^+$09, PL10, GPM11]. Plus précisément, un code de Morton est calculé pour chaque primitive en entrelaçant les bits de ses coordonnées (section 3.3.1). Trier les primitives selon le code ainsi obtenu permet de les grouper de manière cohérente spatialement (figure 3.1).

Cela permet la construction parallèle d'un arbre binaire, niveau par niveau, à partir de la racine, chaque nœud représentant une plage contiguë de primitives. Une idée similaire est exploitée par Zhou et al. [ZGHG10] pour générer des octrees dans le contexte de la reconstruction de surface.

Notre structure GPU sous-jacente est basée sur les travaux de Karras [Kar12], qui maximisent le parallélisme de la construction de l'arbre, atteignant ainsi des performances temps réel pour des modèles de plus d'un million de polygones. Au lieu de générer un niveau de l'arbre à la fois, tous les nœuds sont traités en parallèle, grâce à une structure d'arbre permettant de trouver la plage de primitives couverte par un nœud, ainsi que ses nœuds fils, indépendamment des autres nœuds.

Plus précisément, les nœuds feuilles et les nœuds internes sont stockés dans deux tableaux séparés : L (taille n) et I (taille $n-1$). En assignant les bons indices aux nœuds internes, Karras trouve la plage de feuilles qu'ils couvrent et les indices de leurs nœuds fils, sans avoir à accéder au préalable aux ancêtres ou aux descendants. La méthode est décrite de manière plus détaillée par la suite (section 3.3.1).

3.3 Structures de données

3.3.1 Structure de l'arbre

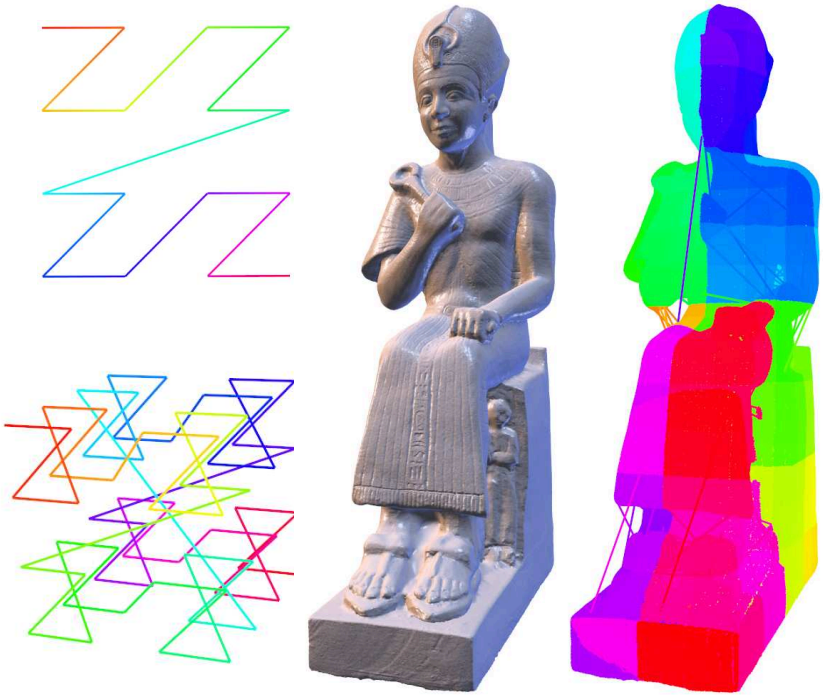


FIGURE 3.1 – **Ordre de Morton** pour les points d'une grille 2D de taille 4^2, une grille 3D de taille 4^3, et les sommets d'un modèle 3D.

La structure que nous utilisons est basée sur les travaux de Karras et al. [Kar12], qui maximisent le parallélisme de la construction d'une hiérarchie sur les données, et permet d'atteindre des performances temps réel sur des modèles de plus d'un million de triangles.

Une approche simple pour paralléliser la construction de la hiérarchie est de se ramener à un problème de tri. En triant nos sommets selon une courbe de remplissage de l'espace, on obtient une séquence 1D que l'on va diviser récursivement, chaque sous-séquence correspondant à un nœud. La racine correspond ainsi à la séquence entière, tandis qu'un nœud feuille correspond à un unique sommet. Chaque nœud interne couvre un intervalle continu dans la séquence triée [LGS$^+$09].

La courbe de Morton (ou courbe de Lebesgue) est particulièrement adaptée dans le cadre de la parallélisation. En effet, contrairement à la majeure partie des courbes de remplissage de l'espace, il est possible de déterminer l'indice d'un point dans la courbe de Morton, son *code de Morton*, directement à partir de ses coordonnées. Parcourir un ensemble de primitives selon l'ordre de Morton revient à faire un parcours en profondeur d'abord dans un octree

sur les données, ou quadtree en 2D (voir figure 3.1). On peut donc dériver directement une hiérarchie à partir de nos sommets triés [LGS+09, PL10, GPM11].

Si on discrétise les coordonnées en entiers (non signés) codés sur k bits, le Morton code d'un sommet se calcule en entrelaçant les bits des ses coordonnées, formant ainsi un entier de $3k$ bits (figure 3.2).

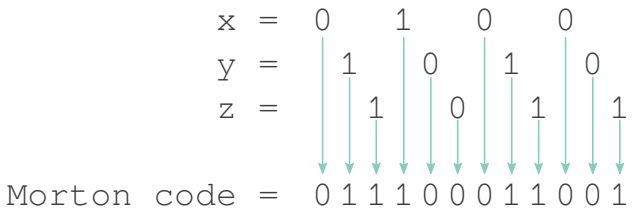


FIGURE 3.2 – **Calcul du code de Morton** pour un point 3D de coordonnées x, y, z.

En utilisant la méthode de Karras [Kar12], nous pouvons construire un arbre radix binaire (interprétable directement comme un kd-tree) sur les données entièrement en parallèle. Au lieu de générer un niveau de l'arbre à la fois, en divisant récursivement la séquence de sommets triés, tous les nœuds sont traités en parallèle grâce à une structure d'arbre particulière qui permet de trouver l'ensemble des feuilles couvertes par un nœud interne indépendamment des autres nœuds.

Dans un premier temps, le tableau des n primitives est trié par code de Morton croissant. Chaque primitive correspond à une feuille de l'arbre final. On alloue également un tableau de taille $n-1$ pour les nœuds internes. Une numérotation spécifique de ces derniers permet de les traiter indépendamment les uns des autres : la racine a pour indice 0, et les indices des enfants d'un nœud seront obtenus à partir de la position de la division pour ce nœud (de chaque côté de la division). L'indice d'un nœud correspond donc toujours à une extrémité de la plage de feuilles qu'il couvre (figure 3.3).

Pour chaque nœud interne i en parallèle, on veut trouver la plage de feuilles qu'il couvre, sachant que l'on connaît une extrémité grâce à la numérotation des nœuds (l'indice i). Pour déterminer si cette extrémité correspond au début ou à la fin de la plage, on examine le Morton code pour les indices voisins $i-1$ et $i+1$. On sait que l'une des feuilles pour ces indices est couverte par le nœud interne i, et l'autre par le nœud frère de i.

On note $lcp(a,b)$ la longueur du *plus long préfixe commun* entre le code de Morton de a et b.

Si $lcp(i,i-1) > lcp(i,i+1)$ alors l'indice i correspond à la fin de la plage, sinon i correspond au début. On sait maintenant dans quelle direction se situe l'autre extrémité de la plage.

On note $lcp_{min} = min(lcp(i,i-1), lcp(i,i+1))$.

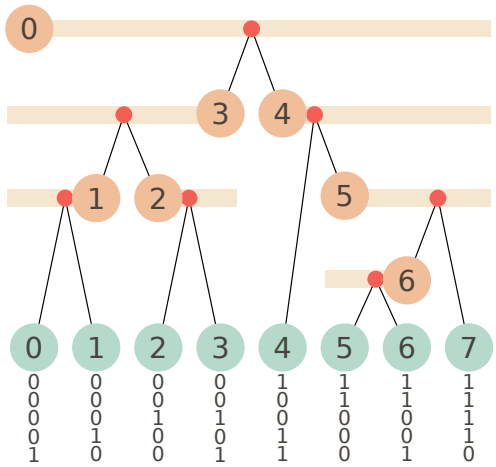


FIGURE 3.3 – **Structure de l'arbre** dans la méthode de Karras et al. [Kar12]. Une barre jaune représente l'ensemble de nœuds feuilles (en vert, avec leur code de Morton) couvert par un nœud interne (en orange). Un point rouge indique la position de la division pour un nœud.

Les feuilles f_i couvertes par le nœud interne i vérifient toutes $lcp(i, f_i) > lcp_{min}$. On effectue une recherche binaire pour trouver la dernière feuille j satisfaisant cette inégalité.

L'étape finale du traitement d'un nœud est de déterminer la position de la division de la plage, et ainsi les indices de ses fils. $lcp(i, j)$ nous donne la longueur du préfixe partagé par l'ensemble des codes de Morton correspondant à i. Grâce à une recherche binaire, on trouve la dernière position d vérifiant $lcp(i, d) > lcp(i, j)$. Les indices des fils de i seront donc d et $d + 1$ si i est le début de la plage, ou $d - 1$ et d si i est à la fin.

3.3.2 Intégrales de Morton

L'approche décrite ci-dessus nous permet de construire très rapidement un arbre sur des données ayant plusieurs dizaines de millions de primitives. Cependant, les applications que nous visons nécessitent de maintenir pour chaque nœud des informations supplémentaires, qui ne sont pas calculables directement dans la méthode de Karras et al. [Kar12].

Dans de nombreux cas (moyenne, nombre de primitives par nœud...), on a besoin pour un nœud interne de la somme des attributs de ses fils. Or, lorsqu'on souhaite traiter chaque nœud interne entièrement indépendamment des autres, les informations sur les fils ne sont pas disponibles.

Nous résolvons ce problème en calculant une *somme cumulée* sur les attributs des feuilles selon l'ordre de Morton. Cette opération est effectuée en parallèle sur GPU grâce à un scan inclusif (section 2.2.2). De façon similaire aux *images intégrales* [VJ01], cette "intégrale de

Morton" permet d'obtenir n'importe quelle somme d'attributs de feuilles consécutives en deux accès mémoire.

Par exemple, dans le cas d'un nœud i couvrant les feuilles r_1 à r_2 :

$$A_i = A_{scan}[r_2] - A_{scan}[r_1 - 1]$$

3.4 Algorithme de simplification

En se basant sur ces structures de données, nous construisons un algorithme de simplification adaptative entièrement parallèle. Il prend en entrée un maillage triangulaire indexé M à n sommets et un seuil d'erreur θ, et fournit en sortie un maillage indexé simplifié M'. L'algorithme se décompose en 5 étapes principales :

1. Les sommets de M, triés par code de Morton, définissent l'ensemble des nœuds feuilles. Une quadrique d'erreur est associée à chacun.

2. On calcule *l'intégrale de Morton*, qui est la somme cumulée des quadriques dans l'ordre de Morton.

3. On crée les nœuds internes (et leurs attributs) du kd-tree K en parallèle, en utilisant l'intégrale de Morton.

4. Une partition S de l'espace est définie comme les nœuds de K les plus hauts dans l'arbre ayant une erreur associée inférieure à θ. Les sommets du maillage simplifié sont les points représentatifs des sous-parties de S.

5. Les triangles du maillage d'entrée partagé par trois groupes de sommets différents de S sont ré-indexés sur leurs points représentatifs [RB93] et forment la connectivité du maillage final.

3.4.1 Initialisation

La première étape de l'algorithme consiste à trier les sommets de M par Morton code. Plutôt que trier directement le tableau des sommets (ainsi que les éventuelles couleurs et normales), on trie par clé un tableau d'indices (de 0 à n), en utilisant le code de Morton des sommets comme clé. On maintient aussi la table des correspondances de l'ordre initial des sommets vers l'ordre de Morton.

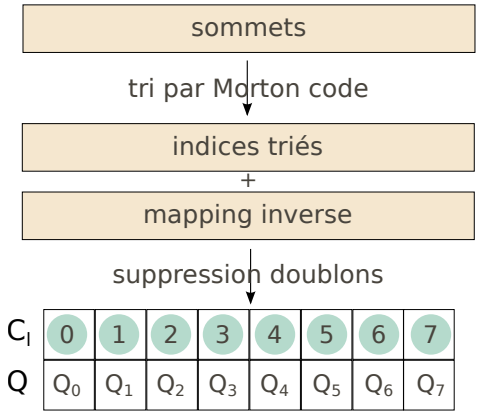


FIGURE 3.4 – **Initialisation.** On commence par trier les sommets dans l'ordre de Morton et générer un tableau de feuilles, pour lequel on calcule les quadriques d'erreur des feuilles.

La structure de données décrite précédemment nécessite que chaque feuille ait un code de Morton unique. Or, étant donné qu'on les calcule sur un nombre de bits limité, il est possible que deux sommets très proches aient le même code de Morton. Éliminer les doublons se fait très simplement en parallèle :

1. On marque la première occurrence de chaque code de Morton à 1, les autres à 0.
2. Une somme cumulée est calculée en parallèle (scan) sur ce marquage, ce qui nous donne la table des correspondances du tableau initial au tableau compacté, et le nombre d'éléments de ce dernier.
3. Enfin le tableau compact C_l est alloué.

On obtient ainsi les correspondances entre le tableau des sommets initial et le tableau représentant les feuilles de l'arbre : $V \rightarrow C_l$ (figure 3.4).

3.4.2 Initialisation des quadriques

Afin de pouvoir calculer la QEM par la suite, on a besoin de maintenir une quadrique par nœud de l'arbre. En effet, la QEM mesurant l'erreur d'approximation entre un point v et un ensemble de triangles P peut s'écrire [GH97] :

$$E(v) = v^T Q_P v \quad avec \quad Q_P = \sum_{t \in P} pp^T$$

où $p = [abcd]^T$ pour le plan d'équation $ax + by + cz + d = 0$ où $a^2 + b^2 + c^2 = 1$.

A partir de Q_P, un *point représentatif* de la géométrie minimisant la QEM peut être calculé en inversant Q_P. Dans le cas où le déterminant de Q_P est nul ou presque nul, on utilisera la position moyenne comme point représentatif [DT07].

Pour chaque nœud feuille l, on calcule une matrice quadrique symétrique de taille 4x4 Q_l. Pour cela, on calcule pour chaque triangle t sa quadrique fondamentale $Q_t = pp^T$, que l'on somme de façon atomique aux quadriques des feuilles correspondant aux sommets de t, en pondérant par l'aire w_t de t :

$$Q_l = \sum_{t \in l} w_t Q_t$$

En supposant que la discrétisation induite par le code de Morton est suffisamment fine, l'utilisation d'opérations atomiques cause peu de collisions entre les threads.

On garde également pour chaque feuille la position moyenne des sommets et leur nombre, ainsi qu'éventuellement la couleur et/ou la normale.

3.4.3 Construction du kd-tree

Grâce à notre tableau trié de nœuds feuilles, on construit notre kd-tree K en parallèle suivant la méthode de Karras et al. [Kar12] décrite précédemment. Pour chaque nœud interne i , nous avons besoin de sa quadrique Q_i et de son "sommet moyen" (position, couleur...) pour pouvoir calculer son point représentatif x_i et l'erreur d'approximation associée E_i. Q_i est la somme des quadriques des nœuds fils de i, ce qui est équivalent à la somme des quadriques de toutes les feuilles ayant i comme ancêtre.

Grâce à l'ordre de Morton, ces feuilles sont toutes contiguës , et par construction [Kar12], on connaît leurs indices (de r_1^i à r_2^i). En utilisant une intégration de Morton sur les attributs des nœuds feuilles, on a accès à leur somme en temps constant (figure 3.5).

On note que pour que cette intégration soit possible, on stocke nos quadriques sous forme de structure de tableaux, et non de tableau de structure (section 2.2.2). En effet, pour que cette intégration ait du sens, il est indispensable que les coefficients des quadriques soient sommés séparément. On stocke donc tous les premiers coefficients à la suite, puis les deuxièmes, et ainsi de suite.

On peut donc associer une quadrique à chaque nœud interne lors de la construction par la méthode de Karras et al. On en déduit le point représentatif x_i et l'erreur quadrique associée E_i (section 2.1).

$$x_i = -A_i^{-1} b_i \qquad\qquad E_i = Q_i(x_i)$$

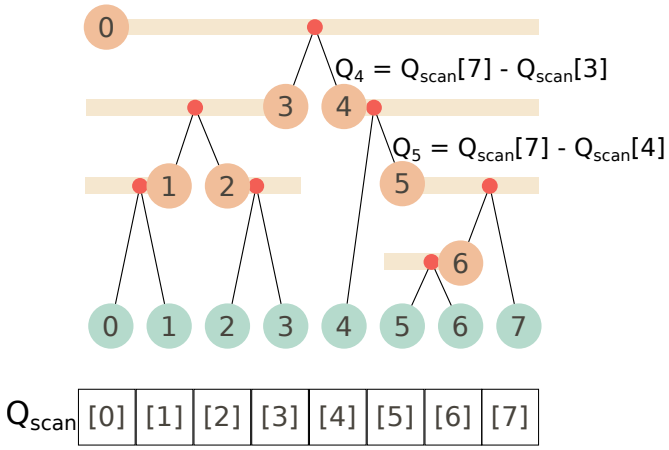


FIGURE 3.5 – **Construction de l'arbre.** On génère un kd-tree, en utilisant l'intégration de Morton pour traiter les nœuds internes en parallèle.

Dans les cas où A_i n'est pas inversible, on utilise le point moyen pour le nœud.

3.4.4 Échantillonnage adaptatif et remaillage

Pour extraire une partition adaptative de M à partir de notre arbre, on cherche la coupe de K formée par les nœuds les plus hauts dans la hiérarchie qui ont une erreur d'approximation associée inférieure à θ. Nous ramenons ce problème au calcul d'une table de correspondance associant chaque élément du tableau des feuilles C_l à un nœud de la coupe souhaitée.

Dans un premier temps, un tableau P, de la taille de C_l, est initialisé avec des zéros (l'indice de la racine). On lance un thread i pour chaque élément de C_l, qui va traverser l'arbre de la racine vers la $i$ème feuille. Lorsque le nœud courant, $P[i]$, a une erreur associée inférieure à θ, la traversée s'arrête. Dans le cas contraire, le nœud courant devient le fils gauche ou droit de $P[i]$, selon la valeur de i : Puisque les indices des nœuds fils correspondent à la position de la division dans le tableau des feuilles, si i est inférieur ou égal à l'indice du fils gauche, $P[i]$ devient le fils gauche. Sinon il devient le fils droit. De cette façon, la traversée se fait toujours vers la feuille d'indice i (algorithme 1 et figure 3.6). Chaque thread s'arrête sur un nœud de la coupe.

Lorsque tous les threads ont terminé, chaque sommet initial x de V peut être associé à un nœud de la coupe dans K en temps constant grâce à notre table donnant les correspondances $V \to C_l$, qui sont équivalente à $V \to P$.

Dans le tableau des points représentatifs des nœuds de K, on marque ceux qui appartiennent à la coupe pour effectuer un compactage. De la même façon que précédemment, on calcule

Algorithme 1 : Parcours parallèle de l'arbre.

pour tous les *nœuds feuilles i en parallèle* **faire**
  $c \leftarrow 0$
  $erreur \leftarrow +\infty$
  tant que $erreur > \theta$ *et* $c \in$ *nœuds internes* **faire**
    $r \leftarrow droite[c]$
    $l \leftarrow gauche[c]$
    si $i > l$ **alors**
      $c \leftarrow r$
    sinon
      $c \leftarrow l$
    fin
    $erreur \leftarrow erreurNoeud[c]$
  fin
  $P[i] \leftarrow c$
fin

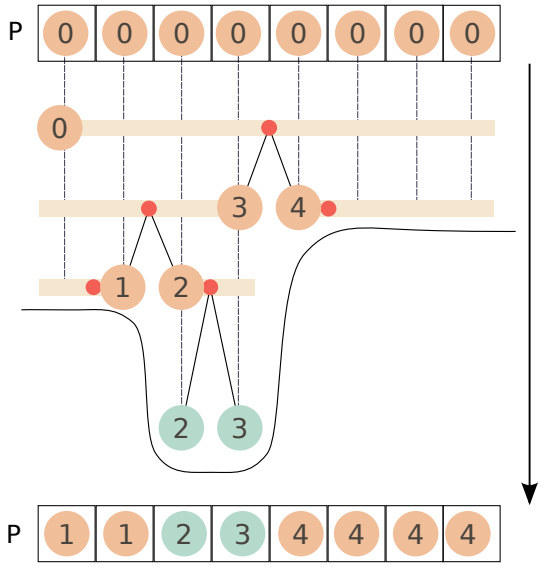


FIGURE 3.6 – **Extraction de la coupe.** On extrait une partition adaptative par rapport à l'erreur.

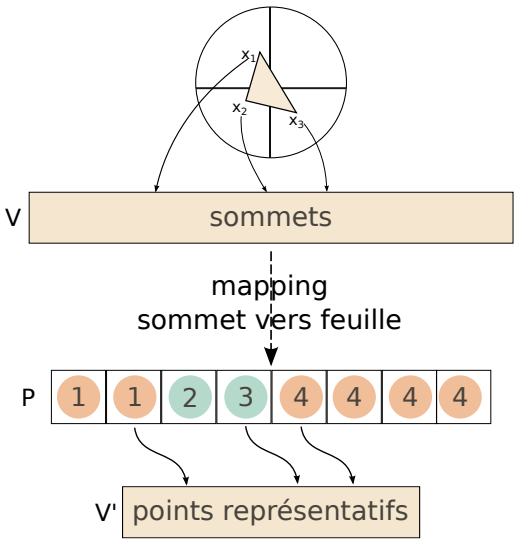


FIGURE 3.7 – **Re-maillage.** Le nouveau maillage est généré en se basant sur la connectivité du maillage en entrée.

en parallèle une somme cumulée sur ces marquages. Cela donne la taille du tableau V' des sommets de M', ainsi que pour chaque nœud de la coupe son indice dans V' (correspondance $P \rightarrow V'$).

Enfin, on génère la connectivité T' du maillage final en faisant une passe sur les triangles du maillage d'entrée M. On garde les triangles dont les sommets sont dans trois cellules différentes dans P, et on les ré-indexe sur les points représentatifs correspondants (figure 3.7).

3.4.5 Variations

Notre méthode de simplification peut prendre en compte différents attributs de sommets, des métriques d'erreur alternatives, et différents types de formats de géométrie en entrée.

Par exemple, l'attribut couleur peut être maintenu pour chaque nœud de la même façon que les matrices quadriques. Il est souvent intéressant de pondérer les couleurs moyennes par l'aire des triangles incidents. Cela demande deux tableaux supplémentaires : le premier accumule l'information de couleur pondérée (Col), l'autre les aires (A). Après intégration de Morton, la couleur de n'importe quel nœud i est donnée par :

$$Col_i = \frac{Col_{scan}[r_2^i] - Col_{scan}[r_1^i - 1]}{A_{scan}[r_2^i] - A_{scan}[r_1^i - 1]}$$

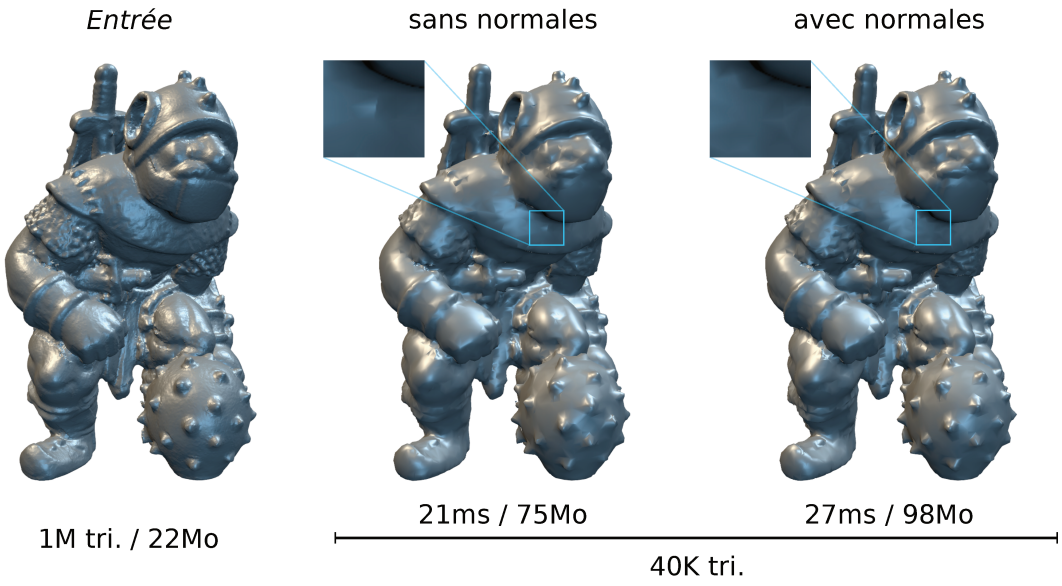


FIGURE 3.8 – **Influence des normales.** Sur ce modèle, des artefacts visuels dus au partitionnement disparaissent lorsque les normales sont maintenues, par exemple au niveau de l'épaule et sous la barbe.

La même technique peut être utilisée pour d'autres attributs comme les normales (figure 3.8).

Dans la dernière partie de l'algorithme, cette information supplémentaire peut influencer l'extraction de la coupe (figure 3.9).En effet, certaines caractéristiques importantes visuellement, qui n'existent qu'à travers la couleur des sommets, sont mieux préservées dans ce cas. Cela est particulièrement utile dans le domaine de la stéréo-vision, où la couleur est parfois plus caractéristique que la géométrie.

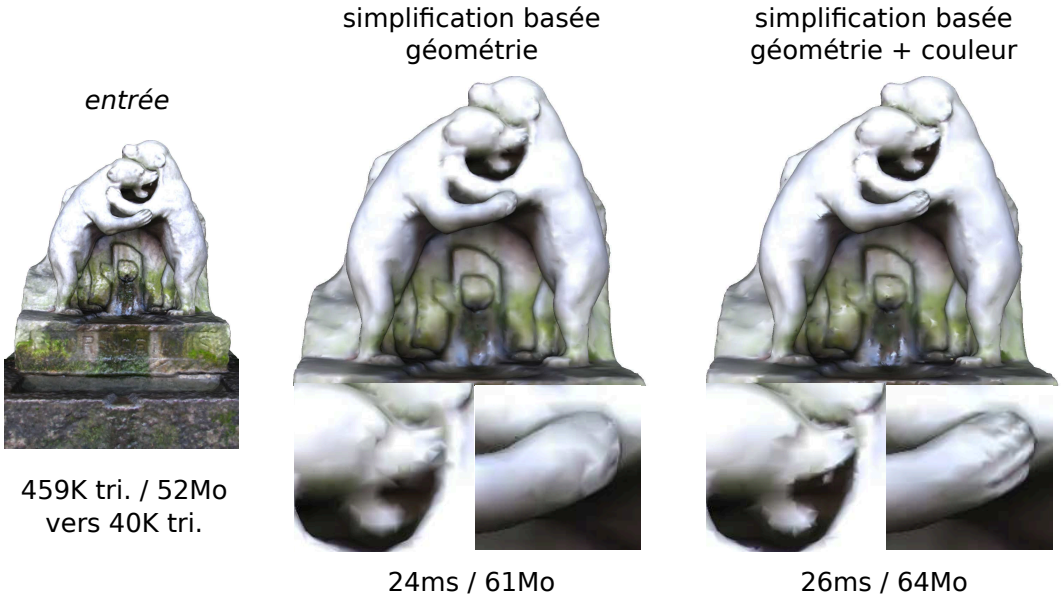
Un exemple simple peut être de fixer un seuil sur l'écart type pour la couleur en plus de l'erreur géométrique. Dans ce cas, un tableau de plus (Col^2) est nécessaire, pour accumuler la couleur pondérée au carré. Après le calcul de l'intégrale de Morton, l'écart type pour la couleur d'un nœud est donné par :

$$S_i = \sqrt{\frac{Col^2_{scan}[r^i_2] - Col^2_{scan}[r^i_1 - 1]}{A_{scan}[r^i_2] - A_{scan}[r^i_1 - 1]} - [Col_i]^2}$$

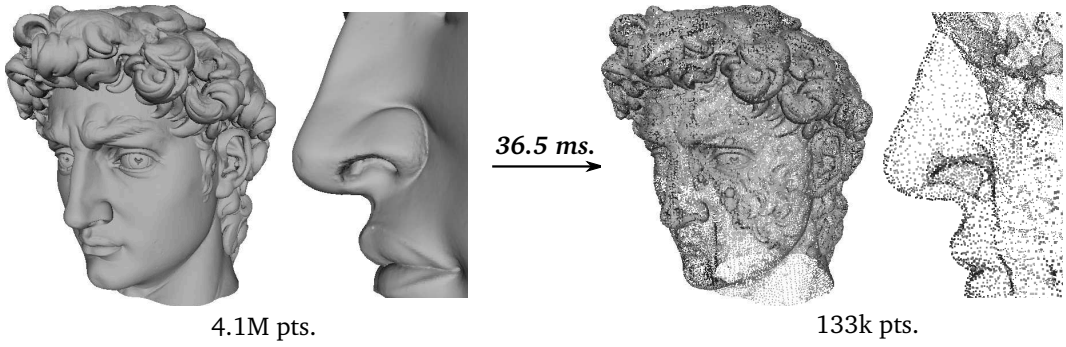
avec Col_i la couleur moyenne pour le nœud i.

Les nuages de points équipés de normales peuvent également être simplifiés avec notre méthode (figure 3.10) :

1. On calcule les matrices quadriques directement à partir des normales des points.



FIGUR E 3.9 – **Influence de la couleur.** Simplification prenant uniquement en compte la géométrie à gauche, simplification utilisant à la fois la géométrie et la couleur pour déterminer la coupe dans l'arbre à droite. Les détails présents uniquement dans l'information de couleur sont préservés.



FIGUR E 3.10 – **Simplification d'un nuage de points.**

2. Les normales sont propagées pour chaque nœud de la même façon que pour la couleur

3. On omet l'étape de maillage finale.

On note cependant qu'en l'absence d'information sur le rayon correspondant à un point, cette méthode ne fonctionne que dans le cas d'un échantillonnage relativement uniforme.

3.5 Implémentation et performances

Notre algorithme de simplification a été implémenté en C++/CUDA et les performances ont été mesurées sur un PC équipé d'une carte graphique GeForce GTX 680 et d'un CPU Intel Xeon 3.6GHz. Nous avons limité les codes de Morton à 30 bits afin de pouvoir les stocker sous forme d'entier 32 bits en mémoire sur le GPU.

Nous avons aussi implémenté une simplification uniforme très rapide (selon une grille), similaire à la méthode de Decoro et Tatarchuk [DT07] pour comparaison. Nous comparons également avec les résultats de haute qualité fournis par QSlim [GH97].

Dans le tableau 3.1, nous détaillons les timings des différentes étapes de notre algorithme pour une collection de modèles. On constate qu'on atteint des performances temps réel pour des maillages allant jusqu'à plusieurs millions de polygones, et que l'algorithme reste interactif même au-delà de 10 millions. Il est à noter que les performances passent à l'échelle linéairement, et que lors d'expérimentations avec une carte graphique plus puissante (GeForce GTX 980Ti), nous obtenons pour le modèle du crabe un temps total de 46 ms pour 11 millions de triangles. Alors que la construction de l'arbre et le ré-échantillonnage constituent une part très faible du temps total, le goulot d'étranglement se situe à l'étape d'initialisation. Remarquons cependant que pour les scénarios d'application générant plusieurs simplifications d'un même modèle (par exemple de la visualisation à distance avec plusieurs utilisateurs, ou du rendu dépendant du point de vue), cette étape est réalisée une unique fois.

Modèle	**#Triangles**	**Sortie #T**	**Init.**	**Arbre**	**Sampling**	**Maillage**	**Total**
Bunny	70K	4,300	3.3	1.8	0.1	1.2	6.4
Dragon	100K	9,300	6.6	2.8	0.1	1.5	11.0
Horse	225K	10,000	5.6	3.0	0.1	1.6	10.3
Buste	510K	19,200	9.0	3.5	0.5	1.9	14.9
Caesar	770K	18,500	11.1	4.4	0.9	2.1	18.6
Grog	1M	41,000	12.5	5.1	0.9	2.3	20.8
Gargoyle	1,7M	44,500	14.9	4.1	0.6	2.5	22.1
Raptor	2M	22,500	16.5	1.7	0.2	2.2	20.6
Neptune	4M	24,500	37.7	2.4	0.3	6.6	47.0
Crab	11M	64,200	88.3	6.0	0.9	11.4	106.5
Lucy	28M	116,500	192.5	5.5	0.8	23.9	222.7

TABLE 3.1 – **Mesures de performance** en ms, sans les transferts mémoire CPU-GPU. **Init.** : tri, calcul des quadriques des feuilles et intégration de Morton. **Arbre** : construction de l'arbre. **Sampling** : traversée de l'arbre et extraction de la coupe. **Maillage** : ré-indexation des triangles.

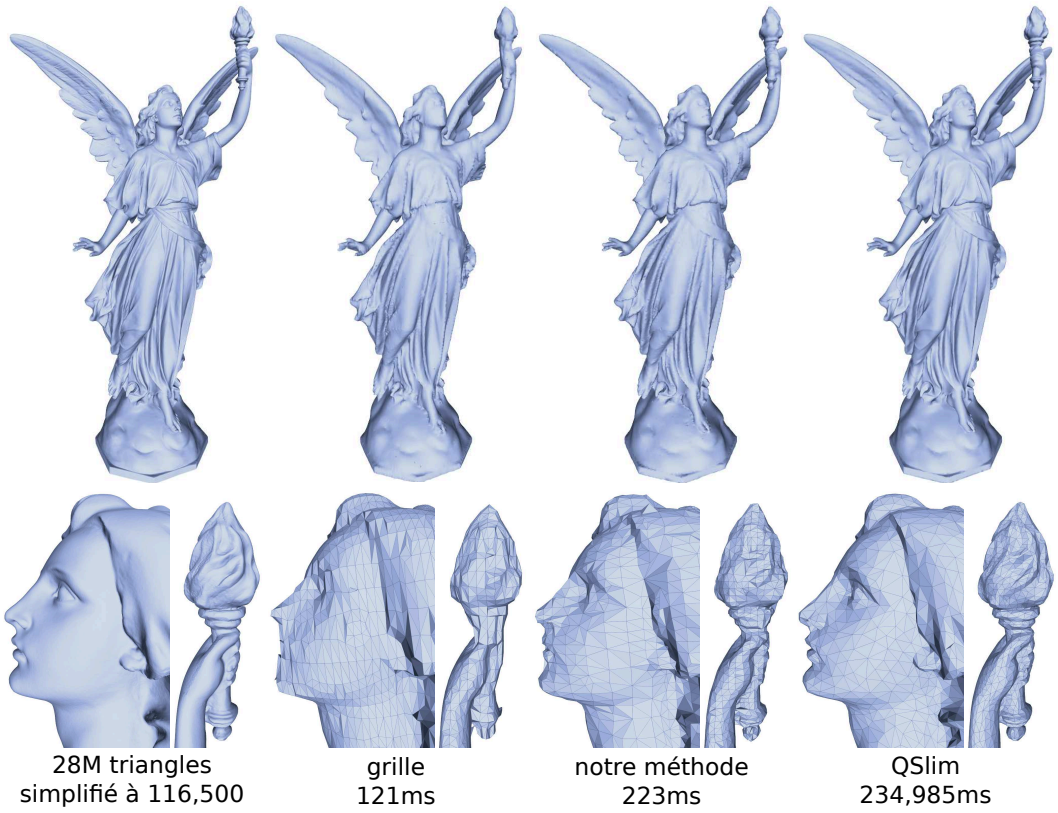


FIGURE 3.11 – **Simplification adaptative** du modèle Lucy. Notre approche donne une meilleure approximation des détails et des structures saillantes qu'un partitionnement uniforme, tout en restant interactive pour ce modèle de 28M de triangles.

En termes de qualité visuelle, notre partition adaptative du modèle préserve les caractéristiques importantes, avec de petits triangles autour des détails et des plus grands dans les régions plus plates (fig. 3.11).

Dans le tableau 3.2, nous présentons l'utilisation mémoire pour le même ensemble de modèles. L'espace nécessaire sur le GPU dépend directement du nombre de sommets de la géométrie en entrée et du nombre de bits utilisés pour les codes de Morton. Dans le pire scénario possible, lorsque le code de Morton est suffisamment précis pour être différent pour chaque sommet en entrée (par exemple, pour le modèle "bunny"), il y aura autant de nœuds feuilles à stocker, et donc autant de quadriques, couleurs, normales... Cependant, quand la taille du modèle augmente, de plus en plus de sommets partagent un code de Morton identique, et les nœuds feuilles ne sont donc pas aussi nombreux que les sommets.

On compare notre méthode avec un partitionnement GPU régulier (selon une grille), qui est très rapide mais non adaptatif. Pour un nombre comparable de triangles, notre méthode pré-

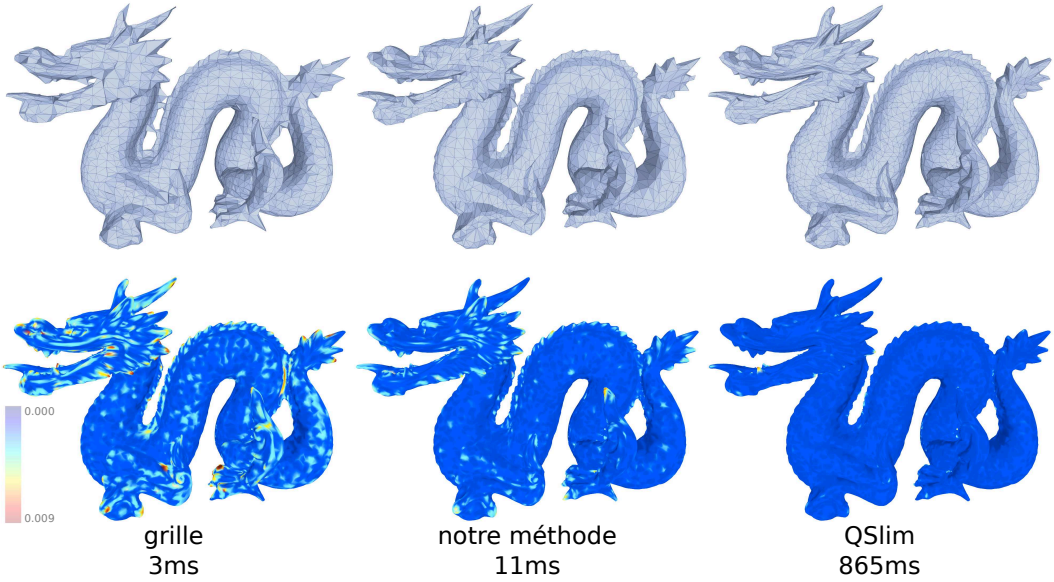


FIGURE 3.12 – **Visualisation de l'erreur** pour le modèle du dragon de Stanford (100K triangles), simplifié à 9300 triangles.

serve des détails visuellement importants qui disparaissent avec le partitionnement régulier. Le temps de simplification, bien que plus long, reste dans le même ordre de grandeur que pour le partitionnement par grille.

On compare également nos résultats avec QSlim, qui privilégie la qualité plutôt que les performances. Nous présentons les timings et les mesures de l'erreur d'approximation pour ces différentes méthodes dans le tableau 3.3. Les mesures ont été réalisées avec l'outil Metro [CRS98]. On représente visuellement cette erreur de simplification pour les trois approches dans la figure 3.12.

Enfin, la figure 3.13 montre des résultats d'expériences effectuées sur des données animées, en particulier en performance capture [dAST$^+$, BHB$^+$11]. D'un côté un modèle relativement simple (40K triangles) est simplifié par un facteur 5, de l'autre un modèle de visage très haute résolution (2,3M triangles) est simplifié drastiquement par un facteur 100 en temps réel (30 frames par seconde).

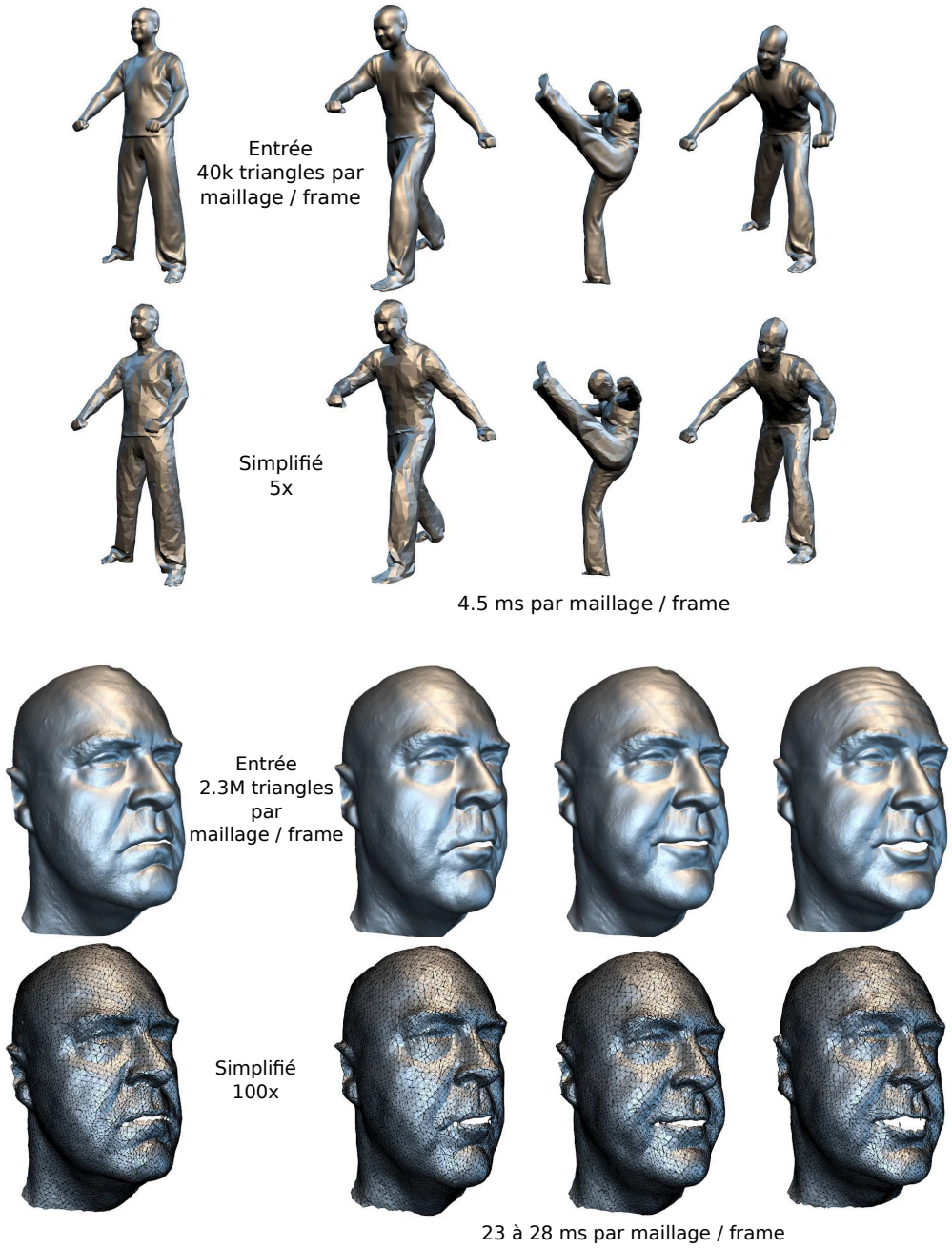


FIGURE 3.13 – **Simplification adaptative** de données issues de performance capture. **En haut :** 500 frames sont traitées indépendamment à environ 200 Hz. Données capturées fournies par le Max-Planck-Center for Visual Computing and Communication (MPI Informatik/Stanford) **En bas :** Le sous-échantillonnage de ce modèle de visage dense (2,3M triangles par frame) est réalisé en temps réel. Données visage fournies par ETH Zurich [BHB+11].

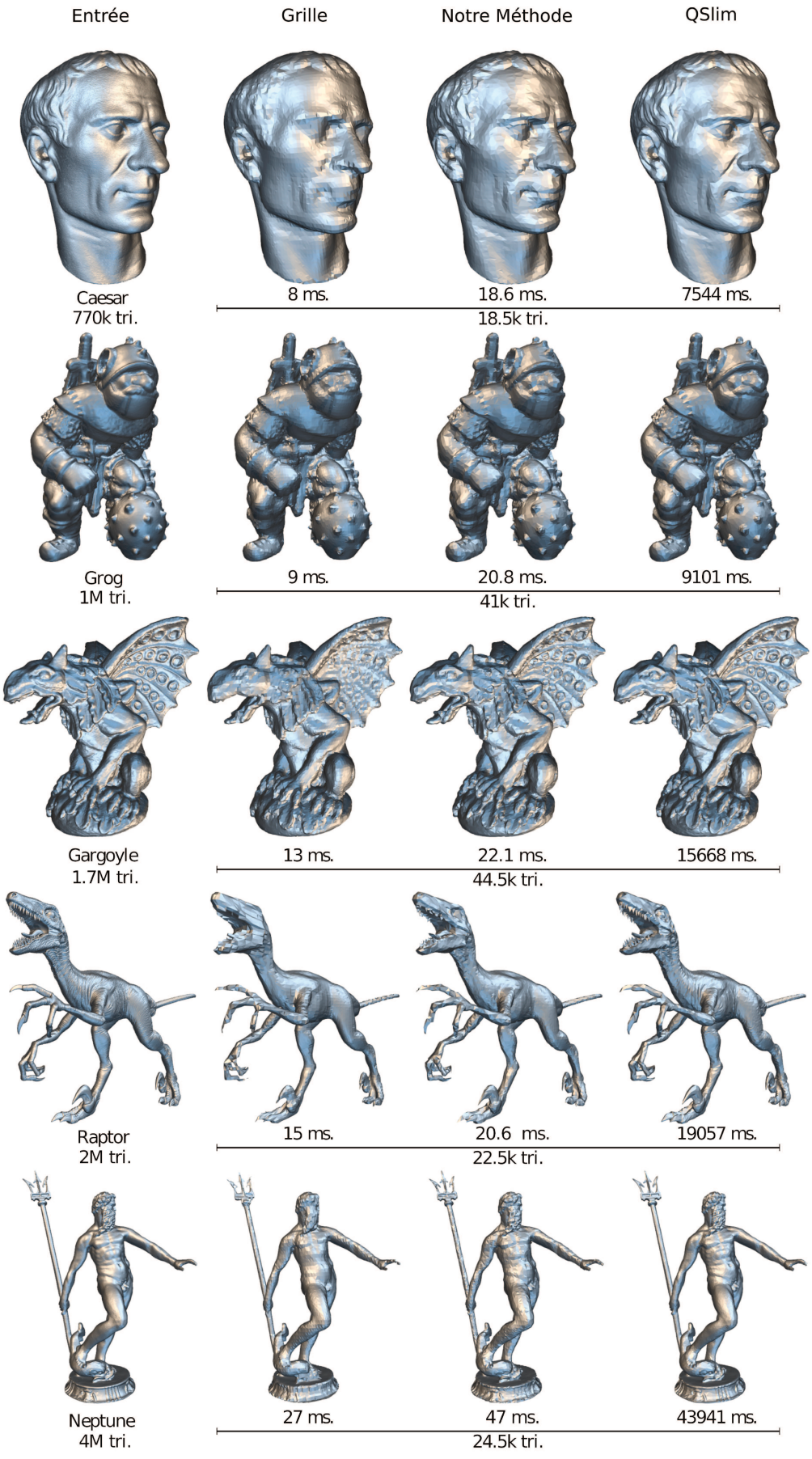


FIGURE 3.14 – Exemples.

3.6 Discussion

Limitations

Bien qu'adaptative, notre approche reste une méthode de *partitionnement*, ce qui implique au moins deux limitations. Tout d'abord, cette famille d'algorithmes ne donne aucune garantie sur la préservation de la topologie. Dans le cas de fortes simplifications, la topologie du modèle peut changer, joignant des parties proches ou fermant des trous. Même si cela constitue parfois un avantage [CSAD04], un contrôle sur ces changements serait intéressant.

Ensuite, le nombre de polygones/sommets du modèle en sortie n'est pas directement contrôlable. L'utilisateur règle le niveau de simplification à l'aide du seuil θ, mais garantir un nombre exact de primitives peut se révéler fastidieux.

Notre intégration de Morton est aussi limitée par la précision de la machine : lorsque l'on calcule des sommes cumulées sur de très grands tableaux, l'imprécision s'accumule et peut causer des instabilités lors de l'inversion des matrices quadriques. On réduit fortement ces instabilités en passant toutes les coordonnées à l'échelle du cube unitaire, et en utilisant une précision double pour le calcul des sommes cumulées. Cependant, l'erreur induite reste significative lorsqu'on traite de très grands modèles (centaines de millions de polygones). Résoudre ce problème tout en préservant le niveau de performance est une des perspectives d'amélioration principales.

Modèle	**taille du modèle**	**espace supplémentaire requis**	
		27 bits MC	30 bits MC
Bunny	1	7	7
Dragon	2	8	9
Horse	5	19	20
Buste	11	32	37
Caesar	17	61	63
Grog	22	75	80
Gargoyle	39	98	120
Raptor	45	44	89
Neptune	91	83	138
Crab	259	217	275
Lucy	642	502	557

TABLE 3.2 – **Utilisation mémoire** en Mo

Modèle	Méthode	#T	Sortie #T	Temps	H	M12	M21
Bunny	HSGS	70K	4,300	6.4	0.013802	0.000500	0.000499
	QSlim			503	0.002425	0.000295	0.000288
	Grille			2.5	0.012880	0.001327	0.001294
Dragon	HSGS	100K	9,300	11.0	0.008810	0.000671	0.000585
	QSlim			865	0.009327	0.000370	0.000251
	Grille			3	0.013838	0.001142	0.001039
Horse	HSGS	225K	10,000	10.3	0.004485	0.000280	0.000280
	QSlim			1,728	0.001862	0.000104	0.000101
	Grille			4	0.009340	0.000588	0.000555
Buste	HSGS	510K	19,200	14.9	0.003257	0.000237	0.000236
	QSlim			4,419	0.001365	0.000095	0.000094
	Grille			6	0.007113	0.000505	0.000490
Caesar	HSGS	770K	18,500	18.6	0.013818	0.000220	0.000209
	QSlim			7,544	0.013894	0.000130	0.000124
	Grille			8	0.014448	0.000431	0.000413
Grog	HSGS	1M	41,000	20.8	0.005253	0.000255	0.000249
	QSlim			9,101	0.003686	0.000128	0.000124
	Grille			9	0.007022	0.000531	0.000482
Gargoyle	HSGS	1,7M	44,500	22.1	0.006374	0.000236	0.000237
	QSlim			15,668	0.004018	0.000120	0.000118
	Grille			13	0.006929	0.000496	0.000469
Raptor	HSGS	2M	22,500	20.6	0.012457	0.000280	0.000277
	QSlim			19,057	0.011525	0.000143	0.000137
	Grille			15	0,012629	0.000423	0.000423
Neptune	HSGS	4M	24,500	47.0	0.005375	0.000272	0.000282
	QSlim			43,941	0.001851	0.000089	0.000082
	Grille			27	0.008222	0.000487	0.000450
Crab	HSGS	11M	64,200	106.5	0.005439	0.000233	0.000244
	QSlim			92,933	0.002617	0.000057	0.000055
	Grille			53	0.004677	0.000319	0.000315
Lucy	HSGS	28M	116,500	222.7	0.008423	0.000185	0.000179
	QSlim			234,985	0.000894	0.000035	0.000033
	Grille			121	0.003576	0.000211	0.000203

TABLE 3.3 – **Comparaison des timings et de la qualité** avec **H** la distance de Hausdorff entre le modèle original et la simplification, **M12** la distance moyenne du modèle original à la simplification et **M21** la distance moyenne de la simplification vers le modèle original. Les mesures de temps sont données en ms.

Conclusion

Nous avons présenté un algorithme de simplification adaptative haute performance capable de traiter des objets constitués de millions de polygones en temps réel sur du matériel grand public. Nous parvenons à atteindre ces performances en introduisant le concept *d'intégration de Morton*, c'est à dire le calcul de sommes cumulées sur les attributs ou les mesures d'erreur le long de la courbe de Morton. Cette opération intermédiaire permet de construire et de traverser en parallèle une structure hiérarchique, de laquelle on extrait une partition adaptative du modèle. Cela permet à notre méthode d'obtenir des maillages simplifiés à la volée et de manière adaptative, avec une meilleure qualité de simplification que les méthodes haute performance actuelles. Nous avons également montré que notre algorithme peut être étendu afin de prendre en compte les éventuels attributs de la surface (normales, couleur), et peut être adapté pour traiter des nuages de points munis de normales.

Au-delà de la simplification, nous pensons que l'intégration de Morton peut bénéficier à d'autres types d'applications, notamment de traitement géométrique, devant manipuler rapidement et en parallèle des données multi-résolutions. En outre, nous avons vu que les quadriques d'erreur constituent de riches descripteurs géométriques locaux, qui ne sont pas limités à la simplification. Nous nous proposons dans la suite de les appliquer à un autre opérateur de traitement géométrique fondamental : le filtrage rapide de surfaces.

CHAPITRE 4

FILTRAGE GÉOMÉTRIQUE BILATÉRAL PAR QUADRIQUES

Comme nous l'avons vu au chapitre précédent avec la simplification, de plus en plus de méthodes de traitement géométrique sont capables de s'exécuter en temps interactif, voire en temps réel, pour des quantités importantes de données 3D.

Cependant, la plupart des méthodes font la supposition que leurs données en entrée ne sont pas, ou peu bruitées, ce qui n'est pas le cas en pratique. Dans ce chapitre, nous détaillons de nouvelles approches de filtrage rapide de ce type de données.

4.1 Contexte

Les maillages générés par des scanners présentent généralement un bruit de haute fréquence et de basse amplitude, c'est à dire des perturbations faibles de la position des sommets, qui ne correspondent pas à des singularités de la forme (arêtes vives, coins).

L'objectif d'un filtre de débruitage est de lisser ces artefacts de façon à ce que la forme globale, les éléments de basse-fréquence, soit préservée. Une autre propriété souhaitable dans ce contexte est la préservation de certaines caractéristiques de la surface propres à la forme, comme les arêtes vives et les coins. En effet, la plupart des formes ne sont lisses que par morceaux, et un débruitage de type filtre passe-bas simple supprime ou atténue ces caractéristiques, qui sont de haute-fréquence comme le bruit.

Les algorithmes de débruitage doivent être capables de gérer des ensembles de données de grande taille efficacement. Il est en général nécessaire d'appliquer un filtre adapté avant la plupart des algorithmes de traitement géométrique, juste après l'acquisition. Dans le cadre

du traitement en temps réel ou en temps interactif, ce filtrage ne doit pas compromettre les performances attendues de l'algorithme auquel sont destinées les données filtrées.

Un élément clé des méthodes de filtrage préservant les détails tient à la capacité à détecter que deux échantillons de surfaces donnés appartiennent à la même zone lisse de la géométrie, autorisant ainsi leur mélange pour filtrer sans risque d'endommager les structures singulières. Cependant, contrairement au cas du filtrage d'image par exemple, paramétrisation (coordonnées d'un échantillon) et signal (valeur d'un échantillon) ne sont pas clairement distincts dans le cas de la géométrie 3D. Il est donc nécessaire d'augmenter la géométrie nue d'une information supplémentaire, permettant d'établir les relations de similarité en amont de la phase de filtrage.

Nous proposons d'enrichir la géométrie des surfaces 3D avec des attributs compacts basés sur la QEM, qui décrivent bien la géométrie localement. En particulier, nous proposons de les exploiter dans le cadre d'un filtrage robuste, et illustrons leur utilisation au travers de plusieurs variations de l'algorithme.

En comparaison avec les filtres bilatéraux classiques, notre méthode s'avère aussi simple à mettre en place, mais s'appuie sur une estimation plus complexe de la géométrie locale, plus robuste dans des conditions de bruit important. Au-delà du débruitage, notre approche permet, à des échelles plus grandes, d'extraire une forme simple, dont les caractéristiques principales sont rehaussées. Selon les paramètres, cet effet ira d'une légère accentuation des arêtes vives à une stylisation extrême de la forme.

L'algorithme est conçu pour être entièrement parallélisable et s'adapter naturellement au GPU. De plus, son coût en termes de mémoire est linéaire. Par conséquent, il permet un filtrage haute performance à la demande pour des applications interactives manipulant des maillages de plusieurs millions de polygones.

4.2 Travaux Antérieurs

Les méthodes de débruitage de maillages sont en général fondées sur les approches utilisées en traitement d'images. L'adaptation de ces algorithmes aux surfaces 3D est cependant loin d'être directe. En effet, dans le cas des images, la position des échantillons et le signal (intensité ou couleur d'un pixel) sont séparés. Dans le cas des maillages, le signal est également une information spatiale, et est confondu avec la position des échantillons.

Les premières méthodes de lissage de maillages se basent sur le principe du filtre passe-bas en traitement du signal, avec Taubin [Tau95], qui introduit l'opérateur Laplacien pour les surfaces 3D discrètes.

Desbrun et al. [DMSB99] partent de l'observation que cet opérateur de lissage Laplacien peut être vu comme une intégration sur le temps de l'équation de la chaleur, et présentent une méthode de lissage basée diffusion, adaptée aux maillages irréguliers. L'inconvénient de ces approches isotropes est qu'elles ne font pas la différence entre le bruit et les caractéristiques de haute-fréquence du maillage, les deux seront lissés sans distinction. Or, dans le cadre du débruitage, on cherche au maximum à conserver ces éléments propres à la forme.

Des méthodes visant à préserver ces caractéristiques ont donc été développées, avec en premier lieu des algorithmes basés sur une diffusion anisotrope [DMSB00, CDR00]. Plusieurs techniques s'appuient sur une étape de lissage des normales, avant de mettre à jour la position des sommets pour que la surface corresponde aux normales calculées [OBS02, YOB02, YOB03, ZFAT11].

Le filtre bilatéral pour les images a été introduit par Tomasi et al. [TM98]. Dans cette méthode, l'échantillon est filtré en utilisant une moyenne des échantillons de son voisinage, pondérés par deux termes. Le premier correspond à une pondération spatiale classique, le second est basé sur la différence de valeur entre le pixel considéré et le voisin. Autrement dit, chaque pixel devient une moyenne pondérée des pixels similaires de son voisinage.

Le principe a été adapté aux maillages 3D par Jones et al. [JD03] et Fleishman et al. [FDCO03], et plus tard au lissage de normales pour le débruitage [ZFAT11]. Nous nous focalisons ici sur la formulation de Jones et al. [JD03]. Dans ces travaux, le lissage se fait par une estimation locale robuste de la forme. La position d'un sommet est calculée comme une moyenne de prédictions obtenues à partir des triangles du voisinage. Pour un sommet à filtrer v, et un triangle t_i de son voisinage V, cette prédiction est définie comme la projection de v sur le plan tangent de t_i. Ce plan tangent est défini par un centre c_i et une normale lissée n_i.

$$P(v) = v + n_i((c_i - v).n_i)$$

Le sommet filtré v' est calculé comme une moyenne de ces prédictions, pondérées d'une part par un terme spatial, la distance au voisin $\|v - c_i\|$, et d'autre part un terme d'influence, la distance à la projection calculée $\|v - P(v)\|$ (figure 4.1). On pondère également par l'aire a_t du triangle.

On utilise des noyaux Gaussiens G_{σ_s} et G_{σ_r} pour le terme spatial et le terme d'influence, d'écarts types σ_s et σ_r respectivement. Ces paramètres permettent de faire varier le niveau de lissage de la surface : σ_s contrôle le nombre de voisins pris en compte pour l'estimation, et σ_r détermine la taille des caractéristiques de la surface à préserver, en donnant plus ou moins d'importance aux voisins situés au-delà d'une arête vive.

$$v' = \frac{1}{w(v)} \sum_{t_i \in V} P_{t_i}(v) a_{t_i} G_{\sigma_s}(\|v - c_i\|) G_{\sigma_r}(\|v - P_{t_i}(v)\|)$$

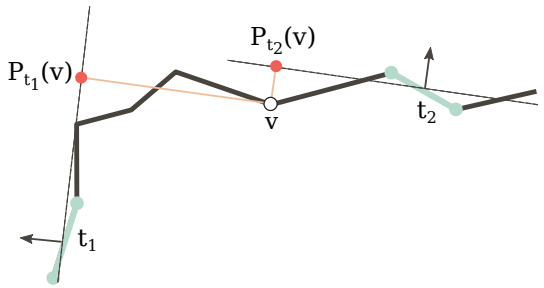


FIGURE 4.1 – Prédictions utilisées dans le filtre bilatéral (normales lissées). La prédiction donnée par le voisin t_2 du même côté de l'arête vive est plus proche de v que celle donnée par le voisin t_1 de l'autre côté. Elle aura donc plus d'influence.

où $w(v)$ est la somme des poids :

$$w(v) = \sum_{t_i \in V} a_{t_i} G_{\sigma_s}(\|v - c_i\|) G_{\sigma_r}(\|v - P_{t_i}(v)\|)$$

Les normales sont lissées en pré-traitement avec un noyau Gaussien d'écart type $\frac{\sigma_s}{2}$. Cela permet de rendre les prédictions basées sur les plans tangents des triangles moins sensibles au bruit.

Sa capacité à préserver les caractéristiques saillantes de la surface, son caractère local et sa simplicité font du filtre bilatéral une référence en matière de lissage et de débruitage. Cela a donné lieu à plusieurs travaux visant à améliorer son efficacité, avec par exemple des structures d'accélération comme la grille bilatérale [CPD07] et le kd-tree Gaussien [AGDL09], ou encore une approximation séparable du filtre [VB11].

Toujours inspiré du débruitage d'images, le filtre par moyennes non-locales (*NL-means* [BCM05]) a été le point de départ de plusieurs adaptations aux surfaces 3D [YBS06, GAB12]. On cherche ici à utiliser parmi tous les sommets, ceux qui ont le voisinage le plus proche du sommet à filtrer, indépendamment de leur position. Cependant, en termes de performances le filtrage bilatéral reste un bon compromis.

Plus récemment, Wang et al. [WFL$^+$15] proposent de filtrer les détails géométriques à une échelle donnée, en appliquant itérativement aux normales un filtre bilatéral joint [KCLU07], suivant l'idée développée pour les images par Zhang et al. [ZSXJ14] avec le *rolling guidance filter*. Le problème est approché différemment par Wang et al. [WLT16], qui utilisent des réseaux de neurones pour apprendre la relation entre la géométrie bruitée et sa version non bruitée.

Nous avons vu dans les chapitres précédents que la QEM [GH97] est un bon moyen d'estimer localement la forme de la surface dans le cadre de la simplification (section 3.2), et

permet d'obtenir des modèles simplifiés où les arêtes saillantes et les coins sont préservés. D'autres travaux étendent la formulation de la QEM pour effectuer une simplification tenant compte du bruit [PSK+02, Gra04]. Mais l'utilisation de la QEM ne se limite pas aux algorithmes de simplification. Par exemple, Ohtake et al. [OBA+05] utilisent des fonctions d'erreur quadriques pour la reconstruction de surfaces. Thiery et al. [TGB13] étendent la QEM pour trouver la sphère correspondant le mieux à un ensemble de plans, et ainsi approximer automatiquement un modèle par un maillage de sphères.

Dans le cadre du débruitage, l'utilisation de la QEM a notamment été examinée par Nociar et al. [NF10]. Après une première étape de filtrage bilatéral itératif des normales, une quadrique est associée à chaque sommet, comme la somme des quadriques fondamentales des triangles adjacents, calculées avec les normales lissées. Les sommets sont ensuite déplacés itérativement, en minimisant la quadrique du sommet, puis en mettant à jour cette quadrique à l'aide des nouvelles positions.

Vieira et al. [VNMC10] proposent un algorithme simple utilisant également une quadrique par sommet. Pour cela, ils définissent un 1-voisinage comme l'ensemble des sommets partageant une arête avec le sommet considéré, un 2-voisinage comme ceux partageant une arête avec le 1-voisinage, et ainsi de suite récursivement. La quadrique d'un sommet est calculée comme la somme des quadriques fondamentales du voisinage, avec un poids basé sur la distance au voisin. Le filtrage est ensuite réalisé en une seule étape, le sommet étant déplacé vers l'optimiseur de sa quadrique. Cet algorithme simple préserve les caractéristiques à l'échelle du voisinage choisi, et va même jusqu'à les rehausser. Il illustre également la capacité de la QEM à estimer localement la surface de manière robuste, en récupérant certains détails dans des conditions de bruit important. Son application reste cependant limitée à des voisinages de petite taille (au maximum d'ordre 2 en pratique), et donne peu de contrôle sur l'échelle à laquelle les caractéristiques sont préservées ou exagérées.

4.3 Algorithmes de filtrage

4.3.1 Filtrage simple basé QEM

Nous proposons tout d'abord d'utiliser la QEM pour effectuer un filtrage simple des sommets.

De manière similaire à la méthode de Vieira et al. [VNMC10], nous enrichissons les sommets du maillage avec une quadrique calculée à partir du voisinage. Toutefois, afin de ne pas dépendre de la régularité d'échantillonnage et de la connectivité du maillage d'entrée, nous optons pour une définition de voisinage différente. Les voisins sont pondérés par un noyau Gaussien d'écart type σ_q. On va donc en pratique examiner tous les sommets dans

un rayon exprimé en fonction de σ_q autour du point considéré. On pondère également par l'aire des triangles.

Une quadrique de base Q_v est associée à chaque sommet v, comme la moyenne, pondérée par l'aire, des quadriques fondamentales des triangles adjacents. Dans un second temps, on va calculer la quadrique Q'_v de v pour son voisinage V_{σ_q} :

$$Q'_v = \frac{1}{w(v)} \sum_{p \in V_{\sigma_q}} Q_p a_p G_{\sigma_q}(\|v - p\|)$$

avec a_p l'aire associée au sommet p, calculée comme la somme des aires des triangles adjacents. Le terme de normalisation $w(v)$ est la somme des poids :

$$w(v) = \sum_{p \in V_{\sigma_q}} a_p G_{\sigma_q}(\|v - p\|)$$

Cette quadrique par sommet est un bon estimateur local de la forme, et capture bien les caractéristiques de la surface à l'échelle du voisinage choisi. On notera néanmoins que la quadrique en question n'est pas un champ scalaire approximant la forme, mais un espace d'optimisation caractéristique à la forme localement.

L'étape suivante consiste à mettre à jour la position des sommets grâce à leur quadrique. Pour cette première méthode, nous suivons l'approche originale utilisée par Garland et al. [GH97], et reprise en débruitage dans [VNMC10, NF10], à savoir placer le nouveau point v' à l'optimiseur de la quadrique (section 2.1). La connectivité du maillage n'est pas modifiée.

Comme observé par Vieira et al. [VNMC10], le filtrage basé sur les quadriques a tendance à rehausser les arêtes vives. En effet, optimiser la quadrique d'un sommet pour un voisinage comportant une caractéristique saillante donnera un point proche de cette caractéristique, car c'est en général le point qui minimise le mieux l'ensemble des distances aux plans tangents du voisinage (figure 4.2). Cette accentuation est illustrée dans la figure 4.3.

Bien que ce comportement puisse être souhaitable dans certains cas, le maillage résultant aura une régularité très différente du maillage d'entrée, avec une concentration de sommets beaucoup plus importante autour des caractéristiques (figure 4.4). Nous proposons donc une extension simple pour les cas où on doit privilégier une régularité du maillage similaire à celle de l'entrée. Plus précisément, afin d'éviter la dérive des sommets dans l'algorithme précédent, on contraint le déplacement du sommet dans la direction de sa normale. Nous utilisons pour cela une normale lissée à l'aide d'un noyau Gaussien d'écart type σ_n, afin d'avoir une direction robuste au bruit.

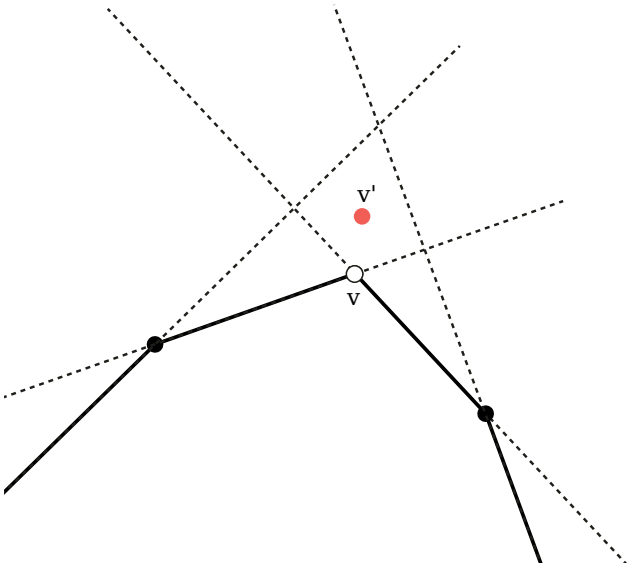


FIGURE 4.2 – Filtrage d'un sommet situé sur une caractéristique saillante.

Au lieu d'utiliser le minimiseur global de la quadrique Q du sommet comme précédemment, on le calcule le long de la normale. On a alors le sommet filtré v' :

$$v' = v + \lambda n$$

où λ est calculé comme suit :

$$\lambda = \frac{-(v^T A n + b^T n)}{n^T A n} \quad \text{avec} \quad Q = \left[\begin{array}{c|c} A & b \\ \hline b^T & c \end{array}\right]$$

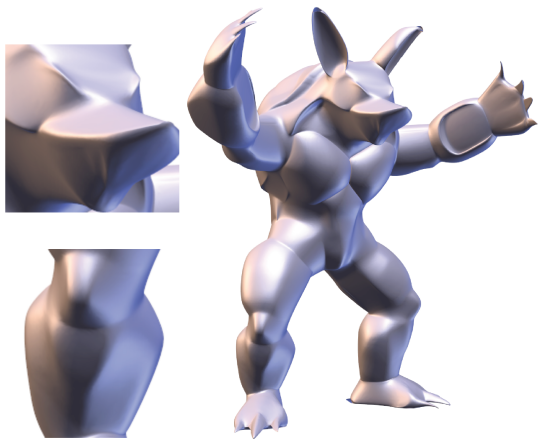
On garde ainsi les bonnes propriétés de préservation des caractéristiques et de robustesse au bruit. Les caractéristiques sont en revanche moins accentuées, on choisira donc la variation appropriée aux traitements auxquels est destiné le maillage filtré (figure 4.4).

(a) maillage initial

(b) $\sigma_q = 1$

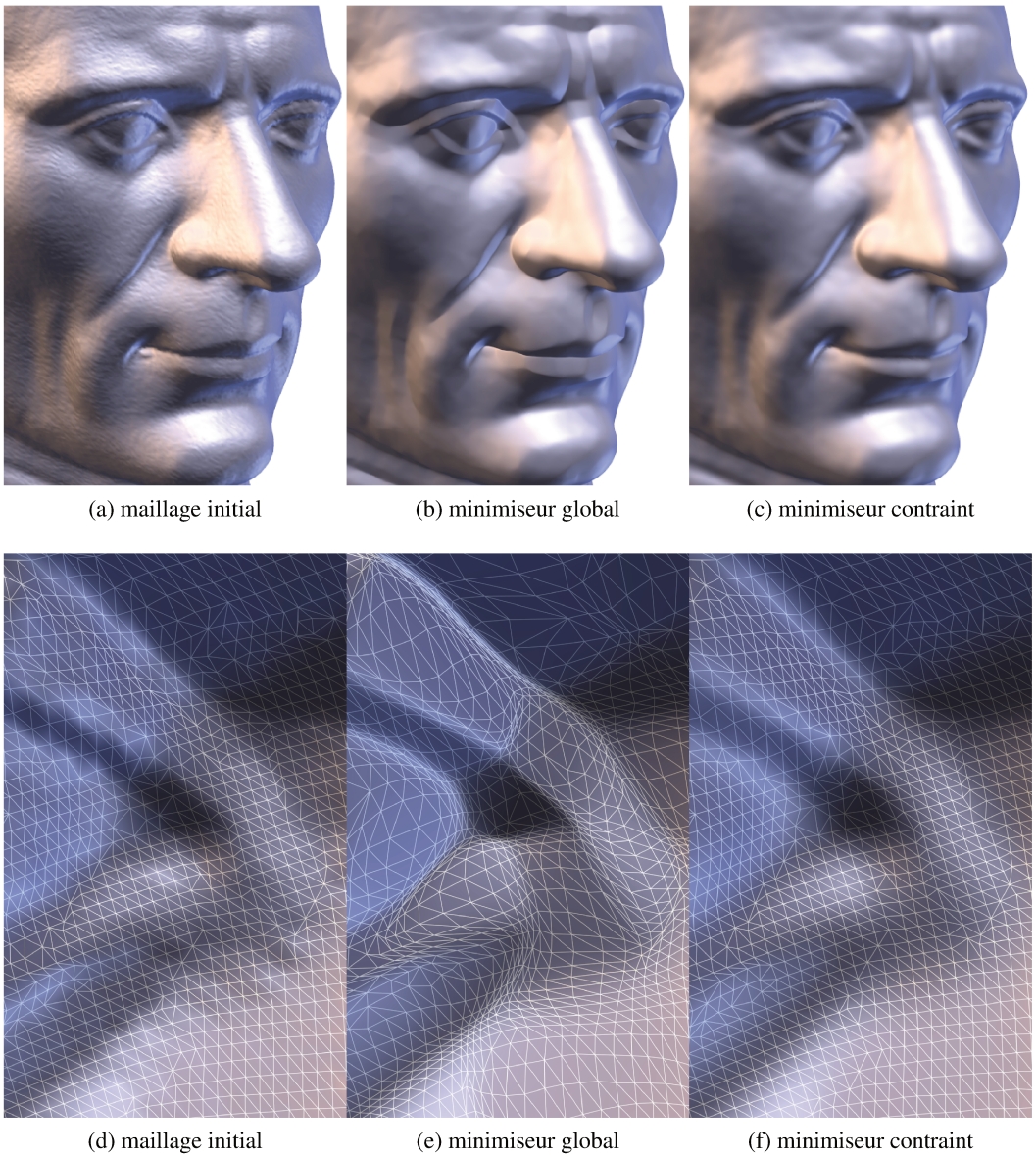
(c) $\sigma_q = 2$

(d) $\sigma_q = 3.5$



(e) $\sigma_q = 6$

FIGURE 4.3 – Filtrage simple basé QEM, pour plusieurs valeurs de σ_q. σ_q est exprimé en fonction de la longueur moyenne des arêtes du maillage. On observe l'accentuation des arêtes vives à l'échelle de σ_q, même pour des niveaux de lissage très élevés. Les structures plus petites disparaissent.



(a) maillage initial (b) minimiseur global (c) minimiseur contraint

(d) maillage initial (e) minimiseur global (f) minimiseur contraint

FIGURE 4.4 – **Filtrage simple basé QEM. Première ligne :** maillage débruité avec la première méthode qui utilise le minimiseur global des quadriques, et la deuxième méthode qui minimise le long de la normale des sommets. $\sigma_q = 2$ dans les deux cas (σ_q exprimé en fonction de la longueur moyenne des arêtes). **Deuxième ligne :** détail du coin interne de l'œil. On observe bien le déplacement des sommets vers les caractéristiques pour la méthode 1. Pour la méthode 2, le maillage a la même régularité que le maillage d'entrée.

4.3.2 Filtrage bilatéral basé QEM

La formulation précédente ne pondère les voisins d'un sommet que par un terme spatial. La similarité du voisin au sommet n'est pas prise en compte, et à distance égale, un sommet situé de l'autre coté d'une arête vive aura le même poids qu'un sommet situé du même coté.

Pour pallier ce problème, nous proposons d'étendre l'algorithme précédent, avec l'ajout de cette notion de similarité. Pour cela, nous reformulons l'algorithme du filtre bilatéral [JD03] pour l'adapter à l'utilisation de quadriques pour décrire localement la surface.

Le filtre bilatéral original, pour un sommet v et les triangles t_i de son voisinage V_v, a la forme suivante :

$$v' = \frac{1}{w(v)} \sum_{t_i \in V_v} P_{t_i}(v) a_{t_i} G_{\sigma_s}(\|v - c_i\|) G_{\sigma_r}(\|v - P_{t_i}(v)\|)$$

où $w(v)$ est la somme des poids, a_{t_i} l'aire du triangle t_i, c_i son centre, G_{σ_s} et G_{σ_r} des noyaux Gaussiens d'écarts types σ_s et σ_r.

$P_{t_i}(v)$ correspond à une prédiction basée sur le voisin t_i. Jones et al. [JD03] utilisent le plan tangent à la surface, défini par le centre du triangle voisin et sa normale. La prédiction $P_{t_i}(v)$ est donnée par la projection de v sur ce plan. Fleishman et al. [FDCO03] utilisent également une approximation de la surface par un plan tangent.

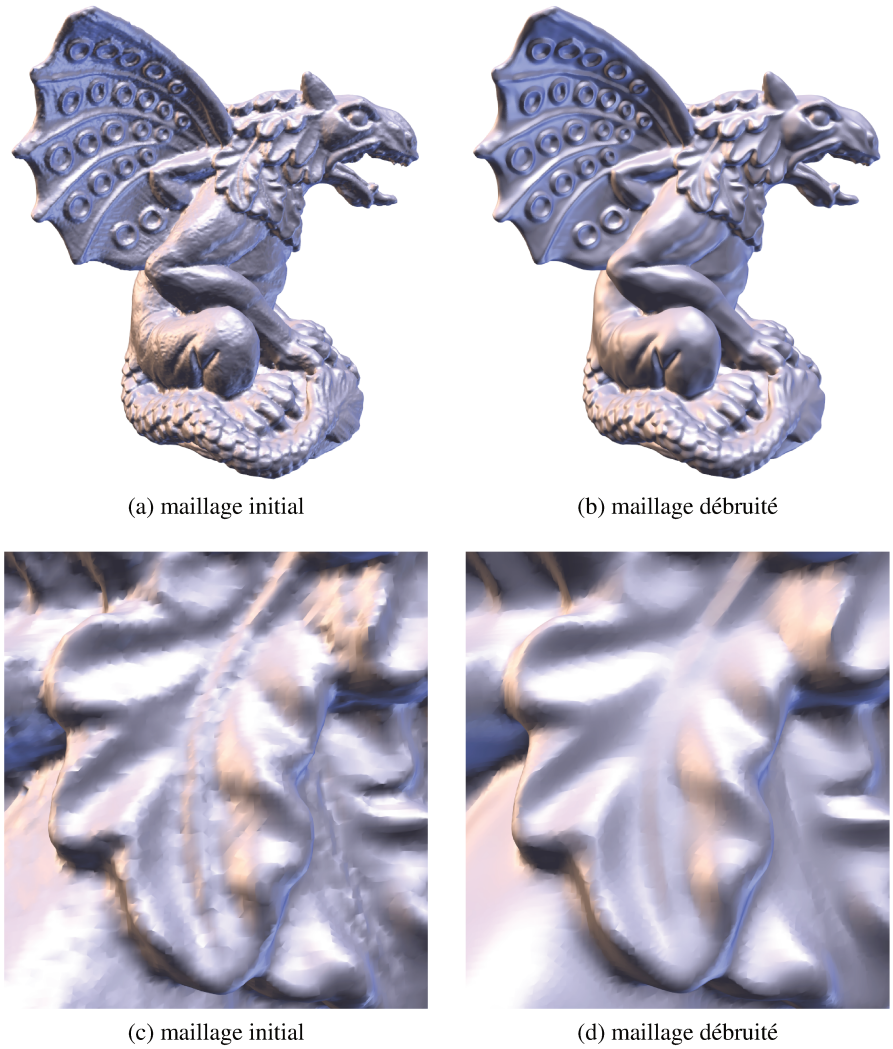
Cette approximation est très sensible au bruit. Jones et al. [JD03] et Fleishman et al. [FDCO03] ajoutent tous les deux une étape de lissage des normales pour répondre à ce problème. Cependant cette solution n'est que partielle et peut entraîner un sur-lissage au niveau des caractéristiques. En effet, comme le font remarquer Jones et al. [JD03], quand les caractéristiques coïncident avec du bruit, il est difficile de séparer les deux correctement (figure 4.7).

Prédictions

Nous proposons d'utiliser un autre type de prédiction, basé sur la QEM. Étant donné un maillage, on calcule pour chaque sommet une quadrique Q', de la façon décrite dans la section précédente, sur un petit voisinage V_{σ_q}. On obtient pour chaque sommet un descripteur de la surface sous-jacente, plus riche et plus robuste qu'un plan tangent basé sur des normales lissées. On remarque que ce plan tangent peut être vu comme une quadrique dégénérée.

On définit la prédiction pour le sommet v par un sommet voisin p comme le minimiseur de la quadrique Q'_p de p.

$$P_p(v) = -A_p^{-1} b_p$$



(a) maillage initial (b) maillage débruité

(c) maillage initial (d) maillage débruité

FIGURE 4.5 – Débruitage d'un modèle scanné par filtrage BLQ-s. Les paramètres utilisés sont $\sigma_s = 3$ et $\sigma_r = 0.8$, exprimés en fonction de la longueur moyenne des arêtes.

avec :

$$Q'_p = \left[\begin{array}{c|c} A_p & b_p \\ \hline b_p^T & c_p \end{array}\right]$$

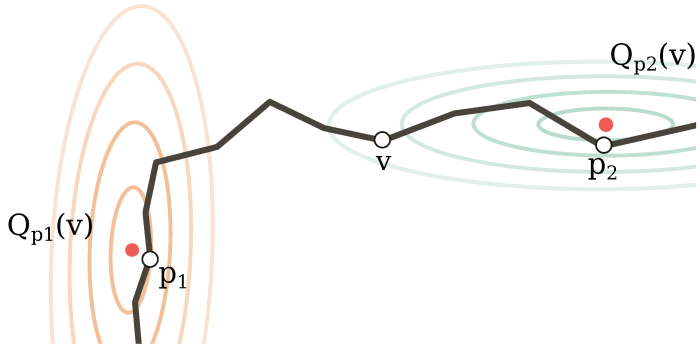


FIGURE 4.6 – Représentation 2D avec les iso-contours des quadriques. L'erreur quadrique en v par rapport à Q_{p_1} est plus forte que par rapport à Q_{p_2}. Le voisin p_1, qui est de l'autre côté de l'arête saillante par rapport à v, aura donc moins d'influence que p_2 qui est du même coté.

Terme d'influence

Considérons maintenant la question de la pondération de ces prédictions. De la même façon que dans l'algorithme de la section précédente et que pour le filtre bilatéral classique, on pondère par un terme spatial, avec un noyau Gaussien G_{σ_s} d'écart type σ_s, sur la distance $\|v - p\|$, et par l'aire.

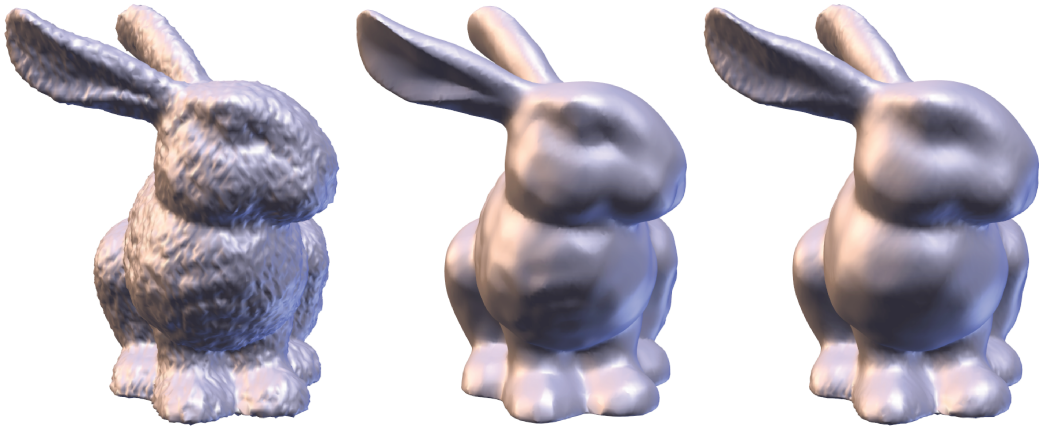
Dans leur formulation, Jones et al. [JD03] pondère également par un terme d'influence, basé sur la distance $\|v - P(v)\|$. En effet, quand on se base sur la projection de v sur le plan tangent de p, les prédictions que l'on souhaite le plus prendre en compte seront proches de v. Ce n'est pas forcément vrai dans notre cas, où on utilise le minimiseur de la quadrique locale de p.

Il s'avère que nos quadriques nous offrent un mécanisme simple permettant de tester si un point de l'espace est éloigné de l'optimiseur local. Pour cela nous calculons notre terme d'influence en évaluant l'erreur quadrique en v par rapport à la quadrique du voisin Q'_p :

$$Q'_p(v) = (v^T|1)Q'_p \begin{pmatrix} v \\ \hline 1 \end{pmatrix}$$

Pour donner une meilleure intuition de la valeur de ce terme, on rappelle que les isosurfaces définies par $Q'_p(v) = \epsilon$ sont, dans les cas non dégénérés, des ellipsoïdes centrés sur l'optimiseur de Q'_p. Ces ellipsoïdes suivent la forme locale de la surface (illustration en 2D figure 4.6).

Plus précisément, le plus petit axe de l'ellipsoïde est aligné avec la normale de la surface, ce qui signifie qu'un point éloigné dans cette direction aura une forte erreur $Q'_p(v)$ associée. Les deux autres axes correspondent aux directions principales de courbure (cf. la thèse de Garland [Gar99], section 4.4.2 pour l'analyse formelle). Un point à une distance donnée



(a) maillage bruité

(b) BLQ-s, $\sigma_s = 3$, $\sigma_r = 0.5$
RMSE = 0. 000 564

(c) bilatéral classique [JD03],
$\sigma_s = 3$, $\sigma_r = 4$
RMSE = 0. 001 418

FIGURE 4.7 – Modèle avec un fort bruit artificiel Gaussien ($\sigma_{bruit} = 0.2$). On observe qu'on débruite mieux les caractéristiques saillantes, comme l'oreille, avec notre méthode, tout en perdant moins de détails qu'avec le filtre bilatéral classique. Cela est confirmé quantitativement avec la RMSE (*root mean square error*), mesurée par rapport au maillage original non bruité. Les paramètres ont été choisis pour lisser au mieux le bruit dans les deux cas. (σ_{bruit}, σ_s et σ_r exprimés en fonction de la longueur moyenne des arêtes).

du centre dans la direction de courbure minimale aura donc une erreur associée plus faible qu'un point à la même distance dans la direction de courbure maximale.

En d'autres termes, on donnera plus de poids à la prédiction $P_p(v)$ d'un voisin p si v est proche de l'approximation locale de la surface au sommet p.

Algorithme

Cela nous amène à cette formulation du filtrage :

$$v^{bl} = \frac{1}{w(v)} \sum_{p \in V_v} (-A_p^{-1} b_p) a_p G_{\sigma_s}(\|v - p\|) G_{\sigma_r}(\sqrt{Q'_p(v)})$$

où $-A_p^{-1} b_p$ est la prédiction, c'est à dire le minimiseur de la quadrique Q'_p de p, et $G_{\sigma_r}(\sqrt{Q'_p(v)})$ est le terme d'influence, calculé à partir de l'erreur de v par rapport à la quadrique de p. La figure 4.5 montre un résultat de ce filtrage.

L'algorithme de filtrage bilatéral basé QEM des sommets (filtre BLQ-s) se décompose ainsi :

- Calcul d'une quadrique Q' par sommet v, pour un petit voisinage V_{σ_q} :

$$Q'_v = \frac{1}{w(v)} \sum_{p \in V_{\sigma_q}} Q_p a_p G_{\sigma_q}(\|v - p\|)$$

On note que cette étape peut s'apparenter à celle du lissage des normales du filtre bilatéral classique, avec une estimation plus riche de la forme non bruitée.

- Calcul des sommets filtrés (algorithme 2).

Algorithme 2 : Filtre BLQ-s - Filtrage bilatéral des sommets basé QEM

pour tous les *sommets v* **faire**
  $wTotal \leftarrow 0$
  $v^{bl} \leftarrow (0, 0, 0)$
  pour tous les *voisins p de v* **faire**
    $w \leftarrow a_p G_{\sigma_s}(\|v - p\|) G_{\sigma_r}(\sqrt{Q'_p(v)})$
    $wTotal \mathrel{+}= w$
    $v^{bl} \mathrel{+}= w(-A_p^{-1} b_p)$
  fin
  $v^{bl} /= wTotal$
fin

Les paramètres à fournir par l'utilisateur sont :

1. σ_q, qui contrôle la taille du voisinage utilisé pour calculer la quadrique d'un sommet, et par conséquent la taille de la plus petite caractéristique qui pourra être préservée. On a obtenu en général de bons résultats pour une valeur inférieure ou égale à $\sigma_s/2$. Dans la suite, sauf mention contraire, on fixe $\sigma_q = \sigma_s/2$.
2. σ_s, qui contrôle la taille du voisinage examiné pour le calcul du sommet filtré.
3. σ_r, qui détermine l'influence des voisins. Une valeur faible pénalisera plus les voisins donnant une erreur quadrique forte pour v.

Dans la suite, nous exprimons ces paramètres relativement à la longueur moyenne des arêtes du maillage.

4.3.3 Filtrage bilatéral des quadriques

Une solution alternative pour adapter le filtre bilatéral original à la QEM est de filtrer non plus les sommets, mais les quadriques elles-mêmes, afin de mieux exploiter leurs propriétés tout au long de l'algorithme.

En utilisant les mêmes poids que précédemment, la formule devient :

$$\mathbf{Q_v^{bl}} = \frac{1}{w(v)} \sum_{p \in V_v} \mathbf{Q'_p} a_p G_{\sigma_s}(\|v - p\|) G_{\sigma_r}(\sqrt{Q'_p(v)})$$

De cette façon, au lieu de moyenner les points minimisant les quadriques locales des sommets du voisinage de v, on calcule une nouvelle quadrique approximant ce voisinage, c'est à dire une moyenne pondérée des quadriques locales des voisins. Le terme d'influence assure que l'on favorisera les quadriques dont v est proche en termes de QEM, et donc l'approximation robuste de la surface. La figure 4.8 montre un modèle filtré avec cette méthode.

Les étapes de cet algorithme de filtrage bilatéral des quadriques (filtre BLQ-q) sont alors :

- Le calcul d'une quadrique Q' par sommet, qui reste identique.
- Le calcul des quadriques filtrées (algorithme 3).

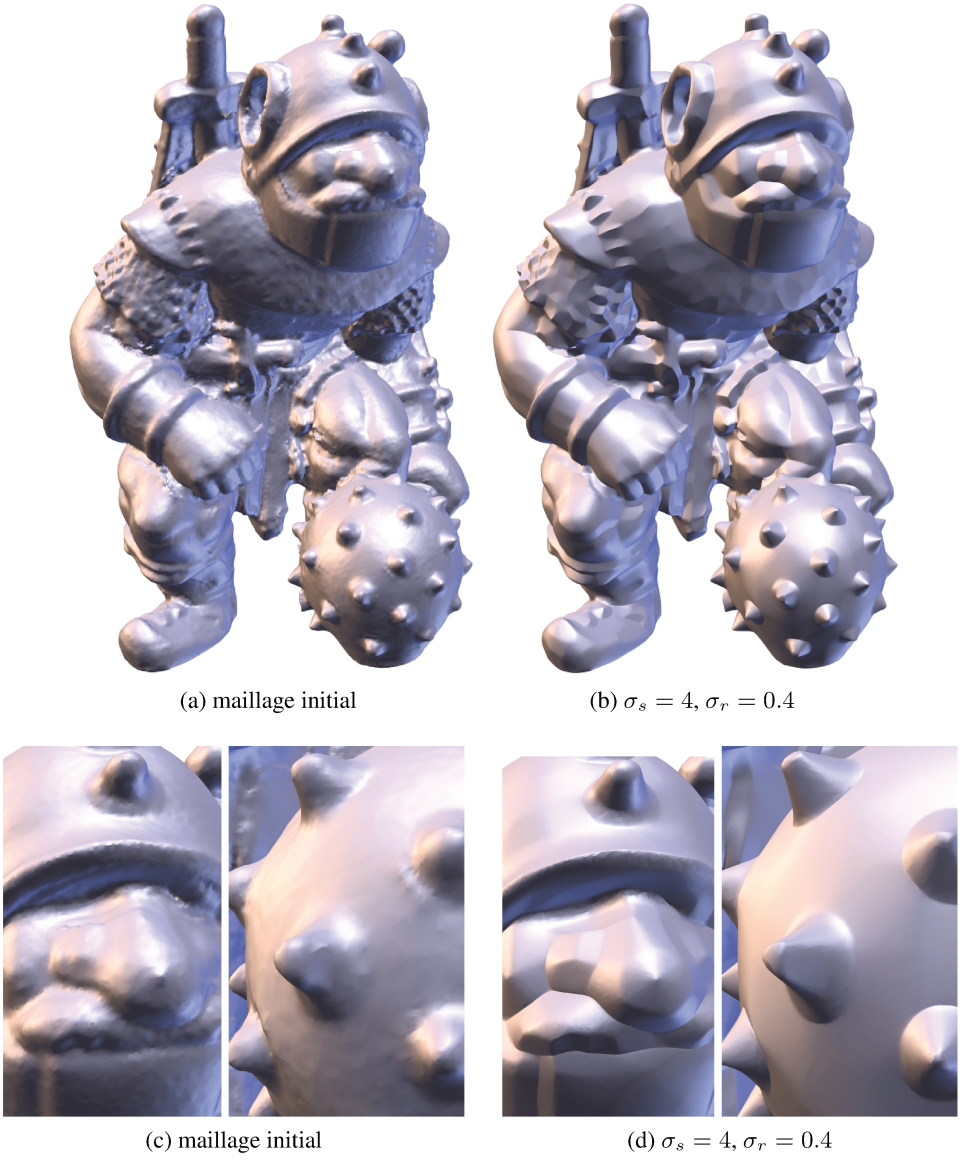
Algorithme 3 : Filtre BLQ-q - Filtrage bilatéral des quadriques

pour tous les *sommets v* **faire**
  $wTotal \leftarrow 0$
  $Q^{bl} \leftarrow 0$
  pour tous les *voisins p de v* **faire**
    $w \leftarrow a_p G_{\sigma_s}(\|v - p\|) G_{\sigma_r}(\sqrt{Q'_p(v)})$
    $wTotal += w$
    $Q^{bl} += wQ'_p$
  fin
  $Q^{bl} /= wTotal$
fin

- Le sommet filtré est donné par le minimiseur de la quadrique filtrée Q_v^{bl} de v.

Comme pour le filtrage simple basé QEM (section 4.3.1), on peut choisir d'utiliser le minimiseur global de la quadrique, ou d'optimiser la position du point le long de la normale (lissée) du sommet. Le filtrage le long de la normale permet de débruiter en préservant la régularité du maillage initial. Il minimise également les auto-intersections qui peuvent se produire pour certains cas où la géométrie est très fortement modifiée. Le minimiseur global des quadriques donnera quant à lui un rehaussement des caractéristiques en plus du débruitage (figure 4.9).

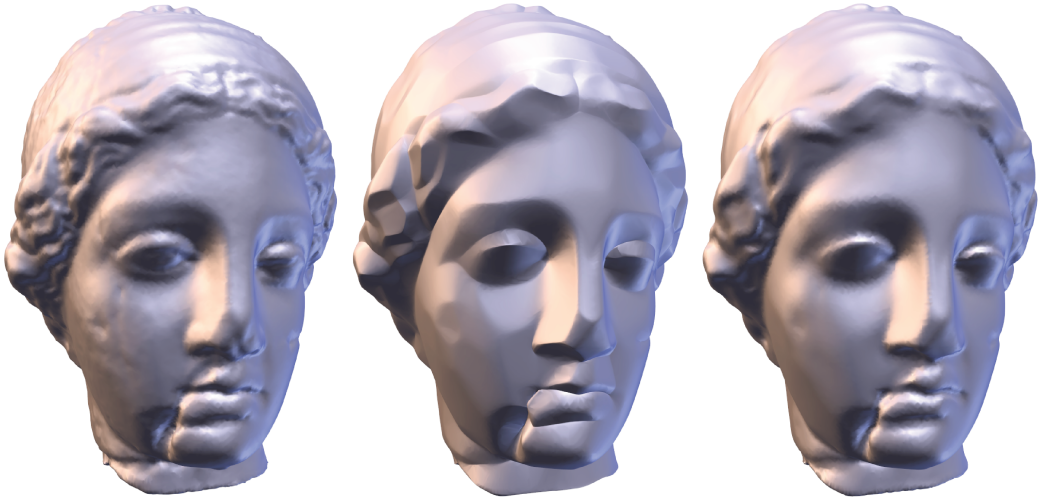
Contrairement au filtrage simple, ce filtrage bilatéral permet de moduler la préservation des caractéristiques indépendamment de la taille du voisinage à considérer (figure 4.10).



(a) maillage initial (b) $\sigma_s = 4$, $\sigma_r = 0.4$

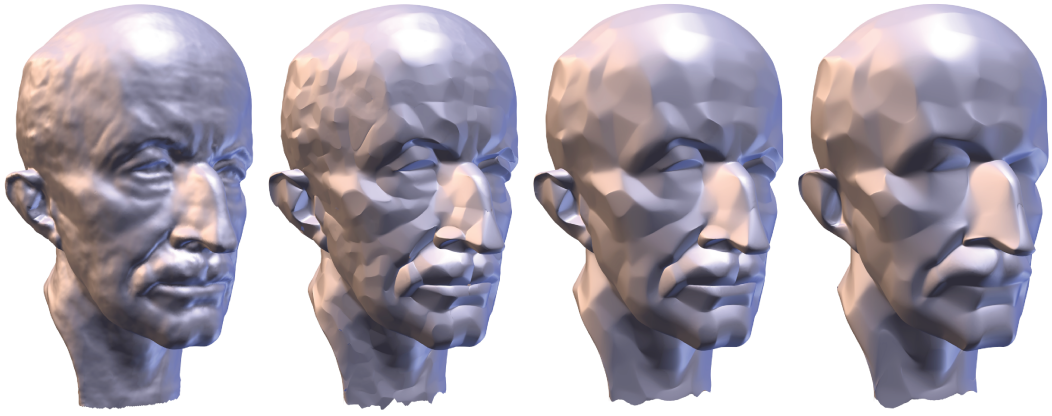
(c) maillage initial (d) $\sigma_s = 4$, $\sigma_r = 0.4$

FIGURE 4.8 – **Filtre BLQ-q.** Filtrage bilatéral des quadriques, puis déplacement des sommets vers le point minimisant leur quadrique filtrée.



(a) maillage initial (b) minimiseur global (c) minimiseur le long de la normale

FIGURE 4.9 – Restriction du déplacement des sommets à la direction de la normale (filtre BLQ-q). Les paramètres utilisés sont $\sigma_s = 5$ et $\sigma_r = 0.4$, exprimés en fonction de la longueur moyenne des arêtes.



(a) maillage initial (b) simple $\sigma_q = 3$ (c) BLQ-q $\sigma_q = 3$, $\sigma_s = 6$, $\sigma_r = 0.8$ (d) simple $\sigma_q = 6$

FIGURE 4.10 – Comparaison filtrage simple, et filtrage BLQ-q. La version bilatérale permet de préserver des structures comme les yeux ou le bord du nez, tout en lissant mieux les zones comme la joue ou le coté de la tête.

4.3.4 Implémentation

Dans l'optique d'un filtrage haute performance à la demande, nous avons implémenté notre méthode sur GPU à l'aide de CUDA. Comme pour le filtre bilatéral classique, notre algorithme est directement adaptable au calcul parallèle. En effet, chaque étape de la méthode a été pensée pour traiter tous les sommets de manière indépendante. Ceci est particulièrement apparent dans les algorithmes 2 et 3, où l'on peut voir que toutes les itérations de la boucle sur les sommets peuvent être exécutées simultanément par un ensemble de threads.

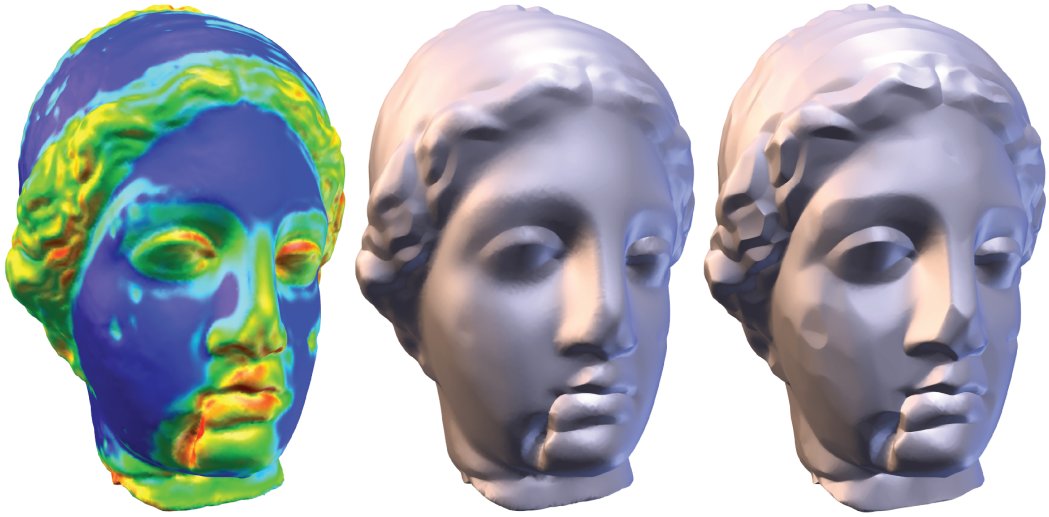
L'étape la plus critique en termes de performance est la recherche des voisins dans un rayon donné autour d'un sommet. Nous utilisons pour cela l'algorithme décrit par Hoetzlein [Hoe14], basée sur un tri comptage parallèle (voir section 2.2.2). Globalement, cette méthode s'appuie sur une partition uniforme de l'espace en cellules, et un tri parallèle des primitives permettant à la fois de les affecter aux cellules correspondantes, et d'y accéder efficacement.

Comme Jones et al. [JD03] et Fleishman et al. [FDCO03], nous limitons le rayon de cette recherche à $2\sigma_s$, ce qui réduit beaucoup le nombre d'échantillons à traiter, mais n'affecte pas significativement la qualité des résultats. Nous avons également ré-implémenté pour comparaison le filtre bilatéral original, dans les mêmes conditions.

4.4 Résultats et discussion

Dans cette section, nous présentons et discutons des résultats obtenus avec les différentes variations de l'algorithme pour un ensemble de modèles variés, avec plusieurs niveaux de bruit naturel ou artificiel. Nous examinons ensuite les performances obtenues avec notre implémentation parallèle. Toutes les expérimentations présentées ici ont été réalisées sur un PC équipé d'une carte graphique GeForce GTX 980 Ti, et d'un processeur Intel Xeon E5-1620 3.6GHz.

La figure 4.3 montre les résultats obtenus avec la version simple du filtre, décrite à la section 4.3.1. Pour la suite, nous restreignons l'analyse des résultats aux filtres plus avancés. Les figures 4.5 et 4.8 illustrent les résultats de nos deux méthodes bilatérales, à savoir le filtrage bilatéral des sommets basé QEM (BLQ-s, décrit à la section 4.3.2), et le filtrage bilatéral des quadriques (BLQ-q, décrit à la section 4.3.3). La première méthode détermine d'abord pour chaque voisin le point minimisant sa quadrique locale, puis calcule la position du sommet filtré à partir de ces points. On observe une certaine accentuation des caractéristiques due à l'étape de minimisation des quadriques locales. Cependant cette accentuation est atténuée dans l'étape suivante, qui calcule une moyenne pondérée des minimiseurs.



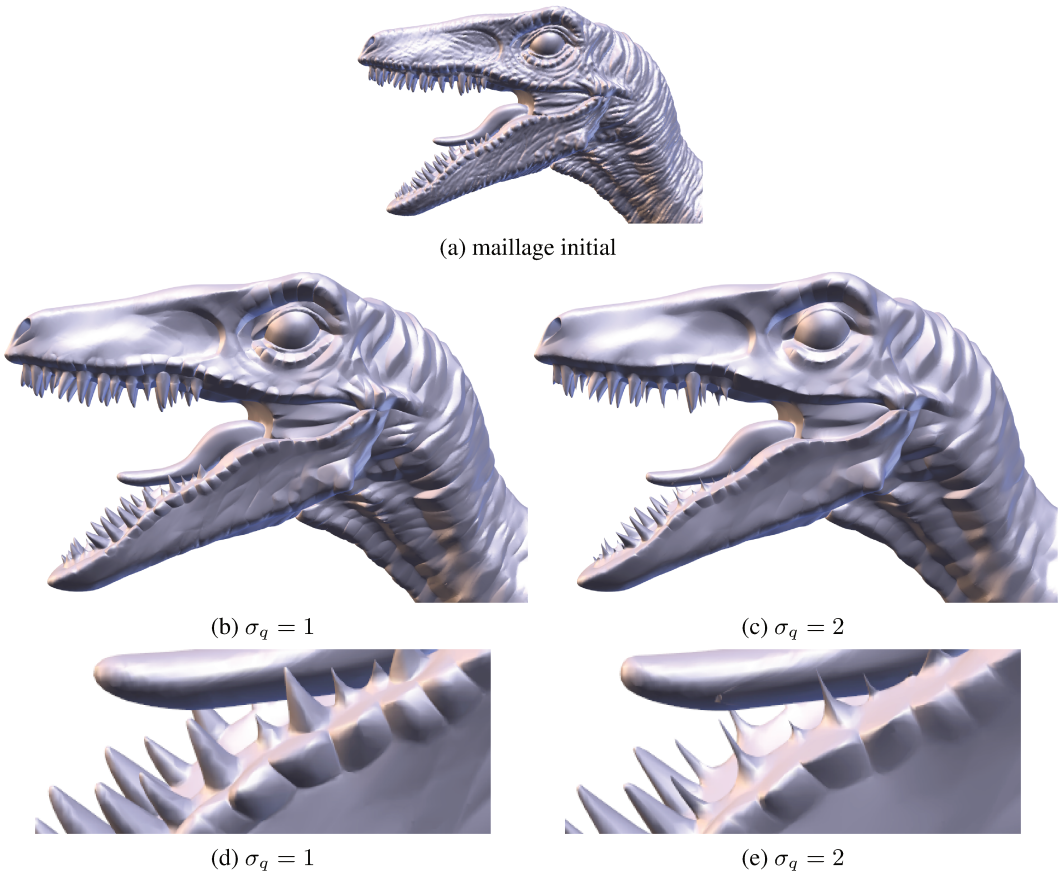
(a) incertitude (b) BLQ-s (c) BLQ-q

FIGURE 4.11 – Comparaison des deux méthodes de filtrage bilatéral, avec les mêmes paramètres, et valeur de l'incertitude en fausses couleurs, bleu correspondant à l'incertitude la plus faible et rouge la plus forte. L'incertitude est donnée par la somme des poids des voisins à chaque sommet ($w(v)$). $\sigma_s = 3$, $\sigma_r = 0.4$.

En revanche pour la deuxième approche, le filtrage est effectué directement sur les quadriques. L'étape de minimisation n'intervient qu'à la toute fin pour placer le sommet filtré, et l'accentuation des caractéristiques apparaît plus nettement. Filtrer les quadriques elles-mêmes plutôt que les points permet de mieux exploiter leur propriétés, en manipulant une information plus riche jusqu'à la fin de l'algorithme. La figure 4.11 montre le résultat des deux méthodes sur un même modèle avec les mêmes paramètres. Dans les deux cas, les poids utilisés pour calculer l'influence de chaque voisin sont calculés de la même façon. Par la suite, nous nous focalisons sur les résultats obtenus avec le filtre BLQ-q.

Durand et al. [DD02] remarquent que la somme des poids des voisins d'un sommet (le facteur de normalisation $w(v)$) donne une mesure de l'incertitude pour ce sommet. En effet, quand $w(v)$ est faible, cela signifie qu'on a trouvé peu de voisins capables de donner une bonne prédiction pour v. C'est souvent le cas au niveau des caractéristiques, comme on peut le voir dans la figure 4.11. On note que les quadriques permettent d'obtenir une bonne approximation de la surface y compris à ces endroits.

Les paramètres σ_q, σ_s et σ_r permettent de faire varier le comportement de l'algorithme du débruitage jusqu'à la stylisation extrême de la forme (figure 4.13). σ_q contrôle la taille du voisinage utilisé pour calculer la quadrique locale initiale de chaque sommet. De ce fait il contrôle également la taille de la plus petite structure qui pourra être préservée par le filtrage.



(a) maillage initial

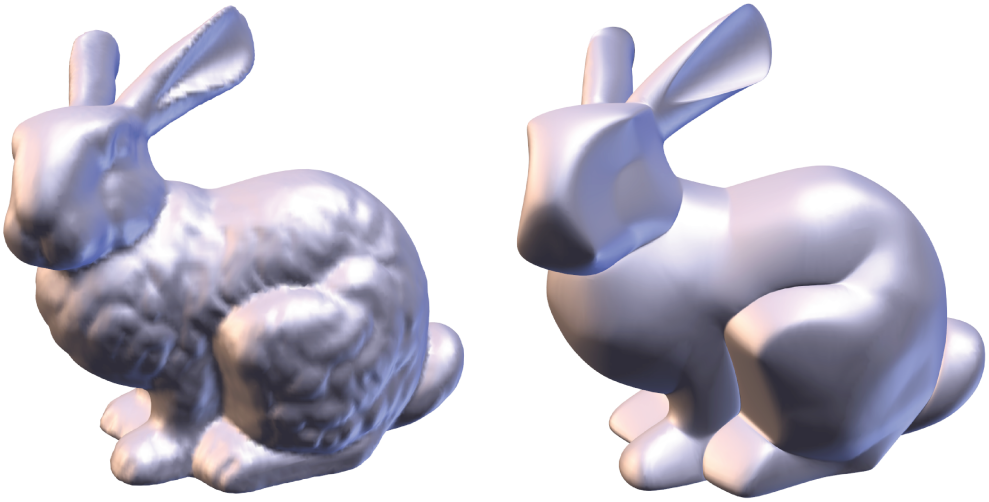
(b) $\sigma_q = 1$ (c) $\sigma_q = 2$

(d) $\sigma_q = 1$ (e) $\sigma_q = 2$

FIGURE 4.12 – **Influence de** σ_q (filtre BLQ-q). σ_q détermine la taille de la plus petite caractéristique qui pourra être préservée par le filtrage. Nos expérimentations ont donné en général de bons résultats avec $\sigma_q = \sigma_s/2$, mais dans certains cas, comme ici pour préserver les dents par exemple, il est préférable de choisir une valeur plus faible. ($\sigma_s = 4$, $\sigma_r = 0.7$)

Nous avons en général fixé sa valeur à $\sigma_s/2$ et avons obtenu de bons résultats, mais il est nécessaire d'adapter cette valeur pour certains modèles, pour lesquels on voudrait conserver des caractéristiques très fines (figure 4.12).

σ_s et σ_r modulent l'influence des voisins au moment du filtrage. On les choisit en fonction du bruit ou des caractéristiques qui doivent disparaître. Faire varier σ_r permettra de lisser plus ou moins les caractéristiques dans la limite du voisinage défini par σ_s, tandis que les caractéristiques restantes seront rehaussées. De fortes valeurs pour ces deux paramètres accentueront uniquement les caractéristiques les plus grandes et les plus saillantes, et lisseront complètement le reste du modèle. Cela résulte en une stylisation de la forme, qui est réduite à sa structure principale (figure 4.13).



(a) maillage initial (b) $\sigma_s = 5, \sigma_r = 2$

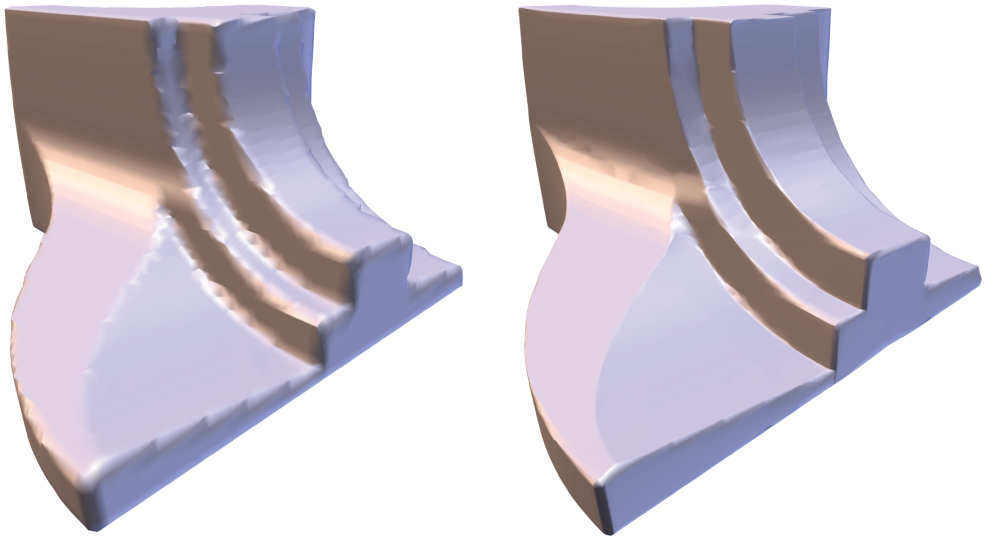
FIGURE 4.13 – Modèle stylisé par filtrage BLQ-q. La structure principale est très accentuée tandis que tout le reste de la forme est lissé.

Modèle	**#Triangles**	σ_s	σ_r	**BLQ-q**	**[JD03]**
Fandisk (fig. 4.14)	20K	3	0.4	15	13
Bunny (fig. 4.13)	70K	5	2.0	68	51
Egea (fig. 4.11)	200K	3	0.4	81	56
Max Planck (fig. 4.10)	400K	6	0.8	332	266
Grog (fig. 4.8)	1M	4	0.4	459	347
Gargoyle (fig. 4.15)	1.7M	5	1.5	1106	855
Raptor (fig. 4.12)	2M	4	0.7	849	649

TABLE 4.1 – **Mesures de performance** en ms pour les modèles utilisés dans les figures, sans les transferts mémoire CPU-GPU.

Dans la figure 4.14, nous illustrons le comportement du filtre sur un modèle mécanique comportant des artefacts de reconstruction au niveau des arêtes. Notre méthode permet de les éliminer en grande partie, tout en retrouvant des arêtes vives. La figure 4.15 montre la robustesse des quadriques au bruit. On parvient avec notre approche à lisser complètement le bruit tout en conservant les caractéristiques, là où le filtre bilatéral classique n'y parvient pas (les paramètres ont été choisis pour lisser au mieux le bruit dans les deux cas).

Dans le tableau 4.1, nous présentons les mesures de temps obtenues pour les modèles des différentes figures. Les mesures n'incluent pas les transferts mémoire. Le temps de



(a) maillage initial (b) $\sigma_s = 3, \sigma_r = 0.4$

FIGURE 4.14 – Accentuation des arêtes saillantes bruitées sur un modèle mécanique, par filtrage BLQ-q.

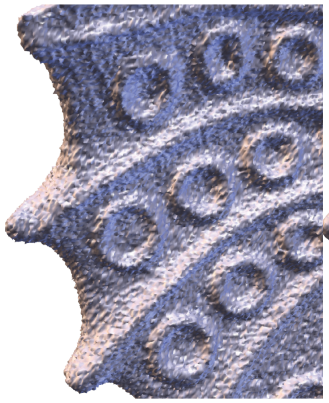
filtrage dépend principalement de la taille du maillage et du paramètre σ_s. On indique également pour chaque modèle le temps mis par notre implémentation du filtre bilatéral classique [JD03] pour les mêmes paramètres. Nous obtenons avec notre approche des temps du même ordre de grandeur qu'avec le filtre bilatéral classique, puisque les deux algorithmes ont une structure très proche.

Conclusion

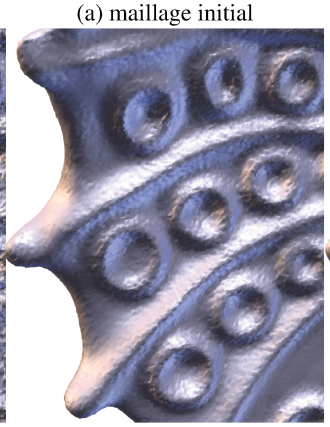
Nous avons présenté différentes variations d'une méthode de filtrage basé sur la QEM. Cette approche a une structure très similaire au filtre bilatéral classique et est aussi simple à mettre en place. Aucune supposition sur la connectivité du maillage n'est faite, et l'extension à des nuages de points munis de normales serait une extension future naturelle.

Notre méthode permet de débruiter efficacement et rapidement des surfaces, y compris dans des conditions de bruit important. Les caractéristiques saillantes sont préservées, et accentuées, tandis que le reste de la forme est lissé. En poussant les paramètres, notre approche ira jusqu'à styliser fortement le modèle tout en restant stable à grande échelle, réduisant la forme à sa structure principale. Ces résultats invitent à explorer la direction de l'analyse de forme multi-échelle rapide basée sur les quadriques dans des travaux futurs.

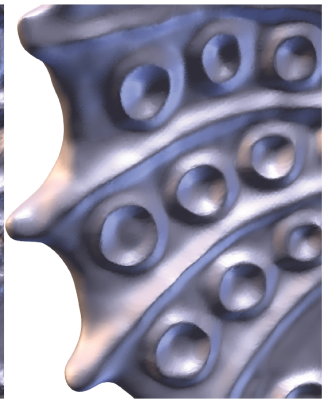
(a) maillage initial



(b) bruit ajouté, $\sigma_{bruit} = 0.4$



(c) **bilatéral classique** $\sigma_s = 5$, $\sigma_r = 3$
RMSE = 0. 000 457



(d) **BLQ-q** $\sigma_s = 5, \sigma_r = 1.5$
RMSE = 0. 000 296

FIGURE 4.15 – Comparaison avec le filtre bilatéral classique [JD03] pour un modèle avec un fort bruit artificiel ajouté. On parvient à retrouver les arêtes vives du modèle avec notre méthode, tandis qu'elles restent bruitées pour le filtre bilatéral classique. Cela est confirmé quantitativement avec la RMSE (*root mean square error*), mesurée par rapport au maillage original non bruité. Les paramètres ont été choisis pour lisser au mieux le bruit dans les deux cas.

CHAPITRE 5

APPLICATION À LA SEGMENTATION DE MAILLAGES PAR QUICK SHIFT

Après avoir abordé la question du traitement géométrique haute performance sous l'angle de la simplification et du filtrage, nous étendons ici nos travaux au domaine de l'analyse géométrique, dont la segmentation est une composante fondamentale. Dans ce chapitre, nous détaillons une méthode de partitionnement exploitant le filtrage de quadriques vu au chapitre précédent. Nous présentons un algorithme de segmentation de surfaces parallèle rapide permettant d'extraire des régions suivant la géométrie. Celui-ci s'appuie sur la caractérisation locale de la surface donnée par les quadriques filtrées, et sur l'algorithme du *Quick Shift*, qui sont tous deux particulièrement adaptés au calcul sur GPU.

5.1 Contexte

Les technologies de capture actuelles permettent de créer des modèles 3D de plus en plus massifs, qu'il est difficile de traiter efficacement en temps raisonnable avec le matériel standard actuel. Les modèles acquis avec des scanners 3D par exemple atteignent aisément plusieurs millions de polygones, et ne peuvent être traités que par des méthodes pouvant passer à l'échelle. Par conséquent, les méthodes d'analyse ou de traitement complexes de maillages requièrent souvent de réduire les ensembles de données 3D massifs à une représentation plus concise, tout en préservant au mieux l'information. Nous avons vu précédemment qu'une approche possible consiste à simplifier rapidement et adaptativement les données avant de les traiter (chapitre 3). Bien qu'elle soit adaptée à un certain nombre de scénarios, cette méthode demeure destructive : la résolution et la structure d'origine du modèle sont affectées de manière irréversible.

Le domaine du traitement d'image, qui a rencontré des défis similaires au cours des dernières années, a introduit le concept de *superpixels*. Ces petits groupes de pixels, correspondant à une sur-segmentation de l'image, jouent le rôle de primitives intermédiaires pour des traitements complexes, et permettent de réduire de plusieurs ordres de grandeur le nombre d'éléments à considérer. Cette approche, alternative au prétraitement par simplification, consiste à n'effectuer la partie la plus coûteuse de l'algorithme (évaluation d'un opérateur, estimation d'un jeu de paramètres) qu'une seule fois par superpixel, et à appliquer le résultat à l'ensemble des pixels du superpixel, en effectuant éventuellement une forme d'interpolation. Les superpixels ont été largement adoptés dans les domaines de la vision et du traitement d'images, et quelques approches ont été proposées pour étendre ce principe au cas des modèles géométriques 3D.

De la même façon qu'une bonne segmentation d'image sera indispensable à son analyse, un bon partitionnement de surface est une étape importante en traitement géométrique. Décomposer une forme en un ensemble de régions permet à la fois de générer des représentations intermédiaires pour un traitement plus rapide, ainsi que des représentations de plus haut niveau dans l'optique d'analyses complexes.

Nous introduisons une nouvelle méthode visant à exécuter ce type de partitionnement aussi rapidement que possible, de façon à aller vers une analyse de forme aussi rapide que la génération de données 3D.

Pour atteindre des temps interactifs, nous nous basons sur l'algorithme du *Quick Shift*, qui est non-itératif et intrinsèquement parallèle, et le filtrage de quadriques vu précédemment pour une approximation locale robuste de la géométrie.

5.2 Travaux Antérieurs

Les techniques de segmentation 3D peuvent être divisées en deux catégories : la segmentation en parties significatives ou en patchs de surface. Nous n'aborderons pas ici la segmentation en parties significatives, qui vise à obtenir des parties de l'objet, souvent volumétriques, sur des critères sémantiques (par exemple séparer les bras, la tête, etc.). Nous nous focaliserons sur la seconde catégorie, qui va plutôt chercher à extraire des patchs qui sont en général des disques topologiques, respectant certains critères géométriques.

Garland et al. [GWH01] proposent un algorithme produisant une hiérarchie de régions basée sur une métrique dérivée de la QEM. Les faces sont contractées deux à deux itérativement formant des régions de plus en plus étendues. *Variational Shape Approximation* (VSA) [CSAD04] est un algorithme itératif de type k-moyennes, adaptant au mieux un certain nombre de proxy planaires à la géométrie. Il alterne entre une phase de croissance de régions

basée sur la métrique $L^{2,1}$ pour créer une partition, et une phase de calcul d'un proxy par région jusqu'à convergence.

Le concept de *superpixel* a été introduit par Ren et al. [RM03], partant de l'observation que les pixels sont la conséquence de la représentation discrète des images, et ne représentent pas une primitive naturelle. De plus, même à des résolutions raisonnables, ils sont très nombreux, ce qui rend les optimisations à l'échelle du pixel difficiles à gérer. Pour réduire le nombre d'éléments à traiter, on va donc grouper les pixels de manière locale, et cohérente avec les structures présentes dans l'image. Ce pré-traitement doit être autant que possible simple et rapide à calculer, et passer facilement à l'échelle. Plusieurs techniques ont été proposées depuis pour les générer, notamment utilisant l'algorithme du Mean Shift [CM02], des k-moyennes [ASS+12], ou encore du Quick Shift [VS08]. Ces superpixels ont été utilisés pour accélérer et améliorer les résultats de nombreux algorithmes, par exemple de détection d'objets ou de segmentation [HZR06, MPW+09, ZSL+13].

Simari et al.[SPDF14] s'inspirent de la méthode de création de superpixels SLIC (*Simple Linear Iterative Clustering*) [ASS+12] pour calculer des "superfacettes" efficacement sur des maillages de grande taille, avec une approche de type k-moyennes.

Souvent vu comme une approche duale aux k-moyennes, le Mean Shift est une méthode générale de recherche de modes et de partitionnement de données en N dimensions. Intuitivement, l'idée est de déplacer chaque échantillon itérativement vers l'optimiseur d'une énergie définie localement, souvent à l'aide d'une somme sur l'ensemble des échantillons de données, pondérée par un noyau centré sur l'échantillon courant. Ainsi, on déplace itérativement les échantillons dans la direction du gradient, vers des régions de plus forte densité jusqu'à convergence. Yamauchi et al. [YLL+05] utilisent un Mean Shift sur des points en 6D, constitués des centres des triangles du maillage et de leur normale. Les normales du maillage sont ensuite mises à jour avec les normales obtenues pour les modes. La seconde phase de l'algorithme alterne entre une étape de croissance de régions basée sur les normales calculées, et une étape de mise à jour des centres.

Le Medoid Shift [SKK07] est un algorithme dérivé du Mean Shift, pour lequel les déplacements sont contraints. Un déplacement ne pourra se faire que vers la position d'un autre échantillon. Cela permet de s'affranchir du caractère itératif du Mean Shift : en calculant uniquement un "suivant" pour chaque point, on obtient toutes les trajectoires jusqu'aux modes. Il n'y a donc pas non plus besoin de définir un critère d'arrêt. La contrainte d'utiliser la distance euclidienne disparaît également, il est ainsi possible d'employer la distance L_1 par exemple, ou toute autre mesure de distance.

Vedaldi et al. [VS08] remarquent qu'il n'est pas nécessaire d'utiliser le gradient pour se déplacer vers les modes. Ils introduisent l'algorithme du Quick Shift, qui va déplacer chaque point vers son plus proche voisin pour lequel la densité estimée est plus forte. Cet algorithme connecte l'ensemble des points en un seul arbre, à partir duquel on peut retrouver une partition en coupant les branches qui sont plus longues qu'un certain seuil. Les modes sont les racines des sous-arbres obtenus. Cet algorithme a été utilisé notamment pour cal-

culer des superpixels [VS08], une segmentation de maillage basée sur la courbure et les normales [LKK15], et s'adapte facilement sur GPU [FS10].

Nous proposons d'adapter cet algorithme pour exploiter les quadriques filtrées vues au chapitre précédent, afin d'obtenir une segmentation de maillages 3D rapide et passant à l'échelle, produisant des régions correspondant à la géométrie locale, à la façon des superpixels et des superfacettes de Simari et al. [SPDF14].

5.3 Algorithme de segmentation

5.3.1 Vue d'ensemble

Nous construisons un algorithme de segmentation prenant en entrée un maillage triangulaire indexé, et calculant une partition des sommets du maillage. Cette partition prend la forme d'un ensemble de modes, obtenus par Quick Shift, et pour chaque sommet du maillage un mode associé. L'algorithme peut être décomposé en 3 étapes principales :

1. le filtrage des quadriques, suivant l'algorithme décrit à la section 4.3.3, donnant une quadrique par sommet.

2. le calcul d'un score de représentativité du voisinage pour chacun de ces sommets munis d'une quadrique.

3. une recherche de modes et un partitionnement par Quick Shift.

La figure 5.1 montre le type de segmentation obtenue par cette méthode.

5.3.2 Initialisation

Dans la section 4.3.3, nous décrivons un algorithme de filtrage de quadriques, produisant un maillage pour lequel chaque sommet est muni d'une quadrique représentative de son voisinage. Nous observons que ces quadriques capturent bien la géométrie locale, et proposons de les utiliser comme base pour une recherche de modes et une segmentation du maillage.

La première étape de notre approche consiste donc à utiliser en pré-traitement l'algorithme décrit à la section 4.3.3 pour associer à chaque sommet v une quadrique filtrée Q_v^{bl}.

(a) $\theta = 0.5$, 2035 modes (b) $\theta = 0.75$, 1120 modes (c) $\theta = 0.99$, 449 modes

FIGURE 5.1 – Plusieurs segmentations d'un modèle en faisant varier le paramètre θ. Les autres paramètres sont fixes : $\sigma_s = 2.0$, $\sigma_r = 1.0$, $\sigma_d = 6.0$, $\sigma_q = 1.0$

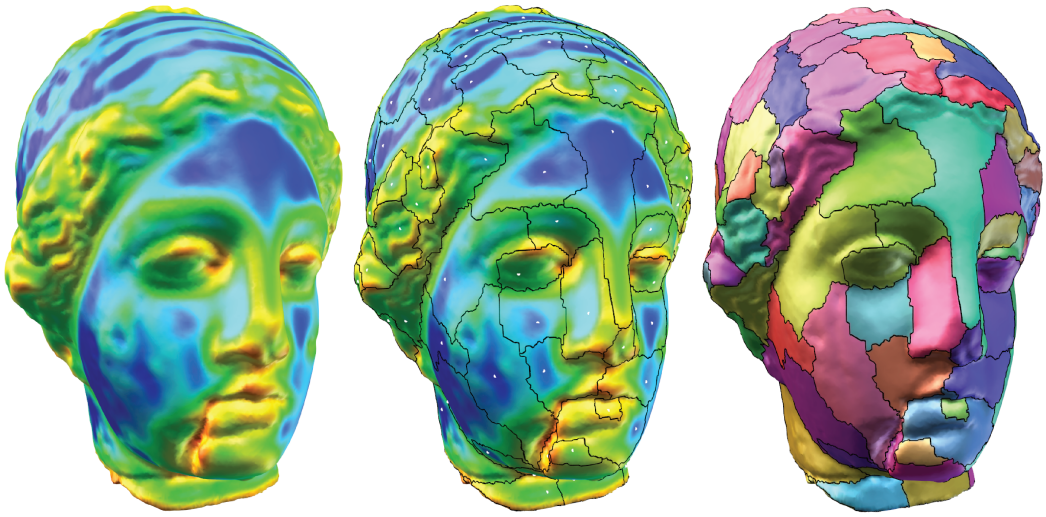
5.3.3 Représentativité du voisinage

La majorité des algorithmes de recherche de modes commencent par calculer une estimation de densité de Parzen. Pour un point x dans un ensemble de points de taille N, cette estimation est :

$$P(x) = \frac{1}{N} \sum_{i=1}^{N} K(x - x_i)$$

où K est un noyau, souvent Gaussien. Les points sont ensuite déplacés progressivement vers les maxima de cette densité. On obtient une partition des données, chaque partie correspondant à l'ensemble des points ayant convergé vers un même mode.

Dans notre cas, les données considérées sont un ensemble de sommets munis de quadriques. Idéalement, on souhaiterait obtenir comme modes les points dont la quadrique représente le mieux son voisinage. L'algorithme du Quick Shift [VS08] nécessite seulement de pouvoir, pour chaque point v, trouver le point v_i le plus proche satisfaisant $P(v_i) > P(v)$. Pour cela, il n'est pas nécessaire que P soit une estimation de densité de Parzen à proprement parler. Nous choisissons de calculer pour chaque sommet v un *score de représentativité* S_v, mesurant à quel point la quadrique Q_v^{bl} représente les sommets de son voisinage spatial $V_{\sigma_{d_1}}$.



(a) score (b) modes et régions (c) partition finale

FIGURE 5.2 – Visualisation du score de représentativité en fausses couleurs, du bleu foncé indiquant un score élevé au rouge indiquant un score faible. Au milieu : les frontières des régions, et les modes représentés par des points blancs. À droite, la partition finale. $\sigma_s = 2.0$, $\sigma_r = 1.0$, $\sigma_d = 6.0$, $\sigma_q = 1.0$ et $\theta = 0.87$. 156 modes.

Ce score de représentativité est donné par :

$$S_v = \frac{1}{\sum a_i} \sum_{v_i \in V_{\sigma_{d_1}}} a_i G_{\sigma_{d_1}}(\|v - v_i\|)\, G_{\sigma_{q_1}}\left(\sqrt{Q_v^{bl}(v_i)}\right)$$

où a_i est l'aire associée au voisin v_i, $Q_v^{bl}(v_i)$ l'évaluation de la QEM en v_i par rapport à Q_v^{bl} (section 2.1), $G_{\sigma_{d_1}}$ et $G_{\sigma_{q_1}}$ des noyaux Gaussiens d'écarts types σ_{d_1} et σ_{q_1} respectivement. La grandeur S_v est plus forte quand l'erreur aux sommets voisins par rapport à la quadrique de v est faible, autrement dit lorsque la quadrique approxime bien le voisinage. La figure 5.2 donne un exemple de visualisation de ce score.

5.3.4 Quick Shift

Une fois ce score S_v calculé pour tous les sommets, on effectue un Quick Shift. Cet algorithme relie chaque point à son plus proche voisin pour lequel la densité, ou le score dans notre cas, est plus forte. Dans le cas de points munis de quadriques , "le plus proche" n'est

pas trivial à définir. Pour rester cohérent avec notre définition du score de représentativité, nous utilisons comme *valeur de proximité* la grandeur suivante :

$$p(v, v_i) = G_{\sigma_{d_2}}(\|v - v_i\|) G_{\sigma_{q_2}}\left(\sqrt{Q^{bl}_{v_i}(v)}\right)$$

En d'autres termes nous considérons à la fois la distance euclidienne entre les sommets, et l'erreur au sommet v par rapport à la quadrique de son voisin $Q^{bl}_{v_i}$. L'importance relative de chacun de ces termes est modulée par σ_{d_2} et σ_{q_2}. On utilise $p(v, v_i)$ comme une *valeur de proximité* de v_i à v (on notera qu'il ne s'agit pas d'une véritable distance, au sens mathématique du terme), et on définit comme voisin le plus proche celui qui maximise $p(v, v_i)$.

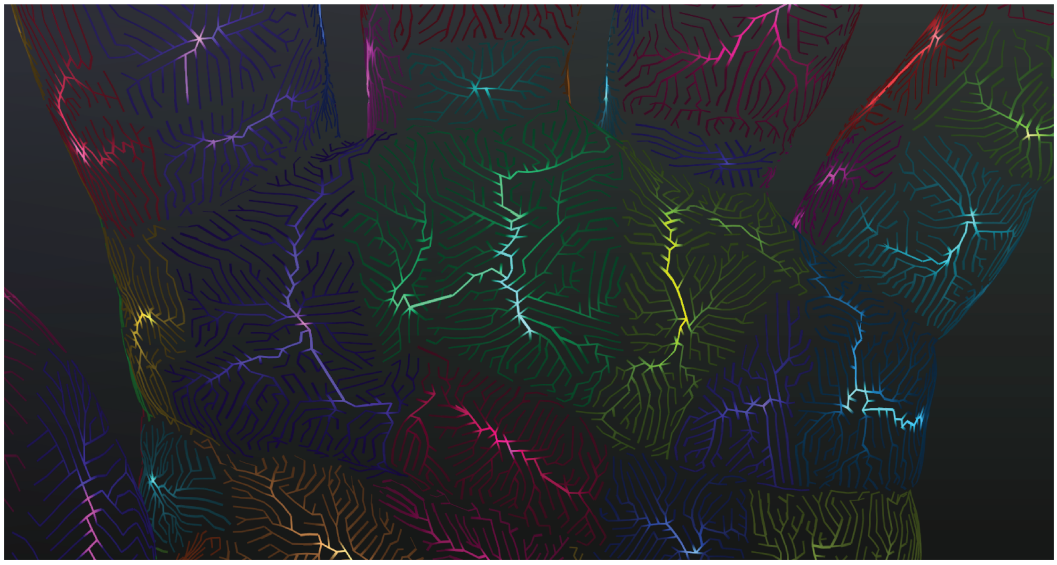
On peut alors trouver le suivant de chaque sommet v, qui est le voisin v_i vérifiant $S_{v_i} > S_v$ qui maximise $p(v, v_i)$. De cette façon, tous les points sauf celui ayant la plus grande valeur de S_v auront un suivant s, et l'ensemble des sommets sera donc connecté pour former un arbre. On associe à chaque branche la valeur $p(v, v_i)$ utilisée pour la construire.

Afin d'obtenir une partition, correspondant à plusieurs modes, on fixe un seuil θ, et on coupe toutes les branches pour lesquelles $p(v, v_i) < \theta$ (algorithme 4). On obtient un ensemble d'arbres, dont les racines sont les modes. Les régions de la partition sont tous les nœuds reliés à un même mode. On associe chaque sommet à son mode en parcourant les suivants jusqu'à arriver à un nœud racine (algorithme 5). La figure 5.3 permet de visualiser les trajectoires des sommets et les arbres formés sur un exemple de segmentation.

Algorithme 4 : Calcul des sommets suivants.

```
pour tous les sommets v du maillage faire
    pmax ← 0
    suivant[v] ← v
    // création de l'arbre
    pour tous les voisins v_i de v faire
        si p(v, v_i) > pmax et S_{v_i} > S_v alors
            pmax ← p(v, v_i)
            suivant[v] ← v_i
        fin
    fin
    // coupure de branches
    si p(v, suivant[v]) < θ alors
        suivant[v] ← v
    fin
fin
```

(a) maillage initial (b) partition (c) arbres



(d) détail arbres

FIGURE 5.3 – Visualisation des trajectoires des sommets lors du Quick Shift. Chaque arbre correspond à une région. La luminance d'un nœud dépend de son nombre de descendants (plus un nœud est clair, plus il a de descendants). $\sigma_s = 2.0$, $\sigma_r = 1.0$, $\sigma_d = 5.0$, $\sigma_q = 1.0$ et $\theta = 0.75$. 142 modes.

Algorithme 5 : Calcul du mode associé à chaque sommet.

pour tous les *sommets v du maillage* **faire**
  $actuel \leftarrow v$
  $parent \leftarrow suivant[v]$
  tant que $actuel \neq parent$ **faire**
    $actuel \leftarrow parent$
    $parent \leftarrow suivant[actuel]$
  fin
  $mode[v] \leftarrow actuel$
fin

5.3.5 Optimisations

De la même façon que pour la segmentation d'images par Quick Shift [FS10], il est possible de limiter fortement le nombre de voisins à examiner aux étapes de calcul de score et de calcul des suivants. Tout d'abord, nous limitons le rayon de recherche des voisins pour le calcul de score de représentativité à $3\sigma_{d_1}$. En effet, au delà de ce rayon, la contribution du voisin au score sera très faible, voire négligeable.

Ensuite, la phase de recherche du suivant de chaque sommet comporte une étape coupant tous les liens pour lesquels la valeur de proximité est inférieure à un seuil θ. La valeur de proximité dépendant de la distance euclidienne entre le sommet et le voisin, on peut limiter le rayon de recherche à la distance à partir de laquelle tous les liens seront coupés.

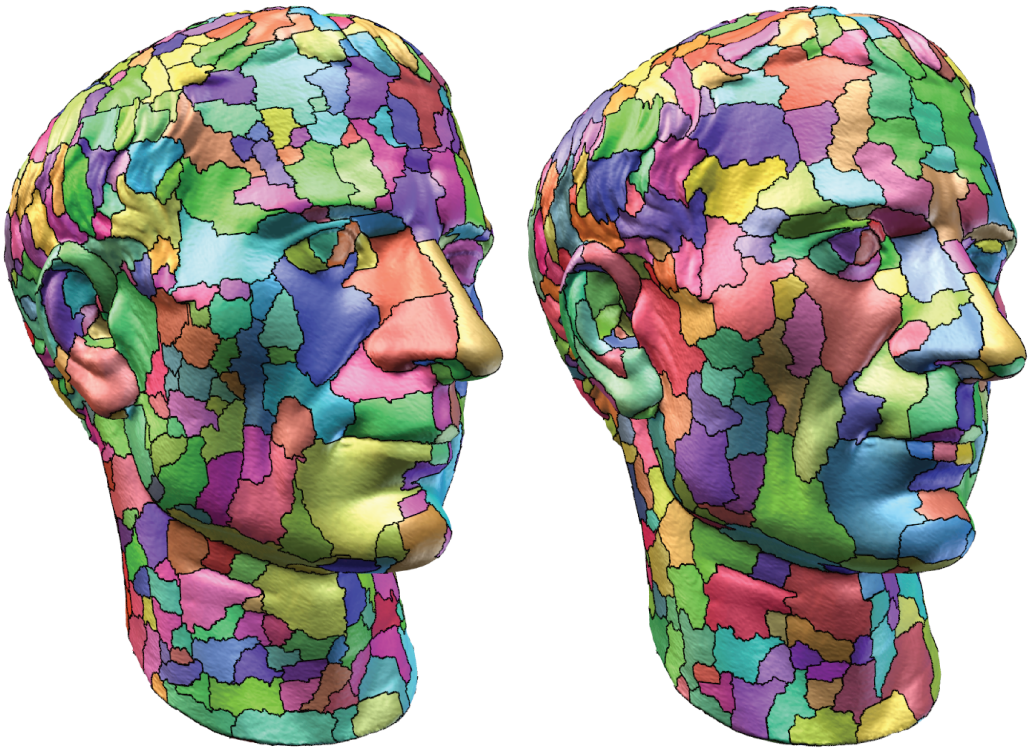
Ces deux optimisations permettent de traiter chaque sommet uniquement à l'aide de son voisinage proche, dépendant de σ_{d_1} et θ, et non de l'ensemble des sommets.

5.3.6 Paramètres

Notre méthode demande de fixer un certain nombre de paramètres :

- les paramètres du filtrage initial, explicités dans la section 4.3.3.
- les paramètres du calcul du score de représentativité σ_{d_1} et σ_{q_1}.
- les paramètres du Quick Shift σ_{d_2}, σ_{q_2} et θ.

Le score de représentativité d'un sommet v peut s'exprimer comme la moyenne (pondérée par les aires) des valeurs de proximité $p(v_i, v)$ de v aux voisins v_i.



(a) $\sigma_q = 3.5$ (b) $\sigma_q = 1.0$

FIGURE 5.4 – **Influence de σ_q.** Ce paramètre contrôle l'importance de la QEM dans le calcul du score et des valeurs de proximité. Lorsque la QEM n'est pas assez prise en compte par rapport à la distance euclidienne (à gauche), les contours des régions adhèrent moins aux caractéristiques du modèles, par exemple au niveau de l'œil ou de la bouche. 600 modes pour les deux segmentations. $\sigma_s = 2.0$, $\sigma_r = 1.0$, $\sigma_d = 7.0$.

$$S_v = \frac{1}{\sum a_i} \sum_{v_i \in V_{\sigma_{d_1}}} a_i \, p(v_i, v)$$

Pour avoir une mesure de proximité cohérente tout au long de l'algorithme, prenant en compte la distance euclidienne et l'erreur quadrique de la même façon, nous proposons donc de fixer $\boldsymbol{\sigma_d} = \sigma_{d_1} = \sigma_{d_2}$ et $\boldsymbol{\sigma_q} = \sigma_{q_1} = \sigma_{q_2}$ pour la suite.

Le paramètre θ donne un seuil sur la valeur de proximité en dessous duquel on coupera tous les liens. C'est donc ce paramètre qui permettra d'ajuster le nombre de modes et de régions de la segmentation finale.

5.4 Résultats et discussion

Comme pour le filtrage, notre algorithme de segmentation a été implémenté à l'aide de CUDA. Encore une fois, la recherche de voisins dans un rayon donné est basée sur la méthode présentée par Hoetzlein [Hoe14]. La parallélisation de notre algorithme est immédiate, on peut voir dans les algorithmes 4 et 5 que tous les sommets peuvent être traités indépendamment en parallèle plutôt que dans une boucle. Les expérimentations présentées ont été réalisées sur un PC équipé d'une carte graphique GeForce GTX 980 Ti, et d'un processeur Intel Xeon E5-1620 3.6GHz. Pour tous les résultats, σ_s, σ_r, σ_d et σ_q sont exprimés en fonction de la longueur moyenne des arêtes, et θ entre 0 et 1 : à $\theta = 1$ aucun lien n'est coupé, à $\theta = 0$ ils le sont tous.

La figure 5.1 montre l'influence du paramètre θ sur le résultat de la segmentation. L'arbre créé par le Quick Shift est le même pour les trois résultats, le seul changement est la tolérance sur la valeur de proximité associée à chaque branche. Plus θ est bas, plus on coupera de branches. Cela met en avant le caractère hiérarchique de la segmentation obtenue par Quick Shift. Les segmentations (a) et (b) peuvent être calculées à partir de la segmentation (c), simplement en parcourant l'arbre avec un nouveau seuil. Il est inutile de répéter les étapes plus coûteuses de calcul de score et de Quick Shift. Cette hiérarchie peut être exploitée lorsque l'on a besoin de plusieurs segmentations d'un même maillage, ou pour ajuster rapidement le nombre de modes obtenus. En effet, notre méthode utilisée telle quelle ne permet pas de fixer le nombre n_r de régions de la partition. Cependant, étant donnée une segmentation avec un nombre de modes $< n_r$, il est simple de couper le nombre de branches nécessaires pour atteindre n_r sans devoir refaire tous les calculs. À l'inverse, pour une segmentation avec un nombre de modes $> n_r$, on connectera les modes les plus proches pour atteindre le nombre de régions souhaité.

La figure 5.4 illustre le rôle de σ_q dans la segmentation. Ce paramètre détermine l'importance donnée à la QEM dans le calcul des scores et des valeurs de proximité lors du Quick Shift. Pour que les frontières des régions adhèrent bien aux caractéristiques présentes dans la géométrie, il est nécessaire que la QEM soit suffisamment prise en compte. Ici les deux segmentations comprennent 600 modes. À gauche, la QEM a une influence suffisamment forte pour que les yeux et la bouche soient correctement segmentés, tandis qu'à droite, les régions suivent beaucoup moins la géométrie.

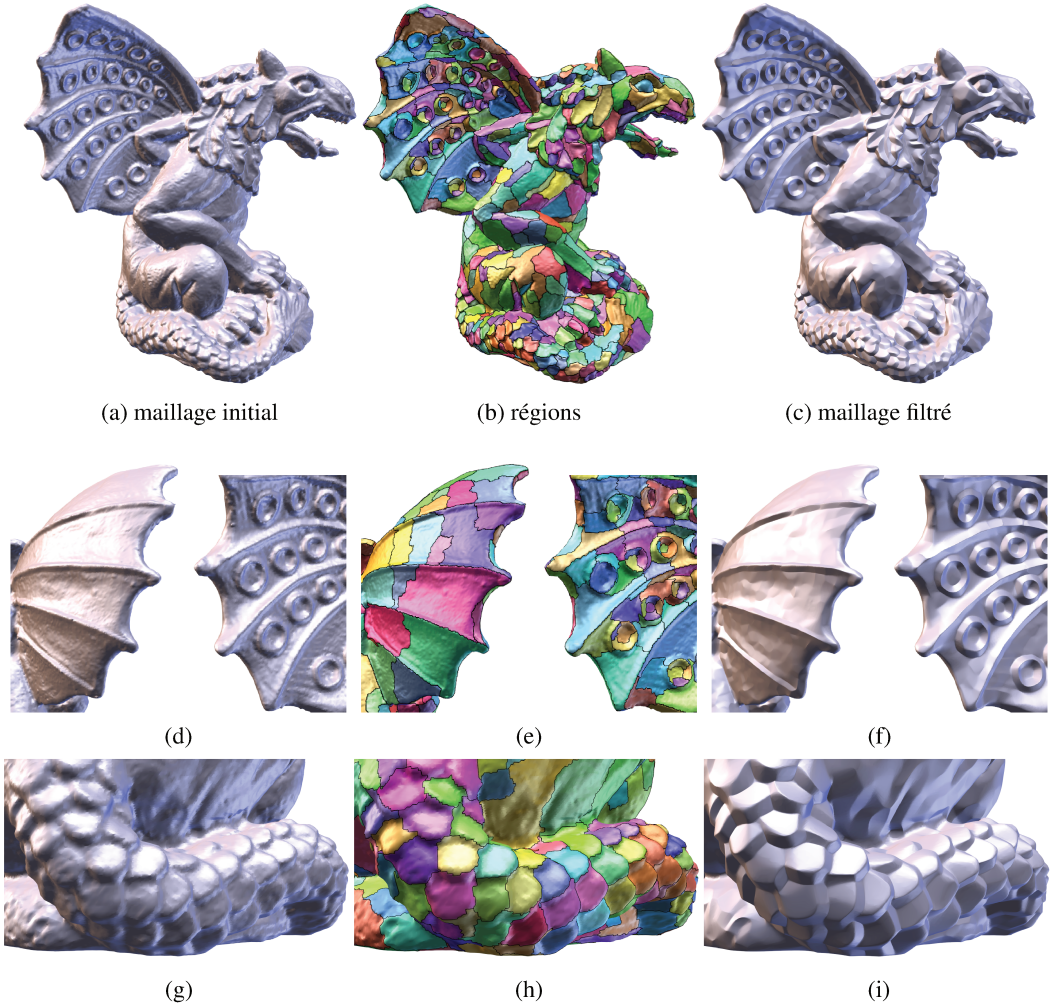
La première étape de notre méthode consiste à associer à chaque sommet une quadrique filtrée, calculée par la méthode présentée à la section 4.3.3. Dans la figure 5.5, nous comparons une segmentation d'un modèle avec le filtrage obtenu en utilisant les mêmes paramètres σ_s et σ_r. On observe bien que les caractéristiques de la surface mises en valeur par les deux algorithmes sont les mêmes, et que les frontières des régions de la segmentation suivent en grande partie les arêtes vives accentuées dans le filtrage. En effet, c'est le filtrage initial des quadriques qui pilote en grande partie la forme des régions obtenues ensuite par le Quick Shift.

Dans la figure 5.6, nous comparons notre méthode à un Quick Shift calculé uniquement à l'aide de la distance euclidienne. Nous obtenons avec notre algorithme une segmentation suivant les caractéristiques de la géométrie, pour un temps de traitement du même ordre de grandeur. Cela met encore une fois en évidence le rapport coût/performance intéressant de l'utilisation des quadriques pour décrire localement une surface. Nous comparons également notre méthode à l'algorithme de calcul de superfacettes décrit par Simari et al. [SPDF14], grâce à l'implémentation CPU mise à disposition en ligne par les auteurs. La figure 5.8 montre des résultats obtenus avec les deux méthodes, pour le même nombre de régions. Notre algorithme permet d'obtenir une segmentation comparable, pour un temps de calcul plusieurs ordres de grandeur plus rapide. Il est à noter néanmoins que, contrairement à Simari et al., nous ne prenons pas en compte de critères de compacité et d'uniformité des régions.

Notre méthode présente certaines limitations qui peuvent dans certains cas demander d'ajouter une étape de post-traitement. La première est illustrée dans la figure 5.7. Lors du Quick Shift, la connectivité du maillage n'est pas utilisée, on considère uniquement un ensemble de sommets munis d'une quadrique et d'un score. Par conséquent, la connectivité des régions n'est pas assurée, comme on peut le constater au niveau de l'oreille. Si nécessaire, on appliquera un post-traitement pour ré-assigner ces sommets isolés à une des régions auxquelles ils sont connectés. Bien que cela puisse constituer une limitation dans le cas des maillages, le fait de ne pas dépendre de la connectivité nous permet d'envisager une extension au traitement des nuages de points dans des travaux futurs. Le Quick Shift n'impose pas non plus de contraintes sur la taille et la régularité des régions.

modèle	**#triangles**	σ_s	σ_r	σ_d	σ_q	θ	**#modes**	**temps** (ms)
Bunny (fig. 5.7)	70K	3.0	1.0	5.0	0.8	0.9	89	161
Hand (fig. 5.3)	100K	2.0	1.0	5.0	1.0	0.75	142	174
Egea (fig. 5.2)	200K	2.0	1.0	6.0	1.0	0.87	156	419
Caesar (fig. 5.4)	750K	2.0	1.0	7.0	1.0	0.75	600	1691
Grog (fig. 5.1)	1M	2.0	1.0	6.0	1.0	0.5	2035	1676
		2.0	1.0	6.0	1.0	0.75	1120	1883
		2.0	1.0	6.0	1.0	0.99	449	2864
Gargoyle (fig. 5.5)	1.7M	3.0	1.0	6.0	1.0	0.95	1119	3928
Raptor (fig. 5.9)	2M	3.0	1.0	4.0	1.0	0.95	1389	2287
Dragon (fig. 5.9)	7M	4.0	1.0	3.0	1.0	0.999	2701	10557

TABLE 5.1 – **Mesures de performance** de notre partitionnement en ms pour les modèles utilisés dans les figures, sans les transferts mémoire CPU-GPU. σ_s et σ_r sont les paramètres de filtrages des quadriques. σ_d, σ_q et θ sont les paramètres du Quick Shift. σ_s, σ_r, σ_d et σ_q sont exprimés en fonction de la longueur moyenne des arêtes.



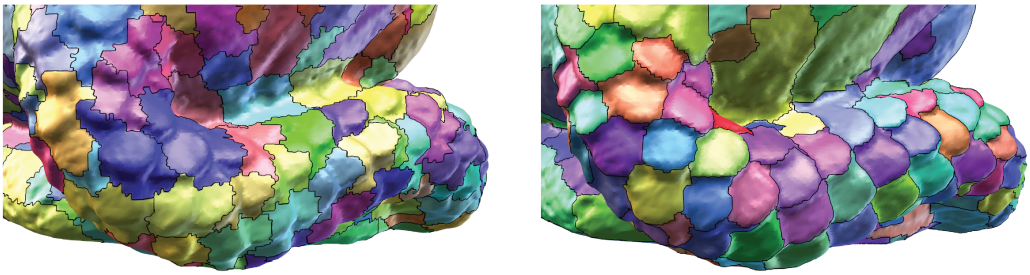
(a) maillage initial (b) régions (c) maillage filtré

(d) (e) (f)

(g) (h) (i)

FIGURE 5.5 – Parallèle avec les résultats obtenus avec le filtrage bilatéral des quadriques présenté en 4, pour les même paramètres σ_s et σ_r. On observe que les régions de la segmentation suivent bien les caractéristiques accentuées par le filtrage, comme les écailles ou les arêtes vives sur les ailes. $\sigma_s = 3.0$, $\sigma_r = 1.0$, $\sigma_d = 6.0$, $\sigma_q = 1.0$, $\theta = 0.95$. 1119 modes.

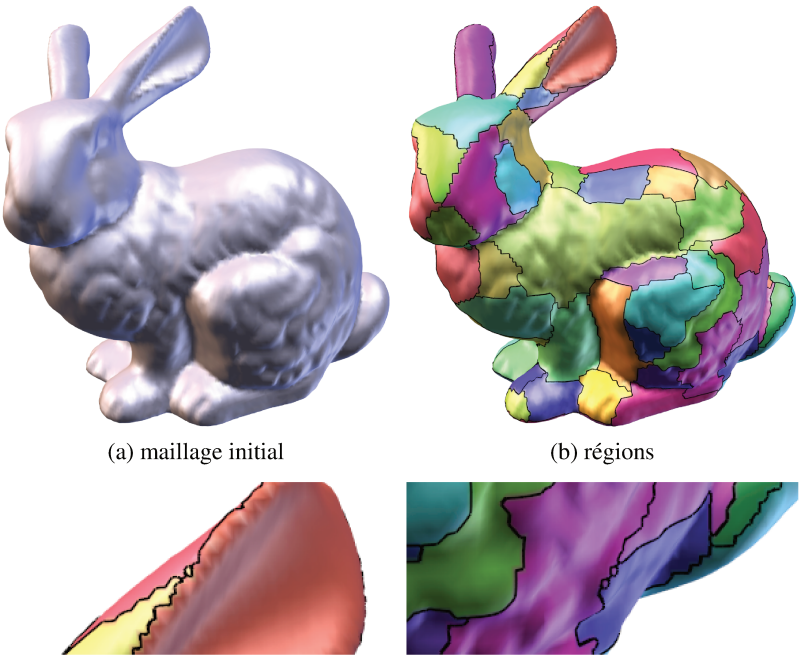
Le tableau 5.1 présente les mesures de temps obtenues pour les modèles des différentes figures avec les paramètres utilisés. Les mesures n'incluent pas les transferts mémoire. Le temps d'exécution est influencé principalement par la taille du maillage, le paramètre σ_d et le paramètre θ. En effet, ce sont ces paramètres qui permettent de limiter le rayon de la recherche de voisins à l'étape de calcul du score et à l'étape de Quick Shift. On note qu'on atteint des temps interactifs pour les modèles d'une taille de l'ordre de 100K triangles. Notre méthode passe à l'échelle et permet de segmenter des maillages ayant plusieurs millions de polygones en quelques secondes (voir figure 5.9).



(a) Quick Shift distance euclidienne : 1877 ms

(b) notre méthode : 3928 ms

FIGURE 5.6 – Comparaison de notre méthode avec un Quick Shift sur la distance euclidienne uniquement. Les temps obtenus sont du même ordre de grandeur, pour une segmentation de bien meilleure qualité. (mêmes paramètres que pour la figure 5.5, 1119 modes dans les deux cas).



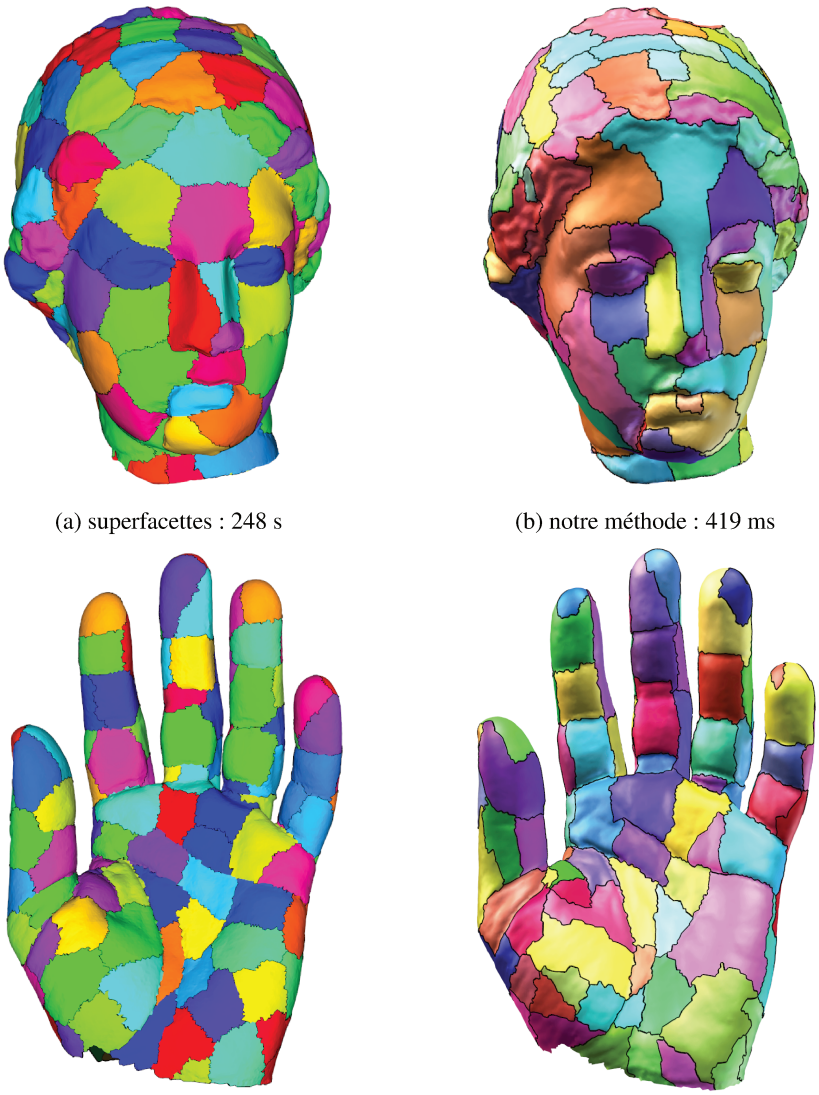
(a) maillage initial

(b) régions

(c) région déconnectée

(d) région déconnectée

FIGURE 5.7 – La connectivité des régions n'est pas contrôlée, ce qui peut engendrer des sommets déconnectés de la région contenant leur mode. Ceux-ci peuvent éventuellement être ré-assignés lors d'un post-traitement. $\sigma_s = 3.0$, $\sigma_r = 1.0$, $\sigma_d = 5.0$, $\sigma_q = 0.8$, $\theta = 0.9$. 89 modes.



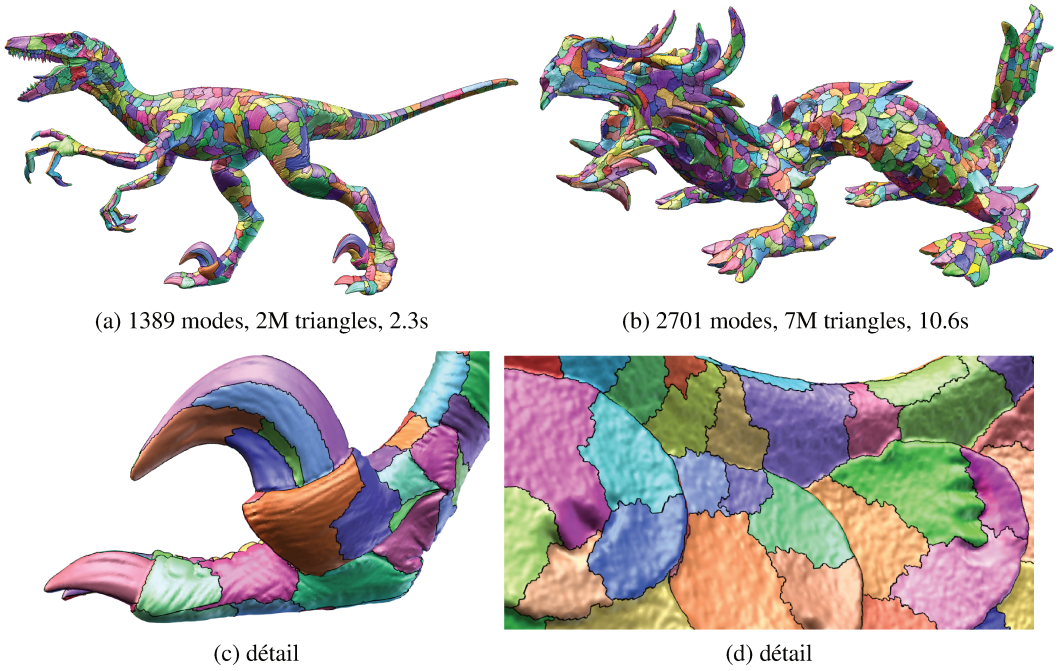
(a) superfacettes : 248 s

(b) notre méthode : 419 ms

(c) superfacettes : 87 s

(d) notre méthode : 174 ms

FIGURE 5.8 – Comparaison de notre méthode avec les superfacettes de Simari et al. [SPDF14]. Notre méthode permet d'obtenir des résultats comparables, en temps interactif. Les paramètres utilisés pour notre méthode sont les mêmes que pour les figures 5.2 et 5.3. Les paramètres choisis pour les superfacettes sont un nombre de régions identique, et $\alpha = 200$ (paramètre recommandé dans l'article original [SPDF14]).



(a) 1389 modes, 2M triangles, 2.3s

(b) 2701 modes, 7M triangles, 10.6s

(c) détail

(d) détail

FIGURE 5.9 – Notre méthode passe à l'échelle et est capable de segmenter des maillages de plusieurs millions de polygones en un temps raisonnable. **Raptor :** $\sigma_s = 3.0$, $\sigma_r = 1.0$, $\sigma_d = 4.0$, $\sigma_q = 1.0$, $\theta = 0.95$. **Dragon :** $\sigma_s = 4.0$, $\sigma_r = 1.0$, $\sigma_d = 3.0$, $\sigma_q = 1.0$, $\theta = 0.999$.

Conclusion

Nous avons présenté une méthode de segmentation de surfaces basée sur des quadriques filtrées et sur l'algorithme du Quick Shift. Cette approche est rapide, simple et intrinsèquement parallèle, et passe à l'échelle pour partitionner des maillages ayant jusqu'à plusieurs millions de polygones. La structure d'arbres obtenue par le Quick Shift est flexible, et permet d'ajuster la segmentation obtenue, ou de dériver plusieurs autres partitions à partir d'un même Quick Shift. Les régions obtenues adhèrent bien aux caractéristiques de la géométrie, et peuvent être vues comme des superpixels 3D (ou superfacettes) à utiliser comme primitives pour des traitements plus complexes.

CHAPITRE 6

CONCLUSION

Au cours de cette thèse, nous nous sommes intéressés au traitement rapide de grandes quantités de données géométriques. Nous avons proposé de nouvelles méthodes permettant de générer rapidement, à partir de ces données, des représentations intermédiaires indispensables pour l'analyse ou les traitements complexes. En particulier, nous avons abordé cette problématique sous l'angle de la *simplification*, avec un algorithme approximant la surface par un nombre réduit de primitives en temps réel. Nous avons également présenté un algorithme de *filtrage* permettant d'obtenir en temps interactif des surfaces à la fois plus lisses et dont les caractéristiques saillantes sont accentuées. Enfin, nous avons introduit une méthode de recherche de modes et de *segmentation* rapide de surface, autorisant l'utilisation de traitements coûteux par régions lorsqu'ils sont impraticables par sommets.

Toutes ces méthodes sont fondées sur l'utilisation de la QEM, et mettent en évidence l'intérêt de décrire localement la surface par des quadriques. Cette représentation est compacte et peu coûteuse à manipuler, notamment en parallèle, tout en capturant efficacement les caractéristiques de la géométrie locale. En cela elle constitue une bonne base sur laquelle construire des algorithmes de traitement géométrique rapides et efficaces.

En particulier, nous avons présenté un algorithme haute performance de simplification adaptative de surface, tirant parti du nouveau concept *d'intégration de Morton* pour générer et exploiter une partition hiérarchique du modèle, tout en maintenant les attributs et les mesures d'erreur à la surface. Cette hiérarchie permet de générer une version simplifiée du modèle, dont chaque sommet est placé de manière à rester sous un seuil d'erreur fixé par l'utilisateur. Cet algorithme, pensé pour les architectures parallèles, simplifie en temps réel des objets constitués de plusieurs millions de polygones, tout en gardant un bon compromis sur la qualité du maillage obtenu, préservant au mieux les caractéristiques.

Deuxièmement, nous avons abordé la question du filtrage, à travers plusieurs variations d'une méthode de filtrage bilatéral basé sur la QEM. Plus particulièrement, nous avons proposé une méthode qui s'appuie sur un filtrage des quadriques aux sommets du modèle, puis

définit les sommets de la nouvelle surface comme les optimiseurs des quadriques filtrées obtenues. Le caractère parallèle de cette approche lui permet de s'exécuter en temps interactif sur GPU, pour des modèles de plusieurs millions de polygones. La surface est lissée, y compris dans des conditions de bruit important, tandis que les caractéristiques saillantes de la forme sont accentuées. Ainsi, selon les paramètres choisis, les résultats obtenus iront du débruitage jusqu'à une stylisation particulière du modèle.

Enfin nous nous sommes intéressés à la segmentation de surface, et avons proposé une approche tirant parti du filtrage des quadriques introduit précédemment. Cette méthode s'appuie sur l'algorithme du *Quick Shift*, et est capable de partitionner des maillages de plusieurs millions de polygones, tout en restant efficace en mémoire et en temps. Sa rapidité et sa capacité à passer à l'échelle en font un pré-traitement idéal pour l'analyse et le traitement géométrique de maillages de grande taille, par exemple des données scannées, ce qui rejoint le concept des superpixels dans les domaines de la vision et du traitement d'image.

Plusieurs directions de recherche sont envisagées dans le prolongement des travaux présentés ici. En premier lieu, nous traitons pour l'instant principalement des maillages statiques. Il nous paraît intéressant d'étendre les approches abordées pour couvrir de manière plus adaptée les différents types de données géométriques capturées. En particulier, les méthodes utilisant la QEM peuvent, dans une certaine mesure, s'appliquer directement aux nuages de points munis de normales, comme nous l'avons vu pour la simplification. En effet, la connectivité du maillage est peu utilisée dans les étapes principales des algorithmes proposés, puisque nous avons privilégié l'utilisation d'un voisinage spatial pour des raisons de performance. Néanmoins, une adaptation directe reste insuffisante, par exemple dans le cas d'un échantillonnage non uniforme. L'importance relative des quadriques, estimée jusqu'ici grâce à l'aire des triangles, n'est pas simple à obtenir efficacement pour un nuage de points. Les problématiques de ce type, propres aux nuages de points, s'inscrivent dans la continuité directe des travaux présentés dans cette thèse.

En outre, dans le cas de flux de données animées, de nombreuses informations peuvent être réutilisées d'un instant à l'autre en exploitant leur cohérence temporelle, afin de gagner en efficacité ou en précision. Par exemple, tout comme les super-pixels sont utilisés pour le traitement vidéo, nous pourrions construire notre segmentation par quadriques en ajoutant une dimension temporelle, pour prédire ou améliorer les données 3D + temps.

Traiter directement les nuages de points permettrait également d'explorer la question de la reconstruction de surface. En effet, les propriétés souhaitables pour les méthodes de reconstruction incluent notamment la rapidité de traitement, l'efficacité en mémoire, la capacité à traiter de grandes quantités de données, la robustesse au bruit et la bonne restitution des caractéristiques saillantes de la forme. Au cours de cette thèse, la QEM est un des éléments qui a permis de remplir ces conditions dans le cadre de la simplification, du filtrage et de la segmentation. Nous pensons qu'il serait intéressant d'explorer son utilisation pour la reconstruction de surface. De plus, la structure hiérarchique utilisant l'intégration de Morton nous paraît particulièrement adaptée dans ce contexte.

ANNEXE A

PRODUCTION

Ces travaux ont donné lieu au développement d'un logiciel implémenté en C++/CUDA, comprenant notamment une implémentation GPU et CPU de notre algorithme de simplification, et ayant fait l'objet d'un dépôt APP de code. Une implémentation GPU des algorithmes de filtrage et de segmentation, une interface permettant de charger, simplifier, visualiser et sauvegarder des nuages de points ont également été développés.

Nos résultats sur la simplification de géométrie ont fait l'objet d'une publication [LB15] :

Morton Integrals for High Speed Geometry Simplification, *Hélène Legrand et Tamy Boubekeur*, ACM SIGGRAPH/Eurographics High Performance Graphics 2015 - Second prix au Wolfgang Strasse Award for the Best Paper

Un brevet a également été déposé (PCT International Application Number PCT/FR2016/051030 filled on 05602-2016).

Un article couvrant les chapitres 4 et 5 est en cours de soumission.

BIBLIOGRAPHIE

[AFS+11] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10) :105–112, 2011.

[AGDL09] Andrew Adams, Natasha Gelfand, Jennifer Dolson, and Marc Levoy. Gaussian kd-trees for fast high-dimensional filtering. *Trans. Graph.*, 28(3) :21 :1–21 :12, 2009.

[ASS+12] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11) :2274–2282, 2012.

[BA09] Tamy Boubekeur and Marc Alexa. Mesh simplification by stochastic sampling and topological clustering. *Computer & Graphics (Proc. Shape Modeling International)*, 33(3) :241–249, 2009.

[BCM05] A. Buades, B. Coll, and J.-M. Morel. A non-local algorithm for image denoising. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 60 – 65, 2005.

[BHB+11] Thabo Beeler, Fabian Hahn, Derek Bradley, Bernd Bickel, Paul Beardsley, Craig Gotsman, Robert W. Sumner, and Markus Gross. High-quality passive facial performance capture using anchor frames. *Trans. Graph.*, 30 :75 :1–75 :10, 2011.

[CDR00] Ulrich Clarenz, Udo Diewald, and Martin Rumpf. Anisotropic geometric diffusion in surface processing. In *Visualization 2000. Proceedings*, pages 397–405. IEEE, 2000.

[CM02] Dorin Comaniciu and Peter Meer. Mean shift : A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5) :603–619, 2002.

[CPD07] Jiawen Chen, Sylvain Paris, and Frédo Durand. Real-time edge-aware image processing with the bilateral grid. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[Cro84] Franklin C Crow. Summed-area tables for texture mapping. In *SIGGRAPH*, pages 207–212, 1984.

[CRS98] P. Cignoni, C. Rocchini, and R. Scopigno. Metro : measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2) :167–174, 1998.

[CSAD04] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. *Trans. Graph.*, 23(3) :905–914, 2004.

[dAST+] E. de Aguiar, C. Stoll, C. Theobalt, N. Ahmed, and H.P. Seidel. Performance capture from sparse multi-view video. *Trans. Graph.*, 27(3) :Art. 98.

[DD02] Frédo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *ACM transactions on graphics (TOG)*, volume 21, pages 257–266. ACM, 2002.

[DMSB99] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 317–324, 1999.

[DMSB00] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H Barr. Anisotropic feature-preserving denoising of height fields and bivariate data. In *Graphics interface*, volume 11, 2000.

[DT07] Christopher DeCoro and Natalya Tatarchuk. Real-time mesh simplification using the GPU. In *ACM I3D*, pages 161–166, 2007.

[FDCO03] Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. Bilateral mesh denoising. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 950–953, 2003.

[FS10] Brian Fulkerson and Stefano Soatto. Really quick shift : image segmentation on a gpu. In *Proceedings of the 11th European conference on Trends and Topics in Computer Vision-Volume Part II*, pages 350–358. Springer-Verlag, 2010.

[GAB12] Thierry Guillemot, Andres Almansa, and Tamy Boubekeur. Non local point set surfaces. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*, pages 324–331. IEEE, 2012.

[Gar99] Michael Garland. *Quadric-based polygonal surface simplification*. PhD thesis, Georgia Institute of Technology, 1999.

[GDG11] Nico Grund, Evgenij Derzapf, and Michael Guthe. Instant level-of-detail. In *Vision, Modeling and Visualization*, pages 293–299, 2011.

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. *SIGGRAPH*, pages 209–216, 1997.

[GPM11] Kirill Garanzha, Jacopo Pantaleoni, and David McAllister. Simpler and faster HLBVH with work queues. In *HPG*, pages 59–64, New York, New York, USA, 2011.

[Gra04] Markus Grabner. Towards multiscale mesh denoising. In *Proceedings of the 20th Spring Conference on Computer Graphics*, pages 212–215. ACM, 2004.

[GWH01] Michael Garland, Andrew Willmott, and Paul S Heckbert. Hierarchical face clustering on polygonal surfaces. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 49–58. ACM, 2001.

[HDD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *SIGGRAPH*, pages 19–26, 1993.

[Hoe14] RC Hoetzlein. Fast fixed-radius nearest neighbors : interactive million-particle fluids. In *GPU Technology Conference*, page 18, 2014.

[Hor05] Daniel Horn. Stream reduction operations for gpgpu applications. *GPU gems*, 2, 2005.

[HSO07] Mark Harris, Shubhabrata Sengupta, and John D Owens. Parallel prefix sum (scan) with cuda. *GPU gems*, 3(39) :851–876, 2007.

[HZR06] Xuming He, Richard Zemel, and Debajyoti Ray. Learning and incorporating top-down cues in image segmentation. *Computer Vision–ECCV 2006*, pages 338–351, 2006.

[JD03] Thouis R. Jones and Frédo Durand. Non-iterative, feature-preserving mesh smoothing. *ACM Transactions on Graphics*, 22 :943–949, 2003.

[Kar12] Tero Karras. Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. In *High Performance Graphics*, pages 33–37, 2012.

[KCLU07] Johannes Kopf, Michael F Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. In *ACM Transactions on Graphics (ToG)*, volume 26, page 96. ACM, 2007.

[LB15] Hélène Legrand and Tamy Boubekeur. Morton integrals for high speed geometry simplification. In *Proceedings of the 7th Conference on High-Performance Graphics*, HPG '15, pages 105–112. ACM, 2015.

[LGS+09] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast BVH Construction on GPUs. *Computer Graphics Forum*, 28(2) :375–384, April 2009.

[Lin00] Peter Lindstrom. Out-of-core simplification of large polygonal models. *SIGGRAPH*, pages 259–262, 2000.

[Lin03] Peter Lindstrom. Out-of-core construction and visualization of multiresolution surfaces. In *I3D*, pages 93–102, 2003.

[LKK15] Jung Lee, Seokhun Kim, and Sun-Jeong Kim. Mesh segmentation based on curvatures using the gpu. *Multimedia Tools and Applications*, 74(10) :3401, 2015.

[LPC+00] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project : 3d scanning of large statues. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 131–144, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[MPW+09] Alastair P Moore, Simon JD Prince, Jonathan Warrell, Umar Mohammed, and Graham Jones. Scene shape priors for superpixel segmentation. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 771–778. IEEE, 2009.

[NF10] Michal Nociar and Andrej Ferko. Feature-preserving mesh denoising via attenuated bilateral normal filtering and quadrics. In *Proceedings of the 26th Spring Conference on Computer Graphics*, pages 149–156. ACM, 2010.

[OBA+05] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. In *Acm Siggraph 2005 Courses*, page 173. ACM, 2005.

[OBS02] Y. Ohtake, A.G. Belyaev, and Hans-Peter Seidel. Mesh smoothing by adaptive and anisotropic gaussian filter applied to mesh normals. In *VISION MODELING AND VISUALIZATION*, pages 203–2010, 2002.

[PL10] Jacopo Pantaleoni and D. Luebke. HLBVH : hierarchical LBVH construction for real-time ray tracing of dynamic geometry. In *High Performance Graphics*, pages 87–95, 2010.

[PSK+02] DL Page, Y Sun, AF Koschan, JK Paik, and MA Abidi. Simultaneous mesh simplification and noise smoothing of range images. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 3, pages 821–824. IEEE, 2002.

[RB93] Jarek Rossignac and Paul Borrel. Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in Computer Graphics*, pages 455–465, 1993.

[RM03] Xiaofeng Ren and Jitendra Malik. Learning a classification model for segmentation. In *null*, page 10. IEEE, 2003.

[SG01] E. Shaffer and M. Garland. Efficient adaptive simplification of massive meshes. In *Visualization*, pages 127–551, 2001.

[SG05] Eric Shaffer and Michael Garland. A multiresolution representation for massive meshes. *TVCG*, 11(2) :139–148, 2005.

[SKK07] Yaser Ajmal Sheikh, Erum Arif Khan, and Takeo Kanade. Mode-seeking by medoidshifts. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[SPDF14] Patricio Simari, Giulia Picciau, and Leila De Floriani. Fast and scalable mesh superfacets. In *Computer Graphics Forum*, volume 33, pages 181–190. Wiley Online Library, 2014.

[SW03] S Schaefer and J Warren. Adaptive vertex clustering using octrees. In *Geom. Design & Computing*, pages 491–500, 2003.

[Tau95] Gabriel Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 351–358, 1995.

[TGB13] Jean-Marc Thiery, Émilie Guy, and Tamy Boubekeur. Sphere-meshes : shape approximation using spherical quadric error metrics. *ACM Transactions on Graphics (TOG)*, 32(6) :178, 2013.

[TM98] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, pages 839–, 1998.

[VB11] Guillaume Vialaneix and Tamy Boubekeur. SBL Mesh Filter : A Fast Separable Approximation of Bilateral Mesh Filtering. In Peter Eisert, Joachim Hornegger, and Konrad Polthier, editors, *Vision, Modeling, and Visualization (2011)*. The Eurographics Association, 2011.

[VJ01] Paul A. Viola and Michael J. Jones. Robust real-time face detection. In *ICCV*, page 747, 2001.

[VNMC10] Antonio W Vieira, Armando A Neto, Douglas G Macharet, and Mario FM Campos. Mesh denoising using quadric error metric. In *Graphics, Patterns and Images (SIBGRAPI), 2010 23rd SIBGRAPI Conference on*, pages 247–254. IEEE, 2010.

[VS08] Andrea Vedaldi and Stefano Soatto. Quick shift and kernel methods for mode seeking. *Computer vision–ECCV 2008*, pages 705–718, 2008.

[WFL+15] Peng-Shuai Wang, Xiao-Ming Fu, Yang Liu, Xin Tong, Shi-Lin Liu, and Baining Guo. Rolling guidance normal filter for geometric processing. *ACM Trans. Graph.*, 34(6) :173 :1–173 :9, October 2015.

[WLT16] Peng-Shuai Wang, Yang Liu, and Xin Tong. Mesh denoising via cascaded normal regression. *ACM Transactions on Graphics (TOG)*, 35(6) :232, 2016.

[YBS06] Shin Yoshizawa, Alexander Belyaev, and Hans-Peter Seidel. Smoothing by example : Mesh denoising by averaging with similarity-based weights. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*, pages 9–, 2006.

[YHGT10] Jason C Yang, Justin Hensley, Holger Grün, and Nicolas Thibieroz. Real-time concurrent linked list construction on the gpu. In *Computer Graphics Forum*, volume 29, pages 1297–1304. Wiley Online Library, 2010.

[YLL+05] Hitoshi Yamauchi, Seungyong Lee, Yunjin Lee, Yutaka Ohtake, Alexander Belyaev, and H-P Seidel. Feature sensitive mesh segmentation with mean shift. In *Shape Modeling and Applications, 2005 International Conference*, pages 236–243. IEEE, 2005.

[YOB02] Hirokazu Yagou, Yutaka Ohtake, and Alexander Belyaev. Mesh smoothing via mean and median filtering applied to face normals. In *Proceedings of the Geometric Modeling and Processing &#8212 ; Theory and Applications (GMP'02)*, GMP '02, pages 124–, 2002.

[YOB03] H. Yagou, Y. Ohtake, and A.G. Belyaev. Mesh denoising via iterative alpha-trimming and nonlinear diffusion of normals with automatic thresholding. In *Computer Graphics International, 2003. Proceedings*, pages 28 – 33, 2003.

[ZFAT11] Youyi Zheng, Hongbo Fu, Oscar Kin-Chung Au, and Chiew-Lan Tai. Bilateral normal filtering for mesh denoising. *IEEE Transactions on Visualization and Computer Graphics*, 17(10) :1521–1530, 2011.

[ZGHG10] Kun Zhou, Minmin Gong, Xin Huang, and Baining Guo. Data-Parallel Octrees for Surface Reconstruction. *TVCG*, 17(5) :669–681, 2010.

[ZSL+13] Luming Zhang, Mingli Song, Zicheng Liu, Xiao Liu, Jiajun Bu, and Chun Chen. Probabilistic graphlet cut : Exploiting spatial structure cue for weakly supervised image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1908–1915, 2013.

[ZSXJ14] Qi Zhang, Xiaoyong Shen, Li Xu, and Jiaya Jia. Rolling guidance filter. In *European Conference on Computer Vision*, pages 815–830. Springer, 2014.

Algorithmes parallèles pour le traitement rapide de géométries 3D

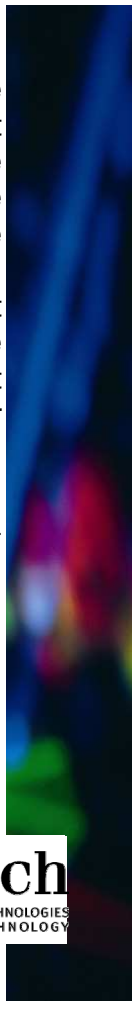
Hélène LEGRAND

RESUME : Au cours des vingt dernières années, les principaux concepts du traitement du signal ont trouvé leur homologue pour le cas de la géométrie numérique, et en particulier des modèles polygonaux de surfaces 3D. Ces traitements requièrent néanmoins un temps de calcul non négligeable lorsqu'on les applique sur des modèles de taille conséquente. Cette charge de calcul devient un frein important dans le contexte actuel, où les quantités massives de données 3D générées à chaque seconde peuvent potentiellement nécessiter l'application d'un sous-ensemble de ces opérateurs. La capacité à exécuter des opérateurs de traitement géométrique en un temps très court représente alors un verrou important pour les systèmes de conception, capture et restitution 3D dynamiques. Dans ce contexte, on cherche à accélérer de plusieurs ordres de grandeur certains algorithmes de traitement géométrique actuels, et à reformuler ou approcher ces algorithmes afin de diminuer leur complexité ou de les adapter à un environnement parallèle. Dans cette thèse, nous nous appuyons sur un objet compact et efficace permettant d'analyser les surfaces 3D à plusieurs échelles : les quadriques d'erreurs. En particulier, nous proposons de nouveaux algorithmes haute performance, maintenant à la surface des quadriques d'erreur représentatives de la géométrie. Un des principaux défis tient ici à la génération des structures adaptées en parallèle, afin d'exploiter les processeurs parallèles à grain fin que sont les GPU, la principale source de puissance disponible dans un ordinateur moderne.

MOTS-CLEFS : Informatique graphique, géométrie, modélisation, algorithmes GPU, traitements parallèles, métrique d'erreur quadrique, ordre de Morton

ABSTRACT : Over the last twenty years, the main signal processing concepts have been adapted for digital geometry, in particular for 3D polygonal meshes. However, the processing time required for large models is significant. This computational load becomes an obstacle in the current context, where the massive amounts of data that are generated every second may need to be processed with several operators. The ability to run geometry processing operators with strong time constraints is a critical challenge in dynamic 3D systems. In this context, we seek to speed up some of the current algorithms by several orders of magnitude, and to reformulate or approximate them in order to reduce their complexity or make them parallel. In this thesis, we are building on a compact and effective object to analyze 3D surfaces at different scales : error quadrics. In particular, we propose new high performance algorithms that maintain error quadrics on the surface to represent the geometry. One of the main challenges lies in the effective generation of the right structures for parallel processing, in order to take advantage of the GPU.

KEY-WORDS : Computer graphics, geometry, modelisation, GPU algorithms, parallel processing, quadric error metric, Morton order



TELECOM ParisTech

EDITE

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY