Université Lumière Lyon 2

Année 2013 $N°$

# THÈSE

pour obtenir le grade de

**Docteur**

Spécialité : **Informatique**

préparée au **Laboratoire d'InfoRmatique en Images et Systèmes d'information**

dans le cadre de l'Ecole Doctorale **Math-Info**

présentée et soutenue publiquement

par

**Ricardo Uribe Lobello**

le

04/12/2013

Titre :

**Génération de maillages adaptatifs à partir de données volumiques de grande taille**

Préparée sous la direction de

Florence Denis et Florent Dupont

## Composition du Jury

| | | |
|---|---|---|
| M. | Pierre Alliez, Directeur de Recherches INRIA, Sophia-Antipolis | Rapporteur |
| M. | Jacques-Olivier Lachaud, Professeur Université de Savoie | Rapporteur |
| M. | Mohamed Daoudi, Professeur Institut Mines-Télécom/Télécom Lille1 | Examinateur |
| M. | Gilles Gesquière, Professeur Université Lyon 2 | Examinateur |
| Mme | Annick Montanvert, Professeur Université Pierre Mendès France | Examinatrice |
| Mme | Florence Denis, Maître de Conferences Université Lyon 1 | Co-directrice de thèse |
| M. | Florent Dupont, Professeur Université Lyon 1 | Directeur de thèse |

# Résumé

Dans cette thèse, nous nous sommes intéressés au problème de l'extraction d'une surface à partir de la représentation volumique d'un objet. Dans ce but, nous nous sommes concentrés sur les méthodes de division spatiale. Ces approches divisent le volume afin de construire une approximation par morceaux de la surface de l'objet. L'idée générale consiste à faire des approximations surfaciques locales qui seront ensuite combinées pour extraire une surface unique représentant l'objet. Les approches basées sur l'algorithme " Marching Cubes " (MC) présentent des défaut par rapport à la qualité et l'adaptativité de la surface produite. Même si une considérable quantité d'améliorations ont été apportées à la méthode originale, la plus grande partie des algorithmes fournissent la solution à un ou deux défauts mais n'arrivent pas à surmonter toutes ses limitations. Les méthodes duales sont plus adaptées pour utiliser un échantillonnage adaptatif sur le volume d'intérêt. Ces méthodes reposent sur la génération de surfaces duales à celles construites par MC ou se basent sur des grilles duales. Elles construisent des maillages moins denses et en même temps capables de mieux approcher les détails de l'objet. De plus, des améliorations récentes garantissent que les maillages extraits ont de bonnes propriétés topologiques et géométriques.

Nous avons étudié les caractéristiques spécifiques des objets volumiques par rapport à leur géométrie et à leur topologie. Nous avons exploré l'état de l'art sur les approches de division spatiale afin d'identifier leurs avantages et leurs inconvénients ainsi que les implications de leur utilisation sur des objets volumiques. Nous avons conclu qu'une approche duale était la mieux adaptée pour obtenir un bon compromis entre qualité du maillage et qualité de l'approximation. Dans un second temps, nous avons proposé et développé un pipeline de génération de surfaces basé sur une combinaison d'une approche duale et de la recherche de composantes connexes n-dimensionnels pour mieux reproduire la topologie et la géométrie des objets originels. Dans un troisième temps, nous avons présenté une extension "out-of-core" de notre chaîne de traitements pour l'extraction des surfaces à partir de grands volumes. Le volume est divisé pour générer des morceaux de surface de manière indépendante et garde l'information nécessaire pour les connecter afin de produire une surface unique topologiquement correcte. L'approche utilisée permet de paralléliser le traitement pour accélérer l'obtention de la surface. Les tests réalisés ont permis de valider la méthode sur des données volumiques massives.

**Keywords :** Extraction de surface, génération de maillages, modèles adaptatifs.

# Abstract

In this document, we have been interested in the surface extraction from the volumetric representation of an object. With this objective in mind, we have studied the spatial subdivision surface extraction algorithms. This approaches divide the volume in order to build a piecewise approximation of the surface. The general idea is to combine local and simple approximations to extract a complete representation of the object's surface. The methods based on the Marching Cubes (MC) algorithm have problems to produce good quality and to handle adaptive surfaces. Even if a lot of improvements to MC have been proposed, these approaches solved one or two problems but they don't offer a complete solution to all the MC drawbacks. Dual methods are more adapted to use adaptive sampling over volumes. These methods generate surfaces that are dual to those generated by the Marching Cubes algorithm or dual grids in order to use MC methods. These solutions build adaptive meshes that represent well the features of the object. In addition, recent improvements guarantee that the produced meshes have good geometrical and topological properties.
In this dissertation, we have studied the main topological and geometrical properties of volumetric objects.In a first stage, we have explored the state of the art on spatial subdivision surface extraction methods in order to identify theirs advantages, theirs drawbacks and the implications of theirs application on volumetric objects. We have concluded that a dual approach is the best option to obtain a good compromise between mesh quality and geometrical approximation. In a second stage, we have developed a general pipeline for surface extraction based on a combination of dual methods and connected components extraction to better capture the topology and geometry of the original object. In a third stage, we have presented an out-of-core extension of our surface extraction pipeline in order to extract adaptive meshes from huge volumes. Volumes are divided in smaller sub-volumes that are processed independently to produce surface patches that are later combined in an unique and topologically correct surface. This approach can be implemented in parallel to speed up its performance. Test realized in a vast set of volumes have confirmed our results and the features of our solution.

**Mots clefs :** Iso-surface extraction, mesh generation, adaptive models.

# Table des matières

# Table des figures

# Liste des tableaux

# Chapitre 1

# Introduction

Dans cette thèse, nous nous sommes concentrés sur deux problématiques : l'extraction de surfaces adaptatives à partir de données volumiques et les techniques de génération de maillages pour pouvoir traiter des données volumiques de grande taille "out-of-core". Au fur et à mesure que les ordinateurs sont devenus des outils fondamentaux dans une myriade de domaines de recherche, ces deux problématiques ont été très explorées et constituent des sujets très actifs de recherche dans le domaine de l'informatique.

Les premiers algorithmes permettant de construire une surface à partir d'une autre représentation d'un objet sont apparus il y a une trentaine d'années. Ils ont eu un fort impact sur plusieurs champs de recherche tels que les algorithmes de rendu et de génération de mondes virtuels ainsi que dans l'exploration et l'analyse des données.

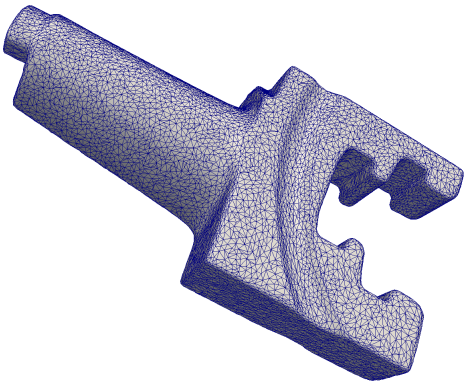
Le traitement out-of-core de données de grande taille n'est pas un sujet exclusivement lié à la génération de surfaces mais à une très grande variété de problèmes qui ont besoin de traiter des données trop volumineuses pour être chargées en mémoire. Cette problématique est apparue au fur et à mesure que la capacité de stockage et la vitesse de traitement des ordinateurs ont augmenté.

Les sections suivantes vont introduire plus en détail le type de données, le contexte de la thèse et les algorithmes qui nous intéressent.

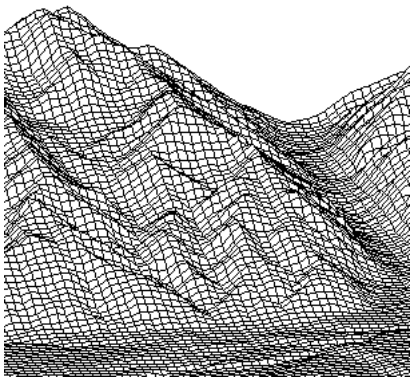
## 1.1 Surfaces

La génération de surfaces est un sujet actif dans le domaine de l'informatique graphique. Les surfaces sont des représentations très utiles en particulier pour la visualisation, l'imagerie médicale, la simulation numérique. Elles peuvent représenter la séparation entre l'intérieur et l'extérieur d'un objet ou la frontière entre deux milieux ayant des propriétés différentes. Elles sont aussi très utilisées parce qu'elles sont plus compactes que les représentations volumiques équivalentes. Une surface va souvent occuper moins d'espace en mémoire que le stockage de tous les voxels contenus à l'intérieur de la surface. La figure 1.1 montre quelques exemples de surfaces.

De plus, les surfaces peuvent contenir diverses propriétés de l'objet qu'elle représentent telles que des textures, par exemple, afin de représenter le matériau de l'objet. Dans les applications de rendu,



(a) Pièce mécanique.



(b) Terrain.

FIGURE 1.1 – Exemples de surfaces. (a) Surface représentant la frontière d'une pièce mécanique. (b) Modèle numérique de terrain.

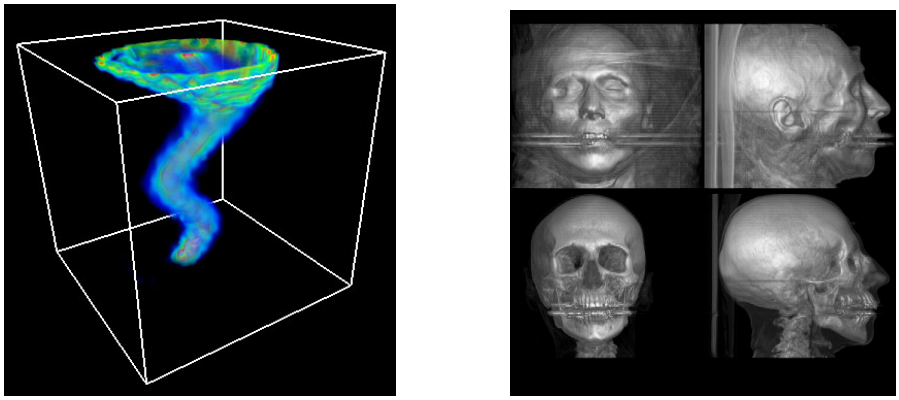
les normales aux facettes sont également utilisées. Les surfaces peuvent être construites à partir de multiples sources : nuages de points, images de profondeur, fonctions implicites ou volumes discrets. Dans cette thèse, nous allons nous intéresser à cette dernière catégorie de données parce qu'elles sont produites en grande quantité par de nombreux systèmes d'imagerie.

## 1.2 Données volumiques

Les données volumiques sont une classe particulière de données structurées sur une grille régulière. Chacune des cases définies par cette grille peut contenir plusieurs types d'information. Dans le cas d'une image, il peut s'agir d'une information de couleur ou de luminance dans chaque pixel. Dans le cas d'une image médicale en 3 dimensions, elle peut contenir des informations locales sur le paramètre physique imagé. Sur la figure 1.2 nous pouvons voir deux exemples de données volumiques. La figure (a) montre un volume contenant les mesures de vorticité d'un tourbillon en chaque point. La figure (b) montre la visualisation volumique obtenue à partir d'un scanner médical.

La notion de volume de grande taille dépend beaucoup des ressources disponibles pour le traitement. Dans les dernières décades, l'évolution de la capacité des ordinateurs a été principalement mesurée par trois indicateurs : 1) la vitesse du processeur, 2) la quantité de mémoire principale disponible et 3) la capacité de stockage sur le disque.

Cependant, en raison de la grande complexité technique pour garantir un accès rapide à la mémoire principale, cette dernière a peu augmenté en comparaison de l'espace disque. Cela a créé un déséquilibre entre la quantité de données qui peuvent être stockées et les ressources mémoire disponibles pour traiter ces données. Le développement de solutions pour surmonter cette limitation a été impulsée par un contexte technologique favorisant la génération de données volumiques de plus en plus grandes. Ce contexte est introduit dans la section suivante.



(a) (b)

FIGURE 1.2 – Exemples de données volumiques. (a) Volume contenant les mesures de vorticité [121] d'un tourbillon. (b) Visualisation volumique [67] d'un empilement d'images du crâne en imagerie médicale.

## 1.3 Contexte

Cette thèse s'inscrit dans un contexte où les données volumiques deviennent des plus en plus fréquentes dans une grande quantité de domaines. L'imagerie médicale est probablement l'une des disciplines où les données volumiques sont les plus répandues. Néanmoins, la physique, la chimie ou la biologie sont de plus en plus utilisatrices de ces données pour essayer de comprendre le comportement des systèmes.

Les données volumiques peuvent être obtenues à partir de multiples sources : dans le domaine médical, l'imagerie par résonance magnétique (IRM), ou l'angiographie CT (CTA) permettent d'obtenir des données de plus en plus détaillées. Les nouveaux microscopes électroniques peuvent aussi produire des images d'une très grande résolution.

(a) Nanozoomer (b) Exemple d'image de grande taille ($60K \times 50K$ pixels) issue d'un Nanozoomer.

FIGURE 1.3 – (a) Le système de scanner électronique NanoZoomer permet d'obtenir des centaines d'images avec une résolution de plusieurs milliers de pixels comme celle illustrée sur (b)

Cette thèse se place dans le contexte d'une rapide évolution des dispositifs de capture d'images qui permettent d'obtenir des volumes d'une précision toujours plus grande. Des dispositifs comme le Nanozoomer commercialisé par Hamamatsu (voir la figure 1.3) permet d'obtenir des empilements d'images avec une énorme résolution et une taille de plusieurs dizaines de milliers de pixels dans chaque direction conduisant à des volumes de centaines de Giga-octets (Go).

Pour des données aussi massives, il faut développer des outils d'exploration. Il existe déjà des algorithmes qui permettent d'extraire des surfaces à partir de volumes mais la majeure partie d'entre eux sont mal adaptés à des volumes qui dépassent la taille de la mémoire vive de l'ordinateur. Il faut donc concevoir de nouvelles méthodes qui puissent organiser et traiter les volumes à partir du disque et produire une surface adéquate non seulement pour la visualisation, comme cela a été l'objectif jusqu'à maintenant, mais aussi pour être utilisée comme entrée pour des algorithmes de remaillage ou de simulation.

## 1.4 Objectifs

Les objectifs principaux de ce travail sont les suivants :

- Étudier les différents types d'algorithmes existant pour l'extraction de surfaces à partir de données volumiques, essayer de comprendre leurs avantages et leurs limitations et en déterminer les causes éventuelles.
- Proposer une méthode d'extraction de surfaces adaptatives et multi-résolution bien adaptée aux données volumiques. Cela implique le développement de critères et d'outils pour la détection des caractéristiques topologiques et géométriques d'un objet volumique.
- Développer une stratégie pour traiter des données volumiques de grande taille avec une capacité de mémoire limitée. Signalons aussi que les surfaces produites peuvent également être massives même pour des données volumiques d'entrée qui ne posent pas de problème mémoire.

## 1.5 Organisation du mémoire

Ce mémoire est organisé en deux parties principales. Dans un premier temps nous allons nous concentrer sur les méthodes d'extraction de surfaces adaptatives. Nous allons introduire les travaux les plus pertinents de l'état de l'art et leur application aux données qui nous intéressent. Ensuite, nous décrirons notre méthode comme une alternative aux méthodes existantes, adaptée aux caractéristiques des données volumiques. Finalement, les résultats obtenus seront présentés, analysés et comparés avec d'autres approches de l'état de l'art.

Dans la deuxième partie, nous traiterons le problème de la génération adaptative de surfaces à partir de données de grande taille. Nous étudierons les approches d'extraction de surfaces "out-of-core" existantes et leur possible application par rapport aux besoins spécifiques de notre problématique, puis, nous présenterons notre solution comme une extension de la méthode utilisée dans la première partie. Enfin, nous évaluerons notre solution sur plusieurs jeux de données de grande taille.

# Première partie

# Génération de surfaces adaptatives

Dans cette partie, nous nous concentrerons sur quelques aspects et défis de l'extraction d'une surface à partir de la représentation volumique d'un objet. Nous allons commencer avec quelques définitions sur les objets volumiques, un analyse de caractéristiques topologiques, géométriques et de qualité des surfaces que nous voulons obtenir.

Ensuite, nous présenterons un état de l'art sur les principales méthodes de génération de surfaces à partir de données discrètes en mettant l'accent en particulier sur les méthodes de division spatiale. Après, nous introduirons notre approche pour l'obtention de surfaces adaptatives avec des garanties sur la topologie et la qualité générale des maillages obtenus. Dans la dernière section de ce chapitre, nous comparerons les résultats de l'application de notre algorithme avec des méthodes de l'état de l'art et analyserons ses implications sur un ensemble de jeux des données.

# Chapitre 2

# Définitions et topologie des données volumiques

## 2.1 Données volumiques

Soit un domaine $\Omega$ de $\Re^3$ contenant un ensemble d'échantillons $S \in \Omega$ distribués sur une grille régulière telle que chaque échantillon $s \in S$ se trouve à une distance euclidienne $d(s, y) = r$ de son plus proche voisin $y \in S$. Un *voxel* $V_S(s)$ est la région de Voronoï de $s$ telle que $V_S(s) = \{\forall x \in \Re^3 \mid d(x, s) \leq d(x, q), \forall q \in S \wedge q \neq s\}$. Maintenant, soit $F(x) : \Re^3 \rightarrow \Re$ une fonction scalaire sur le domaine $\Omega$ et définie sur chaque échantillon. Alors, la représentation volumique d'un objet $A$ par rapport à une valeur $\lambda$ est :

$$A_\lambda = \{\cup V_S(s) \mid F(s) > \lambda \, \forall s \in S\}$$

Un objet volumique est composé d'un ensemble de voxels qui sont connectés par une facette (connexité 6), une arête (connexité 18) ou un sommet (connexité 26) (voir figure 2.1). Afin qu'un objet volumique soit bien composé au sens de Latecki et Stelldinger [64, 106], il doit avoir une connexité 6. Cela veut dire que tous les voxels contenus dans $A_\lambda$ doivent partager au moins une facette.

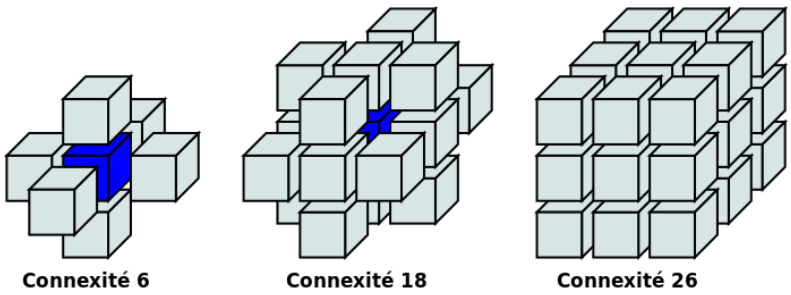



FIGURE 2.1 – Les trois types de connexité dans des objets volumiques. La connexité est illustrée par rapport au voxel central en bleu.

Néanmoins, dans des objets volumiques produits à partir d'un ensemble d'images ou à partir d'une

procédure de segmentation, il peut exister des voxels qui soient dans $A_\lambda$ mais qui ne partagent pas une facette avec d'autres voxels du volume. Ces voxels sont connectés à $A_\lambda$ par une arête ou un sommet. Ces configurations sont illustrées sur la figure 2.2.

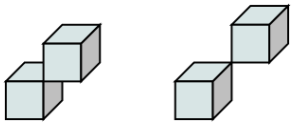


FIGURE 2.2 – Configurations critiques dans la représentation volumique d'un objet. À gauche, les voxels partagent une arête. À droite, les voxels partagent un sommet.

Ces deux configurations critiques dans un objet volumique peuvent favoriser l'apparition des facettes ambiguës. Une *facette ambiguë* est une facette qui contient deux sommets à l'intérieur de la surface séparés par deux sommets à l'extérieur. Elle introduit une ambiguïté dans la topologie de la surface à extraire parce que nous ne pouvons pas être sûrs de la topologie de la surface à l'intérieur de la cellule seulement en regardant les valeurs de la fonction $F$ aux sommets de la facette, comme illustré sur la figure 2.3.

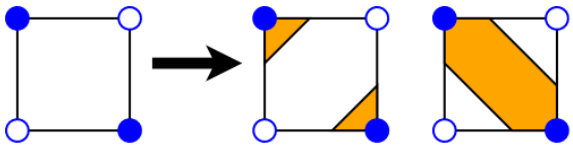


FIGURE 2.3 – Une facette ambiguë génère une ambiguïté dans la topologie de la surface à l'intérieur de la cellule. Les deux cas sont illustrés à droite où l'intérieur de la surface est en orange.

Latecki et Stelldinger ont démontré qu'il est possible d'extraire une surface 2-variété fermée en conservant la topologie de l'objet original seulement à partir des objets volumiques bien formés. Un objet volumique bien formé doit impérativement éviter les configurations critiques et minimiser l'existence de facettes ambiguës.

Néanmoins, si la contrainte topologique est levée, il est encore possible d'obtenir des surfaces 2-variété à partir de n'importe quel objet volumique si nous appliquons de manière cohérente le même critère pour les deux configurations critiques : soit séparer systématiquement deux composantes de l'objet connectées par une arête ou un sommet, soit toujours les connecter. Ces choix sont illustrés sur la figure 2.4.

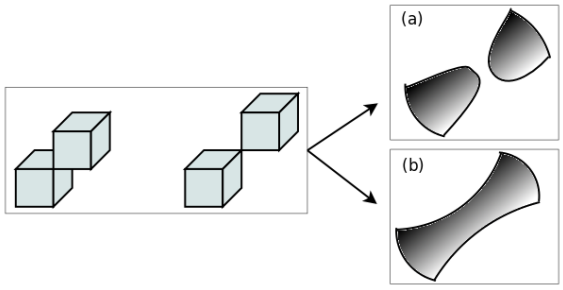



FIGURE 2.4 – Deux traitements sont possibles dans le cas des configurations critiques. En (a), les deux voxels sont systématiquement séparés en deux composantes. En (b), les voxels sont connectés pour former une seule composante "tunnel" entre les deux (en noir).

La stratégie adoptée peut changer les propriétés topologiques du maillage obtenu par rapport à l'objet original, notamment, le genre de la surface ou le nombre de composantes connexes. Afin d'éviter ce problème, nous pouvons aussi nous assurer que l'échantillonnage utilisé pour discrétiser un objet continu produise toujours un objet r-régulier à la manière de Stelldinger [106]. Pour définir ce qu'est un objet *r-régulier*, Stelldinger a utilisé la notion de *balle osculatrice*.

**Définition 1.** Soit $\partial A_\lambda$ la surface de l'objet volumique $A_\lambda$. Une balle ouverte $B_r^{A_\lambda}(x)$ est une *balle osculatrice* pour $x \in \partial A_\lambda$ si elle a un rayon $r$, elle est tangente à $\partial A_\lambda$ au point $x$ et elle est contenue entièrement dans $A_\lambda$ ou dans son complément $A_\lambda^c$. La figure 2.5 illustre cette définition.

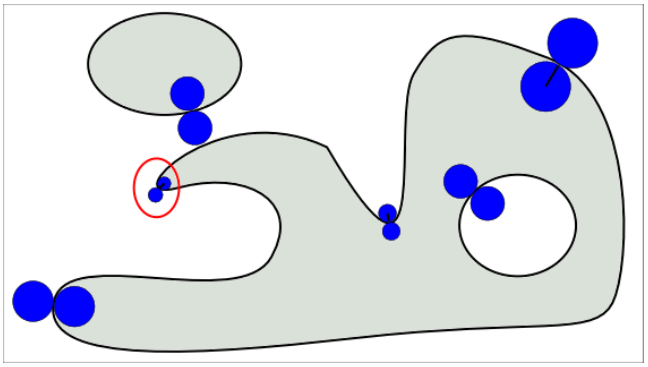


FIGURE 2.5 – Illustration 2D des disques osculateurs. L'intérieur de la figure est plus foncé que l'extérieur. Cet objet est r-régulier par rapport au $r$ défini par les disques osculateurs entourés en rouge. Même si plusieurs disques osculateurs peuvent être trouvés, la valeur de $r$ coïncide avec le rayon minimal des disques osculateurs sur les zones les plus fines du volume $V$.

Ensuite, il est possible de donner une définition des objets r-réguliers :

**Définition 2.** Un objet $A_\lambda \in \Re^3$ est *r-régulier* si et seulement si pour tout point $x \in \partial A_\lambda$, il existe deux *balles osculatrices* ouvertes de rayon $r$ tangentes à $\partial A_\lambda$ en $x$, dont l'une est à l'intérieur de $A_\lambda$ et l'autre dans son complément $A_\lambda^c$.

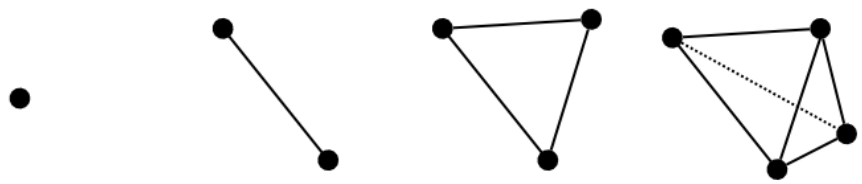
En conséquence, afin de bien pouvoir conserver la topologie d'un objet volumique, il faut s'assurer que l'objet volumique est un objet r-régulier. Une manière de le faire est d'identifier la valeur minimale de $r$ pour l'objet $A_\lambda$ et d'utiliser une r-grille afin de pouvoir détecter les composantes géométriques les plus fines de $A_\lambda$. Utiliser une r-grille revient à utiliser des voxels d'une taille supérieure à $r$ et qui pourront représenter la topologie de l'objet original avec un objet volumique de connexité 6. Dans l'exemple de la figure 2.5, même si une grande quantité de disques osculateurs peuvent être dessinés, l'objet est r-régulier seulement par rapport au rayon des disques osculateurs entourés en rouge sur la figure et qui est localisé sur les structures les plus fines de la figure.

En conclusion, nous considérons que la topologie d'un objet volumique ne pourra être conservée que si l'ensemble des voxels qui forment $A_\lambda$ est r-régulier par rapport à l'objet original.

## 2.2 La représentation de la surface d'un objet volumique

La représentation de la surface d'un objet volumique est composée d'un ensemble de polygones $S$ qui séparent l'intérieur de l'objet $V$ de son complément $V^c$. Dans cette thèse, $S$ sera défini comme

un complexe simplicial plongé dans $\Re^3$. Un complexe simplicial est constitué par un ensemble de simplexes définis comme l'enveloppe convexe de $n+1$ points. Ainsi, un 0-simplexe est un sommet, un 1-simplexe est une arête, un 2-simplexe est une facette et un 3-simplexe est un tétraèdre (voir figure 2.6).



(a) (b) (c) (d)

FIGURE 2.6 – Les quatre types de simplexes d'un 4-simplexe. (a) Un sommet (0-simplexe), (b) une arête (1-simplexe), (c) une facette (2-simplexe) et (d) un tétraèdre (3-simplexe).

Nous nous intéressons à l'extraction des surfaces conformes. Afin que $S$ soit une surface conforme, elle doit être 2-variété et fermée. Pour garantir cela, elle doit vérifier les propriétés suivantes :
– l'intersection de deux n-simplexes doit être un (n-1)-simplexe ;
– en tout point, la surface $S$ doit être équivalente topologiquement à un disque.
La première propriété garantit qu'il n'y aura pas d'auto-intersections ni de trous dans la surface. La deuxième propriété force la surface $S$ à séparer effectivement l'intérieur de l'objet de l'extérieur et qu'un (n-1)-simplexe ne soit pas partagé par plus de deux n-simplexes. Quelques cas sont illustrés sur la figure 2.7.

(a) (b) (c)

FIGURE 2.7 – (a) Un complexe simplicial non conforme parce qu'il y un trou dans la surface, (b) une intersection entre deux triangles qui n'est pas une arête et (c) un complexe simplicial conforme.

En plus d'obtenir une surface conforme qui isole l'intérieur de l'objet de son complément, nous souhaitons en extraire une surface lisse qui approxime bien la géométrie du volume. Une autre propriété souhaitée est que la surface obtenue soit adaptative. Une telle surface adapte le nombre de facettes utilisées pour approximer une région de la surface par rapport à sa courbure discrète locale. En conséquence, les zones de faible courbure seront représentées avec une moindre quantité de polygones que les zones de courbure élevée. La forme et la topologie des régions va fortement dépendre de la nature de la méthode d'extraction de surface choisie.

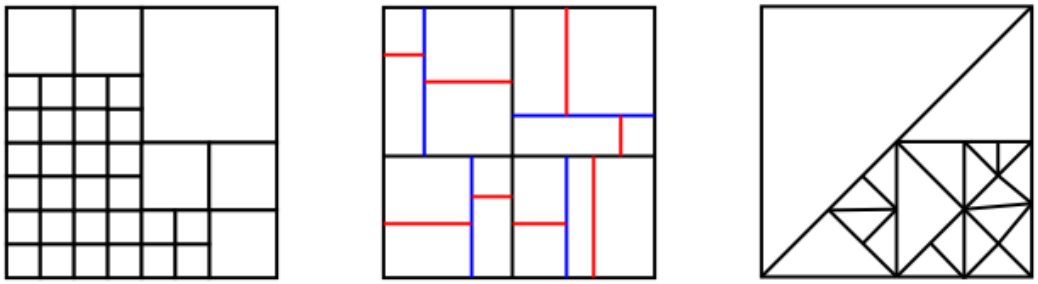
## 2.3 Méthodes de division spatiale

Dans cette thèse, nous allons nous concentrer spécifiquement sur les méthodes de division spatiale. Ces méthodes utilisent des structures de données auxiliaires afin de diviser le volume. La plus

grande partie de ces structures utilisent des cellules de géométrie simple du type hexaédrique ou tétraédrique. Cela permet de simplifier le calcul de l'approximation de la surface et le raccordement des morceaux de surface générés par des cellules adjacentes. L'estimation locale de la topologie et de la géométrie de l'objet consiste à produire un complexe simplicial qui approxime la surface $\partial V$. Chaque structure de données a ses propres avantages et inconvénients qui vont déterminer les caractéristiques de la surface produite et les limites dans la qualité de l'approximation qui peut être obtenue. Les deux principaux types de divisions spatiales sont :

- les divisions où toutes les cellules utilisées sont au même niveau de résolution comme c'est le cas pour une grille régulière ou pour un diagramme de Voronoï ;
- les structures de subdivision hiérarchiques qui permettent de garder une relation hiérarchique entre les cellules en plus d'une relation de voisinage géométrique.

Afin de traiter des données volumiques, nous considérons que les structures plus adaptées sont celles de nature hiérarchique. Ces structures vont nous permettre d'adapter plus facilement la résolution de l'approximation par rapport à la topologie ou à la géométrie du volume. Parmi ces types de structures on trouve les Octrees, les Kd-trees et les subdivisions simpliciales.



(a) (b) (c)

FIGURE 2.8 – Trois types de divisions spatiales en 2D utilisées pour l'extraction des surfaces. (a) Quadtree, (b) Kd-tree, (c) Bisection simpliciale régulière.

Les octrees sont des structures hiérarchiques tridimensionnelles construites à partir des cellules cubiques et alignées sur les axes du repère. Chaque cellule d'un octree est régulièrement divisée en huit cellules en utilisant trois plans alignés sur chacun des trois axes (voir figure 2.8a). Un Kd-tree est une structure hiérarchique inspirée des arbres binaires et composée de cellules hexaédriques. A la différence des octrees, les Kd-trees ne divisent pas chaque cellule en huit mais en deux cellules en utilisant un plan aligné sur un des axes du repère comme l'illustre la figure 2.8b. Finalement, une subdivision simpliciale utilise des cellules tétraédriques et sa structure hiérarchique est construite sur celle d'un arbre binaire (voir figure 2.8c). Au niveau géométrique, les cellules sont divisées en utilisant une des méthodes de division simpliciale telle que la bissection par l'arête la plus longue qui sera détaillée dans le chapitre suivant. Les trois structures de données mentionnées peuvent faire des divisions adaptatives comme illustré sur la figure 2.8.

Dans la section suivante, nous allons présenter un état de l'art sur les méthodes d'extraction de surfaces basées sur des divisions spatiales dans le contexte des objets volumiques. Nous exposerons leurs principes et analyserons leurs avantages et leurs inconvénients.

# Chapitre 3

# État de l'art sur les méthodes d'extraction de surface

Il existe une myriade d'algorithmes pour l'extraction d'une surface à partir de données volumiques. La plus grande partie de ces algorithmes se basent sur une division spatiale des données afin de réduire le problème à la construction des triangulations locales de topologie simple qui peuvent ultérieurement être connectées pour former la surface. Concernant ce type d'algorithme, nous avons identifié deux familles de méthodes. Les méthodes primales utilisent des structures de division spatiale de géométrie fixe directement sur les données. Elles se basent généralement sur des modifications de l'algorithme très connu "Marching Cubes". Les méthodes duales, comme leur nom l'indique, utilisent des représentations duales de la surface ou des structures de division spatiale de géométrie variable adaptée aux données. Ces méthodes capturent mieux les arêtes vives et sont plus pertinentes pour la génération de maillages adaptatifs. Ces deux types de méthodes seront mieux détaillés dans les deux sections suivantes.

## 3.1 Méthodes primales

Les méthodes *primales* sont des méthodes qui réalisent une division du domaine en cellules de géométrie connue et simple. Ces méthodes localisent les sommets du maillage, ou la plus grande partie des sommets, sur les arêtes des cellules afin de simplifier la triangulation des morceaux de la surface. Ils peuvent être classés par rapport à la géométrie de la cellule utilisée. Les deux types de cellules les plus utilisées sont les hexaèdres (dont la topologie est celle d'un cube) et les tétraèdres (donc la topologie est celle d'un 3-simplexe).

### 3.1.1 Algorithmes à subdivision hexaédrique

L'algorithme des "Marching cubes" (MC) [72] permet d'extraire une surface à partir d'un champ scalaire $\{F(x) \mid \Re^3 \rightarrow \Re\}$ qui délimite un objet volumique $A_\lambda$ comme il a été défini dans la section précédente. Il utilise une division régulière du domaine en cellules cubiques telle que chaque sommet de la cellule est localisé au centre d'un voxel. Ainsi, chaque sommet peut être à l'intérieur ou à

l'extérieur de $A$. Cette division produit des cellules qui peuvent avoir au plus 256 configurations différentes de valeurs des sommets de chaque cellule. Ces 256 configurations peuvent se réduire par rotation et symétrie à 15 cas (voir figure 3.1) afin de créer une table avec les triangulations qui reproduisent la forme de la surface à l'intérieur de chaque cellule.

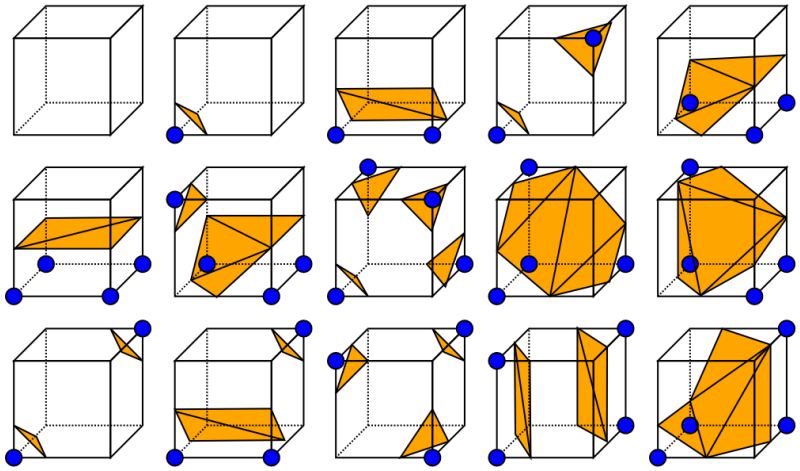


FIGURE 3.1 – Les 15 configurations possibles de l'algorithme des "Marching Cubes" après rotation et symétrie. Les sommets en bleu sont à l'intérieur de la surface $\partial A_\lambda$ et la triangulation générée pour chaque cas est représentée en jaune.

Dans l'algorithme MC, les sommets de la surface sont localisés sur les arêtes des cellules et leur position est estimée par interpolation linéaire sur chaque arête. Cette caractéristique limite grandement la liberté de localisation des sommets de la surface et ne permet pas toujours de bien approcher d'éventuelles arêtes vives à l'intérieur d'une cellule.

De plus, comme MC utilise une grille régulière, il ne peut pas générer de maillages adaptatifs et des effets de crénelage peuvent apparaître résultant de la division utilisée (voir la figure 3.2). Finalement, si le volume à reconstruire contient des configurations critiques, MC peut générer des trous et d'autres artefacts topologiques dans la reconstruction.

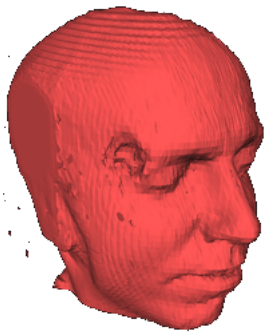


FIGURE 3.2 – Exemple de l'application de l'algorithme des "Marching Cubes" sur une tête humaine.

Plusieurs méthodes ont été proposées afin de surmonter les limitations de MC. Nielson et Hamann [82] ont développé un outil basé sur le comportement asymptotique de l'interpolateur tri-linéaire pour détecter la topologie de la surface sur les facettes ambiguës. Ce *décideur asymptotique* permet d'éviter les trous générés par MC à cause d'une mauvaise sélection de connectivité sur deux cellules adjacentes comme illustré sur la figure 3.3.

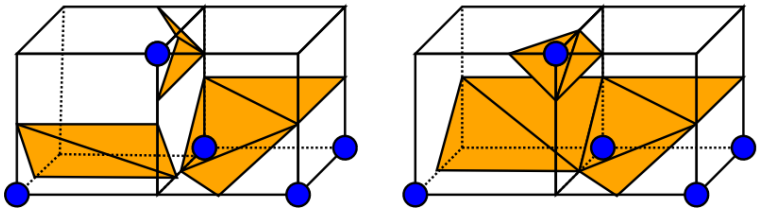


FIGURE 3.3 – Deux cellules voisines qui partagent une facette ambiguë. Dans la cellule de gauche, MC connecte les composantes et produit un tunnel à l'intérieur de la surface alors que dans la cellule de droite il sépare les composantes. En conséquence, il se produit une cassure dans l'approximation de la surface. À droite, les deux cellules choisissent la même topologie et séparent les deux composantes pour construire un tunnel extérieur à la surface en gardant la continuité de la surface.

Néanmoins, cet outil peut seulement être utilisé avec des cellules de même taille et ne peut pas être facilement étendu à des grilles adaptatives. Dans le but d'extraire des maillages adaptatifs avec MC, il faut résoudre les problèmes de continuité de l'interpolateur tri-linéaire utilisé par MC sur des régions qui présentent des changements de résolution de la grille. Ce problème est visible sur la figure 3.4 (à gauche) où quatre cellules sont adjacentes à une cellule plus grande.

Dans "Adaptive Marching Cubes" [101], Shu *et al.* ont présenté un algorithme de fermeture des trous ("patch cracking") pour détecter et réparer ces problèmes. Cette solution consiste à regarder les intersections de la surface avec les cellules plus petites afin de détecter si le contour de la surface (une polyligne) est une courbe simple et, dans ce cas, de la remplacer par une ligne qui relie les extrémités de celle-ci (voir la ligne rouge sur la figure 3.4 à droite). Dans le cas où le contour n'est pas simple, Shu *et al.* proposent de diviser la cellule voisine plus grande.

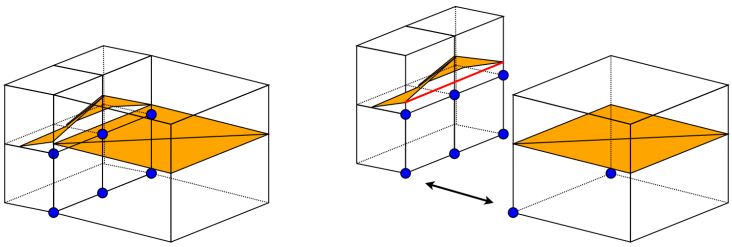


FIGURE 3.4 – A gauche, origine du problème de "patch cracking" dans l'algorithme de MC sur une grille adaptative. À droite, la polyligne simple détectée dans les cellules plus petites est remplacée par une ligne connectant les deux extrémités de la polyligne *[101]*.

L'algorithme de Shu a l'inconvénient de trop simplifier la surface dans les zones de changement de résolution. Westermann *et al.* [126] ont proposé une autre solution pour produire des maillages adaptatifs en remplaçant la grille par un octree. Afin d'éviter des cassures dans la surface, ils insèrent un nouveau sommet dans le centre de masse du morceau de surface à l'intérieur de la cellule grande (3.5 à gauche) et construisent un éventail de triangles entre ce sommet et les points d'intersection de la surface avec les arêtes des cellules plus petites (3.5 à droite).

Même si ces solutions produisent une surface adaptative, elles ont tendance à simplifier la surface sur les zones de changement de résolution, limitent la capacité d'adaptation du maillage par rapport à la courbure de la surface et ne résolvent pas les problèmes de MC par rapport aux arêtes vives et aux artefacts topologiques.

Kobbelt *et al.* [62] ont pré-calculé un champ de distance signé sur le volume pour améliorer la

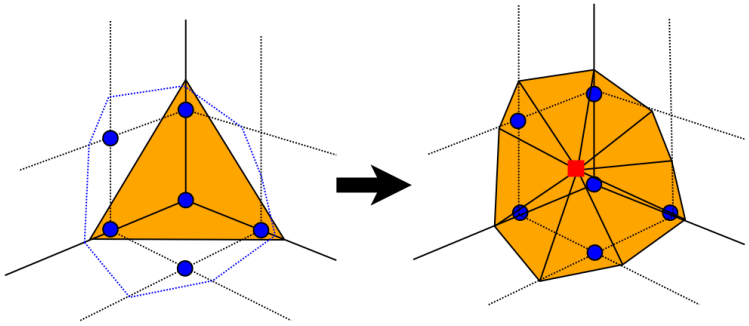


FIGURE 3.5 – Vue d'une cellule grande adjacente à des cellules plus petites. Le morceau de surface généré par MC (à gauche) ne correspond pas aux points d'intersection de la surface avec les cellules voisines (lignes en pointillés). Ces cassures sont résolues par la création d'un sommet au centre de gravité du morceau de surface (carré) et la construction d'un éventail de triangles entre ce sommet et les points d'intersection de la surface avec les arêtes des cellules plus petites (à droite).

détection des points d'intersection. Cette méthode utilise un octree et introduit un sommet dual à l'intérieur des cellules pour mieux approximer les arêtes vives comme le montre la figure 3.6. Malheureusement, comme toutes les cellules qui traversent la surface sont au même niveau, cet algorithme ne génère pas de maillages adaptatifs.

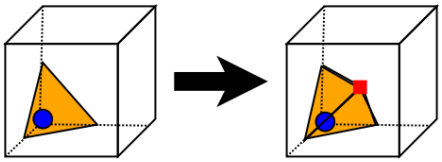
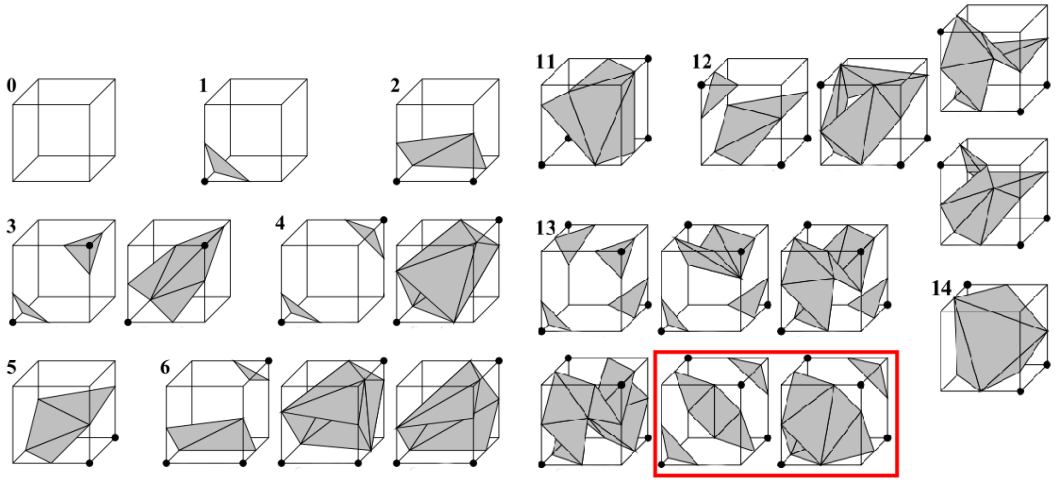


FIGURE 3.6 – Si une arête vive est détectée, un sommet dual (carré) est ajouté au centre de la cellule et le patch créé par MC est remplacé par un éventail de triangles.

Même si les méthodes précédentes résolvent quelques problèmes relatifs à la géométrie des surfaces générées, elles n'examinent pas les problèmes topologiques de MC. Nielson [79] a présenté un critère pour éliminer toutes les ambiguïtés topologiques à l'intérieur des cellules et a fourni une look-up table étendue avec les triangulations possibles en se basant entièrement sur les propriétés analytiques de l'interpolateur tri-linéaire. Carr [17] a fourni une explication plus intuitive de la look-up table proposée par Nielson en se basant sur l'algorithme de cubes divisés ("Dividing Cubes "), un algorithme utilisant une approche "diviser pour régner" et des arguments topologiques.

Afin d'assurer que la topologie de la surface générée par MC est la même que l'objet de départ, il faut fournir des outils analytiques et géométriques pour estimer la forme de la surface à l'intérieur de chaque cube. Chernyaev [22] a aussi proposé des techniques analytiques basées sur le comportement de l'interpolateur trilinéaire et une look-up table étendue avec 33 cas. Ces 33 cas sont des subdivisions des cas contenus dans la look-up table de MC (voir figure 3.7) et qui reproduisent toutes les topologies possibles de la surface à l'intérieur d'un cube. Plus récemment, [36] Custodio *et al.* ont démontré que "Marching Cubes 33" pouvait encore générer des surfaces qui ne sont pas correctes topologiquement à cause d'une ambiguïté non résolue dans le cas 13.5 de la table proposée par Chernyaev. Malheureusement, ces deux algorithmes n'offrent pas une implémentation adaptative de MC.

Une autre approche pour générer des surfaces adaptatives avec des garanties topologiques à été proposée par Varadhan *et al.* [112]. Cette approche établit deux critères sur la topologie de la

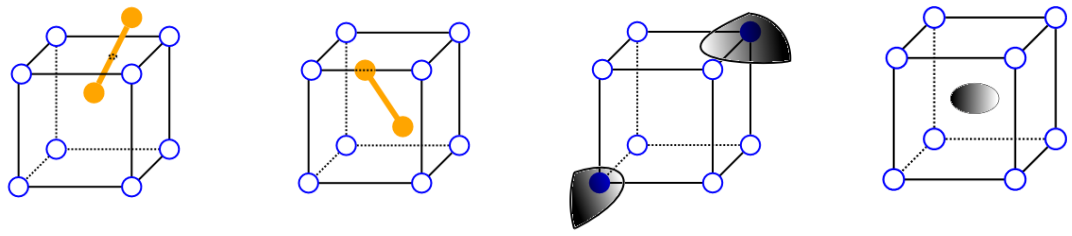



(a) Extension des sept premiers cas de MC.

(b) Derniers cas de "Marching Cubes 33" avec les cas 13.5 entourés d'un cadre.

FIGURE 3.7 – (a) Subdivisions des sept premiers cas de la look-up table de MC proposées par Chernyaev afin de reproduire toutes les possibles topologies à l'intérieur d'un cube. (b) Derniers cas de MC 33 avec les deux cas 13.5 encadrés. L'ambiguïté topologique dans ces deux cas a été résolue par Custodio *et al.*

surface à l'intérieur d'une cellule afin de déterminer si sa topologie peut être bien représentée par l'un des cas de MC. Le premier critère permet de déterminer si la topologie de la surface ne va pas créer d'arête non-variété et d'assurer qu'il n'y ait pas d'ambiguïtés topologiques. Le second détecte l'existence de composantes isolées à l'intérieur des cellules qui n'intersectent pas leurs facettes. Quelques exemples des topologies prises en compte sont montrés sur la figure 3.8.



(a) (b) (c) (d)

FIGURE 3.8 – Cas testés par l'algorithme de Varadhan *et al.* (a) La surface traverse une facette de la cellule sans toucher aucun des sommets de la facette. (b) La surface traverse plus d'une fois une arête de la cellule. (c) La topologie de la surface dans la cellule n'est pas unique topologiquement. (d) Un morceau de l'objet ne touche pas les facettes ni les arêtes de la cellule.

Cet algorithme permet d'extraire des surfaces adaptatives en appliquant ces deux critères à des cellules de différentes tailles pour s'assurer qu'il n'existe pas de configuration complexe dans les zones de changement de résolution. Cependant, il n'améliore pas l'approximation de la surface ni la qualité des triangles générés et ne tient pas compte des arêtes vives de la surface.

Plusieurs algorithmes présentent des améliorations concernant la qualité des surfaces produites par MC. Dietrich *et al.* [38] ajoutent des sommets et changent la triangulation des polygones de la look-up table de MC pour améliorer la qualité des triangles. Shekhar *et al.* [97] utilisent une

stratégie de combinaison des patches sur plusieurs cellules et une stratégie de fermeture de trous (patch cracking) sur les zones de changement de résolution.

Plus récemment, Kazhdan *et al.* [61] ont présenté un algorithme pour extraire des surfaces adaptatives sur n'importe quel octree. Cette solution utilise la look-up table de MC et introduit le concept de *Edge-tree* pour fermer les polylignes qui génèrent des cassures entre les cellules grossières et les cellules plus fines. Sur la figure 3.9, il est possible de visualiser le mécanisme utilisé par Kazhdan pour fermer les iso-polylignes. Dans des régions de changement de résolution, la surface peut traverser plusieurs fois l'arête d'une cellule plus grande. Kazhdan *et al.* proposent de construire des arbres binaires qui reflètent les intersections des arêtes et des sous-arêtes avec la surface.

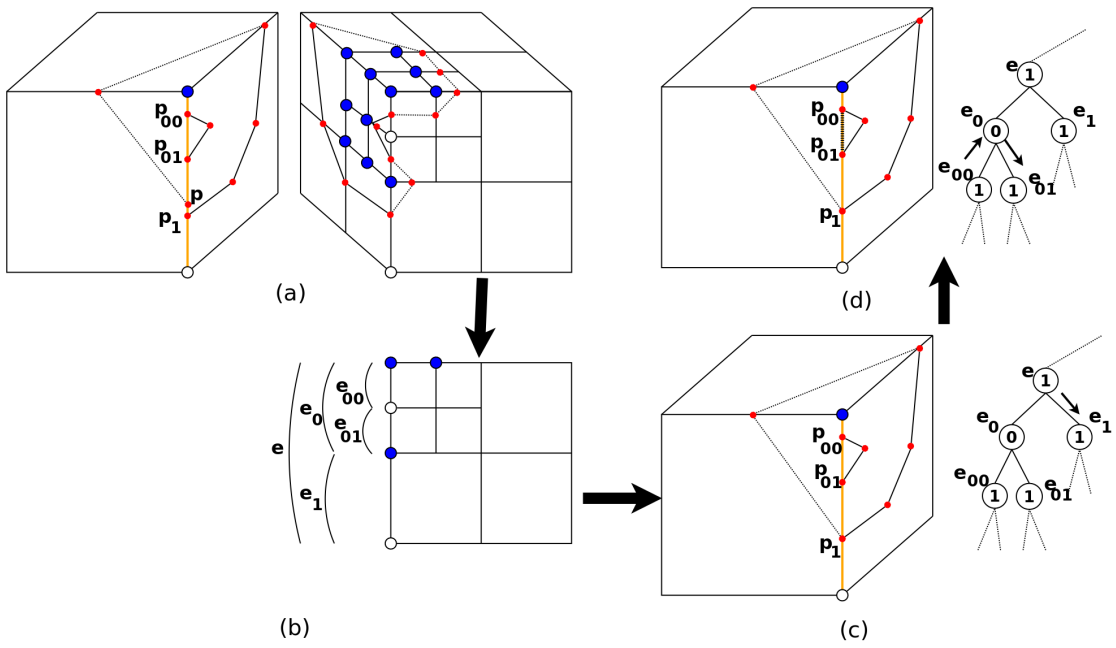



FIGURE 3.9 – (a) Dans les régions de contact entre des cellules de différentes tailles, MC génère des discontinuités sur la surface. (b) Sur les arêtes traversées plus d'une fois par l'iso-surface, Kazhdan *et al.* construisent des edge-trees en tenant compte des valeurs des extrémités sur les arêtes et sous-arêtes. En traversant ces arbres, il est possible de détecter les points d'intersection qui doivent être déplacés (c) ou connectés sur l'arête pour fermer l'iso-surface (d).

Par exemple, par rapport aux cellules de la figure 3.9a, les deux cellules partagent une arête qui est plus divisée à droite qu'à gauche. Dans ce cas, nous allons nommer les arêtes et les sous-arêtes de l'arête partagée comme il est indiqué sur la figure 3.9b. Ainsi, l'arête complète $e$ devient le nœud racine de l'edge-tree et comme elle affiche un changement de valeur aux deux extrémités, la valeur stockée dans ce nœud est 1. Pour la sous-arête $e_0$, les deux extrémités ont la même valeur, alors la valeur sur son nœud est 0. Par contre, la sous-arête $e_1$ aura une valeur de 1 parce que ces deux extrémités présentent une valeur différente (elle est traversée par la surface). En continuant de la même manière avec les sous-arêtes $e_{00}$ et $e_{01}$, nous arrivons à construire l'arbre de la figure 3.9c. Une fois cet arbre construit, son parcours permet de détecter qu'il y a plusieurs intersections de la surface sur l'arête. Dans l'arbre de la figure 3.9c, l'existence des nœuds avec des valeurs "1" dans la hiérarchie descendante d'un nœud à 0, implique qu'il y a plusieurs intersections et que la localisation du sommet $p$ sur la grande arête $e$ doit être déplacée vers la position du sommet $p_1$, de la sous-arête $e_1$ pour bien fermer la surface. Aussi, si un parcours entre deux nœuds qui intersectent la surface (valeur à 1) traverse un nœud à 0, cela veut dire qu'il existe une double

intersection de la surface entre ces deux sommets et qu'ils doivent être connectés comme c'est le cas des sommets $p_{00}$ et $p_{01}$ sur la figure 3.9d.

Pourtant, même si cette solution génère des surfaces adaptatives, sa capacité d'adaptation à la géométrie de l'objet d'origine reste très limitée parce que les sommets peuvent seulement être placés sur les arêtes de l'octree.

Finalement, même si MC est très efficace sur des surfaces lisses, quand il s'agit d'extraire des surfaces à partir d'objets discrets, il produit des effets de crénelage et des oscillations autour de la surface réelle. Manson et Smith [74] ont proposé une méthode pour remplacer l'interpolateur linéaire de MC par une nouvelle fonction basée sur l'analyse de l'intersection d'un objet discret avec les arêtes de deux cellules voisines qui s'adapte bien aux fonctions indicatrices discrètes. Cette approche est simple et génère des surfaces plus lisses en éliminant la plus grande partie du crénelage (voir figure 3.10). Malheureusement, cette algorithme ne marche que sur des grilles régulières et, en conséquence, ne permet pas d'extraire des surfaces adaptatives.

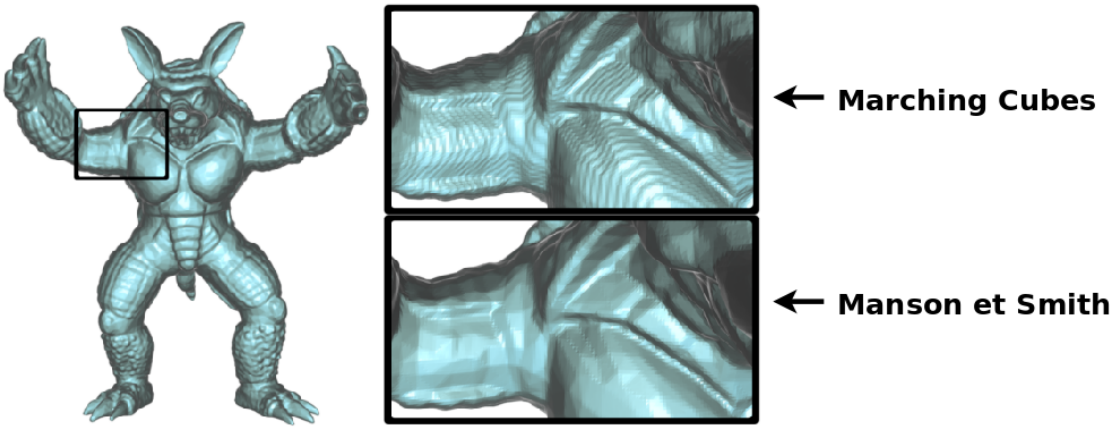


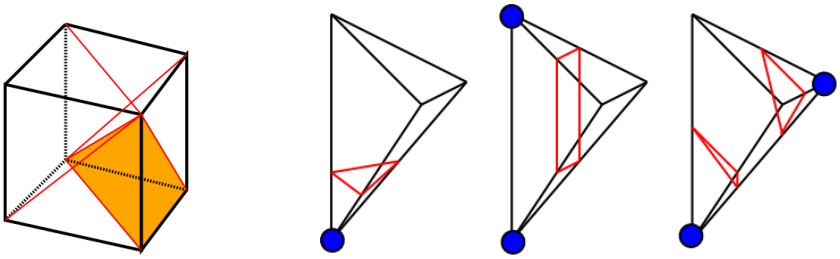

FIGURE 3.10 – Application de l'algorithme de localisation des sommets de Manson et Smith sur un objet "Armadillo" dans une grille de $512^3$ voxels et comparaison avec la surface produite par l'algorithme MC [74].

### 3.1.2 Algorithmes à subdivision tétraèdrique

L'algorithme "Marching Tetrahedra" (MT) [100] a été conçu comme une manière de simplifier la look-up table utilisée pour extraire les morceaux de surface. Cet algorithme fait une division simpliciale des cellules cubiques de MC avec afin d'obtenir une grille tétraédrique. Ensuite, cette grille est parcourue pour extraire la surface. Cela permet de réduire la taille de la LUT à trois cas et d'éliminer les cas des configurations ambiguës comme le montre la figure 3.11.

Comme il n'existe pas une unique subdivision tétraédrique d'un cube, plusieurs subdivisions ont été proposées [109, 16, 18]. La qualité de la surface obtenue va beaucoup dépendre de l'option choisie mais, en général, l'utilisation de MT augmente la quantité de triangles d'un rapport 2.5 environ et dégrade la qualité des triangles par rapport à MC.

Une manière d'obtenir des surfaces adaptatives avec une subdivision tétraédrique consiste à construire une hiérarchie de diamants [123] qui garantit que la subdivision reste conforme et que l'approximation de la surface est continue. Les diamants sont des éventails de tétraèdres qui partagent



(a) Division d'une cellule cubique en tétraèdres.

(b) Les trois configurations de la look-up table de Marching Tetrahedra.

FIGURE 3.11 – La surface obtenue par MT dépend de la subdivision tétraédrique choisie. La figure (a) montre une possible subdivision d'un cube en tétraèdres autour de sa diagonale, un tétraèdre est affiché en orange. (b) LUT pour MT, les sommets à l'intérieur de la surface sont en bleu et la surface est construite en connectant des sommets sur les arêtes du tétraèdre.

une arête appelée épine et qui sont divisés simultanément en utilisant la bissection de l'arête la plus longue (Longest Edge Bisection) (LEB). La LEB choisit l'arête la plus longue d'un tétraèdre, crée un nouveau sommet au milieu de cette arête et divise le tétraèdre en connectant le nouveau point et les deux sommets de l'arête opposée comme le montre la figure 3.12.

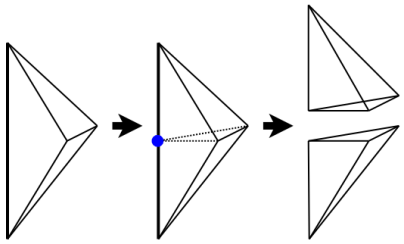


FIGURE 3.12 – Le mécanisme de bissection de l'arête la plus longue (LEB) divise un tétraèdre en deux en ajoutant un nouveau sommet (en bleu) sur l'arête la plus longue (en gras) et en le connectant aux sommets de l'arête opposée.

Weiss *et al.* [124, 125, 122] ont identifié les trois types de diamants 3D illustrés sur la figure 3.13. Chaque type de diamant a une épine orientée sur la diagonale d'un cube, sur la diagonale d'une facette ou sur l'arête du cube. Si le point de bissection est localisé au centre de l'épine, ces trois types de tétraèdre auront toujours les même angles dièdres, ce qui permet de prédire la qualité des triangles générés. Cette méthode permet de construire des hiérarchies adaptatives de diamants qui peuvent être combinées avec MT afin d'obtenir des surfaces 2-variété adaptatives.

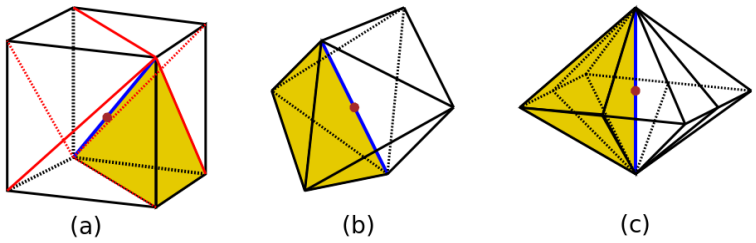



FIGURE 3.13 – Les trois types de diamants en 3D. Les épines des trois diamants contiennent un sommet au centre qui sera ajouté lors de la subdivision. (a) 0-diamant avec son épine alignée sur la diagonale d'un cube. (b) 1-diamant avec son épine alignée sur la diagonale d'une facette. (c) 2-diamant avec son épine alignée sur l'arête d'un cube.

Malheureusement, cette méthode ne résout pas les problèmes de qualité dans la triangulation produite par MT. De plus, comme il utilise un codage implicite qui localise toujours le nouveau sommet au milieu de l'arête la plus longue, il ne peut pas s'adapter facilement aux arêtes vives ou profiter de la géométrie de l'objet original pour réduire le nombre de triangles générés ou améliorer leur qualité.

Les méthodes présentées jusqu'à maintenant utilisent des grilles à géométrie fixée dans le sens que la géométrie de la cellule utilisée est prédéfinie et orientée par rapport aux axes. En conséquence, ses capacités d'adaptation à la surface pour produire une bonne subdivision de la structure hiérarchique sont réduites. De plus, les sommets sont majoritairement localisés sur les arêtes de cellules ce qui donne une plus grande importance à la subdivision et limite la liberté pour approximer la surface. Pour conclure, même si la plus grande partie des limitations de l'algorithme de MC ont été levées par des nombreuses extensions, la quantité de triangles et la qualité de la triangulation produite restent encore des problèmes ouverts.

## 3.2 Méthodes duales

Les méthodes duales s'intéressent à l'un des principaux problèmes de méthodes basées sur l'algorithme des "Marching Cubes". Comme les extensions de MC placent les sommets de la surface sur les arêtes des cellules, ils ont une capacité limitée pour reproduire les caractéristiques de la surface qui ne sont pas alignées avec les axes principaux de la grille. Les méthodes duales résolvent ce problème en localisant les sommets de la surface à l'intérieur des cellules et en produisant un maillage dual de celui généré par MC. Les surfaces ainsi construites seront appelées *surfaces duales* et les sommets, *sommets duaux*. Ces méthodes peuvent aussi être divisées en deux familles. La première comprend les méthodes qui génèrent des sommets à l'intérieur des cellules de grilles primales tels des octrees, Kd-trees, etc. La seconde regroupe des méthodes qui localisent les sommets sur les arêtes des cellules de grilles duales.

### 3.2.1 Méthodes duales sur de grilles primales

L'une des premières méthodes proposées pour construire des surfaces duales a été l'algorithme "Dual Contouring" (DC) [59, 92]. DC extrait une surface adaptative à partir d'un champ scalaire en utilisant un octree adaptatif. Chaque cellule de l'octree stocke des informations sur les points d'intersection entre la surface et les arêtes de la cellule, ainsi que les normales de la surface en ces points d'intersection (données de Hermite). La dualité et les différences fondamentales entre DC et MC sont illustrées sur la figure 3.14.

Les données de Hermite permettent d'estimer la courbure de la surface dans la cellule. Une bonne estimation de la courbure est calculée avec l'aide d'une fonction d'erreur quadratique (QEF) [48, 47]. DC définit une QEF par l'équation suivante :

$$E[x] = \sum_i (\vec{n}_i \cdot (x - p_i))^2 \tag{3.2.1}$$

Où $p_i$ représente un point d'intersection et $\vec{n_i}$ la normale à la surface en ce point. La minimisation de cette erreur $E[x]$ permet d'obtenir une estimation précise de la qualité de l'approximation de la

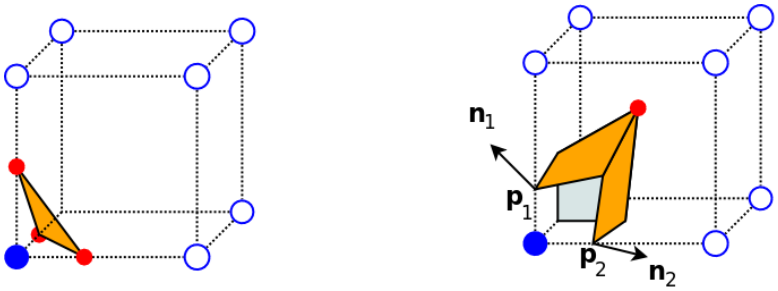
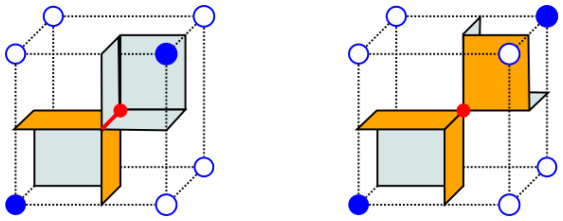



FIGURE 3.14 – Comparaison entre MC et DC. A gauche, dans MC, trois sommets sont créés sur les arêtes qui traversent la surface et un patch de surface est construit. À droite, DC crée un seul sommet dual à l'intérieur de la cellule et un patch est construit pour chaque arête traversant la surface.

surface à l'intérieur de la cellule. La minimisation de l'erreur $E[x]$ ci-dessus est géométriquement équivalente à chercher le point $x$ qui se trouve à une distance minimale des plans définis par l'ensemble des normales et des points d'intersection. Ce point est alors utilisé comme le sommet dual de la cellule comme illustré sur la figure 3.14 à droite. Certains aspects techniques peuvent affecter la localisation du sommet dual. Dans le cas où les plans sont presque parallèles, le point $x$ n'est pas nécessairement unique et, dans d'autres cas, le sommet peut même se trouver à l'extérieur de la cellule. Afin de résoudre ces deux problèmes, la minimisation de l'équation 3.2.1 doit se faire par rapport à la norme au centre de la cellule.

DC permet d'extraire des surfaces adaptatives qui conservent bien les arêtes vives mais comme il ne peut pas contenir plus d'un sommet dual par cellule, il peut générer des sommets et des arêtes non-variété dans les cellules ambiguës comme le montre la figure 3.15. De plus, les QEF sont très dépendantes de l'estimation des normales et ne garantissent pas que le sommet dual soit bien localisé sur la surface.



(a) Arête non-variété. (b) Vertex non-variété.

FIGURE 3.15 – La condition d'avoir seulement un sommet dual (en rouge) dans DC peut faire apparaître des configurations non-variété dans des cellules à topologie ambiguë.

L'un des principaux problèmes de DC est qu'il ne permet pas d'avoir plus d'un sommet dual par cellule et ne gère pas les arêtes complexes (présentant plus d'une intersection avec la surface $\partial V$). Il est donc nécessaire de diviser les cellules jusqu'à obtenir des cellules simples. Dans le but de mieux traiter les arêtes complexes et réduire le nombre de cellules nécessaires pour la reconstruction, Varadhan *et al. [111]* ont proposé un algorithme de "Dual Contouring" étendu (EDC) qui utilise un champ de distance orienté pour bien détecter les points d'intersection de la surface avec les arêtes de la cellule. Ce champ de distance est défini comme une fonction en chaque point $(x, y, z) \in \Re^3$ :

$$f(x, y, z) = dist(\mid x, y, z \mid, V)$$

Pour un objet volumique $V$, on aura $f(x, y, z) > 0$ si $(x, y, z) \in V$ et $f(x, y, z) < 0$ si $(x, y, z) \notin V$.

Cette méthode est capable de générer plusieurs sommets duaux à l'intérieur d'une cellule dans le cas d'une surface intersectant une arête plusieurs fois. Les principes de cette solution sont illustrés en 2D sur la figure 3.16.

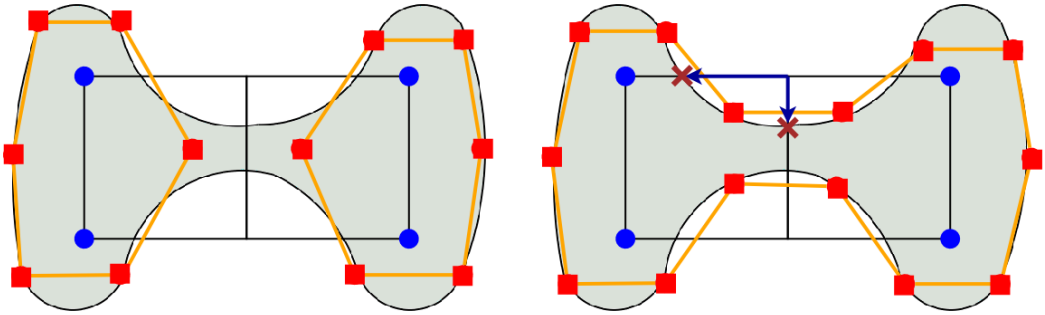


FIGURE 3.16 – Comparaison entre les surfaces générées par DC et par EDC dans le cas d'une arête complexe. A gauche, DC génère seulement un sommet dual par cellule et, en conséquence, il sépare l'objet en deux composantes. À droite, EDC utilise un champ de distance orienté discret (flèches) pour reconnaître les points d'intersection (croix) de l'arête complexe et crée un sommet dual (carrés) pour chaque composante. Les sommets duaux sont connectés par des arêtes afin de former la surface.

Le principal avantage de cette approche est qu'elle permet de générer des surfaces adaptatives et de mieux représenter les arêtes vives. Ainsi, la possibilité de contenir plus d'un sommet dual par cellule permet de reproduire les zones fines de l'objet. Néanmoins, cette méthode ne considère pas le cas général d'une arête traversant la surface plus de deux fois parce que cela introduit des ambiguïtés topologiques qui peuvent ne pas être nécessairement résolues en ajoutant plus de sommets. D'autre part, comme elle explore seulement les arêtes de la cellule, elle ne détecte pas si il existe des morceaux de la surface isolés à l'intérieur de la cellule ou si la surface coupe une facette de la cellule sans toucher ses arêtes.

Une alternative pour reproduire des géométries plus complexes est la méthode de Cuberille proposée par Che *et al.* [21]. Cette méthode place un sommet dual au centre de la cellule et génère des polygones à travers les arêtes traversant la surface. Le principal avantage de cette méthode est qu'ils s'adapte bien à la génération des surfaces à partir de données volumiques. Ses inconvénients sont nombreux 1) elle utilise une grille régulière, 2) comme elle utilise un seul sommet dual, elle peut produire très facilement des arêtes et sommets non-variété et 3) la localisation du sommet dual n'est pas adaptée à la géométrie de la surface à l'intérieur de la cellule.

Dans le but de résoudre les problèmes de configurations non-variété dans DC et dans la méthode de Cuberille, Nielson a proposé un algorithme appelé "Dual Marching Cubes" (DMC) [80] qui génère une surface duale de celle produite par MC. DMC est dual à MC dans le sens que, pour chaque polygone de surface construite par MC, DMC génère un sommet dual (à l'intérieur de la cellule) et pour chaque sommet généré par MC, il existe un polygone de DMC traversant cette même arête. Finalement, une bijection est établie entre les arêtes des surfaces produites par MC et DMC. Cette dualité est illustrée sur la figure 3.17 pour le cas 6 de la LUT de MC (à gauche) et le cas 13 de DMC. Les polygones et les sommets sont numérotés pour mettre en évidence la bijection existante entre MC et DMC.

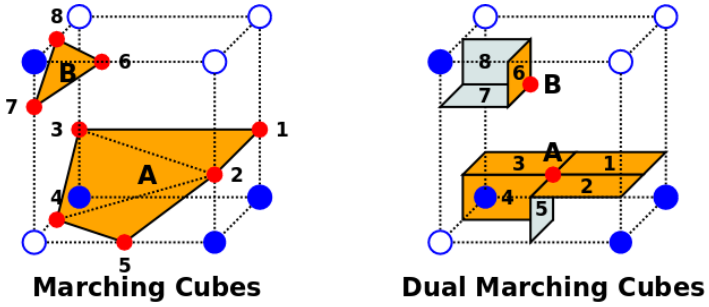



FIGURE 3.17 – Illustration de la dualité entre les surfaces produites par MC et DMC. Les cas montrés sont le cas 6 de MC et son équivalent topologique en DMC (cas 13). Les sommets de la surface (sur les arêtes) de MC son numérotés afin de pouvoir voir son polygone dual correspondant de DMC. Les polygones de MC sont aussi notés avec des lettres afin d'identifier les sommets duaux correspondants (à l'intérieur de la cellule) de DMC.

Cette dualité permet d'affirmer que si MC peut construire une surface 2-variété à partir d'un objet volumique $V$, DMC le fait aussi. Nielson a aussi proposé une table (look-up-table) pour DMC reproduite sur la figure 3.18.

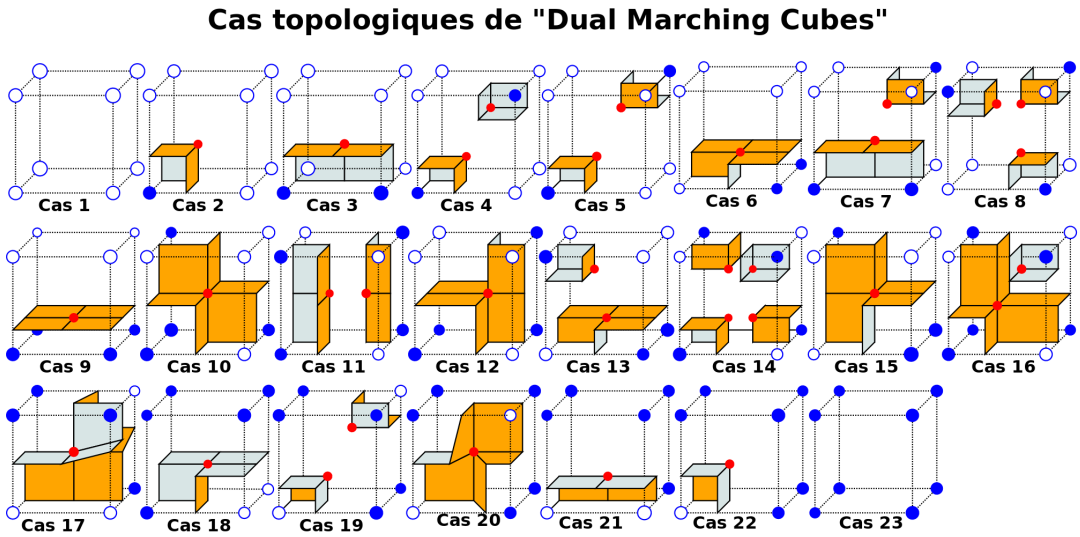



FIGURE 3.18 – Table utilisée pour la génération des sommets duaux dans l'algorithme "Dual Marching Cubes". Chaque nœud d'une cellule peut être à l'intérieur (plein) ou à l'extérieur (vide) de la surface $\partial V$. Les sommets du maillage sont à l'intérieur de la cellule et sont connectés entre eux par des arêtes traversant les facettes des cellules. Les morceaux de la surface sont représentés en orange du côté extérieur et gris du côté intérieur. Chaque patch traverse une arête intersectant $\partial V$.

Comme le montre la figure 3.18, DMC permet d'avoir jusqu'à 4 sommets duaux à l'intérieur de la cellule et sépare systématiquement les composantes à l'intérieur d'une cellule pour éviter les configurations ambiguës (voir le cas 5 de la figure 3.18). Ainsi, il n'apparaît pas de sommets non-variété comme dans l'algorithme DC. Cependant, il existe encore des cas où DMC produit des arêtes non-variété quand deux cellules partagent une facette ambiguë. Un exemple est fourni sur la figure 3.19.

Le cas illustré sur la figure 3.19 peut être détecté lors d'une étape de post-traitement afin d'obtenir un maillage 2-variété. DMC permet alors d'obtenir des maillages 2-variété sur une grille de n'im-

porte quelle résolution.

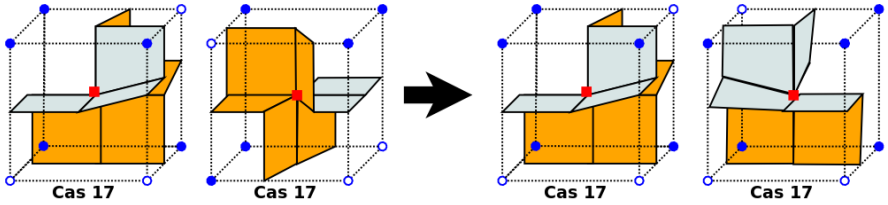



FIGURE 3.19 – DMC peut produire des arêtes non-variété quand deux cellules du cas 17 de la table de DMC sont connectées par une facette de topologie ambiguë.

Une autre méthode qui permet de gérer plusieurs sommets dans une cellule a été proposée par Zhang *et al.* [130] dans le contexte de la simplification de surfaces avec des octrees. Cette méthode utilise l'algorithme DC complété avec des cellules enrichies afin de pouvoir stocker des composantes topologiques de la surface pendant la simplification. Dans cette méthode, plusieurs sommets peuvent être fusionnés s'ils appartiennent à une même composante connexe de la surface. Sur la figure 3.20, nous illustrons un exemple de l'application de cet algorithme en 2D. Si nous avons des sommets duaux (carrés rouges) qui sont contenus dans des cellules à un niveau de profondeur $i$ (voir figure à gauche), cette solution détecte les composantes connexes des sommets de cellules qui se trouvent à l'intérieur de la surface (sommets en bleu). Dans le cas de la figure 3.20, nous avons deux composantes connexes qui contiennent à leur tour plusieurs sommets duaux. En conséquence, nous pouvons simplifier ces contours en éliminant les cellules au niveau $i$ et en fusionnant les sommets duaux de chaque composante en un seul sommet qui va représenter chaque composante. Ces sommets duaux simplifiés seront contenus dans une seule cellule de niveau $i-1$ et ils seront placés dans la position donnant la courbure la plus élevée.

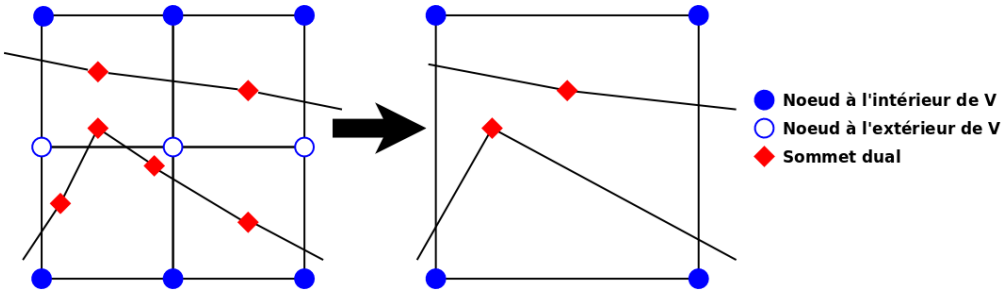



FIGURE 3.20 – Illustration en 2D de l'algorithme de simplification proposé par Zhang *et al.* via l'utilisation d'une représentation améliorée des cellules. Les noeuds en bleu sont à l'intérieur de $V$ et ceux en blanc sont en dehors. Les noeuds en rouge sont les sommets duaux.

Les limitations principales de cette méthode sont qu'elle ne permet pas de gérer plus de deux sommets duaux par cellule et qu'elle est conçue pour la simplification de surfaces et non pour une génération "top-bottom" des surfaces adaptatives.

Dans le but d'obtenir des surfaces sans auto-intersection, Ju et Udeshi [60] ont proposé une approche hybride de DC qui détecte si un patch peut générer des cas d'intersection et les résout. En premier lieu, comme avec DC chaque morceau de surface doit traverser une arête partagée par quatre cellules (dans le cas régulier), Ju et Udeshi suggèrent qu'une manière d'éviter que des patches s'intersectent est de s'assurer qu'ils soient placés dans des régions mutuellement exclusives. Cette région est définie à partir des tétraèdres formés par :

– les sommets placés sur les facettes des cellules $V_f$ ;
– les sommets à l'intérieur des cellules $V_c$ ;
– les sommets de l'arête partagée par les cellules.
La figure 3.21 à gauche illustre cette construction. Une fois tous les tétraèdres construits autour de l'arête qui traverse la surface (voir la figure 3.21 au centre), nous devons vérifier que le patch de surface soit contenu entièrement à l'intérieur de cette région. Si tel est le cas, le patch ne va pas générer d'auto-intersection, sinon, un sommet est ajouté sur l'arête $V_e$ et le patch est remplacé par un éventail de triangles comme sur la figure 3.21 à droite.

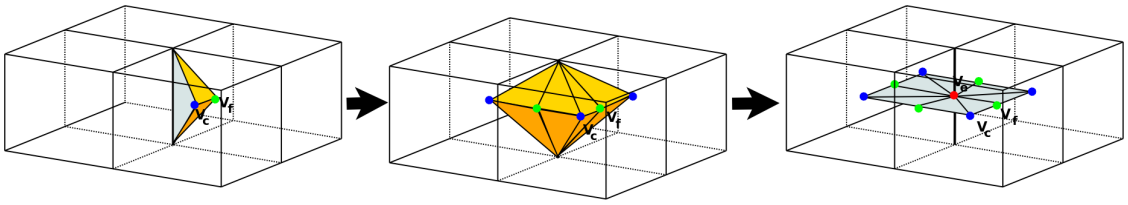



FIGURE 3.21 – Illustration de la méthode proposée par Ju et Udashi. Les sommets duaux sont affichés en bleu. L'algorithme construit des tétraèdres avec des sommets placés à l'intérieur de la cellule $V_c$, des sommets sur les facettes des cellules $V_f$ et par les sommets de l'arête partagée par les cellules (à gauche). Tous les tétraèdres contenant l'arête partagée par les quatre cellules permettent de créer un éventail qui délimite une zone exclusive pour ce patch (au centre). Si le patch n'est pas entièrement contenu dans cette zone, il est divisé en un éventail de triangles, en ajoutant un sommet $V_e$ (en rouge) sur l'arête centrale.

Cette algorithme élimine les problèmes d'auto intersection de DC mais réintroduit la limitation d'avoir seulement un sommet dual par cellule et, par conséquent, peut générer des surfaces qui ne sont pas 2-variété.

Pour obtenir des surfaces 2-variété adaptatives, Schaefer *et al.* *[93]*(DMC) utilisent une variante de DMC sur un octree appelée "Manifold Dual Contouring" (MDC). Cet algorithme commence par créer un octree adaptatif jusqu'à un niveau de profondeur qui garantit que la topologie de la surface est conservée et que toutes les cellules qui intersectent la surface ont la même taille. Cela leur permet d'appliquer la table de DMC pour générer les sommets duaux de la surface. Ensuite, ils appliquent un algorithme de simplification basé sur les arêtes des cellules de l'octree et le calcul de la caractéristique d'Euler afin d'obtenir une surface simplifiée qui conserve la topologie de la surface originale.

Le critère utilisé pour s'assurer qu'une simplification conserve la topologie de la surface $S_v$ à l'intérieur d'une cellule, se base sur le calcul de la caractéristique d'Euler définie comme :

$$\chi(S_v) = V(S_v) - E(S_v) + F(S_v) \tag{3.2.2}$$

Où $V(S_s)$, $E(S_v)$ et $F(S_v)$ sont respectivement le nombre de sommets, d'arêtes et de facettes de la surface $S_v$. Cette équation permet de calculer le genre de la surface défini comme le nombre maximum de courbes fermées simples que l'on peut tracer sur une surface sans la déconnecter. L'équation 3.2.2 combinée avec le calcul des intersections de la surface avec les arêtes des facettes de la cellule permet d'établir le critère de simplification à utiliser pour rendre la surface adaptative. Ainsi, la simplification d'un patch de surface $S_v$ contenu dans une cellule génère une surface 2-variété si et seulement si $\chi(S_v) = 1$ et le nombre d'intersections de $S_v$ avec les arêtes de chaque facette de la cellule est 0 ou 2 comme l'illustre le cas de la figure 3.22.

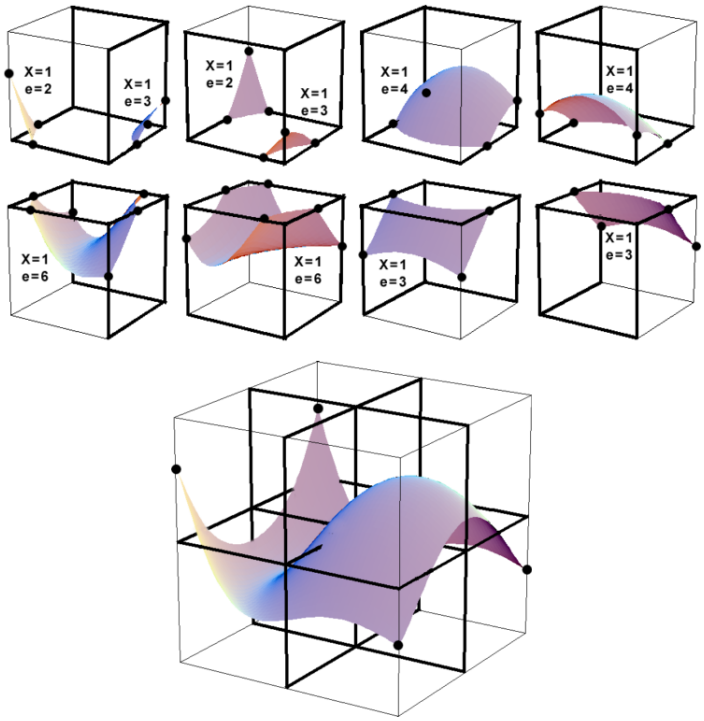



FIGURE 3.22 – Application du critère topologique de MDC sur les huit sous-cellules d'une cellule $C_v$. Dans ce cas, la simplification est possible. Figure extraite de l'article "Manifold Dual Contouring" [93].

MDC permet d'obtenir des surfaces adaptatives et de simplifier des surfaces tout en conservant leur propriétés topologiques (voir l'exemple de la figure 3.23).

Néanmoins, comme MDC utilise un approche "bottom-up", il a besoin de construire un octree qui représente la topologie originale de l'objet et dans lequel toutes les cellules qui intersectent la surface doivent avoir la même taille. Cette approche n'est pas adéquate pour traiter des volumes de grande taille où il n'est pas toujours possible d'aller jusqu'à la résolution maximale pour simplifier la surface. D'autres méthodes [118] utilisent la table de DMC pour obtenir des surfaces adaptatives à partir d'images de profondeur mais elles ne résolvent pas les problèmes de configurations non-variété illustrées sur la figure 3.19 et doivent appliquer des post-traitements afin de dupliquer l'arête non-variété et éliminer les problèmes sur la surface.

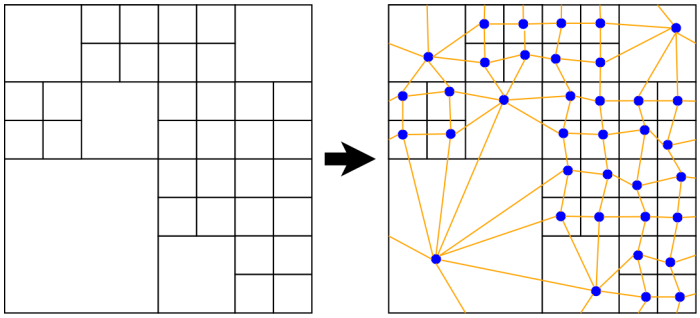
### 3.2.2 Méthodes primales sur des grilles duales

Ces méthodes construisent des structures duales de subdivision spatiale à partir des structures de subdivision primales tels les octrees. Ces subdivisions duales sont générées et guidées par les caractéristiques géométriques et topologiques du volume original.

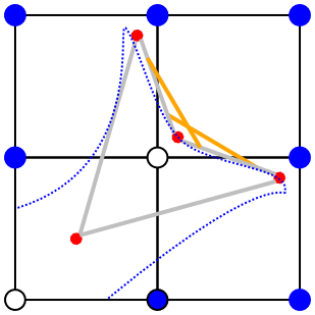
Schaefer *et al. [94]*ont présenté un algorithme dual (DMC2) pour générer, en plusieurs étapes, des surfaces adaptatives avec une quantité réduite de triangles. Dans la première étape, ils construisent un octree adaptatif guidé par un critère d'erreur géométrique basé sur des QEF en utilisant des données d'Hermite afin de décider si les cellules doivent être divisées. Une fois l'octree construit, ils génèrent un sommet dual par cellule et le placent dans la position qui minimise l'erreur de

FIGURE 3.23 – Application de l'algorithme de MDC afin de simplifier une surface de "Vierge".

la QEF de la cellule. Ensuite, l'octree est parcouru pour connecter les sommets duaux dans le but de construire une grille (duale) de cellules hexaédriques. Finalement, comme les cellules de la grille duale sont topologiquement équivalentes à des cubes, un algorithme MC peut être utilisé afin d'extraire la surface. Cette procédure est affichée sur la figure 3.24a.



(a) Génération d'une grille duale à partir d'un octree.



(b) Auto-intersection générée par DMC2.

FIGURE 3.24 – Illustration en 2D de l'algorithme DMC2. (a) Des QEF sont utilisées pour construire un octree adaptatif. Ensuite, DMC2 crée un sommet dual à l'intérieur de chaque cellule et une grille duale est construite en connectant ces sommets. Ensuite, la table de MC est appliquée pour extraire une surface adaptative. (b) Exemple en 2D d'un cas d'auto-intersection. Les sommets duaux ne produisent pas toujours des cellules convexes et la surface (ligne pointillée) peut s'auto-intersecter.

Le principal problème de cette solution est qu'elle ne garantit pas que les cellules de la grille duale soient convexes (voir figure 3.24b), cela peut causer des auto-intersections dans la surface. De plus, comme elle utilise la triangulation de MC et que les sommets du maillage sont localisés sur les arêtes des cellules, elle génère des triangulations de mauvaise qualité et beaucoup de triangles dégénérés.

Une alternative pour obtenir des maillages plus lisses est de changer la grille hexaédrique par une grille tétraédrique. Nielson [81] a proposé une méthode appelée "Dual Marching Tetrahedra" (DMT) qui utilise une grille tétraédrique construite à partir des voxels. Cet algorithme est dual parce qu'il crée les sommets de la surface à l'intérieur de chaque tétraèdre au lieu de les placer sur les arêtes des tétraèdres à la manière de MT. Cette solution divise la grille de voxels en tétraèdres. Néanmoins, comme il existe plusieurs manières de diviser un cube afin de produire des tétraèdres

et que chaque subdivision a ses caractéristiques de continuité et d'approximation [18], la surface obtenue est fortement dépendante de la division choisie. Une fois la grille tétraédrique construite, Nielson localise un sommet dual à l'intérieur de chaque tétraèdre et les connecte avec des arêtes qui traversent les facettes des tétraèdres. Nielson a regroupé ces cas topologiques dans les trois cas sur la figure 3.25.

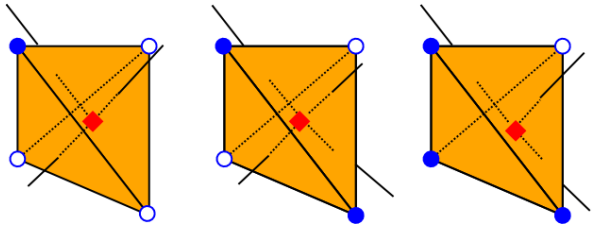


FIGURE 3.25 – Les trois cas de la table de DMT. Les sommets du tétraèdre qui sont à l'intérieur de la surface sont pleins, ceux à l'extérieur sont creux. Un sommet dual (carré) est créé à l'intérieur de chaque tétraèdre. Deux sommets duaux sont connectés entre eux avec des arêtes qui traversent la facette partagée par les deux tétraèdres qui les contiennent.

Cette méthode génère des surfaces qui sont un peu plus lisses que MT et beaucoup plus lisses que MC. Un exemple de son application sur des volumes discrets sur la figure 3.26.

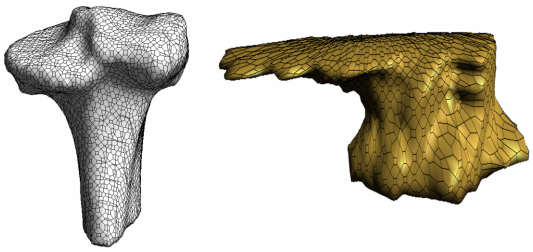


FIGURE 3.26 – Application de l'algorithme de DMT proposé par Nielson pour l'extraction de la surface de deux volumes osseux à partir d'une segmentation. Les maillages obtenus sont lisses mais la surface n'est pas adaptative.

Le principal problème de cette solution et qu'elle utilise une grille régulière. Une possible extension pour des surfaces adaptatives peut être envisagée si nous nous assurons de réaliser une subdivision conforme à partir des idées proposées par Weiss et Floriani [123, 124] sur des hiérarchies de diamants adaptatives déjà résumées dans la première partie de ce chapitre.

Une combinaison intéressante entre un algorithme dual et MT que nous appellerons "Dual Marching Tetrahedra" (DMT)[73] a été proposée par Manson et Schaefer . Cette solution construit un octree adaptatif raffiné par rapport à l'erreur d'approximation de la surface. Cette erreur est estimée avec une fonction d'erreur quadratique (QEF) construite avec les normales de la surface sur les points d'intersection avec les arêtes des cellules. Ensuite, il localise un sommet dual dans chacune des 0-cellules (un sommet), 1-cellule (une arête), 2-cellule (une facette) et 3-cellule (une cellule) comme illustré sur la figure 3.27. Puis, une grille tétraédrique duale est générée avec les sommets duaux (voir la figure 3.27d). Finalement, MT est utilisé sur la grille duale afin d'extraire une surface 2-variété.

Cette solution présente l'avantage de toujours produire une surface 2-variété adaptative et de conserver les arêtes vives. Néanmoins, sa principale faiblesse est qu'aligner les sommets duaux aux caractéristiques principales de la surface peut produire des triangles allongés ou aplatis. En

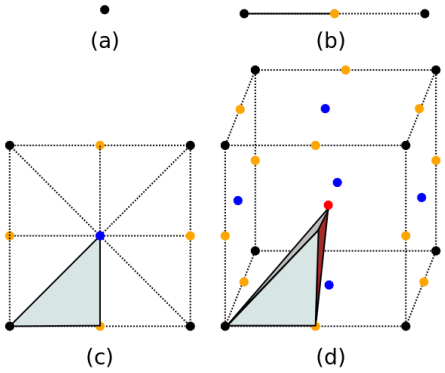



FIGURE 3.27 – Configuration des sommets duaux générés par l'algorithme de DMT dans chaque cellule de l'octree. Un sommet dual est créé sur chaque 0-cellule (en noir), 1-cellule (en jaune), 2-cellule (en bleu) et 3-cellule (en rouge). Ensuite, une grille tétraédrique est construite.

conséquence, les surfaces contiennent une grande quantité de triangles dégénérés comme nous pouvons l'observer sur la figure 3.28 à gauche et même après l'application d'un post-traitement à droite. Finalement, même si la division de l'espace avec une grille tétraédrique améliore la qualité de l'approximation, elle va aussi nécessairement produire des maillages avec plus de triangles.

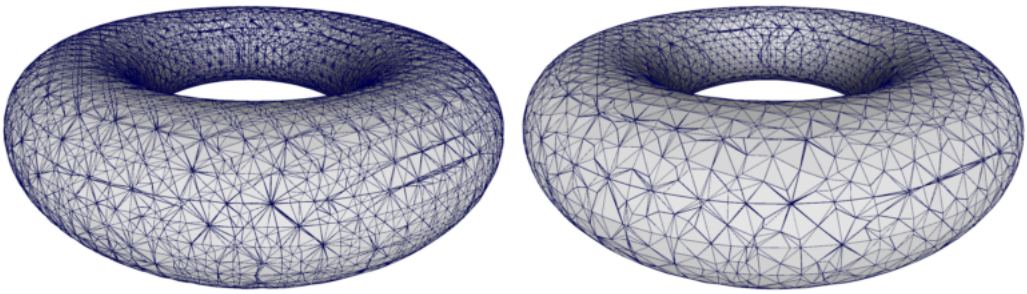


FIGURE 3.28 – Exemple d'utilisation de DMT pour l'extraction de la surface d'un objet "Donut". À gauche, la triangulation obtenue initialement. À droite, même après l'application d'un post-traitement, il reste encore beaucoup des triangles dégénérés.

## 3.3 Discussion

A partir de l'étude des solutions dans l'état de l'art pour l'extraction de surface à partir de données volumiques, nous voyons qu'il existe une forte relation entre la quantité d'information sur un volume et la capacité d'obtenir une bonne approximation de sa surface. La plus grande partie des méthodes décrites ci-dessus ont besoin d'extraire des informations complémentaires afin de pouvoir détecter les caractéristiques de la surface. Les informations les plus couramment utilisées sont les normales sur les points d'intersection de la surface ou les champs de distance discrets par rapport à la surface. Pratiquement aucune méthode de l'état de l'art n'analyse le coût et les problèmes liés au calcul de ces informations sur des données volumiques. De plus, la quasi totalité des méthodes ne mettent pas en évidence ces difficultés et n'expliquent pas comment ces calculs impactent les performances et l'efficacité de sa solution. Généralement, l'extraction de ce type d'information est faite avec un pré-traitement et peut impliquer un ou plusieurs parcours des données ou une connaissance

approfondie des caractéristiques du volume à traiter.

Pour cela, nous considérons qu'il est important d'essayer de comprendre les problèmes fondamentaux de l'extraction de surface à partir de données volumiques et de présenter une méthode qui tienne compte de toutes ces propriétés. Dans la section suivante, nous présenterons notre méthode pour l'extraction de surfaces.

# Chapitre 4

# Notre approche

Nous avons développé une approche pour l'extraction d'une surface 2-variété à partir de la représentation volumique d'un objet. Nous cherchons à obtenir un bon compromis entre la qualité de l'approximation obtenue et la qualité des éléments géométriques (triangles) de la surface. Pour cela, nous avons décidé d'utiliser une division spatiale adaptée pour traiter des données volumiques segmentées sans aucune information préalable. Nous avons fait le choix d'une méthode duale afin d'avoir plus de liberté dans la localisation des sommets. Cela va nous permettre d'améliorer la qualité de l'approximation de la surface et d'obtenir des surfaces plus lisses en éliminant une bonne partie du crénelage généré par les algorithmes de division spatiale du type "Marching Cubes" (MC). Notre solution est divisée en plusieurs étapes qui sont représentées graphiquement sur la figure 4.1.

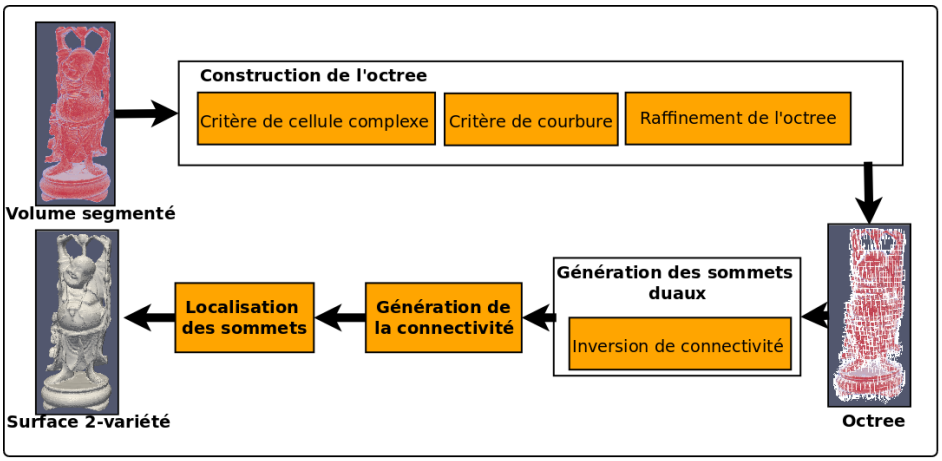



FIGURE 4.1 – Pipeline général de notre algorithme. À partir d'un objet volumique $V$, nous construisons un octree basé sur la topologie et la géométrie de $V$. Ensuite, nous générons les sommets duaux et la connectivité du maillage puis nous appliquons un algorithme pour localiser les sommets et améliorer l'approximation géométrique de la surface.

Les principales étapes de notre algorithme sont les suivantes : une division spatiale de l'objet d'intérêt avec un octree généralisé en tenant compte de la topologie et de la géométrie de l'objet. Ensuite, nous générons les sommets qui feront partie du maillage et nous construisons la connectivité de la surface à partir de la structure de division spatiale. Finalement, nous utilisons une nouvelle technique de localisation afin de placer les sommets du maillage sur la surface de l'objet

discret. Nos contributions à chacune des ces étapes seront décrites plus en détail dans les sections suivantes. Pour commencer, nous allons introduire la structure de données utilisée pour diviser l'objet.

## 4.1 Structure de données

L'implémentation de la structure de données pour diviser le volume $V$ est l'une des parties les plus critiques car nous avons besoin d'un accès rapide et efficace aux cellules voisines et un encodage compact qui nous permette d'explorer des volumes de grande taille. Notre structure de données est basée sur celle proposée par Lewiner *et al.* [68] et implémentée à l'aide d'une courbe de remplissage d'espace connue sous le nom de code de Morton.

### 4.1.1 Code de Morton

Le code de Morton permet d'obtenir un codage linéaire de la subdivision régulière d'un espace d-dimensionnel. Pour des raisons de clarté, les principes des codes de Morton seront présentés en dimension 2, avec un bit par axe à chaque subdivision. Leur généralisation à des dimensions plus élevées ne présente pas de difficulté. Ainsi, dans le cas d'une division régulière d'un espace bidimensionnel, nous pouvons établir les coordonnées de chaque 2-cellule comme illustré sur la figure 4.2a. Ensuite, nous pouvons construire le code de Morton correspondant à chaque cellule comme le montre la figure 4.2. Afin de pouvoir stocker de manière efficace un code, nous devons utiliser une quantité fixe de bits. Le code sera construit à partir du bit le moins significatif en stockant alternativement les bits de chaque coordonnée depuis les moins significatifs jusqu'aux plus significatifs. Ensuite, nous ajouterons toujours un bit à 1 afin d'identifier la fin du code. Enfin, les bits restants seront mis à zéro. Par exemple, pour stocker le code 0110, nous pouvons utiliser 8 bits et le stockage du code deviendra 00010110. Cela permet de produire une subdivision codée de manière scalaire (voir figure 4.2c) et qui offre aussi un parcours en $Z$ de l'espace comme l'illustre le chemin fléché sur la figure 4.2d.

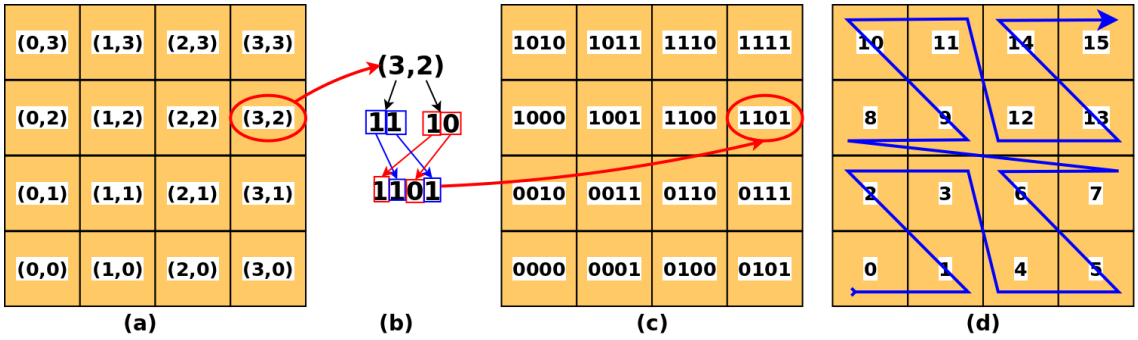
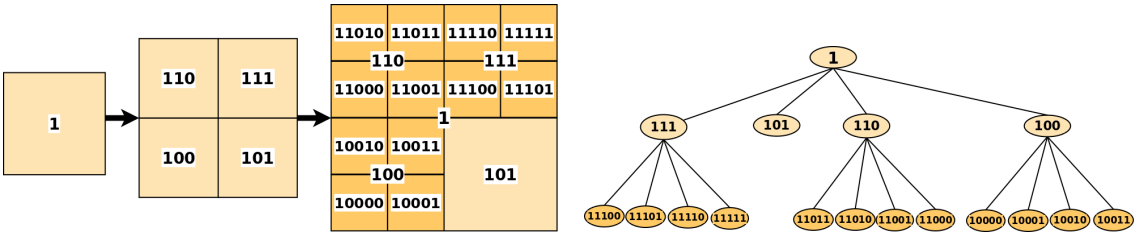



FIGURE 4.2 – (a) Illustration d'une division spatiale régulière en 2D avec les coordonnées de chaque case. (b) Le code de Morton peut être construit en alternant les bits qui représentent les coordonnées de chaque case. (c) Nous ajoutons un bit à 1 dans le code de Morton afin d'identifier la fin du code. (d) La suite de codes de Morton pour une division spatiale 2D induit un parcours de l'espace en Z comme illustré par le chemin fléché.

De plus, le code de Morton peut exprimer implicitement une relation hiérarchique entre les subdivisions successives d'un espace. Par exemple, en 2D, la figure 4.3 affiche les subdivisions spatiales

produites avec un quadtree et son codage avec le code de Morton. Comme nous venons de l'expliquer, le code de chaque cellule est construit par la concaténation des bits utilisés pour représenter la position d'une cellule à un niveau donné. Lors d'une subdivision, les bits du nouveau niveau deviendront les bits les moins significatifs du code. Le nombre $N$ de bits disponibles pour stocker le code par rapport à la dimension $d$ de l'espace à diviser déterminera le nombre de niveaux que la structure peut contenir, soit $\lfloor N/d \rfloor$.




(a) Division spatiale. (b) Structure hiérarchique.

FIGURE 4.3 – Équivalence entre un code de Morton pour une subdivision spatiale 2D et la représentation hiérarchique de la subdivision. (a) Subdivision spatiale d'un espace 2D avec un quadtree. (b) Structure hiérarchique équivalente.

Comme il a été expliqué précédemment, nous aurons besoin d'un bit par axe afin de stocker la position relative de chaque code par rapport au repère. Dans le cas bidimensionnel, nous avons besoin de deux bits pour représenter chaque niveau hiérarchique. Dans le cas d'un octree, trois bits seront nécessaires afin de coder les huit cellules à ajouter lors de chaque division.

### 4.1.2 Code de Morton et coordonnées spatiales

A partir d'un code de Morton, il est aussi possible de récupérer la position spatiale d'une cellule. Si nous avons un octree ($d = 3$), le nombre de bits dans le code de Morton doit être un multiple de $d$ plus le bit indicateur de la fin du code, soit $N = dk + 1$ avec $k$ le nombre de niveaux de l'octree. Donc, le bit situé à la position $i$ à partir du bit le moins significatif ($i \in [0, N-1]$) appartient à la coordonnée $m = (i \mod d)$ tel que $0 < m \leq d - 1$. La procédure d'extraction des coordonnées spatiales à partir d'un code de Morton est illustrée sur la figure 4.4.

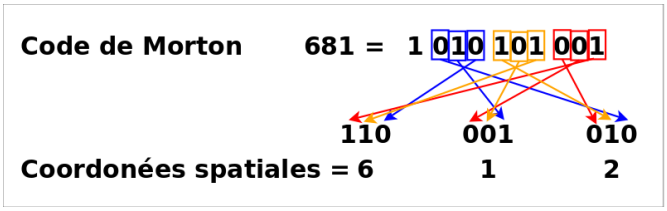



FIGURE 4.4 – Détermination des coordonnées spatiales à partir d'un code de Morton.

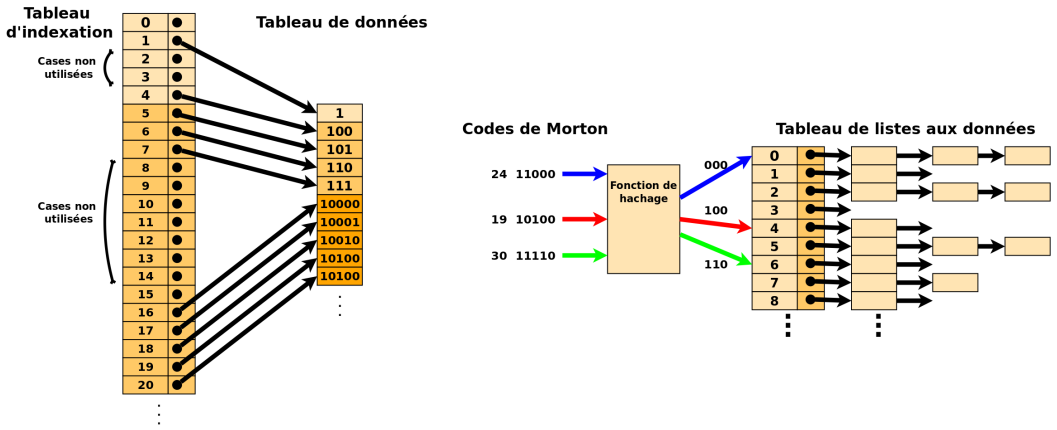
Nous avons implémenté un octree qui utilise les codes de Morton pour encoder le niveau hiérarchique et la localisation spatiale de chaque cellule. Les avantages de cette solution sont :
– un accès en temps constant aux cellules à partir du code ;
– un accès en temps logarithmique à partir des coordonnées spatiales d'une cellule.
Selon le même principe, il est aussi possible d'avoir un accès rapide aux cellules voisines de n'importe quelle cellule. Concernant l'usage de la mémoire, notre octree élimine la nécessité d'avoir huit

pointeurs dans chaque cellule pour faire référence aux sous-cellules parce que leurs positions et codes d'accès sont implicitement inclus dans la structure du code.

### 4.1.3 Implémentation de l'octree

Il existe plusieurs possibilités d'implémentation pour un octree qui utilise un code de Morton. L'une d'elles consiste à utiliser un tableau qui stocke les références vers les cellules dans des cases indexées avec les valeurs des codes de Morton. Cette implémentation linéaire fournit un vrai accès en temps constant $O(1)$ à n'importe quelle cellule de l'octree mais elle a un fort impact sur l'utilisation de la mémoire parce qu'elle a besoin d'allouer un espace mémoire contigu nécessaire pour stocker les cellules ou les références aux cellules. Par exemple, un octree avec un niveau de profondeur maximal de 8 aura besoin d'un tableau de $8^8$ cases. De plus, le code de Morton ne remplit pas totalement l'espace et une bonne partie des éléments du tableau ne seront pas utilisés. Cet aspect est particulièrement important dans le cas d'un octree adaptatif où toutes les cellules ne sont pas également divisées. Cette structure de données est illustrée sur la figure 4.5a.

Une autre implémentation consiste à stocker les codes dans une structure de données ordonnée de type arbre binaire. La représentation des codes de Morton sous forme d'entiers permet de les organiser et de fournir un accès logarithmique $O(\log n)$ aux cellules. L'avantage est d'optimiser la quantité de mémoire utilisée pour stocker les codes d'indexation mais l'accès aléatoire aux cellules est moins performant.




(a) Octree linéaire.

(b) Octree avec une table de hachage.

FIGURE 4.5 – Implémentations possibles d'un octree avec le code de Morton d'un espace en 2D (deux bits pour chaque niveau). (a) Dans un codage linéaire, le tableau de codage est indexé par les codes de Morton et contient des pointeurs vers la structure de données des cellules. Néanmoins, comme le code de Morton en 2D doit ajouter deux bits à chaque nouveau niveau, certaines cases ne seront pas utilisées. Cette problème peut s'accentuer dans le cas d'un octree adaptatif. (b) Un octree de hachage permet de trouver un bon compromis entre l'espace mémoire et l'accès rapide aux cellules à partir de leur code de Morton.

Une implémentation basée sur une table de hachage (figure 4.5b) permet d'améliorer l'utilisation de la mémoire sur des octrees irréguliers et d'offrir un temps d'accès constant en moyenne mais, suivant la fonction de hachage utilisée, il peut atteindre $O(n)$ dans le pire des cas. Pour les codes de Morton, une fonction de hachage classique utilise les $b$ derniers bits (les moins significatifs) qui peuvent être extraits en appliquant au code $K_n$ de $n$ bits les opérations $K_n \mod 2^b$ (ou

l'opération binaire $K_n \,\&\, 01_b \ldots 1_0$). Cette fonction de hachage permet de disperser les codes d'une manière uniforme dans le cas d'un octree régulier et d'obtenir une bonne distribution pour les octree adaptatifs. Cela est dû au fait que deux codes peuvent seulement entrer en collision s'ils partagent le même chemin hiérarchique à partir de leur ancêtre $\frac{b}{3}$. En utilisant $b$ bits, notre table de hachage doit avoir une taille de $2^b$ cases mémoire consécutives mais un $b$ grand peut éviter la plus grande partie des collisions (au prix d'une augmentation de la quantité de cases potentiellement vides). Il est nécessaire de trouver un bon équilibre entre la taille de la table de hachage et l'optimisation du temps d'accès. Nous avons décidé d'utiliser cette dernière implémentation parce qu'elle offre le meilleur compromis performances/utilisation raisonnablement efficace de la mémoire.

### 4.1.4 Accès aux cellules adjacentes

L'opération pour retrouver une cellule voisine est particulièrement importante dans notre algorithme. Elle va nous permettre de valider la conformité de notre subdivision et d'assurer que la surface ne contient pas de trous. Pour avoir accès aux cellules voisines, nous nous basons sur la conversion du code de Morton en coordonnées. Les cellules voisines sont atteintes par l'ajout d'un offset aux coordonnées spatiales. Enfin, ces coordonnées sont à nouveau converties en codes de Morton. En principe, cet algorithme est adéquat pour des cellules qui se trouvent au même niveau de profondeur dans l'octree mais afin de pouvoir retrouver des cellules voisines à n'importe quel niveau, nous devons nous assurer que la cellule de départ se trouve au niveau le plus profond de la hiérarchie. Ainsi, si une cellule $c$ est adjacente à une cellule plus grande $C$, nous pouvons toujours retrouver $C$ si nous détectons que la cellule voisine au même niveau n'existe pas et que nous réalisons une opération de décalage afin d'éliminer les 3 bits les moins significatifs jusqu'à trouver le code de Morton de la cellule $C$. Cette opération est décrite dans l'algorithme 1.

**Algorithme 1:** Obtention de la cellule voisine par un offset à partir d'une cellule feuille $c$.

**Entrées** : Cellule $c$ et offset[3].
**Sorties** : Cellule adjacente $a$.
*niveau* ← `ObtenirNiveau` *(c)*
codeDeMorton ← `ObtenirCodeDeMorton` ($c$) ;
coordonnees[3] ← `ObtenirCoordonneesPourCode` (codeDeMorton) ;
**pour** $i = 0 \rightarrow 2$ **faire**
  coordonnees[i] ← coordonnees[i] + offset[i] ;
**fin**
MortonCodeAdjacent ← `ObtenirCodePourCoordonnees` (coordonnees[3]) ;
**tant que** *niveau* $\geq$ *0* **faire**
  a ← `ObtenirCellulePourCode` (CodeDeMortonAdjacent) ;
  **si** `CelluleNeExistePas` *(a)* **alors**
    CodeDeMortonAdjacent ← `DecalerCode` (CodeDeMortonAdjacent,3) ;
  **fin**
  **sinon**
    Finir ;
  **fin**
**fin**

L'algorithme 1 utilise le code de Morton de la cellule pour déterminer le niveau de la cellule $c$ et les techniques décrites dans cette section pour convertir un code de Morton vers des coordonnées et vice-versa. De plus, il est capable de monter dans la hiérarchie avec des décalages binaires afin

de détecter des cellules qui se trouvent sur les niveaux supérieurs de l'octree. L'algorithme 2 décrit brièvement la fonction *ObtenirCoordonneesPourCode* qui nous permet d'obtenir des coordonnées spatiales à partir d'un code de Morton. Dans l'algorithme 2, la position des bits est considérée à partir du bit le moins significatif.

**Algorithme 2:** Algorithme pour convertir un code de Morton en coordonnées spatiales.

**Entrées** : Code de morton $c$.
**Sorties** : Coordonnées de la case codée par le code $c$.

```
compteur ← 0 ;
niveau ← ObtenirNiveau (c);
x ← 0, y ← 0, z ← 0 ;
tant que compteur ≤ niveau faire
    x ← SetBit (x,niveau - compteur, ObtenirBitParPosition (compteur*3)) ;
    y ← SetBit (y,niveau - compteur, ObtenirBitParPosition (compteur*3 + 1)) ;
    z ← SetBit (z,niveau - compteur, ObtenirBitParPosition (compteur*3 + 2)) ;
    compteur ← compteur + 1 ;
fin
```

Dans la section suivante, nous allons utiliser la structure de données et les algorithmes décrits ci-dessus pour présenter un algorithme pour construire l'octree.

## 4.2 Construction de l'octree

La première étape de notre algorithme consiste à construire un octree adaptatif divisant l'objet volumique $V$. Pour cela, nous avons utilisé une approche "top-bottom" qui décide si une cellule de l'octree doit être divisée en fonction de la topologie de la surface $\partial V$ et de ses caractéristiques géométriques. L'algorithme construit un octree compatible avec la look-up-table (LUT) de Dual Marching Cubes (DMC) et il a besoin de garantir quelques propriétés topologiques des cellules traversées par la surface. Afin de pouvoir représenter la topologie de la surface $\partial V$ à l'intérieur d'une cellule, il faut s'assurer qu'elle est correctement représentée par l'un des cas de la figure 4.6.

DMC est l'algorithme dual de MC. Ainsi, pour chaque arête intersectée par la surface $\partial V$, MC génère un sommet sur cette arête et DMC construit un polygone qui coupe l'arête. Pour chaque polygone construit par MC, DMC génère un sommet dual à l'intérieur de la cellule. Finalement, il existe une bijection entre les arêtes construites par les deux méthodes. Cette dualité est expliquée par Nielson [80] et permet d'affirmer que si l'application de MC sur un ensemble des cellules produit une surface 2-variété, DMC le fait aussi.

### 4.2.1 Critère Topologique

Selon le paragraphe précédent, comme DMC permet d'avoir plusieurs sommets duaux à l'intérieur d'une cellule, nous devons assurer que, dans la cellule, chaque morceau de surface représenté par un sommet dual soit topologiquement équivalent à un disque. De plus, comme chaque arête de la cellule ne peut être liée qu'à un seul sommet dual, il faut s'assurer qu'aucune arête ne soit intersectée plus d'une fois par la surface $\partial V$. Afin d'identifier ces cas, nous avons proposé une première définition comme suit :

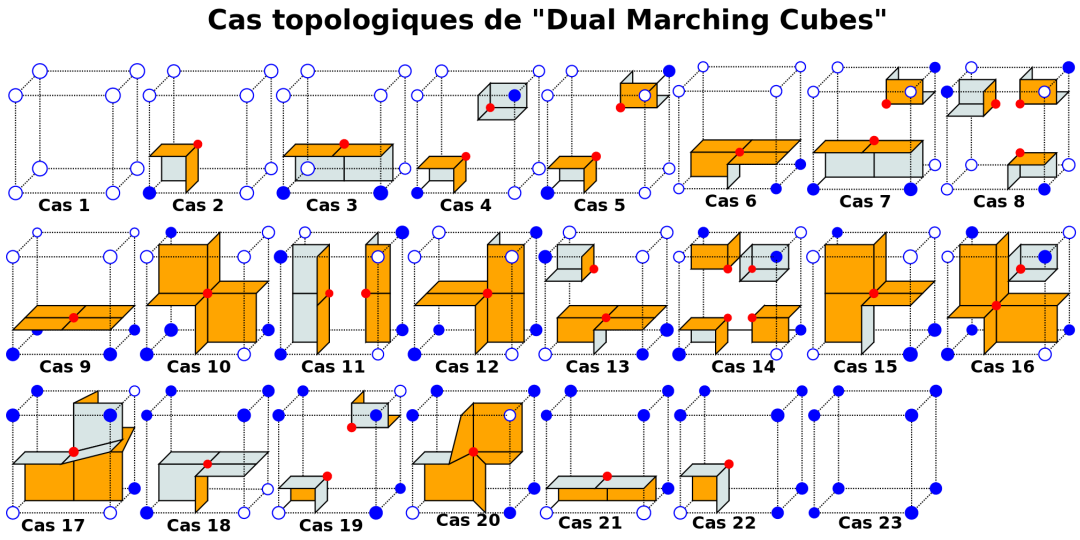



FIGURE 4.6 – Tous les cas topologiques considérés par la LUT de l'algorithme de Dual Marching Cubes. Les sommets des cellules peuvent être, soit à l'intérieur de la surface $\partial V$ (en bleu), soit en dehors (en blanc). Les sommets duaux sont localisés à l'intérieur de la cellule et l'orientation de la surface est représentée en foncée vers l'extérieur et plus claire vers l'intérieur.

**Définition 1.** Une cellule $C$ est complexe si au moins une de ses facettes ou une des ses arêtes est complexe.

- Une facette est complexe si elle coupe la surface $\partial V$ et si tous ses sommets sont soit à l'intérieur, soit à l'extérieur de $\partial V$.
- Une facette $f$ est aussi complexe si le nombre de composantes connexes de $f \cap \partial V$ est plus grand que le nombre de composantes connexes extraites à partir des sommets qui forment la facette.
- Une arête est complexe si elle traverse plus d'une fois la surface $\partial V$.

La définition 1 introduit le concept de *facette complexe* qui va nous permettre de détecter des morceaux de surface qui sont plus fins que la taille des cellules. Ainsi, nous pourrons raffiner l'octree jusqu'à identifier toutes les structures de $V$. La détection des *arêtes complexes* évitera l'apparition de configurations non-variété dans les zones de changement de résolution entre les cellules de l'octree. Par exemple, sur la figure 4.7, nous présentons, en (a) une cellule avec une facette complexe parce qu'un morceau de surface traverse la facette sans toucher les sommets qui la forment. En (b), deux facettes complexes forment un tunnel qui traverse la cellule et en (c), une configuration non-variété apparaît à cause d'une arête qui traverse plus d'une fois la surface.

A partir de la définition 1, nous pouvons établir un premier critère de subdivision des cellules comme suit :

**Critère 1.** Si la topologie de la surface $\partial V$ à l'intérieur de la cellule $C$ correspond à celle d'une cellule complexe, la cellule $C$ doit être divisée.

Nous appliquons le critère de subdivision 1 afin de garantir que toutes les cellules feuilles de l'octree, même si elles n'ont pas la même taille, ne sont pas complexes. En conséquence, nous pouvons établir que ces cellules sont compatibles avec MC et, par dualité, avec DMC. Nous appellerons un tel octree *conforme* dans le sens où nous pourrons extraire une surface 2-variété en appliquant la LUT de DMC. Cette affirmation sera prouvée dans le lemme suivant :

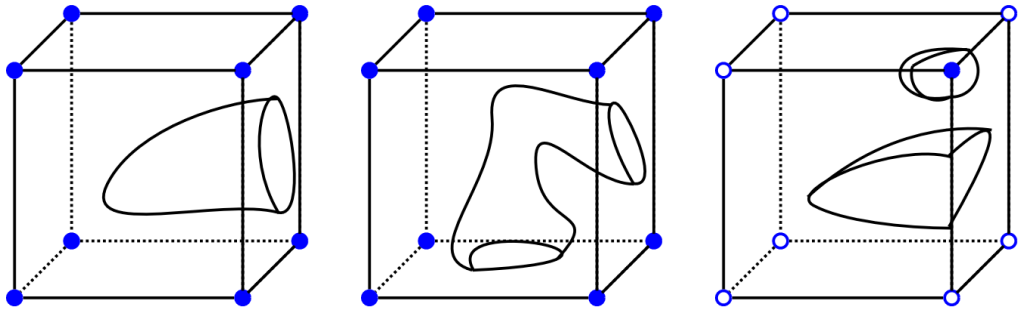



FIGURE 4.7 – Cas considérés dans la définition des cellules complexes. (a) Cellule avec une facette complexe, (b) cellule avec deux facettes complexes qui forment un tunnel, (c) cellule avec une arête complexe.

**Lemme 1.** L'application de la LUT de DMC sur un octree qui ne contient pas de cellule complexe génère toujours une surface 2-variété.

***Preuve :*** Comme nous l'avons déjà établi, DMC construit un patch de surface pour chaque arête intersectée par la surface $\partial V$. Aussi, les configurations non-variété ne peuvent apparaître qu'autour des arêtes et facettes complexes. En conséquence, nous devons prouver que s'il n'existe pas d'arêtes complexes ou de facettes complexes, il ne peut pas se produire de configurations non variété. Ainsi, nous divisons la preuve en deux parties :

Dans le cas d'une arête complexe,

Soit $C$ la cellule d'un octree à la profondeur $p$ et $S$ l'ensemble des cellules de profondeur $\{p + x \mid x > 0\}$ adjacentes à $C$ par une de ses arêtes $E$. Soit $X$ le sous-ensemble d'arêtes des cellules en $S$ telles qu'elles sont des sous-arêtes de $E$, elles sont disjointes et l'union de toutes ces arêtes $e_i \in X$ couvre totalement $E$. De plus, les arêtes en $X$ sont disjointes. Ainsi, sans perte de généralité, nous pouvons affirmer que si nous évitons que l'arête $E$ soit coupée plus d'une fois par $\partial V$, nous pouvons aussi assurer que la surface intersecte une et une seule des arêtes en $X$. Cela vient du fait que la seule intersection entre deux arêtes contenues dans $X$ est un sommet. Par conséquent, si la surface traverse $E$, elle traverse seulement une des sous arêtes contenues dans $X$.

Dans le cas d'une facette complexe,

Soit $F$ une facette de la cellule $C$ et $S$ un ensemble de cellules de profondeur $\{p + x \mid x > 0\}$ adjacentes à $C$ par la facette $F$ tel que pour chaque cellule $s_i \in S$, il existe une facette $f_i$ contenue dans $F$. De plus, les facettes $f_i$ sont toutes disjointes. Ainsi, si nous supposons que la facette $F$ ne contient pas d'arêtes complexes, $F$ ne pourrait être complexe que s'il existe au moins une composante de la surface $\partial V$ qui la traverse. Alors, s'il n'y a pas de composante qui traverse $F$, aucune composante ne traverse une facette $f_i$ et, en conséquence, aucune configuration non-variété ne peut exister■.

L'implémentation de notre critère topologique a une complexité $O(n)$ en fonction du nombre de voxels sur les facettes de la cellule. Afin d'optimiser son application, nous commençons par vérifier que les arêtes de la cellule ne sont pas complexes. Pour cela, il suffit de parcourir les douze arêtes de la cellule et de compter le nombre de changements de valeur sur chaque arête. Si la valeur change plus d'une fois, l'arête est complexe et, en conséquence, la cellule est complexe et doit être divisée

(voir figure 4.8a). Dans le cas où toutes les arêtes sont simples, nous passons à la validation des facettes. Pour déterminer si une facette est complexe, nous devons évaluer deux cas :

- La facette $f$ est homogène si ses quatre sommets sont dans $V$ ou dans son complément $V^c$. Dans ce cas, la facette est complexe s'il existe au moins un voxel appartenant à $V$ si $f \in V^c$ ou, inversement, s'il existe un voxel appartenant à $V^c$ si $f \in V$ comme l'illustre la figure 4.8b.
- La facette $f$ n'est pas homogène si les quatre sommets qui la forment n'ont pas la même valeur. Dans ce cas, afin de détecter si la facette est complexe, il faut extraire le nombre de composantes connexes des sommets appartenant à $V$ , et le comparer avec le nombre de composantes connexes prévues par DMC. Si le nombre effectif de composantes connexes de la facette est supérieur au nombre de composantes prévus, la facette est complexe (voir l'exemple sur la figure 4.8c).

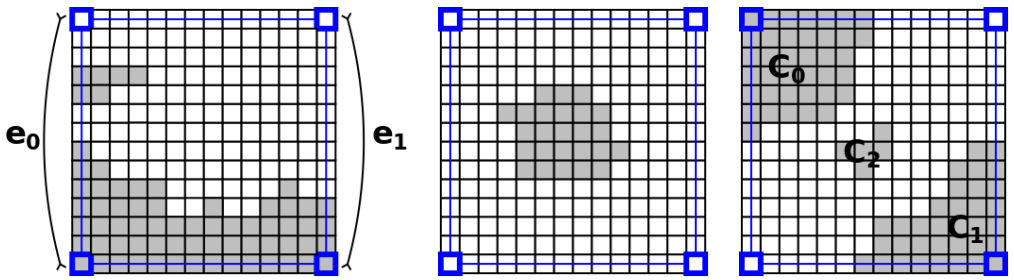



FIGURE 4.8 – Illustration de l'implémentation du test topologique. Les voxels en gris sont dans $V$ et les voxels en blanc sont dans $V^c$. (a) Les arêtes d'une facette sont parcourues pour détecter des changements de valeurs dans les voxels. Dans l'exemple de la figure, $e_0$ est une arête complexe et $e_1$ est une arête simple. (b) Exemple d'une facette homogène mais complexe, il est nécessaire de parcourir tous les voxels de la facette afin de s'assurer que la surface ne la traverse pas. (c) Si la facette n'est pas homogène et s'il n'y a pas d'arêtes complexes, il faut comparer le nombre de composantes connexes de voxels sur la facette ($C_0$, $C_1$ et $C_2 = 3$) par rapport au nombre de composantes prévues par DMC pour ce type de facette ($C_0$ et $C_1 = 2$). Dans cet exemple, il y a effectivement trois composantes au lieu de 2 prévues par DMC et la facette est complexe.

Dans le but d'extraire les composantes connexes d'une facette, nous pouvons utiliser l'un des algorithmes existant pour l'étiquetage des composantes connexes d'une image [39, 51, 107, 35, 69]. Par simplicité, nous avons utilisé un algorithme de fast-scanning avec double passage sur l'image en connexité 4. Nous avons choisi la connexité 4 afin de rester cohérents avec le choix de séparer systématiquement toutes les composantes qui sont connectées par une arête ou un sommet $3D$ (configurations critiques) et qui projetées en $2D$ coïncident avec les composantes connectées par un sommet.

L'application du critère 1 pour décider si une cellule doit être divisée permet de construire un octree adaptatif à partir duquel nous pouvons extraire des surfaces. Néanmoins, l'adaptabilité de ces surfaces sera guidée seulement par des critères topologiques peu en relation avec les caractéristiques géométriques de la surface. Afin de tenir compte de la géométrie, nous proposons un deuxième critère de subdivision basé sur l'estimation de la courbure pour chaque composante de la surface à l'intérieur d'une cellule.

### 4.2.2 Critère géométrique

Le critère géométrique que nous utilisons pour décider si une cellule doit être divisée est la courbure estimée pour l'approximation de chaque morceau de la surface. Il existe plusieurs approches pour

estimer la courbure qui sont plus ou moins adaptées aux données volumiques. Pottman *et al.* [85] ont présenté des estimateurs de courbure qui se basent sur l'intégration de l'intersection de la surface avec un noyau simple défini par l'utilisateur. Un noyau couramment utilisé est un disque en $2D$ et une boule en $3D$ en raison de la simplicité d'obtenir une solution analytique et une estimation du comportement asymptotique. Une fois défini le noyau à utiliser, l'estimateur est appliqué sur tous les points de la surface qui nous intéressent. Afin d'obtenir une estimation générale de la courbure d'une surface il faudrait échantillonner des points sur la surface et appliquer l'estimateur sur eux. Même si ces estimateurs ont été proposés initialement dans le domaine continu, les travaux récents de Coeurjolly *et al.* [34] démontrent aussi leur convergence sur des données volumiques. La figure 4.9 illustre en $2D$ l'utilisation de ces estimateurs dans le domaine continu (à gauche) et leur équivalent sur un objet discret (à droite).

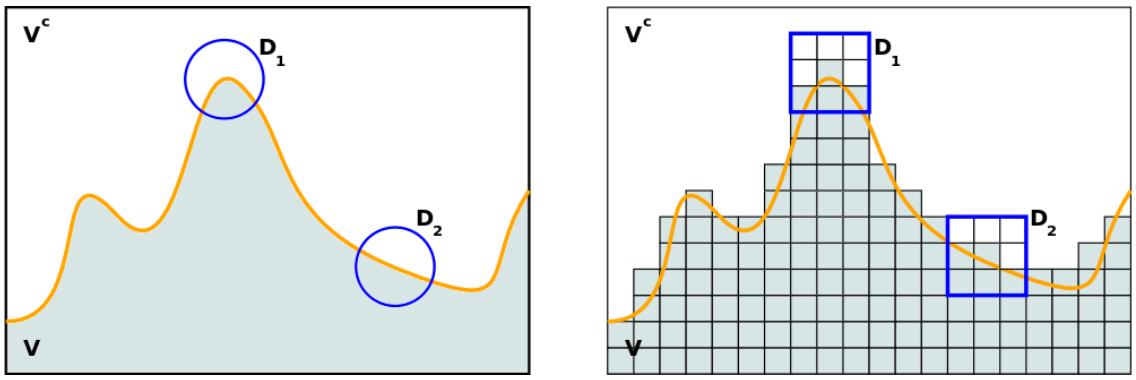



FIGURE 4.9 – Illustration des estimateurs de courbure proposés par Pottman *et al. en* $2D$. *À* gauche, leur application sur une courbe continue avec un disque comme noyau. À droite, l'équivalent discret de la courbe de gauche et les noyaux d'intégration représentés comme un 8 voisinage autour du voxel considéré.

Malheureusement, l'utilisation de ces estimateurs est très coûteuse en temps de calcul parce qu'ils nécessitent un parcours complet du volume pour trouver la surface. De plus, comme sur des volumes discrets une intégration devient une addition, l'estimation de la courbure pour chaque voxel $v$ a besoin de parcourir un ensemble des voxels autour de $v$, augmentant la complexité de l'algorithme.

Une autre approche pour estimer la courbure d'une surface utilise les normales de la surface sur quelques points bien choisis. Flin *et al.* [45] et Lenoir *et al.* [66] ont proposé des méthodes qui reposent directement sur les caractéristiques des données volumiques. Nous avons choisi de calculer la normale avec une méthode dite de "normales unitaires sur les surfels". Cette méthode consiste à utiliser la moyenne de normales unitaires sur les surfels à l'intérieur d'un voisinage géodésique, défini par un rayon $r$, autour du voxel dans lequel nous voulons estimer la normale. Soient $N_r^p$ l'ensemble des normales unitaires définies dans un voisinage géodésique de rayon $r$ autour du surfel $p$. Ainsi, la normale estimée pour le surfel $p$ est définie comme :

$$\overrightarrow{n_p} = \frac{\sum_i b_i \overrightarrow{n_i}}{\mid N_r^p \mid} = \frac{\left[ \sum_i b_i n_i^x \quad \sum_i b_i n_i^y \quad \sum_i b_i n_i^z \right]^T}{\mid N_r^p \mid} \tag{4.2.1}$$

Où $b_i$ est un facteur de pondération inversement proportionnel à la distance géodésique du surfel de la normale par rapport au surfel central $p$. Sur la figure 4.10, le calcul de la normale sur le

surfel $s$ est réalisé en utilisant la moyenne des normales unitaires dans le voisinage de rayon 1 délimité par le contour clair. Le contour plus foncé illustre le voisinage géodésique de rayon 2 qui peut être aussi utilisé afin d'améliorer la précision de l'estimation. La précision et la robustesse vont forcement dépendre du rayon $r$, et leur qualité est directement proportionnelle au nombre de voxels considérés et, en conséquence, au temps de calcul. Il faut ainsi trouver un équilibre entre la précision voulue et la contrainte de temps de calcul.

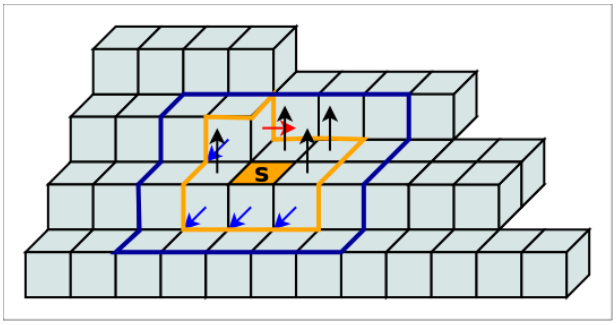



FIGURE 4.10 – Estimation de la normale sur une surface discrète. La normale du surfel $s$ est estimée en utilisant les normales unitaires des surfels contenus dans un voisinage de rayon $r = 1$ délimité par le contour clair. En plus foncé, nous délimitons un voisinage géodésique de rayon $r = 2$ pour illustrer l'idée générale.

Dans DMC, chaque arête est liée à une unique composante connexe de la cellule. En conséquence, nous calculons les normales sur chaque arête de la cellule liées à cette composante et nous les utilisons dans le critère 2.

**Critère 2.** Soient $C$ une cellule, $i_0, i_1, ...i_n$ ses points d'intersection avec $\partial V$ et $\overrightarrow{n_0}, \overrightarrow{n_1}, ...\overrightarrow{n_n}$ les normales associées. $\delta$ est un paramètre dont la valeur est comprise entre 0 et 1. Si $\text{Max}\{\overrightarrow{n_i} \bullet \overrightarrow{n_j} | i \neq j\} > \delta$, $C$ doit être divisée.

Une valeur de $\delta$ proche de 1 implique que même les cellules où la courbure est faible (les normales sont presque parallèles) seront divisées. Ainsi, nous obtiendrons une surface presque régulière parce que la plus grande partie des cellules feuilles de l'octree contenant la surface auront la même taille. Par contre, si $\delta$ est proche de 0, même les cellules pour lesquelles les normales sont presque opposées ne seront pas divisées. Cela entraînera aussi la construction d'un octree presque régulier mais qui ne reflétera pas bien les caractéristiques fines de la surface $\partial V$.

Nous illustrons sur la figure 4.11, l'utilisation des normales discrètes pour d'obtenir une information sur la forme de $\partial V$ à l'intérieur de la cellule (boîte en noir). Dans cet exemple nous utilisons le voisinage de rayon 1. La figure à gauche montre un exemple avec quatre normales. Même si certaines normales semblent presque parallèles, les normales qui sont prises en compte sont celles qui maximisent le produit scalaire. Dans ce cas, il existe deux normales légèrement opposées révélant l'existence d'une dépression à l'intérieur de la cellule. À droite, deux normales qui peuvent paraître presque parallèles se trouvent dans deux plans différents. Dans ce cas, nous mesurons la distance entre les deux surfels dans la direction de la normale afin de nous assurer qu'ils ne sont pas sur le même plan.

D'après les travaux de Flin et Lenoir, nous avons aussi observé que, au même titre que le rayon du voisinage utilisé pour estimer les normales a un fort impact sur la précision de l'approximation, il a aussi une grande influence sur la quantité de seuils que nous pouvons utiliser pour décider si

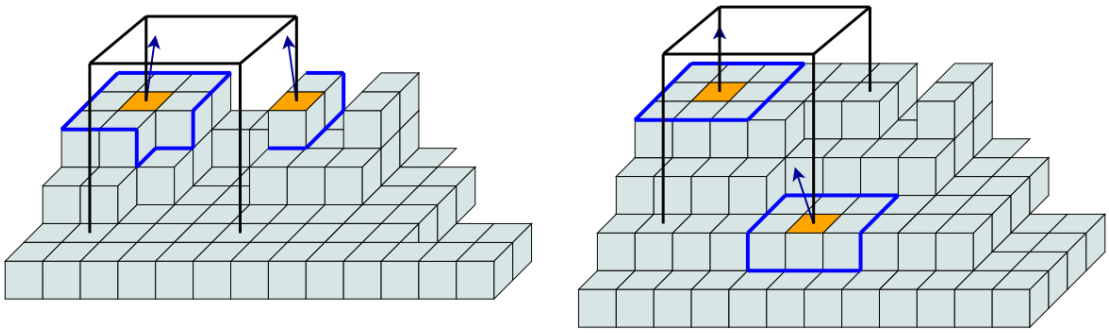


FIGURE 4.11 – Utilisation des normales discrètes afin d'estimer la géométrie de la surface à l'intérieur d'une cellule (boîte en noir). À gauche, même s'il y a quatre intersections entre les arêtes de la cellule et le plan, les normales qui seront prises en compte sont celles qui maximisent la valeur du produit scalaire du critère 2. À droite, l'existence de deux normales qui sont presque parallèles sur des plans différents peut être détectée en regardant la distance entre les deux surfels dans la direction de la normale pour détecter s'ils sont placés sur le même plan. Si cela n'est pas le cas, la cellule doit être divisée.

une cellule doit être divisée. Par exemple, si nous utilisons un rayon $r = 0$ (normale unitaire sur le surfel), les seules valeurs possibles pour le produit scalaire entre les normales de ce type sont 0 ou 1 (parallèles ou orthogonales). En augmentant le rayon $r$, nous augmentons aussi l'éventail des valeurs du produit scalaire et la sensibilité au paramètre $\delta$. Expérimentalement avec un rayon $r = 1$, nous considérons qu'une valeur de $\delta = 0.9$ est adéquate pour assurer que les structures fines de $\partial V$ soient bien préservées et que la surface produite soit bien adaptée à la géométrie de la surface originale.

Comme mentionné précédemment, nous devons calculer la normale aux points d'intersection de la surface avec les arêtes de la cellule. En conséquence, pour que l'estimation de la courbure fournie par le critère 2 ait du sens, il doit être appliqué sur une cellule non complexe. Ainsi, l'application du critère de cellule complexe doit intervenir avant de pouvoir appliquer le critère de courbure. Sur la figure 4.12, nous présentons en 2D, quelques exemples de l'utilisation de notre critère de courbure dans certains cas où son application n'offre pas de garantie d'obtenir une bonne estimation (figure 4.12a).

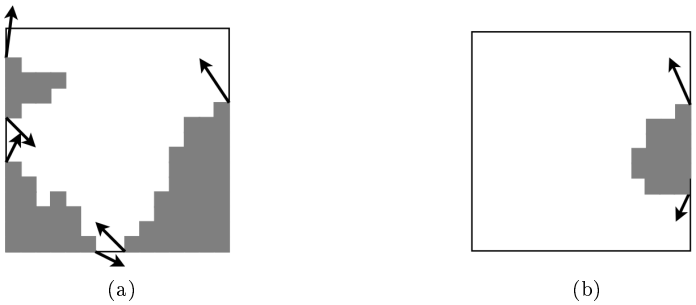



FIGURE 4.12 – Utilisation des normales dans l'estimation de la courbure d'un contour discret. (a) L'existence d'une arête complexe ne permet pas de tirer de conclusion unique par rapport à la forme de la composante. (b) Les deux normales appartiennent à la même arête et ne peuvent pas être utilisées comme référence pour DMC.

En utilisant le critère de cellule complexe, nous arrivons à détecter les configurations qui peuvent faire apparaître des problèmes dans la topologie du maillage. Ensuite, l'utilisation du critère de

courbure nous permet de raffiner les cellules où la géométrie de la surface est trop courbée pour être bien représentée par un plan minimisant l'erreur d'approximation.

### 4.2.3 Algorithme de construction de l'octree

Notre algorithme de construction de l'octree reçoit trois paramètres : la profondeur de subdivision minimale de l'octree (généralement 3 ou 4) qui nous permet de représenter les principales caractéristiques de $V$, la profondeur maximale afin de pouvoir reproduire les structures les plus fines de l'objet et enfin la courbure maximale (dans l'intervalle $[0, 1]$) estimée pour bien approximer la géométrie de la surface. Ainsi, nous commençons avec une unique cellule qui englobe l'objet $V$. Ensuite, une subdivision régulière est réalisée jusqu'au niveau minimal. À partir de là, nous utilisons le critère 1 afin de décider si les cellules doivent être divisées. Une fois qu'une cellule n'est plus complexe, nous lui appliquons notre critère de courbure jusqu'à ce que la courbure estimée ne dépasse pas le seuil établi par l'utilisateur ou que le niveau maximal de subdivision soit atteint. À ce stade, toutes les cellules qui n'ont pas été subdivisées jusqu'au niveau de profondeur maximal sont marquées comme *compactes*. Nous résumons cette procédure dans l'algorithme 3.

**Algorithme 3:** Construction d'un octree conforme à DMC.

**Entrées** : Cellule racine $c$. Niveaux de subdivision minimal $min$ et maximal $max$. Seuil de courbure $\delta$.
**Sorties** : Un octree régulier jusqu'au niveau $min$ et une division adaptative de cellules entre le niveau $min$ et $max$.

```
Ajouter c à la liste aTraiter;
tant que aTraiter n'est pas vide faire
    celluleActuelle ← Premier(aTraiter)
    n ← Niveau (celluleActuelle)
    si (n≥ min) and (n ≤ max) alors
        diviser ← non
        if Complexe (celluleActuelle) then
            diviser ← oui
        sinon
            if (Inhomogeneous (celluleActuelle) and (Courbure (celluleActuelle) > δ) then
                diviser ← oui
            sinon
                Marquer celluleActuelle comme compacte.
            fin
        fin
        si diviser alors
            Commentaire : celluleActuelle doit être divisée;
            S = {q1, q2, q3, ..., q8} ← Diviser(celluleActuelle);
            pour chaque cellule qi dans S faire
                Ajouter qi dans aTraiter
            fin
        fin
    fin
fin
```

L'algorithme 3 implémente une stratégie descendante qui nous permet de construire un octree conforme à DMC en raffinant progressivement la résolution des cellules pour obtenir une meilleure approximation de l'objet volumique $V$. Il peut être utilisé à partir de n'importe quelle cellule de l'octree pour raffiner l'approximation dans une zone particulière du modèle.

### 4.2.4 Algorithme de raffinement local

Notre approche topologique basée sur un octree nous permet de raffiner certaines cellules afin d'améliorer l'approximation avec des critères différents de ceux basés sur la topologie et la courbure. Par exemple, un utilisateur pourrait vouloir raffiner la surface par rapport au point d'observation. Cela impliquerait que les cellules proches de l'observateur soient divisées de manière indépendante afin de fournir des détails plus fins. Cette procédure est illustrée en 2D sur la figure 4.13 où des cellules *compactes* englobent un contour. Si l'utilisateur se trouve plus proche de la cellule à droite (voir figure 4.13a), une subdivision adaptative de la cellule améliorera l'approximation du contour (voir figure 4.13b). En même temps, elle peut affecter la topologie du contour extrait (sommets non-variété sur la figure 4.13c) si nous ne contrôlons pas l'existence d'arêtes complexes dans les zones de changement de résolution.

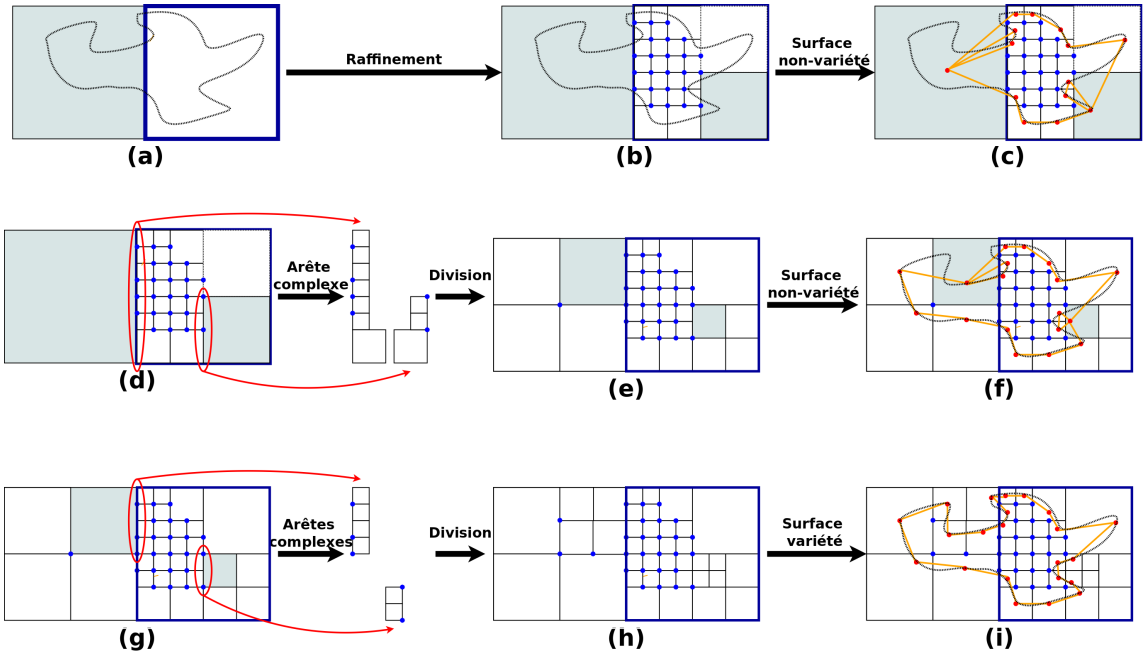



FIGURE 4.13 – Illustration en 2D du processus de raffinement des cellules *compactes* (plus foncées) afin de mieux représenter la géométrie de l'objet. Les sommets représentés par des points sont à l'intérieur de $\partial V$. (a) Un contour contenu dans deux cellules, (b) la cellule à droite est raffinée pour améliorer l'approximation du contour, (c) Des arêtes non-variété vont apparaître sur l'approximation du contour. (d) Nous parcourons les arêtes des cellules compactes afin de détecter les arêtes complexes et, si elles existent, nous les divisions comme en (e). Nous répétons cette procédure (f) et (g) jusqu'à ce qu'il n'y ait plus de cellules compactes avec des arêtes complexes (h). Cela garantit que le contour généré est une variété (i).

Le raffinement isolé des cellules peut rendre l'octree non-conforme. La figure 4.14 illustre les cas de deux cellules compactes adjacentes par une facette. Si la cellule de gauche est raffinée de manière indépendante, la facette partagée peut refléter une topologie plus complexe. Sur la figure 4.14a, une arête complexe $e$ est découverte et sur la figure 4.14b des composantes isolées de la surface apparaissent. Ces configurations vont nécessairement produire des surfaces non-variété parce que le critère de cellule complexe n'est plus respecté.

La solution que nous proposons consiste à raffiner les cellules *compactes* choisies, puis, pour chaque facette de la cellule divisée, nous allons extraire un quadtree construit à partir des sous-cellules

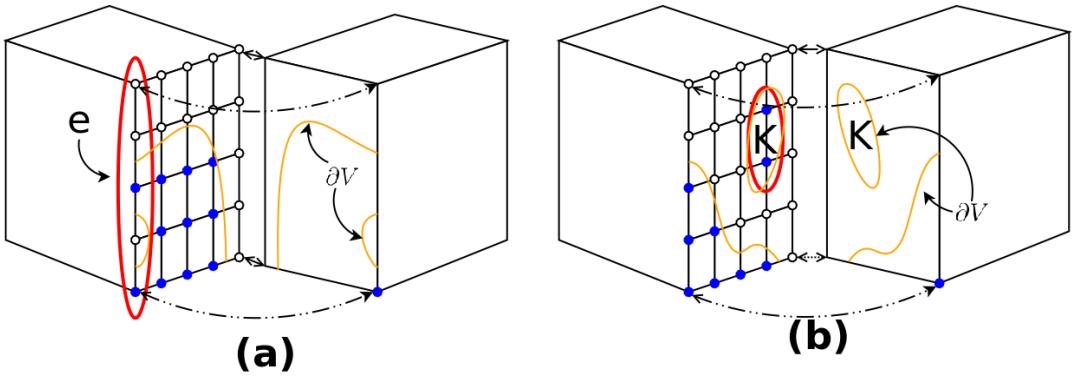



FIGURE 4.14 – La cellule compacte à gauche est subdivisée de manière indépendante de la cellule voisine à droite. Les sommets à l'intérieur de la surface $\partial V$ sont remplis et les sommets à l'extérieur sont vides. La surface réelle est aussi affichée. La subdivision peut entraîner l'apparition de configurations non-variété sur la facette partagée. (a) Une arête complexe $e$ apparaît au moment du raffinement. (b) Une composante isolée $K$ traversant la facette est visible sur les deux cellules.

adjacentes à la facette. À partir de ce quadtree, nous pourrons construire une image de la facette qui va nous permettre d'appliquer notre critère topologique afin de détecter les subdivisions qui pourraient produire une surface avec des sommets ou des facettes non-variété.

Sur la figure 4.14, nous pouvons voir l'ensemble des sommets contenus dans une facette $f$ comme une "image" de la discrétisation produite par le découpage de $f$. Pour obtenir l'information sur la subdivision de $f$ d'une cellule $C$, il faut pouvoir lister les sous-cellules de $C$ qui ont une facette contenue dans $f$. Cela est équivalent à parcourir le sous-octree qui a comme racine la cellule $C$ en ne gardant que les sous-cellules qui touchent la facette d'intérêt.

| **Facette** | **Direction** | **Position** | **Valeur** |
|---|---|---|---|
|  | $X$ Négatif | 1 | 0 |
|  | $Y$ Positif | 2 | 1 |
|  | $X$ Positif | 1 | 1 |
|  | $Y$ Négatif | 2 | 0 |
|  | $Z$ Négatif | 3 | 0 |
|  | $Z$ Positif | 3 | 1 |

Tableau 4.1 – Orientation spatiale des cellules par rapport aux bits du code de Morton. La première colonne montre la facette concernée. La deuxième la direction relative des cellules concernées par rapport aux axes. La troisième colonne affiche la position et la valeur de chaque bit par rapport à la position de la cellule encodée.

Ce parcours est possible en profitant des propriétés du code de Morton utilisé pour encoder la position des cellules. Il faut rappeler que, pour chaque niveau de profondeur, trois bits sont nécessaires pour encoder la position des sous-cellules dans le niveau suivant et que le code de Morton d'une cellule correspond à l'enchaînement de ces séries de codes. La table 4.1, présente les index utilisés pour chaque facette dans une cellule, la direction dans laquelle se trouve la facette référencée et

la position et valeur du bit utilisé pour coder la position de la cellule dans ce niveau. Ainsi, pour récupérer les cellules qui touchent une facette, il faut parcourir le sous-octree de la cellule $C$ en gardant seulement les cellules feuilles pour lesquelles le bit à la position requise a la valeur cherchée par rapport à la configuration donnée dans la table 4.1. Par exemple, si nous avons une cellule $C$ dans $\Re^3$ dont le code de Morton est 1 101, le nombre de bits (3 + 1) nous permet de savoir qu'elle se trouve à un profondeur 1. Si nous divisons cette cellule afin d'améliorer la qualité de l'approximation, tous les codes de Morton de ses sous-cellules commenceront par le code 1 101 et se poursuivront avec la position des sous-cellules à l'intérieur. Ainsi, pour une cellule avec le code **1 101** 111 011, les bits situés en deuxième position du code de Morton pour chacun des deux derniers niveaux (**1**11 et 0**1**1) sont à 1. Ceci nous indique que la cellule est contenue dans la cellule $C$ et qu'elle est adjacente à la facette orientée dans la direction positive de l'axe $X$. La figure 4.15 illustre visuellement le résultat d'une telle opération sur la facette (en rouge) dans la direction positive de l'axe $X$. Les cellules obtenues n'ont pas besoin de se trouver au même niveau de profondeur.

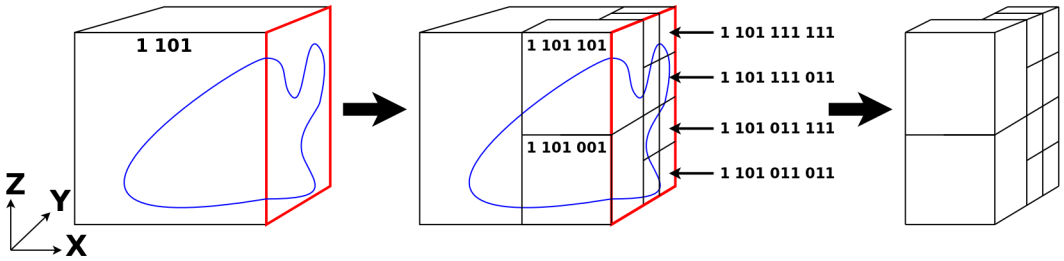



FIGURE 4.15 – Extraction de l'ensemble des cellules qui sont adjacentes à une facette $f$ dans la direction positive de l'axe $X$. Quelques codes de Morton sont illustrés. Il n'y a pas de restrictions dans la taille des cellules sur la facette, elles peuvent définir une subdivision très irrégulière de $f$.

Une fois que nous avons obtenu les cellules adjacentes à une facette $f$, nous regardons les sommets des cellules contenus dans la facette $f$. Nous gardons toujours la même orientation pour les cellules à tous les niveaux de l'octree. Pour cela, nous pouvons facilement obtenir les valeurs des voxels sur chaque sommet d'une cellule. La subdivision de l'octree projetée sur la facette $f$ peut être vue comme un quadtree $Q$ qui la subdivise. Ainsi, nous pouvons utiliser les valeurs des sommets et les dimensions des cellules $2D$ contenues dans $Q$ afin de construire une "image" comme illustré sur la figure 4.16. Cette image est une version sous-échantillonnée de l'image des voxels contenus dans $f$.

Finalement, l'application de notre critère de cellule complexe sur cette image pour chaque facette de la cellule nous permettra de détecter si l'octree reste correct au niveau local. Lorsque ce n'est pas le cas, nous avons choisi d'appliquer une stratégie de subdivision récursive des cellules compactes voisines jusqu'à revenir à un octree correct par rapport à la LUT de DMC comme l'illustre la figure 4.13 en 2D. Nous utilisons cette méthode sur toutes les cellules touchées par la subdivision jusqu'à assurer qu'il ne reste aucune arête ou facette complexe dans l'octree raffiné. Il faut néanmoins considérer que si les cellules touchées par le raffinement sont très dispersées dans l'octree, cet algorithme peut générer des effets de "vague" qui peuvent s'étaler sur une grande partie de l'octree. Un tel effet est très dépendant de la différence de profondeur entre deux cellules voisines et de la complexité du volume $V$. Néanmoins, des résultats obtenus dans l'étude d'arbres équilibrés considèrent que s'il n'y a pas une grande différence de profondeur entre deux cellules voisines, l'effet "vague" d'une telle opération de raffinement reste local dans la plus grande majorité des cas. Une description algorithmique de la procédure est présentée dans l'algorithme 4.

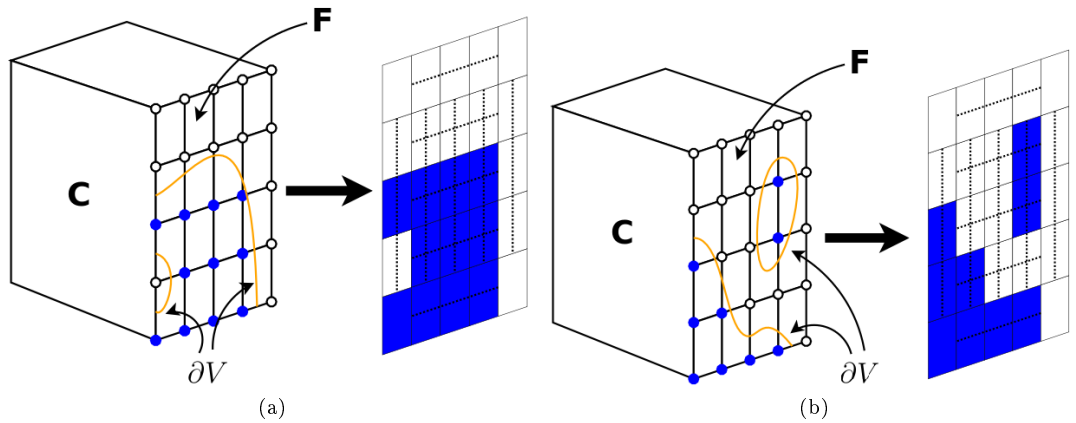



FIGURE 4.16 – Si la cellule adjacente à une cellule compacte $C$ est subdivisée de manière indépendante, il se produit une subdivision sur la facette partagée $F$. Les sommets à l'intérieur de la surface $\partial V$ sont pleins tandis que les sommets à l'extérieur sont vides. La surface réelle $\partial V$ est affichée sur la facette. La subdivision peut entraîner l'apparition de trous ou de tunnels de $\partial V$ sur la facette partagée. (a) La surface $\partial V$ coupe plus d'une fois une arête de la cellule (arête complexe). (b) La facette est devenue complexe parce qu'il existe une composante isolée qui n'était pas détectée au niveau supérieur.

**Algorithme 4:** Algorithme de raffinement local.

**Entrées** : Ensemble $T$ de cellules compactes $c$.
Niveau maximal de l'octree $maxLevel$.
**Sorties** : Soit $A$ la liste des cellules à traiter.
Initialisation : A ← T
**pour chaque** *cellule* $c \in A$ **faire**
  **pour chaque** *facette* $\in c$ **faire**
    **si** *facette est complexe* **alors**
      `RaffinementOctree` ($c$, `Niveau` ($c$), `Niveau` ($c$) +1)
      Ajouter les sous-cellules de c dans A ;
      Retirer c de A ;
    **fin**
  **fin**
**fin**

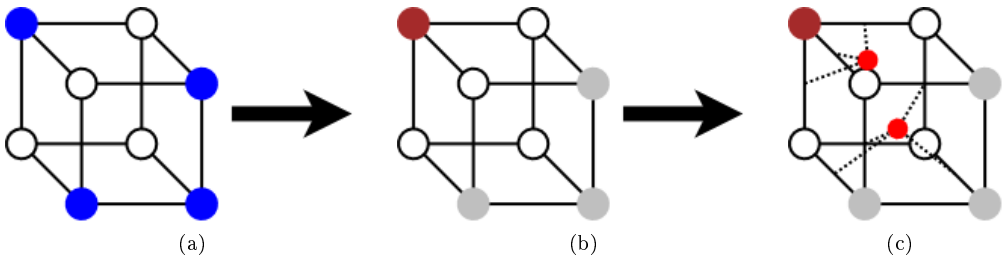
Dans cette section, nous avons présenté des critères topologiques et géométriques adaptés aux données volumiques. Ces critères sont utilisés dans la construction d'un octree qui sera utilisé pour extraire une surface 2-variété. De plus, nous avons fourni une stratégie pour raffiner progressivement les cellules dans l'octree avec des garanties quant à la topologie du maillage obtenu. Dans la section suivante nous allons présenter la méthode pour générer les sommets du maillage et déterminer leur localisation par rapport à la surface $\partial V$.

## 4.3 Génération des sommets du maillage et localisation

### 4.3.1 Génération des sommets duaux

Dans la section précédente, nous avons construit un octree compatible avec la look-up-table de l'algorithme de Dual Marching Cubes (DMC) (voir figure 4.6). La caractéristique principale des sommets créés par les méthodes duales est qu'ils sont placés à l'intérieur de la cellule et qu'ils sont liés aux arêtes de la cellule qui intersectent la surface $\partial V$. Comme chaque arête est coupée une seule fois par la surface, chaque arête est liée à un seul sommet dual. Tel qu'il a été montré précédemment, la LUT fournit une manière de générer des sommets duaux en se basant uniquement sur la configuration des sommets qui sont à l'intérieur de $\partial V$.

Afin de pouvoir généraliser la génération des sommets duaux à l'intérieur des cellules, nous avons utilisé la notion de composante connexe avec l'objectif de déduire le nombre et la localisation des sommets duaux. Chaque cellule de l'octree peut être considérée comme un graphe $G$ composé de huit sommets $v_1, v_2, \ldots v_8$ et douze arêtes $e_1, e_2, \ldots e_{12}$. Chaque sommet de ce graphe, contient une valeur scalaire que nous pouvons utiliser pour déterminer si le sommet se trouve à l'intérieur ou à l'extérieur de la surface $\partial V$. Un sommet $v_i$ est à l'intérieur de la surface si sa valeur associée est plus grande qu'un seuil donné $\lambda$. Nous appellerons $\phi_\lambda(c) : \Re \rightarrow \mathbb{Z}$ la fonction qui permet d'extraire les composantes connexes de la cellule $c$ par rapport à un seuil $\lambda$. En particulier, $\phi^i_\lambda(c)$ fournira le nombre de composantes connexes des sommets de la cellule $c$ qui se trouvent à l'intérieur de la surface et $\phi^e_\lambda(c)$ le nombre de composantes connexes par rapport aux sommets à l'extérieur de la surface. On trouve dans la littérature différentes implémentations [39, 91] de la fonction $\phi$. Le nombre de composantes connexes existant dans la cellule va déterminer le nombre de sommets duaux à construire. Chaque sommet dual sera positionné par rapport à la composante connexe qui coupe les arêtes avec lesquelles il est connecté. La figure 4.17 illustre la manière dont la localisation initiale des sommets duaux est estimée par rapport aux composantes connexes à l'intérieur de $\partial V$. Sur la figure 4.17a, nous avons une cellule possédant quatre sommets à l'intérieur de la surface (pleins) et quatre à l'extérieur (vides). Donc, si nous utilisons les sommets pleins, nous allons extraire deux composantes connexes dans la cellule (voir figure 4.17b) et placer deux sommets duaux liés aux arêtes qui sortent de chaque composante (lignes en pointillées sur la figure 4.17c).



(a) (b) (c)

FIGURE 4.17 – Calcul des sommets duaux. (a) Une cellule traversée par la surface où les sommets en bleu sont à l'intérieur de $\partial V$. (b) Étiquetage des composantes connexes de la cellule. (c) L'existence de deux composantes connexes nous permet de placer les sommets duaux. Les arêtes avec lesquelles les sommets sont connectés sont matérialisées par des lignes pointillées.

Les valeurs de $\phi^s_\lambda(c)$ et $\phi^v_\lambda(c)$ pour les 23 cas de la LUT de DMC sont présentées dans le tableau 4.2. A partir de ce tableau, il est possible de vérifier que l'application de cette technique avec les

sommets à l'intérieur de la surface va produire les mêmes configurations de sommets duaux que ceux prévus par la LUT de DMC. Même si toutes les cellules de l'octree sont compatibles avec DMC, des arêtes non-variété peuvent néanmoins être générées à cause de l'existence des facettes ambiguës partagées et du mécanisme de génération des sommets duaux. Ce type de configuration ne peut pas être détecté en regardant une cellule individuellement. Nous avons établi que ces cas problématiques apparaissent entre deux cellules des cas 17 et 20 qui partagent une facette ambiguë. Pour ces deux cas le nombre de composantes connexes à l'extérieur dépasse celui des composantes à l'intérieur. Cela génère un tunnel qui se ferme sur lui même produisant une arête non-variété comme l'illustre la figure 4.18.

| **Cas** | $\phi_\lambda^s(c)$ | $\phi_\lambda^v(c)$ |
|---|---|---|
| 1 | 0 | 1 |
| 2 | 1 | 1 |
| 3 | 1 | 1 |
| 4 | 2 | 1 |
| 5 | 2 | 1 |
| 6 | 1 | 1 |
| 7 | 2 | 1 |
| 8 | 3 | 2 |

| **Cas** | $\phi_\lambda^s(c)$ | $\phi_\lambda^v(c)$ |
|---|---|---|
| 9 | 1 | 1 |
| 10 | 1 | 1 |
| 11 | 2 | 2 |
| 12 | 1 | 1 |
| 13 | 2 | 2 |
| 14 | 4 | 4 |
| 15 | 1 | 1 |
| 16 | 2 | 3 |

| **Cas** | $\phi_\lambda^s(c)$ | $\phi_\lambda^v(c)$ |
|---|---|---|
| **17** | **1** | **2** |
| 18 | 1 | 1 |
| 19 | 1 | 2 |
| **20** | **1** | **2** |
| 21 | 1 | 1 |
| 22 | 1 | 1 |
| 23 | 1 | 0 |
| | | |

Tableau 4.2 – Nombre de composantes connexes pour les différents cas de la LUT de DMC.

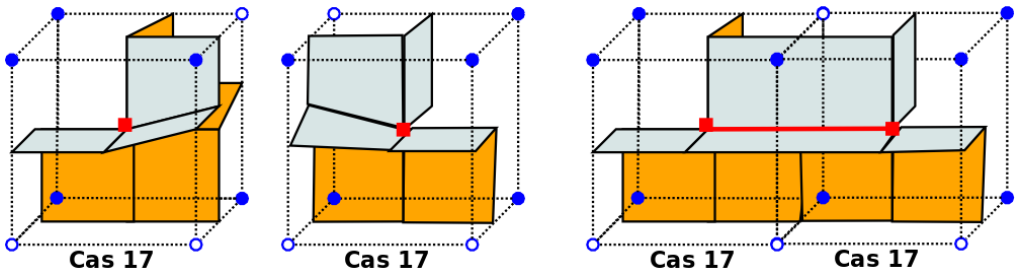



FIGURE 4.18 – Génération d'une arête non-variété entre deux cellules du cas 17 de DMC. Les cellules doivent partager une facette ambiguë. L'utilisation d'un sommet dual dans chaque cellule (carrés rouges) crée une arête non-variété (en rouge).

Afin de surmonter cet inconvénient, il faut parfois pouvoir inverser les composantes connexes qui sont utilisées pour déterminer les sommets duaux. En utilisant les composantes connexes à partir des sommets qui sont à l'extérieur de la surface, nous pouvons changer la topologie estimée de la surface à l'intérieur de la cellule. Néanmoins, nous ne pouvons pas seulement inverser la connectivité dans une seule cellule parce que cela conduirait à la génération d'un sommet non-variété comme illustré sur la figure 4.18.

Par contre, comme nous avons deux cellules avec une unique facette ambiguë, nous devons inverser la sélection des composantes dans les deux cellules à la fois (voir figure 4.20). Dans ces cas, l'inversion ne va pas affecter la connectivité du maillage par rapport aux cellules voisines parce que la facette est contenue seulement par les deux cellules. En conséquence, les sommets duaux qui sont générés sont liés aux mêmes ensembles d'arêtes intersectant la surface $\partial V$.

Afin d'automatiser cette procédure de détection, il faut pouvoir détecter les facettes ambiguës d'une cellule. Ensuite, il faut avoir accès aux cellules adjacentes à ces facettes afin de déterminer si elles correspondent à un cas 17 ou 20. Si tel est le cas, nous devons inverser la connectivité utilisée

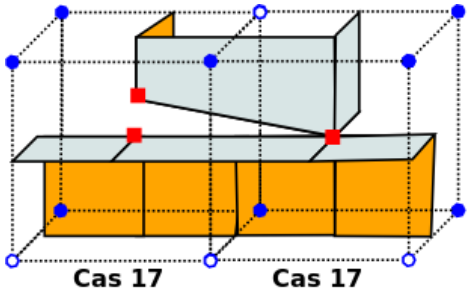



FIGURE 4.19 – L'utilisation de composantes connexes de sommets à l'extérieur de la surface va générer un sommet non variété à l'intérieur de la cellule à droite. Les cellules qui partagent une facette ambiguë doivent utiliser les mêmes composantes connexes par rapport aux sommets qui sont à l'extérieur ou à l'intérieur de la surface.

pour générer les sommes duaux dans les deux cellules.

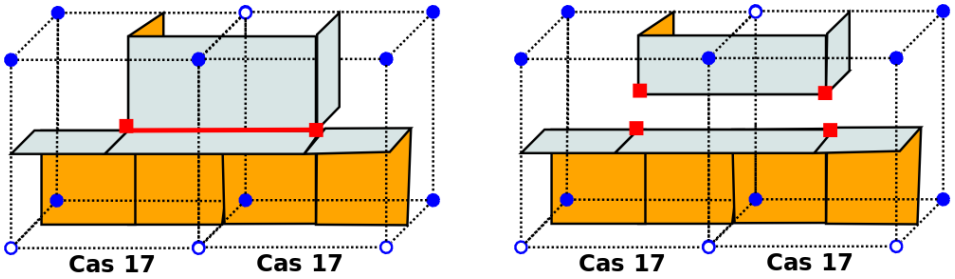



FIGURE 4.20 – Génération de deux arêtes variété entre deux cellules du cas 17 de DMC. À gauche, l'utilisation des composantes connexes des sommets à l'intérieur de $\partial V$ génère une arête non-variété. À droite, les composantes connexes des sommets à l'extérieur de $\partial V$ permettent d'éliminer l'arête non-variété en la remplaçant par deux arêtes. Les deux cellules partageant une facette ambiguë doivent utiliser les composantes connexes par rapport aux sommets qui sont à l'extérieur de la surface.

Dans le but de vérifier que notre approche résout tous les problèmes possibles entre deux cellules, nous avons testé les $2^{12}$ combinaisons possibles des valeurs de sommets pour deux cellules adjacentes par une facette (il y a 4 sommets en commun). Nos résultats prouvent que nous évitons l'apparition de sommets ou arêtes non-variété dans toutes les configurations.

### 4.3.2 Localisation des sommets duaux

La génération des sommets nous permet de reproduire la topologie de la surface $\partial V$ à l'intérieur d'une cellule. Ensuite, la localisation des sommets va nous permettre d'améliorer l'approximation géométrique de la surface générée par rapport à la surface réelle. Dans l'état de l'art, nous avons identifié deux types principaux de techniques pour placer les sommets : la première est la technique utilisée dans les algorithmes primaux et qui place les sommets sur les arêtes des cellules (voir figure 4.21b) ; la deuxième est utilisée par les méthodes duales et consiste à placer les sommets à l'intérieur des cellules en utilisant un champ de distance ou une fonction d'erreur quadratique (QEF) du type $E[x] = \sum_i \overrightarrow{n_i} \bullet (x - p_i)$ comme l'illustre la figure 4.21a.

Sur les figures 4.21(a) et (b), nous pouvons constater les limitations des méthodes précédentes.

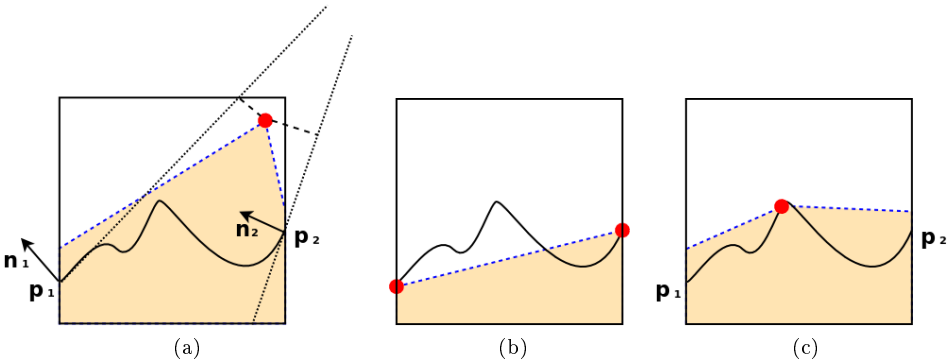



FIGURE 4.21 – Illustration en 2D des surfaces possibles obtenues par l'utilisation des différentes classes de méthodes de localisation des sommets (disques pleins). La surface extraite est en traits pointillés. (a) La minimisation proposée par les méthodes duales ne garantit pas que le sommet soit proche de la surface mais qu'il soit le plus proche possible des plans définis par les normales $\overrightarrow{n_i}$ et les points d'intersection $p_i$. (b) Le fait de localiser les sommets sur les arêtes (MC) peut faire disparaître une bonne partie de la courbure de la surface. (c) Un sommet dual placé sur la surface améliore dans la plupart des cas l'approximation.

Placer les sommets sur les arêtes ne tient pas compte de la courbure de la surface à l'intérieur de la cellule et les QEF ne garantissent pas que le sommet soit placé sur la surface $\partial V$. Nous considérons que la meilleure approche afin d'améliorer l'approximation géométrique de la surface combine les deux approches présentées. Nous allons donc placer des sommets duaux à l'intérieur de la cellule mais en nous assurant que le sommet soit placé sur la surface discrète de l'objet $V$. Cette solution est illustrée sur la figure 4.21c.

### Méthode proposée

Afin que la méthode reste locale, nous utilisons une approche basée seulement sur les composantes connexes du volume $V_c$ contenu à l'intérieur d'une cellule $C$. Comme notre algorithme place un sommet dual pour chaque composante connexe de $V_c$ dans la cellule, nous cherchons à trouver une position pour ce sommet qui reflète bien la géométrie de la surface recouvrant la composante. Dans notre méthode, le sommet dual va être localisé à l'intersection de la surface et du segment qui relie les barycentres de la composante connexe et de son complément.

De cette façon, soit $V_s$ une composante volumique tel que $V_s \subset V_c$, la méthode commence avec la localisation du barycentre $b_s$ de la composante avec un parcours itératif des voxels dans $V_s$ qui nous permet de calculer $m_s$ comme le nombre de voxels qui en font partie. Comme nous connaissons les dimensions de la cellule $C$, nous pouvons calculer le nombre de voxels $m_c$ qu'elle contient et son barycentre $b_c$. Ensuite, nous utilisons l'équation linéaire $m_c b_c = m_s b_s + m_e b_e$ avec $m_e$ le nombre de voxels du complément $V_e$ tel que $m_c - m_s = m_e$. Cette équation nous permet d'obtenir le barycentre $b_e$ du complément qui se localise dans le complément de $V_s$. Une fois les barycentres de la composante et de son complément détectés, nous disposons d'un segment qui traverse la surface et sur lequel nous pouvons trouver le point d'intersection avec la surface discrète. Toute cette procédure est illustrée en 2D sur la figure 4.22.

Pour une composante avec une forme plus concave, son barycentre peut se trouver en dehors du volume $V$ mais nous pouvons toujours garantir qu'en suivant la droite définie par le barycentre

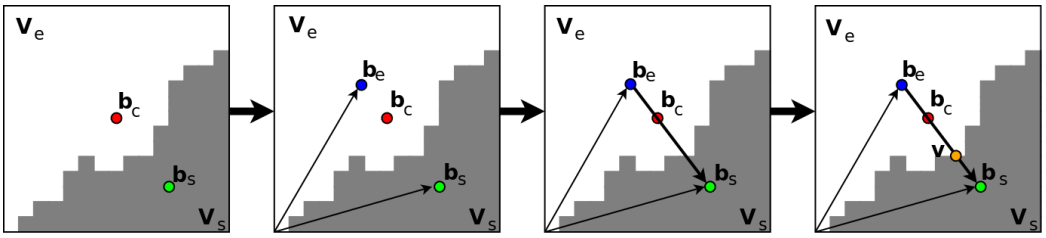



FIGURE 4.22 – Illustration 2D de la méthode pour localiser des sommets duaux sur la surface discrète à l'intérieur d'une cellule $C$. Le volume $V \cap C$ est en gris. Les barycentres du composant $b_s$, de la cellule $b_c$ et du complément $b_e$ sont utilisés pour définir un segment sur lequel le sommet dual sera localisé.

de la composante et de son complément à l'intérieur de la cellule, nous allons trouver un point sur la surface $\partial V$. Dans un cas limite, les positions des barycentres peuvent s'inverser dans le sens géométrique et la direction donnée par le vecteur défini par les barycentres ne nous guide pas vers la surface. Dans ce cas, il peut être nécessaire de parcourir toute l'étendue de la cellule le long de la droite. Afin d'avoir un bon point de départ $p$ pour la recherche de la surface, nous utilisons la relation entre le nombre de voxels de la composante $m_s$ et celui de son complément $m_e$ pour placer ce point sur le segment de recherche. En conséquence, $p$ peut se trouver à l'intérieur ou à l'extérieur de la surface et il faut le déplacer vers la surface discrète de $V$. Pour cela, nous déplacerons $p$ en utilisant une logique d'intervalles sur la droite jusqu'à trouver une région ou intervalle $3D$ d'un voxel qui se trouve à l'intérieur (si $p$ se trouvait à l'extérieur) ou à l'extérieur (si $p$ était à l'intérieur) de la surface $\partial V$.

Dans le cas où la cellule contient plus d'une composante connexe appartenant au volume $V$, il n'est pas possible d'extraire la localisation de tous les sommets duaux en même temps, donc nous traiterons une seule composante à la fois. Un exemple de l'application de cette stratégie est illustré sur la figure 4.23. La localisation de chaque sommet dual est calculée de manière indépendante en utilisant seulement la composante qui lui correspond et en considérant toutes les autres composantes comme faisant partie du complément. Le coût de calcul de cette stratégie reste strictement lié au coût du parcours des voxels qui sont à l'intérieur de $V$.

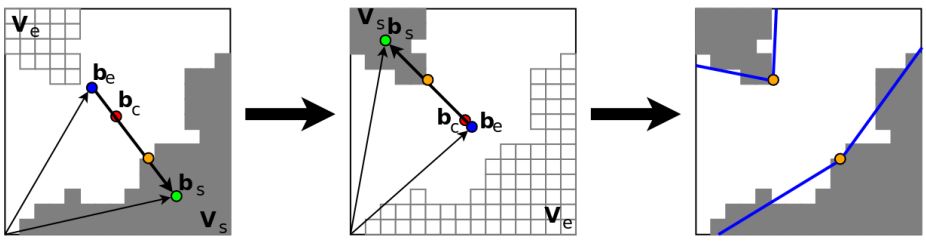



FIGURE 4.23 – Dans le cas où une cellule contient plus d'une composante connexe dans le volume $V$. Nous appliquons notre algorithme de localisation avec une composante à la fois en considérant toutes les autres composantes comme une partie du fond. Dans le cas illustré, deux composantes généreront deux sommets duaux. Finalement, la surface produite reflétera les deux composantes existant dans la cellule.

Notre technique de localisation permet de garantir que le sommet dual généré se trouve sur la surface discrète du volume $V$. Dans ce contexte, notre stratégie garantit, comme la méthode de localisation utilisée par "Marching cubes", que les sommets sont placés sur la surface. Un autre avantage de notre méthode est qu'elle tend à localiser le sommet près du *noyau* de la surface . Le

*noyau* du morceau de surface à l'intérieur de la cellule est l'ensemble des points de la surface qui peuvent être connectés à n'importe quel autre point de la surface par un segment qui ne coupe pas la surface. Dans le cas de composantes convexes, le noyau contient toute la composante. Dans le cas de composantes concaves le noyau peut se retrouver vide. Une conséquence importante de cette approche est qu'elle maximise la distance moyenne entre sommets duaux qui se trouvent dans des cellules différentes et sur la surface du volume.

*Preuve :* Cette caractéristique peut être démontrée si nous considérons que les sommets duaux sont toujours placés sur la droite $L$ qui connecte le barycentre de la composante à l'intérieur de la surface avec le barycentre du complément. Comme la droite $L$ est calculée en utilisant l'équation linéaire de centre de masse, la droite $L$ doit passer par le centre de la cellule. De plus, nous savons que pour maximiser la distance moyenne d'un sommet dual par rapport à tous les autres sommets duaux à l'intérieur des cellules voisines, il faut le placer au centre de la cellule. Avec notre algorithme, le sommet dual va toujours être placé au point de la surface le plus proche du centre de la cellule. En conséquence, notre solution a la propriété de placer le sommet dual à la position qui maximise la distance moyenne entre les sommets duaux de toutes les cellules adjacentes ■.

Cette propriété améliore grandement la qualité des triangles obtenus sans affecter la qualité de l'approximation comme nous le verrons dans la section des résultats.

Cependant, les régions de changement de résolution peuvent encore contenir beaucoup de triangles dégénérés. Une grande différence entre la résolution des cellules voisines réduit progressivement l'amplitude de l'angle minimal des triangles construits avec des sommets appartenant à des cellules de niveaux différents comme l'illustre la figure 4.24. Un niveau de subdivision additionnel engendre la division du triangle $T$ en trois triangles $T_1$, $T_2$ et $T_3$ et la division de son angle minimal $a$ en trois angles $a_1$, $a_2$ et $a_3$ tels que $a = a_1 + a_2 + a_3$. Cela produit des triangles de plus en plus allongés qui peuvent affecter la qualité globale de la triangulation.

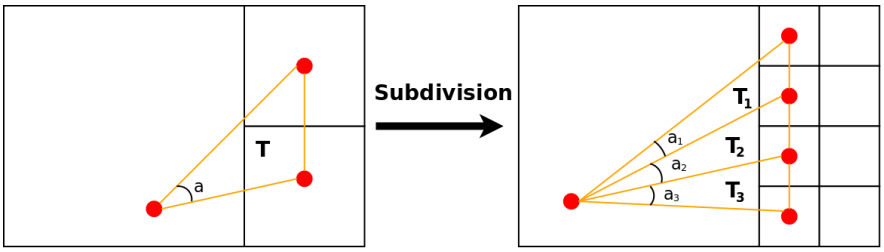



FIGURE 4.24 – Angles minimaux des triangles dans des régions de changement de résolution. Au fur et à mesure que la différence de profondeur entre les cellules adjacentes augmente, les angles minimaux des triangles connectant les deux régions devient plus petit et les triangles plus allongés. À gauche, un niveau de différence. À droite, deux niveaux de différence. Le triangle minimal $a$ est divisé en trois triangles $a_1$, $a_2$ et $a_3$ tels que $a = a_1 + a_2 + a_3$.

Cet inconvénient trouve son origine dans un manque d'équilibre entre les profondeurs des branches adjacentes dans l'octree. Elle peut être surmontée avec l'utilisation d'un algorithme de rééquilibrage des octrees. L'idée générale est d'imposer une différence de profondeur maximale entre des branches adjacentes. De tels algorithmes ont été copieusement proposés dans la littérature des structures de données. Dans cette thèse, nous allons nous contenter d'utiliser une de ces solutions afin de nous assurer que la différence de profondeur entre branches n'excède pas 1 niveau. Cette condition permet également de générer des surfaces avec des transitions progressives entre les régions à haute résolution et les régions à faible résolution.

## 4.4 Génération de la connectivité

Pour la génération de la connectivité, nous utilisons la méthode proposée par Tao Ju *et al.* [59]. Cet algorithme réalise un parcours récursif afin d'énumérer toutes les arêtes de l'octree. Pour cela, il utilise trois méthodes, CellProc, FaceProc et EdgeProc. CellProc permet de parcourir toutes les cellules, FaceProc parcourt toutes les facettes et EdgeProc toutes les arêtes.

L'algorithme commence avec un appel à CellProc avec la cellule racine de l'octree. La méthode CellProc fait huit appels récursifs avec ses sous-cellules, douze appels à FaceProc avec toutes les combinaisons possibles de sous-cellules qui partagent une facette et six appels à EdgeProc avec toutes les combinaisons de quatre cellules qui partagent une arête. Les appels à CellProc sont illustrés graphiquement sur la figure 4.25.

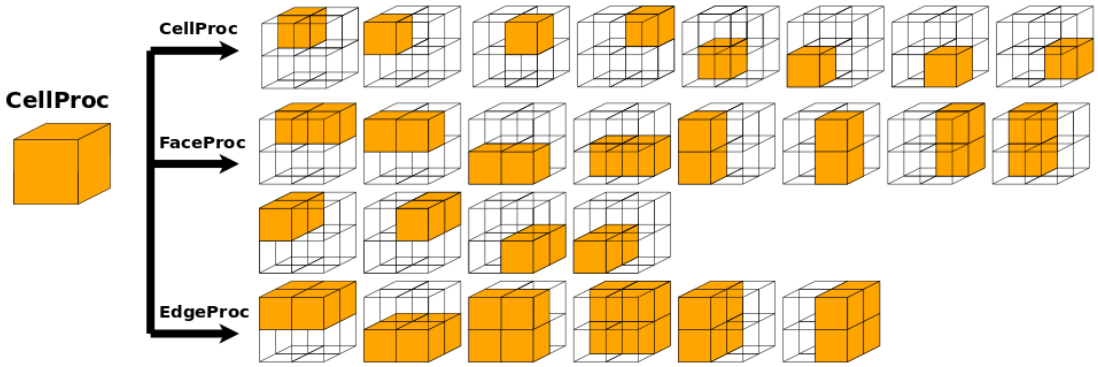



FIGURE 4.25 – Appels récursifs dans la méthode CellProc sur une cellule $C$. La méthode CellProc reçoit une cellule en argument et elle réalise huit appels récursives à CellProc pour chaque sous-cellule de $C$, douze appels à FaceProc pour chaque couple de sous-cellules qui partagent une facette et six appels à EdgeProc pour les différentes combinaisons de quatre cellules qui partagent une arête interne de $C$.

La méthode FaceProc reçoit en argument deux cellules $c_1$ et $c_2$ partageant une facette $f$. Pour commencer, FaceProc fait quatre appels récursifs à FaceProc avec une sous-cellule de $c_1$ et une sous-cellule de $c_2$ qui partagent une sous-facette de $f$. Pour finir, FaceProc fait quatre appels à EdgeProc avec deux sous-cellules de $c_1$ et deux sous-cellules de $c_2$ partageant une arête contenue dans la facette $f$. Ces appels sont présentés sur la figure 4.26.

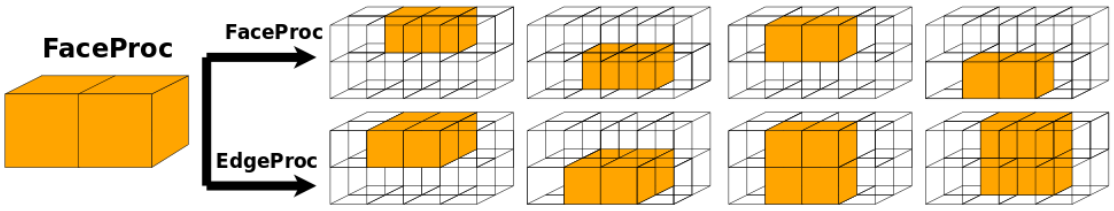



FIGURE 4.26 – FaceProc reçoit comme argument deux cellules partageant une facette $f$ . Alors, FaceProc réalise quatre appels récursifs à FaceProc avec les combinaisons de deux cellules qui partagent une sous-facette contenue dans $f$. Après, elle fait quatre appels à EdgeProc avec les combinaisons de quatre cellules qui partagent une arête contenue dans $f$.

La méthode EdgeProc reçoit en argument quatre cellules $c_1$,$c_2$, $c_3$ et $c_4$ partageant une arête $a$. Elle fait deux appels récursifs avec quatre sous-cellules qui partagent une demi arête contenue dans

$a$ comme illustré sur la figure 4.27.

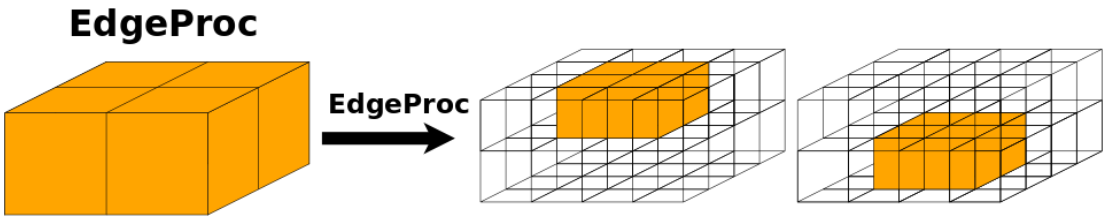



FIGURE 4.27 – La méthode EdgeProc reçoit comme arguments, quatre cellules partageant une arête $a$. Elle réalise deux appels récursifs avec les deux combinaisons des sous-cellules qui partagent une demi arête contenue dans l'arête $a$.

Ces trois méthodes permettent de parcourir toutes les arêtes de l'octree afin de pouvoir identifier les arêtes intersectées par la surface $\partial V$. Les explications précédentes s'appliquent pour un octree régulier mais, dans le cas d'un octree adaptatif, nous pouvons nous retrouver avec des cellules de dimensions différentes. Par exemple, nous pouvons nous trouver dans le contexte de la méthode EdgeProc avec quatre cellules $c_{1,}c_2, c_3$ et $c_4$ tel que $c_1$ et $c_2$ soient des cellules feuilles tandis que $c_3$ et $c_4$ sont divisées en sous-cellules $c_3^1, c_3^2, \ldots$ et $c_4^1, c_4^2, \ldots$ respectivement. Dans ce cas, nous réaliserons les deux appels récursifs à EdgeProc avec les sous-cellules de $c_3$ et $c_4$ et avec les cellules $c_1$ et $c_2$ en argument. La fonction EdgeProc$(c_1, c_2, c_3, c_4)$ réalisera deux appels récursifs EdgeProc$(c_1, c_2, c_3^i, c_4^i)$ afin de relier les sommets des cellules qui se trouvent à des niveaux différents de l'octree.

En plus d'être intersectée par la surface, une arête doit être minimale. Une *arête minimale* est une arête qui ne contient pas une autre arête. Des exemples de ces types d'arêtes sont donnés sur la figure 4.28. L'arête $e$ n'est pas minimale parce qu'elle contient deux autres arêtes, les arêtes $m_1$ et $m_2$ sont minimales.

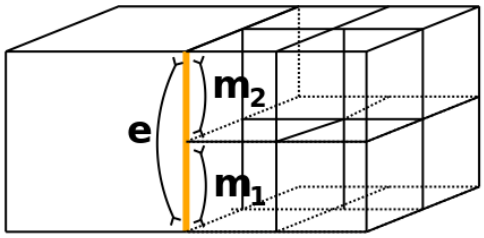



FIGURE 4.28 – Exemple d'une arête qui n'est pas minimale $e$ contenant deux arêtes minimales $m_1$ et $m_2$.

La méthode EdgeProc est en charge de détecter les arêtes minimales intersectées par la surface et de générer le polygone correspondant. Comme nous l'avons vu dans la section précédente, chacune de ces arêtes est liée à un sommet dual dans chacune des cellules qui lui sont adjacentes. Un polygone est alors construit en connectant les sommets duaux de chacune des cellules partageant l'arête. Dans le cas d'une zone de subdivision régulière, quatre cellules partagent l'arête, en conséquence, nous construisons un quadrangle (à diviser en deux triangles), dans le cas d'une zone à division adaptative, trois cellules partagent l'arête et nous construisons un triangle entre les sommets de ces cellules. La construction des polygones est illustrée sur la figure 4.29.

L'application de cet algorithme sur toutes les cellules feuilles de l'octree situées sur la surface produit une surface adaptative qui entoure tous les sommets de l'octree à l'intérieur de la surface $\partial V$.

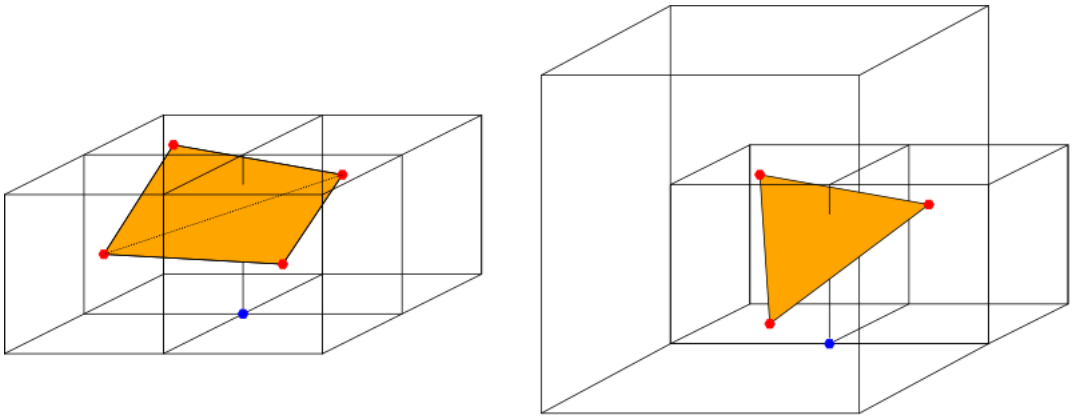


FIGURE 4.29 – Génération des polygones dans la méthode EdgeProc. L'arête centrale est coupée par la surface $\partial V$ et les sommets duaux sont à l'intérieur des cellules. À gauche, une zone divisée régulièrement va produire un quadrangle qui peut être divisé en deux triangles. À droite, une zone divisée de manière adaptative va produire un triangle.

## 4.5 Discussion

Dans ce chapitre, nous avons présenté un pipeline complet pour la génération des maillages adaptatifs à partir de données volumiques. Notre algorithme n'a besoin d'aucune information préalable ni de pré-traitement sur les données pour générer une surface 2-variété et fermée. La méthode présentée analyse les caractéristiques topologiques du volume dans le but d'assurer que la surface reflète la forme de l'objet. Notre critère géométrique permet de mieux adapter le nombre de triangles utilisés par rapport à la courbure locale de la surface $\partial V$. De plus, une nouvelle technique de localisation des sommets permet d'assurer qu'ils sont effectivement sur la surface afin d'améliorer l'approximation de l'objet à l'intérieur de chaque cellule. Enfin, un nouvel algorithme, combiné avec des critères topologiques sur l'octree, permet à l'utilisateur de raffiner le maillage dans les zones d'intérêt sans compromettre la topologie de la surface obtenue.

Dans la section suivante, nous présentons une évaluation générale de notre algorithme sur un ensemble de volumes. Nous faisons aussi une comparaison des caractéristiques de notre solution par rapport aux travaux de l'état de l'art.

# Chapitre 5

# Résultats

Dans cette section, nous présentons les résultats obtenus avec notre méthode sur un ensemble d'objets volumiques de référence. Nous comparons notre solution avec les méthodes les plus représentatives de l'état de l'art et nous mettons en évidence les avantages et les limitations de notre solution. Les tests d'utilisation de notre algorithme sont réalisés sur des objets volumiques construits à partir d'empilements d'images. Par contre, les objets volumiques utilisés dans les tests comparatifs sont générés à partir de la discrétisation de modèles polygonaux de référence. Cette discrétisation nous permet de choisir la résolution du modèle et consiste à construire une fonction indicatrice $F : \mathbb{Z}^3 \rightarrow (0, 1)$ définie sur une grille régulière telle que $F(x, y, z) = 1$ si $(x, y, z)$ est à l'intérieur de la surface et 0 sinon. Concrètement, la discrétisation produit un volume avec des voxels qui peuvent avoir une valeur de 1 s'ils sont à l'intérieur de la surface et 0 sinon comme illustré sur la figure 5.1.

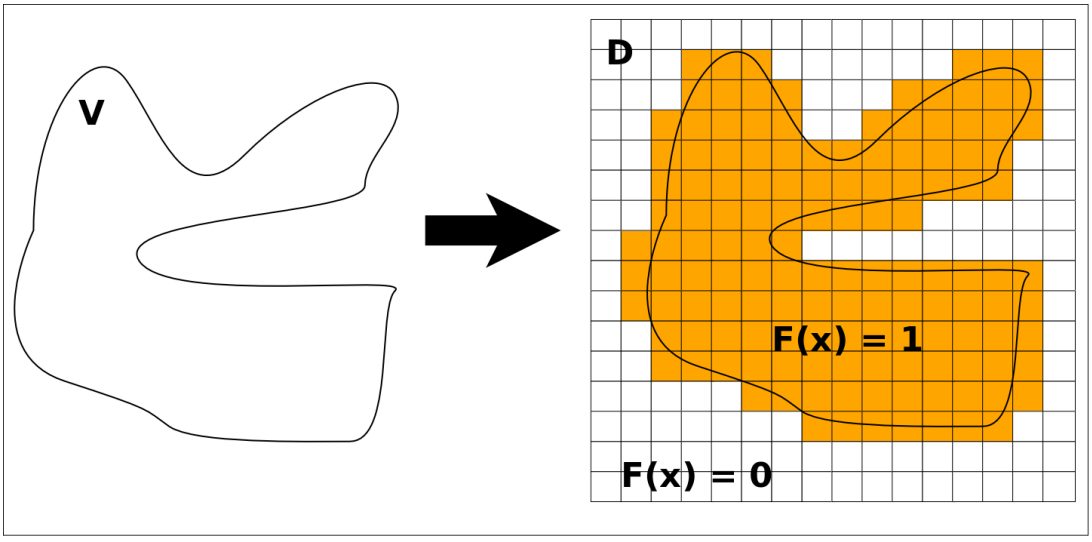



FIGURE 5.1 – Discrétisation $2D$ d'une courbe afin de produire une représentation volumique basée sur une fonction indicatrice qui sépare les voxels qui sont à l'intérieur et ceux à l'extérieur de la courbe.

Afin de faire une évaluation complète de notre solution, nous avons identifié quatre aspects pertinents : 1) la qualité de l'approximation obtenue, 2) la forme et la quantité d'éléments géométriques

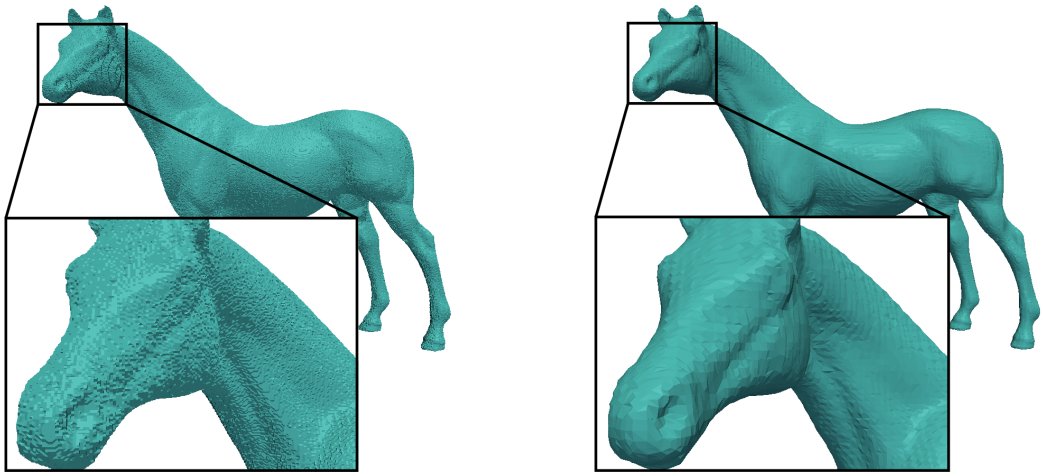
générés, 3) la mesure des temps d'exécution et 4) la mémoire nécessaire par rapport aux données d'entrée et à la taille de la surface obtenue.

## 5.1 Qualité de l'approximation

Il existe plusieurs manières de mesurer la qualité de l'approximation d'une surface à partir d'un objet volumique. Les critères liés à la perception ont une caractéristique plus subjective et permettent d'estimer la qualité visuelle de l'approximation par rapport à l'observateur. Par contre, les critères géométriques sont basés sur des mesures objectives de la distance entre la surface obtenue et un modèle de référence. Dans cette évaluation, nous considérons comme modèle de référence la surface polygonale à partir de laquelle nous avons construit le volume.

### 5.1.1 Mesures de qualité visuelle et perception

Ces critères mesurent principalement l'existence d'artefacts liés à la discrétisation ou la perte de détails pertinents sur la surface reconstruite. La figure 5.2a, présente la surface générée par l'algorithme Marching Cubes sur le volume "Horse" de $512^3$ voxels ainsi que le résultat de notre algorithme sur le même volume avec un octree adaptatif de profondeur maximale 7 (voir la figure 5.2b).



(a) Marching Cubes à résolution maximale.

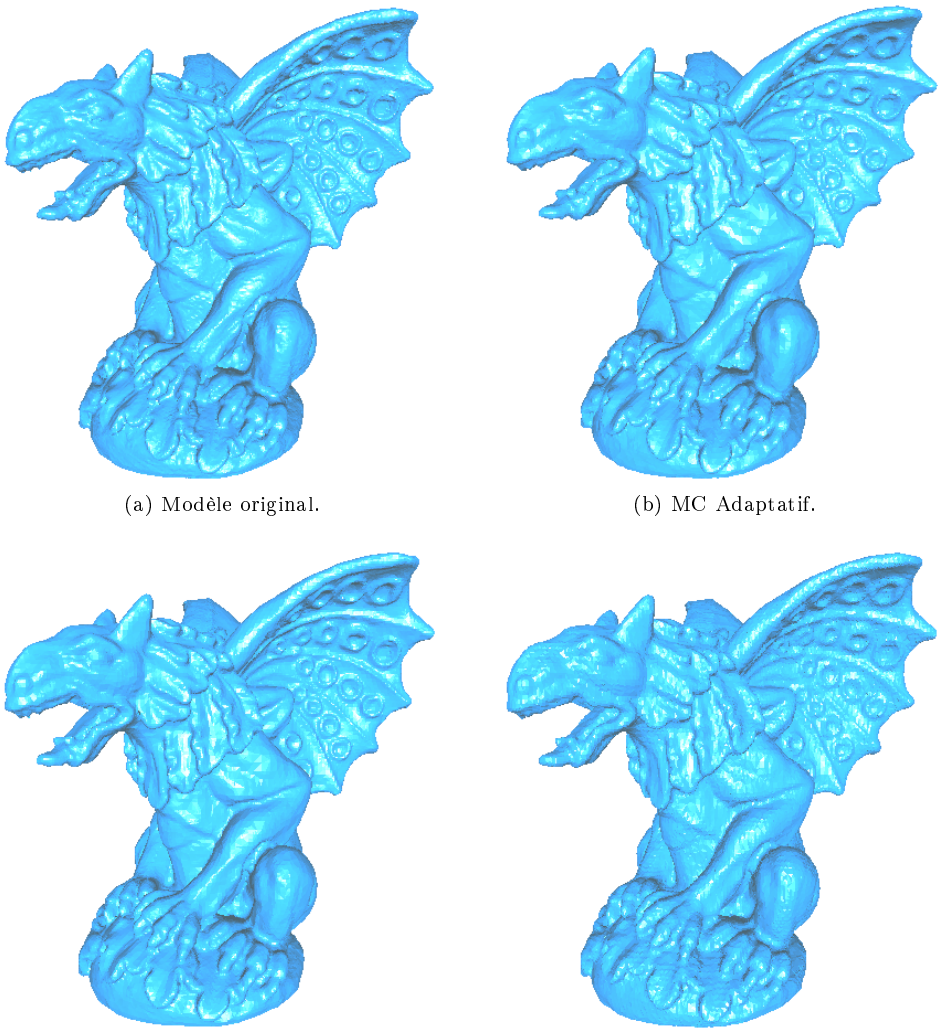
(b) Notre algorithme avec un octree de profondeur maximal 7.

FIGURE 5.2 – Comparaison de l'algorithme Marching Cubes et de notre algorithme sur le volume "Horse" de $512^3$ voxels.

Selon la figure 5.2, il est clair que l'algorithme MC produit une surface bruitée avec des effets de crénelage très marqués. Cela est dû principalement au fait que MC utilise une grille régulière pour diviser le volume et que les sommets de la surface sont placés en utilisant un interpolateur linéaire sur les arêtes des cellules. Par contre, notre solution produit une surface beaucoup plus lisse et qui contient moins de triangles, ce qui améliore fortement sa qualité visuelle.

La comparaison avec des méthodes d'extraction de surfaces adaptatives telles que le Marching Cubes adaptatif (AMC) de Kazhdan *et al.* ou la méthode duale adaptative de Schaefer et Warren

(ADMC) montre des surfaces de qualité visuelle équivalente à la notre et qui ressemblent beaucoup plus au modèle original. La figure 5.3 présente le modèle original et les maillages résultant de l'application d'AMC, d'ADMC et de notre solution sur un modèle "Gargoyle" de $512^3$ voxels avec des octrees de profondeur minimale 3 et maximale 8.



(a) Modèle original. (b) MC Adaptatif.

(c) MC Dual. (d) Notre algorithme.

FIGURE 5.3 – Comparaison visuelle entre le modèle original et les reconstructions faites par AMC, MCD et notre algorithme sur un volume "Gargoyle" de $512^3$ voxels.

Comme les trois méthodes comparées sur la figure 5.3 utilisent des divisions hiérarchiques adaptatives, les surfaces obtenues ne présentent pas de différences fondamentales au niveau visuel. Cependant, AMC et ADMC génèrent des surfaces légèrement plus lisses que le modèle original et que la surface produite par notre algorithme. Cela est dû au fait que notre méthode tient compte des petites irrégularités de la surface à l'intérieur des cellules alors que AMC et ADMC approximent la surface à l'intérieur de la cellule avec des plans.

### 5.1.2 Mesures d'approximation géométrique

Les critères d'approximation géométrique mesurent le niveau d'erreur géométrique introduit par la reconstruction par rapport à la surface originale. Ils consistent en un calcul de distance euclidienne entre la surface produite et une surface de référence. Ici, la surface de référence est le modèle qui a servi à générer le volume discret. Les volumes utilisés dans ces tests ont $512^3$ voxels et ils sont présentés dans la table 5.1 avec le nombre de triangles générés pour chacun des algorithmes : Marching Cubes (MC), MC adaptatif proposé par Kahzdan *et al.* (AMC), Dual Marching Cubes Dual de Schaefer et Warren (ADMC) et notre algorithme avec un octree de profondeur maximale 7.

| | **# total de triangles** | | | |
|---|---|---|---|---|
| **Méthode** / **Modèle** | MC | AMC | ADMC | Notre algorithme |
| Cow | 131988 | 54250 | 63314 | 64024 |
| VaseLion | 189732 | 176228 | 195722 | 127606 |
| Horse | 113416 | 32776 | 37782 | 47576 |
| Bunny | 138552 | 57534 | 67662 | 61182 |
| Armadillo | 107892 | 137098 | 156820 | 81056 |
| Dragon | 183916 | 107342 | 122074 | 105624 |
| Buddha | 230656 | 119124 | 134342 | 138424 |
| Gargoyle | 172460 | 224854 | 256020 | 123013 |
| WalesDragon | 208880 | 220008 | 245508 | 152630 |
| AsianDragon | 104256 | 107284 | 118760 | 83204 |

Tableau 5.1 – Nombre de triangles de la surface générée à partir de modèles volumiques de $512^3$ voxels pour différents objets. Nous avons comparé notre algorithme avec "Marching Cubes" (MC), Marching cubes adaptatif (AMC) de Kazhdan *et al.* et MC dual adaptatif de Schaefer et Warren (ADMC).

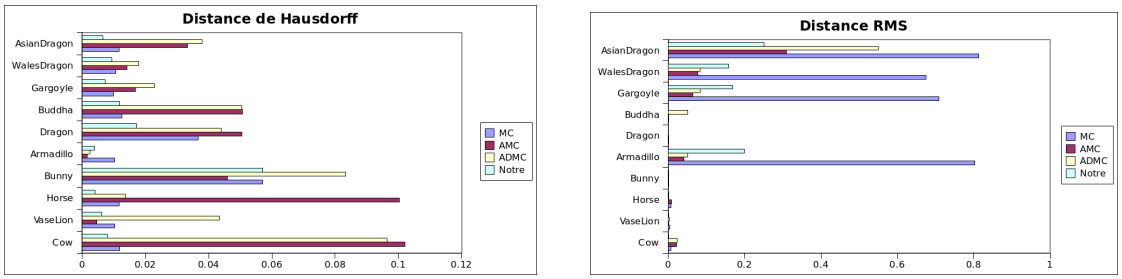
Nous avons appliqué notre algorithme sur les volumes et nous comparons la surface obtenue avec la surface originale en utilisant la librairie de calcul des distances géométriques Metro [33]. Cette dernière permet de calculer les distances de Hausdorff, maximale, moyenne et "Root-Mean-Square" (RMS) entre deux maillages. Ces distances sont estimées en utilisant un échantillonnage sur les facettes et sur les arêtes des maillages afin d'améliorer leur précision et leur robustesse au bruit. La table 5.2 contient les mesures des distances de Hausdorff et RMS entre différents modèles générés avec les algorithmes listés dans la table 5.1.

Les graphiques de la figure 5.4 confirment que, même si nous n'obtenons pas toujours la meilleure approximation en terme de distance de Hausdorff et RMS, l'erreur d'approximation de notre algorithme reste sous la moyenne des erreurs géométriques introduites par les algorithmes de l'état de l'art.

Les données de la table 5.2 confirment que l'obtention d'une bonne approximation est le résultat d'un grand nombre de facteurs. Les deux facteurs que nous avons identifiés comme les plus déterminants sont la division spatiale et la technique utilisée pour localiser les sommets du maillage. Dans les trois méthodes que nous avons testées, les cellules utilisées pour diviser le domaine sont homéomorphes à un cube. Dans le cas d'AMC et de notre méthode, les cellules sont alignées sur les trois axes principaux. Néanmoins, comme AMC est limité à localiser les sommets de la surface sur les arêtes de la cellule, il ne tient pas compte de la géométrie de la surface à l'intérieur de la cellule entraînant des erreurs d'approximation. Par contre, notre méthode place les sommets à l'intérieur

| | Distance RMS | | | | Distance de Hausdorff | | | |
|---|---|---|---|---|---|---|---|---|
| Méthode / Modèle | MC | AMC | ADMC | Notre | MC | AMC | ADMC | Notre |
| Cow | .0119 | .1020 | .0964 | **.0080** | .0068 | .0226 | .0231 | **.0019** |
| VaseLion | .0102 | **.0046** | .0435 | .0063 | .0048 | .0018 | .0022 | **.0016** |
| Horse | .0117 | .1004 | .0136 | **.0041** | .0072 | .0087 | **.0013** | .0017 |
| Bunny | .0571 | **.0458** | .0832 | .0571 | .0014 | **.0007** | .00105 | .0012 |
| Armadillo | .01032 | **.0017** | .0024 | .0039 | .8022 | **.0416** | .0506 | .1993 |
| Dragon | .0367 | .0506 | .0439 | **.0171** | .0009 | .0007 | .0005 | **.0002** |
| Buddha | .0126 | .0507 | .0505 | **.0118** | .0008 | .0007 | .0507 | **.0002** |
| Gargoyle | .0100 | .0169 | .0228 | **.0072** | .7083 | **.0643** | .0838 | .1689 |
| Wales Dragon | .0106 | .0141 | .0178 | **.0094** | .6756 | **.0783** | .0831 | .1585 |
| Asian Dragon | .0117 | .0333 | .0379 | **.0065** | .8130 | .3091 | .5509 | **.2504** |

Tableau 5.2 – Mesures des distances RMS et Hausdorff pour les algorithmes “Marching Cubes” (MC), MC adaptatif (AMC) de Kazhdan *et al.*, ADMC de Schaefer et Warren et notre algorithme. Les mesures ont été réalisées par rapport aux surfaces originales.




(a) (b)

FIGURE 5.4 – Erreurs géométriques (Hausdorff et RMS) pour les 10 modèles utilisés dans le test. Même si notre algorithme n'obtient pas toujours la meilleure approximation, il reste très compétitif par rapport aux autres méthodes testées.

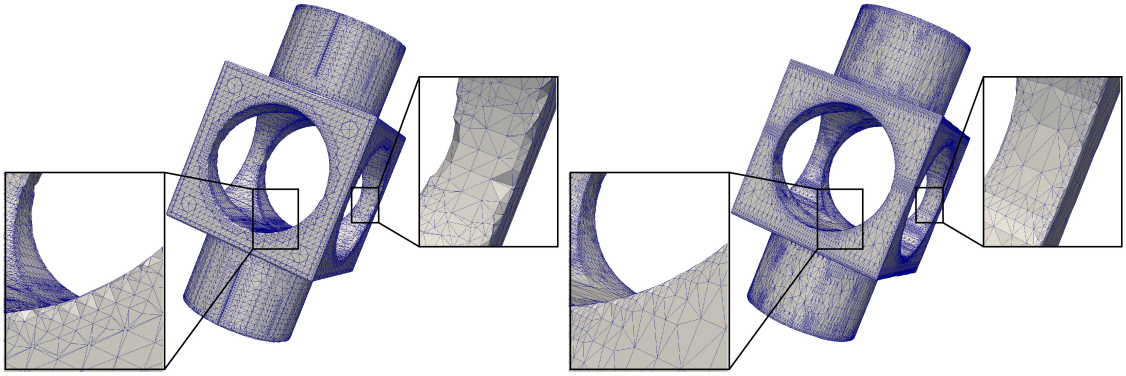
de la cellule, ce qui nous offre une plus grande liberté pour améliorer leur localisation. ADMC construit une grille duale alignée avec les caractéristiques du volume mais utilise la triangulation de MC pour générer la surface.

Cela explique en bonne partie la raison pour laquelle, à un même niveau de subdivision, les erreurs qui peuvent être corrigées avec la localisation des sommets à l'intérieur des cellules et sur la surface peuvent être compensées négativement par la création de morceaux de triangles qui ne tiennent pas compte de la géométrie de la surface. Ce problème affecte toutes les méthodes de division spatiale et n'est pas exclusive à notre méthode. Il faut toujours trouver un équilibre entre la profondeur maximale de la subdivision (qui détermine la résolution), la forme des cellules utilisées pour approximer la surface au niveau local et les stratégies de localisation des sommets du maillage. En conclusion, notre méthode offre un bon compromis entre l'approximation géométrique de la surface et l'aspect visuel.

## 5.2 Qualité des éléments géométriques

Un autre facteur important dans toute méthode de génération de surface est la qualité des éléments géométriques construits. Ici, nous nous concentrerons sur la qualité des triangles par rapport aux

critères géométriques classiques tel que ses angles minimal et maximal. Ces propriétés sont particulièrement pertinentes parce qu'elles peuvent affecter fortement la performance des algorithmes de remaillage ou d'amélioration de surface. De plus, des angles proches de $0^o$ altèrent la performance et la convergence des algorithmes de simulation numérique. Des angles proches de $180^o$ nuisent à la précision de l'approximation obtenue.



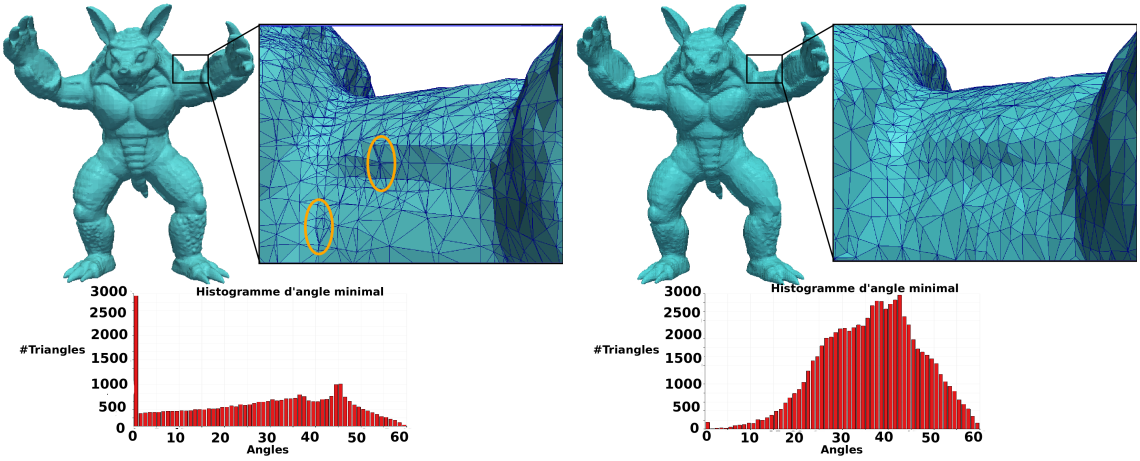
(a) MC Dual Adaptatif. (b) Notre algorithme.

FIGURE 5.5 – Comparaison d'ADMC et de notre algorithme sur l'objet volumique "Block" de $512 \times 512 \times 256$ voxels. L'utilisation d'une grille duale ne garantit pas l'élimination des artefacts, spécialement sur les arêtes vives.

L'une des stratégies les plus utilisées afin d'améliorer l'approximation géométrique est de créer des grilles duales alignées avec les caractéristiques et les arêtes vives du volume. Néanmoins, même si ces stratégies clament pouvoir toujours s'adapter à la forme du volume, cela n'est pas toujours le cas. Sur la figure 5.5, nous comparons les surfaces produites par l'algorithme ADMC et notre algorithme sur un volume "Block" de $512 \times 512 \times 256$ voxels. Il apparaît que, même si ADMC arrive à bien reproduire la géométrie générale du volume, il génère des artefacts clairement visibles sur les arêtes vives. Ceux-ci sont principalement dûs au fait que les sommets du maillage doivent être localisés sur les arêtes des cellules. Ainsi, malgré la possibilité d'aligner les arêtes des cellules sur la géométrie du volume, la liberté dans la localisation des sommets demeure une contrainte majeure.

De la même manière, la génération d'une grille duale peut améliorer l'approximation mais affecte la qualité des triangles obtenus. Sur la figure 5.6, nous montrons une surface extraite du volume "Armadillo" de $512^3$ voxels avec ADMC et notre algorithme. Les surfaces sont très similaires visuellement mais un zoom permet de mieux observer les différences entre les triangulations générées. La surface produite avec ADMC contient beaucoup plus de triangles dégénérés (encerclés) que celle extraite avec notre algorithme. Les histogrammes d'angle minimal affichés permettent de confirmer l'existence d'une grande quantité de triangles dégénérés sur la surface (a) et montrent une distribution plus intéressante des angles sur notre surface (b). Ces différences s'expliquent parce que ADMC construit une grille hexaédrique adaptative qui peut contenir des cellules inversées ou très aplaties. De plus, l'utilisation de la triangulation de MC à l'intérieur des cellules peut conduire à l'apparition de triangles allongés.

Sur la figure 5.7 nous comparons la triangulation obtenue par l'algorithme "Dual Marching Tetrahedra" (DMT) proposé par Manson et Schaefer avec notre solution. Nous avons appliqué ces deux algorithmes sur un objet "Block" de $512^3$ voxels. La figure montre que notre algorithme obtient une meilleure distribution des sommets du maillage par rapport à la géométrie de la surface.

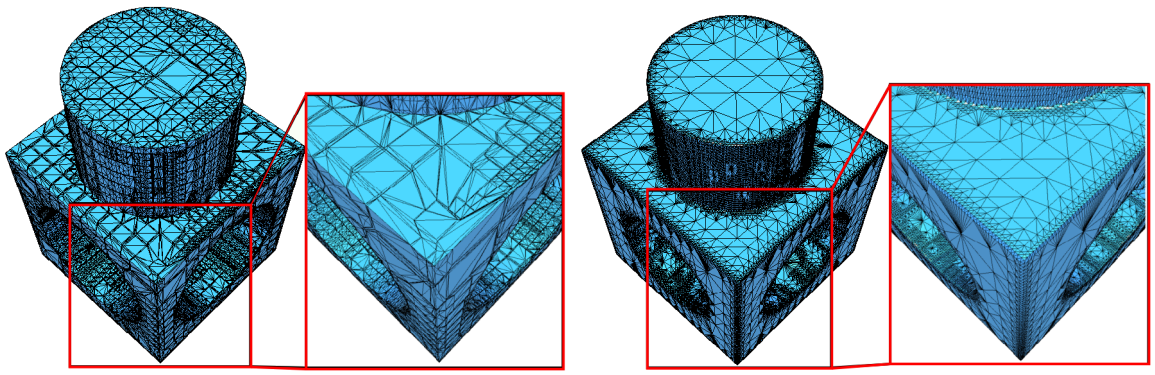



(a) MC Adaptatif Dual. (b) Notre algorithme.

FIGURE 5.6 – Surfaces extraites à partir d'un volume "Armadillo" de $512^3$ voxels. Les histogrammes permettent de comparer la qualité des triangulations générées.

En général, nous pouvons observer que dans un algorithme qui localise les sommets sur les arêtes des cellules, la subdivision spatiale utilisée devient un facteur fondamental dans la distribution finale des points sur la surface. La faiblesse de l'algorithme AMC est qu'il utilise la même triangulation que MC et que la méthode utilisée pour fermer les polylignes sur les régions de changement de résolution produit très souvent des triangles allongés alignés sur les arêtes des cellules. Cet aspect est illustré sur la figure 3.9. L'inconvénient des algorithmes duaux ADMC et DMT est qu'ils essaient d'améliorer la qualité de l'approximation en générant des grilles (hexaédrique pour ADMC et tétraédrique pour DMT) duales des octrees utilisés pour diviser le domaine. Dans le cas d'ADMC, la limitation principale est l'utilisation de la triangulation de MC. Concernant DMT, l'utilisation de la look-up-table de MT, qui place les sommets sur les arêtes de tétraèdres, génère des triangles qui sont bien alignés sur les caractéristiques de la surface mais cette méthode ne garantit pas la qualité de ces triangles.

En revanche, placer les sommets à l'intérieur des cellules donne plus de liberté pour améliorer la qualité de triangles générés. Dans notre cas, l'algorithme de localisation des sommets basé sur les



(a) Marching Tetrahedra Dual. (b) Notre algorithme.

FIGURE 5.7 – Comparaison de la qualité de triangles produits pour l'algorithme de "Dual Marching Tetrahedra" proposé par Manson et Schaefer.

composantes connexes du volume à l'intérieur de la surface aura tendance à toujours placer les sommets duaux près du *noyau étoilé* de la composante correspondante. Afin de confirmer cette propriété, nous avons comparé les taux de triangles dégénérés sur les surfaces produites par les algorithmes MC, AMC, ADMC et le nôtre. Les résultats de cette expérience sont présentés dans le tableau 5.3.

| | **Pourcentage de triangles dégénérés** | | | |
|---|---|---|---|---|
| **Méthode** / **Modèle** | **MC** | **AMC** | **ADMC** | **Notre algorithme** |
| Cow | 31.90 | 2.08 | 1.99 | **0.15** |
| VaseLion | 34.13 | 2.09 | 1.88 | **0.18** |
| Horse | 34.44 | 2.38 | 11.86 | **0.06** |
| Bunny | 33.68 | 2.21 | 12.90 | **0.06** |
| Armadillo | 35.76 | 2.31 | 11.01 | **0.05** |
| Dragon | 34.39 | 2.21 | 10.76 | **0.05** |
| Buddha | 28.92 | 2.02 | 9.34 | **0.08** |
| Gargoyle | 35.50 | 2.31 | 10.64 | **0.07** |
| WalesDragon | 32.38 | 2.13 | 9.24 | **0.12** |
| AsianDragon | 34.42 | 2.20 | 8.64 | **0.19** |

Tableau 5.3 – Comparaison du taux de triangles dégénérés produits par les méthodes "Marching Cubes" (MC), le MC adaptatif (AMC) de Kazhdan *et al.*, le MC dual de Schaefer et Warren (ADMC) et notre algorithme (CCDMC).

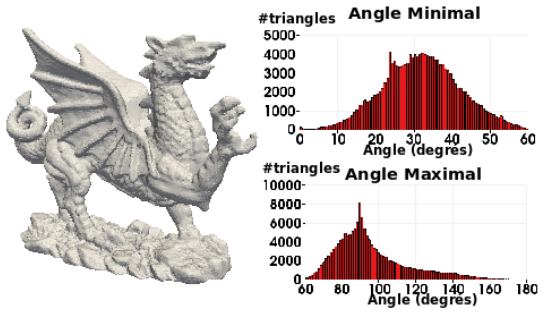
Le tableau 5.3 montre le pourcentage de triangles contenant un angle minimal de moins de 2°. Ces chiffres confirment que, même si notre algorithme génère encore quelques triangles allongés, ils sont très peu nombreux par rapport au nombre de triangles dégénérés produits par les algorithmes similaires de l'état de l'art.

L'existence de triangles dégénérés n'est pas la seule mesure importante. Une bonne distribution générale des angles à l'intérieur des triangles est également fondamentale. La maximisation de l'angle minimal permet d'améliorer la convergence des algorithmes de remaillage ou de simplification topologique. Les algorithmes de simulation numérique peuvent utiliser une surface afin de produire des maillages tétraédriques et la qualité de leurs résultats et très dépendante de la qualité géométrique de la surface fournie. La figure 5.8 donne un exemple de reconstruction d'un modèle "Wales Dragon" à partir d'un volume de $512^3$ voxels ainsi que les histogrammes des angles minimaux et maximaux pour les quatre méthodes testées.

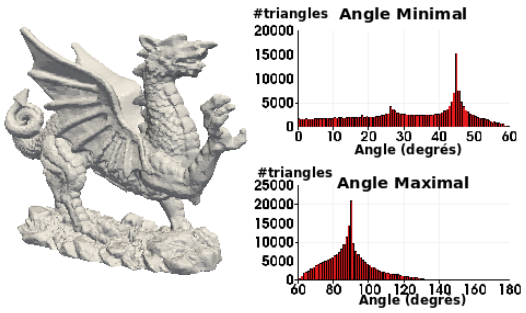
Comme l'illustre la figure 5.8, même si visuellement les surfaces obtenues par les quatre algorithmes sont très similaires, les histogrammes des angles minimaux et maximaux montrent que notre méthode produit une distribution beaucoup plus équilibrée autour des valeurs standard de 35° pour l'angle minimal et 90° pour l'angle maximal. Les algorithmes d'AMC et ADMC ont un pic d'angle minimal vers 45° mais, contrairement à notre méthode, ils ont aussi une grande proportion d'angles qui se trouvent dans des valeurs inférieurs à 20°. Nos expériences sur une grande quantité de modèles et volumes variés confirment ces résultats.

## 5.3 Temps d'exécution et utilisation de la mémoire

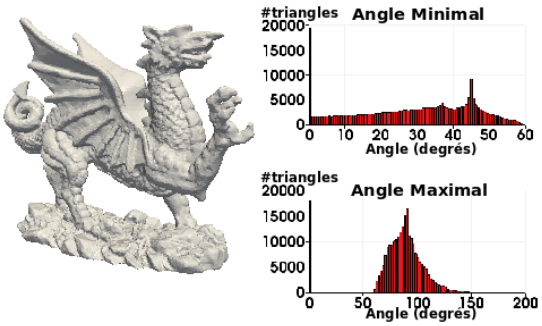
Cette section est divisée en deux parties : dans un premier temps, les caractéristiques de notre algorithme en termes de temps d'exécution et d'utilisation de la mémoire sont étudiées sur un



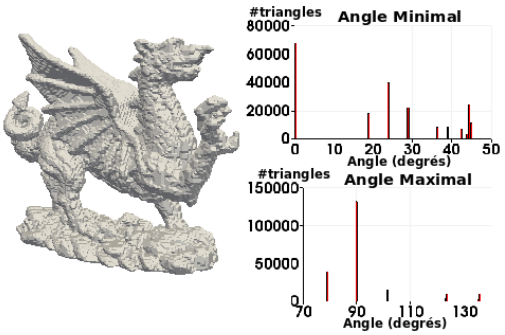

(a) Notre algorithme.




(b) Marching Cubes adaptatif.




(c) Marching Cubes Dual Adaptatif.




(d) Marching Cubes.

FIGURE 5.8 – Comparaison des histogrammes des angles maximal et minimal pour les surfaces reconstruites à partir du Volume "WalesDragon" de $512 \times 512 \times 256$ voxels.

exemple concret. Ensuite, une étude comparative de ses performances par rapport aux algorithmes de la littérature est proposée.

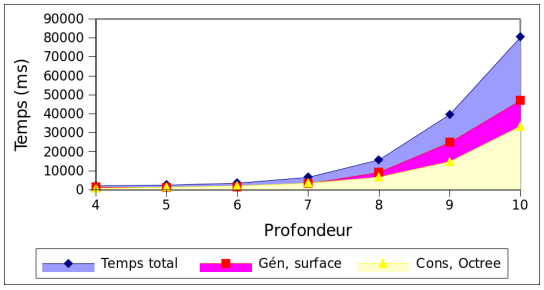
### 5.3.1 Étude de cas (volume "Asian Dragon")

Nous avons étudié en détail les temps d'exécution et l'encombrement mémoire nécessaire au traitement d'un volume "Asian Dragon" de $1024^3$ voxels. Nous avons réalisé ces tests sur un ordinateur conventionnel possédant huit cœurs de 2.6 GHz chacun et 8 Go de mémoire principale. Le tableau 5.4 résume les temps d'exécution pour une profondeur maximale variant entre 4 et 10. Les temps d'exécution sont répartis entre les deux parties principales de notre algorithme : la construction de l'octree et l'extraction de la surface.

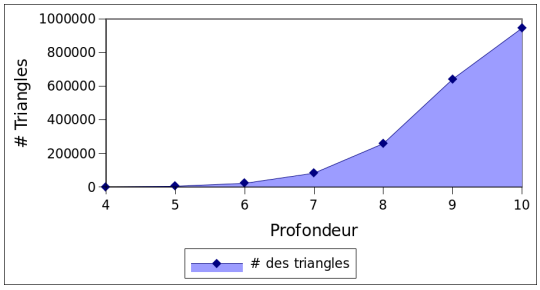
Dans le tableau 5.4, le temps d'exécution indiqué dans la colonne "Octree" contient les temps de construction des cellules de l'octree ainsi que l'application de nos critères topologiques et géométriques assurant que l'octree est correct. Le temps indiqué dans la colonne "Surface" inclut la génération des sommets duaux, la génération de la connectivité de la surface et la localisation des sommets. Les deux dernières colonnes présentent le nombre total de triangles de la surface et le nombre de triangles générés par seconde. Sur la figure 5.9, nous pouvons observer graphiquement la proportion du temps dédié à chaque étape de l'algorithme et le nombre de triangles générés en fonction du niveau de profondeur.

| Profondeur | Temps d'exécution (ms) | | | # triangles | |
|---|---|---|---|---|---|
| | Octree | Surface | Total | Total | Par seconde |
| 4 ($16^3$ voxels) | 834 | 1231 | 2065 | 1232 | 600 |
| 5 ($32^3$ voxels) | 1300 | 1149 | 2449 | 5662 | 2310 |
| 6 ($64^3$ voxels) | 2026 | 1413 | 3439 | 22816 | 6630 |
| 7 ($128^3$ voxels) | 3419 | 3110 | 6529 | 83204 | 12740 |
| 8 ($256^3$ voxels) | 6634 | 9090 | 15724 | 259011 | 16470 |
| 9 ($512^3$ voxels) | 14788 | 24808 | 39596 | 642168 | 16220 |
| 10 ($1024^3$ voxels) | 33425 | 46987 | 80412 | 945306 | 11760 |

Tableau 5.4 – Temps d'exécution de notre algorithme sur le volume "Asian Dragon" de $1024^3$ voxels (exprimés en millisecondes) pour la construction de l'octree et l'extraction de la surface. Nombre total de triangles générés et quantité de triangles générés par seconde.




(a) Proportion du temps d'exécution par étape.




(b) Nombre de triangles générés par niveau de profondeur.

FIGURE 5.9 – Graphique d'aires pour les temps d'exécution de notre algorithme sur le volume "Asian Dragon" ($512^3$ voxels) par rapport à la profondeur maximale de l'octree. (a) Le temps total est illustré avec l'aire sous la ligne bleue, le temps de génération de la surface est l'aire sous la ligne rouge et le temps de construction de l'octree est constitué par l'aire sous la ligne jaune. Le temps total est égal à l'addition de l'aire sous les lignes rouge et jaune. (b) Évolution du nombre de triangles par rapport à la profondeur de l'octree.

La figure 5.9a illustre la proportion croissante de l'étape de génération de la surface (aire sous la ligne de carrés) par rapport à l'étape de construction de l'octree (aire sous la ligne de triangles) dans le temps total (aire sous la ligne de losanges). Ceci est dû à la nature de notre algorithme de localisation des sommets qui parcourt les composantes connexes à l'intérieur de chaque cellule. Comme nous parcourons les voxels contenus dans les facettes et arêtes, plus on divise le volume, plus le recouvrement entre les voxels sera important et leur parcours plus long, du fait qu'ils sont partagés entre cellules adjacentes. Le temps total est composé de l'addition d'aires sous les courbes de carrés et de triangles. Par rapport à la quantité de triangles, la figure 5.9b confirme l'idée intuitive que l'utilisation d'une structure hiérarchique va fortement augmenter la quantité de triangles générés de manière presque exponentielle à chaque nouveau niveau de subdivision.

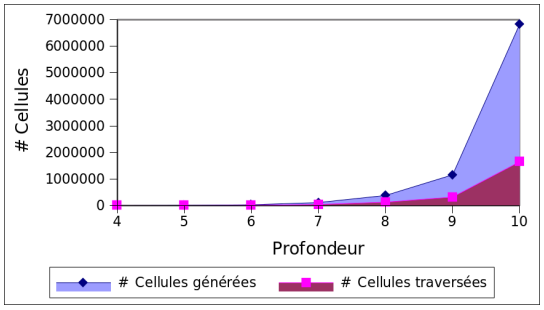
Concernant l'utilisation de la mémoire pour le même exemple, le tableau 5.5 présente le nombre de cellules de l'octree comparé aux cellules situées sur la surface. Il indique également la quantité de mémoire (en Moctets) occupée par l'octree et par les sommets.

Même si le tableau 5.5 confirme la tendance exponentielle de la mémoire nécessaire à chaque niveau de profondeur, la complexité topologique et géométrique du volume à approximer demeure

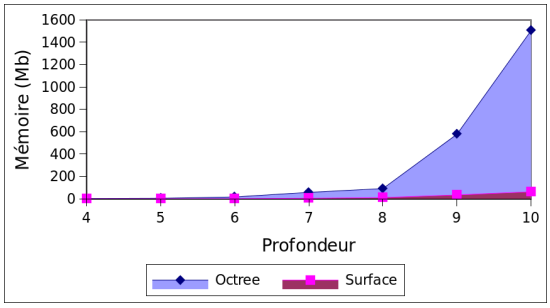
| Critères / Profondeur | # Cellules dans l'octree | | Mémoire (Moctets) | | |
|---|---|---|---|---|---|
| | Total | Sur $\partial V$ | Octree | Surface | Total |
| 4 | 1841 | 1611 | 2.4 | 0.4 | 2.8 |
| 5 | 7441 | 2796 | 5.2 | 0.8 | 6 |
| 6 | 30617 | 11333 | 16.7 | 1.3 | 18 |
| 7 | 113625 | 41475 | 57.7 | 5.1 | 62.8 |
| 8 | 380337 | 129265 | 91.5 | 11.1 | 102.6 |
| 9 | 1156577 | 320735 | 577.8 | 34.7 | 612.5 |
| 10 | 6831577 | 1648258 | 1504 | 63 | 1567 |

Tableau 5.5 – Nombre total de cellules dans l'octree par rapport aux cellules intersectées par la surface $\partial V$ et quantité de mémoire utilisée par l'octree et la surface (en Moctets).

le facteur le plus important afin d'estimer la quantité de cellules nécessaires et, en conséquence, la mémoire à utiliser. Une manière de réduire l'espace mémoire nécessaire consiste à ne pas garder toutes les cellules de l'octree en mémoire mais seulement celles qui sont situées sur la surface $\partial V$. La figure 5.10a illustre la proportion de ces cellules par rapport au nombre total de cellules et la figure 5.10b l'espace mémoire utilisé pour stocker la surface comparé à la mémoire totale pour l'octree.




(a) Graphique d'aires de cellules générées (en bleu) par rapport aux cellules traversées par la surface (en rouge).




(b) Quantité de mémoire utilisé par l'octree (en bleu) et par la surface générée (en rouge) par rapport au niveau du profondeur de l'octree.

FIGURE 5.10 – Ces graphiques d'aires illustrent la relation étroite entre le nombre de cellules construites dans l'octree par rapport à la profondeur et à la mémoire occupée. (a) La plus grande partie des cellules construites ne sont pas traversées et peuvent donc être potentiellement éliminées. (b) L'octree utilise la plus grande partie de la mémoire utilisée par notre algorithme.

La figure 5.10a confirme que l'utilisation d'un octree augmente très rapidement le nombre de cellules construites avec chaque nouveau niveau de subdivision. Néanmoins, elle montre aussi que la plus grande partie des cellules construites ne sont pas traversées par la surface et peuvent, par conséquent, être éliminées afin de limiter la quantité de mémoire nécessaire pour stocker l'octree.

### Analyse comparative

Dans cette section, nous comparons les performances en termes de mémoire et de temps d'exécution de notre algorithme par rapport aux autres algorithmes de l'état de l'art. Les méthodes que nous utilisons sont "Marching Cubes" (MC), "Adaptative Marching Cubes" (AMC) de Kazhdan *et al.* et "Dual Marching Cubes : Primal contouring of Dual Grids" (ADMC) de Schaefer et Warren.

L'une des caractéristiques de notre algorithme est qu'il n'a pas besoin de pré-traitements sur les

données et peut être appliqué directement sur des modèles volumiques binaires ou à niveaux de gris. Néanmoins, ce choix a des conséquences sur le temps d'exécution. Comme notre approche est fortement basée sur les composantes connexes 2D pour la topologie et 3D pour la localisation des sommets, le temps d'exécution de notre algorithme dépend forcément de la taille du volume à traiter et de la complexité géométrique et topologique du modèle.

| **Volume** | | **Temps d'exécution (millisecondes)** | | | | **Mémoire (Moctets)** | | | |
|---|---|---|---|---|---|---|---|---|---|
| Modèle | Voxels $^3$ | MC | AMC | ADMC | Le notre | MC | AMC | ADMC | Le notre |
| Buddha | **128** | 773 | 902 | **749** | 3774 | **112.1** | 240 | 242 | 176 |
| | **256** | 3788 | 3607 | **3072** | 12087 | **226.2** | 277 | 278 | 377 |
| | **512** | 16790 | 24734 | **3952** | 30924 | 736.9 | **334** | 335 | 920 |
| Dragon | **128** | 660 | 941 | **564** | 2871 | **65.7** | 194 | 193 | 105 |
| | **256** | 3048 | 2989 | **1870** | 8202 | **158.5** | 240 | 230 | 238 |
| | **512** | 14542 | 16197 | **3540** | 20507 | 586.4 | **274** | 275 | 568 |
| Wales Dragon | **128** | **746** | 1798 | 1853 | 3962 | **116.5** | 472 | 480 | 196 |
| | **256** | **3381** | 10183 | 6866 | 14562 | **221.1** | 532 | 535 | 440 |
| | **512** | 15921 | 70532 | **15003** | 44887 | 709 | **690** | 695 | 1150 |
| Horse | **128** | **135** | 307 | 228 | 1505 | **39.3** | 56 | 54.5 | 56 |
| | **256** | **657** | 685 | 392 | 3615 | 102.3 | **63** | 65 | 104 |
| | **512** | 3686 | 2348 | **867** | 8190 | 410 | 86 | **67** | 290 |
| Gargoyle | **128** | **617** | 2599 | 1739 | 3548 | **96.3** | 383 | 384 | 167 |
| | **256** | **2731** | 19799 | 4445 | 12996 | **182.3** | 460 | 462 | 412 |
| | **512** | **13743** | 179927 | 28600 | 43966 | **596.1** | 651 | 651 | 1150 |
| Armadillo | **128** | **404** | 1167 | 861 | 2435 | **35.1** | 94 | 96 | 79 |
| | **256** | **1833** | 6227 | 2502 | 9097 | **94.3** | 147 | 150 | 226 |
| | **512** | **10014** | 79329 | 10124 | 28231 | 395.1 | 255 | **252** | 770 |
| Cow | **128** | 479 | 473 | **314** | 2078 | 25.1 | 13.5 | **13.3** | 57 |
| | **256** | **740** | 2069 | 1087 | 6416 | 97.8 | 37.9 | **38** | 159 |
| | **512** | 4276 | 9467 | **3355** | 18106 | 439.3 | **97.9** | 102.9 | 487 |

Tableau 5.6 – Comparaison des temps d'exécution et de la mémoire utilisée par les algorithmes de "Marching Cubes"(MC), Marching Cubes Adaptatif (AMC) de Kahzdan *et al.*, "Dual Marching Cubes sur une grille duale" (ADMC) de Schaefer et Warren et notre algorithme. Les méthodes ont été appliquées sur un ensemble de volumes avec trois résolutions différentes. Les temps sont exprimés en millisecondes et la mémoire en Moctets.

Les parties les plus coûteuses en temps sont le calcul des normales en chaque point d'intersection de la surface $\partial V$ avec les arêtes des cellules et l'estimation de la localisation des sommets duaux. Le calcul des normales est nécessaire pendant la construction de l'octree afin de décider si la surface est suffisamment lisse à l'intérieur de la cellule. La complexité de cette procédure est de $O(k \bullet n)$ où $n$ est le nombre d'intersections de $\partial V$ avec les arêtes des cellules et $k$ est défini par rapport au rayon choisi pour estimer la normale.

La localisation des sommets est de loin l'opération la plus coûteuse. Afin d'assurer une robustesse au bruit et une bonne estimation de la forme de la surface, notre méthode doit potentiellement parcourir une bonne partie des voxels à l'intérieur de la cellule. Cela garantit que le sommet dual sera placé à l'intérieur ou proche du noyau étoilé du morceau de la surface dans la cellule, ce qui améliore grandement la qualité des triangles générés. Soit $k$ le nombre de voxels de la composante qui sont à l'intérieur de la surface. En général, $k \leq n$ avec $n$ le nombre total de voxels à l'intérieur

de chaque cellule. En conséquence, en prenant $k_i$ comme le nombre de voxels à l'intérieur de $\partial V$ dans la cellule $c_i$ qui appartiennent aux cellules qui traversent la surface, la somme $K = \sum_i^N k_i$ produit une valeur $K \gg N$ où $N$ est le nombre total des voxels contenus dans le volume $V$.

| **Volume** | | **nombre de triangles** | | | | **Nombre de sommets** | | | |
|---|---|---|---|---|---|---|---|---|---|
| Modèle | Voxels$^3$ | MC | AMC | ADMC | Le notre | MC | AMC | ADMC | Le notre |
| Buddha | 128 | 230656 | **119124** | 134342 | 120665 | 115294 | **59606** | 62027 | 60310 |
| | 256 | 934700 | **291036** | 335618 | 301884 | 967400 | **145652** | 152137 | 150944 |
| | 512 | 3756636 | **535998** | 625506 | 617388 | 1878798 | **268261** | 280138 | 308724 |
| Dragon | 128 | 183916 | 107342 | 122074 | **71844** | 91984 | 53799 | 55690 | **35925** |
| | 256 | 738648 | 208636 | 240866 | **174197** | 369364 | 104668 | 108655 | **87104** |
| | 512 | 2965056 | **321712** | 372406 | 342843 | 1482606 | **161522** | 166941 | 171435 |
| Wales Dragon | 128 | 208880 | 220008 | 245508 | **121479** | 104436 | 110008 | 113816 | **60734** |
| | 256 | 849400 | 549836 | 635438 | **364973** | 424718 | 274930 | 286964 | **182486** |
| | 512 | 3417760 | **969850** | 1134398 | 945645 | 1708938 | **484951** | 506966 | 472821 |
| Horse | 128 | 113416 | **32776** | 37782 | 34648 | 56710 | **16395** | 17043 | 17326 |
| | 256 | 456420 | **54964** | 64218 | 66420 | 228212 | **27495** | 28763 | 33212 |
| | 512 | 1829256 | 123818 | 104892 | **118806** | 914648 | 54920 | 52467 | **59413** |
| Gargoyle | 128 | **172460** | 224854 | 256020 | 107110 | **86232** | 112437 | 116962 | 53558 |
| | 256 | 697768 | **529048** | 614280 | 352727 | 348886 | **264532** | 275540 | 176367 |
| | 512 | 2802372 | **1137768** | 1330630 | 964788 | 1401188 | **568918** | 592772 | 482398 |
| Armadillo | 128 | 107892 | 156820 | 137098 | **65253** | 53948 | 71261 | 68551 | **32627** |
| | 256 | 435792 | 272210 | 316328 | **205915** | 217898 | 136109 | 141560 | **102961** |
| | 512 | 1748412 | **541336** | 634218 | 551974 | 874208 | **270674** | 281813 | 275988 |
| Cow | 128 | 132012 | 54250 | 63314 | **53136** | 65998 | 27089 | 28571 | **26570** |
| | 256 | 534028 | 149392 | 176042 | **141307** | 266980 | 74600 | 78516 | **70642** |
| | 512 | 1074756 | 462828 | 390984 | **351313** | 2149608 | 205114 | 195256 | **175668** |

Tableau 5.7 – Comparaison de nombres de triangles et de sommets des surfaces générées dans le tableau 5.6.

Les algorithmes avec lesquels nous allons comparer notre algorithme n'utilisent pas de techniques de localisation des sommets et ne tiennent pas compte du calcul des normales dans les estimations de temps d'exécution. Cela explique la différence des temps d'exécution et d'utilisation de la mémoire. Nous avons effectué nos tests sur un ordinateur conventionnel avec un processeur de 2.2GHz et 8 Go de mémoire. Le tableau 5.6 présente les temps d'exécution et la mémoire utilisée par les trois algorithmes de l'état de l'art et par notre méthode. Nous avons appliqué les méthodes sur sept modèles volumiques avec trois résolutions différentes. Les algorithmes ont été exécutés avec un critère de courbure de 0.9 défini dans un intervalle fermé entre $[0, 1]$.

Le tableau 5.6 contient les temps d'exécution et la mémoire utilisée par les algorithmes de "Marching Cubes"(MC), "Marching Cubes" Adaptatif (AMC) de Kahzdan *et al.*, "Dual Marching Cubes sur une grille duale" (ADMC) de Schaefer et Warren et notre algorithme (CCDMC). Le tableau 5.7 présente le nombre de sommets et de triangles dans chacune des surfaces. Même s'il est évident que les surfaces ne contiennent pas la même quantité de triangles, nous avons extrait ces surfaces afin d'obtenir un approximation géométrique équivalente. Néanmoins, la taille de nos maillages oscille régulièrement entre le nombre de triangles produits par AMC et ADMC. En général, même si notre algorithme a besoin de plus de temps pour extraire la surface, il faut signaler qu'il génère une surface qui contient beaucoup moins de triangles allongés (slivers) et une meilleure distribution

des angles minimal et maximal dans les triangles. C'est une propriété importante parce qu'elle peut fortement limiter les performances des algorithmes de post-traitement.

Afin de pouvoir améliorer la qualité générale des surfaces générées par MC, AMC et ADMC, il est possible d'utiliser des techniques variationnelles de remaillage qui font une optimisation globale [52, 34] de la qualité des triangles ou des algorithmes avec des heuristiques locales qui se concentrent sur l'amélioration locale de la qualité des triangles. Dans le cas des algorithmes variationnels, ils ne tiennent pas compte du modèle volumique original et leur application peut affecter fortement la qualité de l'approximation géométrique obtenue. De plus, la plus grande partie des algorithmes de l'état de l'art n'ont été appliqués que sur des maillages de quelques milliers de triangles parce que l'utilisation d'une minimisation globale induit une forte augmentation de la complexité algorithmique de la méthode. Finalement, d'autres algorithmes de remaillage doivent être appliqués afin de s'assurer que l'existence de sommets avec une valence élevée n'affecte pas la convergence des solutions.

Afin de confirmer ces idées, nous avons testé quelques algorithmes de remaillage sur quelques surfaces produites par MC, AMC et ADMC en essayant d'améliorer la qualité des triangles afin de réduire la distance entre les histogrammes d'angle minimal et maximal entre les surfaces améliorées et l'histogramme de la surface extraite directement par notre méthode. Par exemple, la figure 5.11 illustre le résultat de l'application de notre algorithme sur un objet volumique "Asian Dragon" de $512 \times 512 \times 256$ voxels. La figure montre le maillage et deux zooms permettant d'observer la qualité générale de la triangulation obtenue. L'histogramme contient la distribution de l'angle minimal pour tous les triangles de la surface.

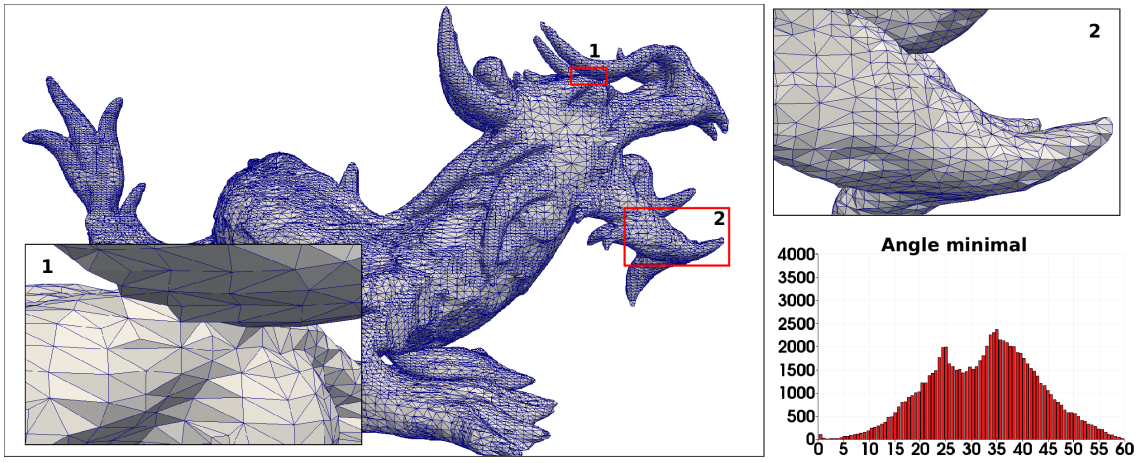
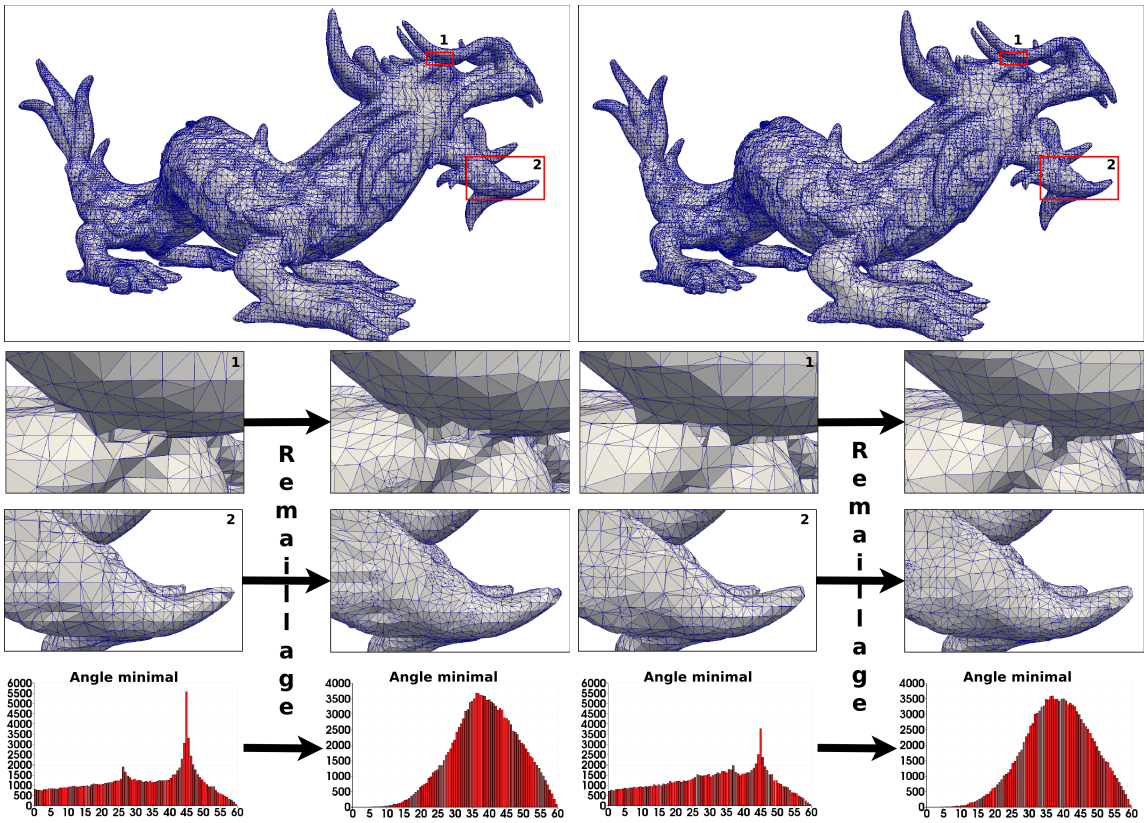



FIGURE 5.11 – Maillage produit par notre algorithme à partir d'un volume "Asian Dragon" de $512 \times 512 \times 256$ voxels. Les zooms 1 et 2 sur la figure permettent d'observer les triangles générés et l'histogramme montre la distribution de l'angle minimal des triangles du maillage.

La figure 5.12 montre le résultat de l'application des algorithmes AMC et ADMC sur le même objet "Asian Dragon". La figure 5.12a contient les résultats obtenus par AMC, les zooms 1 et 2 dans la colonne de gauche permettent d'apprécier l'existence des triangles aplatis et l'histogramme d'angle minimal confirme l'existence d'une grande quantité de triangles dégénérés. La colonne de droite montre le résultat de l'application d'un algorithme de remaillage qui améliore beaucoup la qualité de la triangulation et élimine tous les triangles aplatis. Néanmoins, ce post-traitement a requis entre 15 et 20 secondes et il a fortement augmenté la quantité de triangles dans le maillage

en passant de 106784 à 137694 facettes. Un analyse similaire peut être faite avec la figure 5.12b où le maillage est passé de 110192 à 135914 facettes.




(a) Le maillage produit par AMC passe de 106784 à 137694 facettes.

(b) Le maillage produit par ADMC passe de 110192 à 135914 facettes.

FIGURE 5.12 – Comparaison de la qualité de la triangulation obtenue par les algorithmes d'AMC (a) et ADMC (b) pour le même modèle volumique "Asian Dragon" de $512 \times 512 \times 256$ voxels avant et après l'application d'un algorithme de remaillage pour améliorer sa qualité. Néanmoins, l'algorithme a beaucoup augmenté la taille de la surface.

L'analyse des résultats obtenus nous montrent que, afin d'obtenir une surface de bonne qualité qui soit fidèle à l'objet volumique original, il vaut mieux utiliser l'information contenue dans l'objet volumique au lieu d'utiliser un post-traitement qui ne tient pas compte des caractéristiques de l'objet original. En outre, même si les méthodes d'optimisation locale arrivent à améliorer la qualité de la surface, ils consomment un temps potentiellement supérieur à celui utilisé par notre algorithme pour extraire une surface de qualité équivalente.

## 5.4 Discussion

Dans ce chapitre, nous avons analysé avec plusieurs critères les performances de notre méthode. Nous avons pu confirmer que toutes les méthodes existantes dans l'état de l'art résolvent certains problèmes liés à la génération d'une surface mais n'arrivent pas à proposer une solution complète. Notre méthode n'est pas l'exception ; néanmoins, une différence fondamentale est que notre solution cherche à fournir un bon compromis entre les différents facteurs qui sont considérés comme impor-

tants au moment de générer une surface. Notre méthode est particulièrement adaptée aux données volumiques parce qu'elle utilise intensivement les composantes connexes comme outil pour la détection et l'amélioration de l'approximation topologique et géométrique de l'objet original. Cette caractéristique est obtenue au détriment de la performance de notre solution mais elle permet aussi d'obtenir les informations nécessaires du volume sans avoir besoin de pré-traitements ou de transformer l'entrée dans une représentation plus convenable (par exemple un champ de distance). Nous avons aussi observé que, même s'il est toujours possible d'appliquer un post-traitement dans le but d'améliorer la qualité de triangles dans une surface, les algorithmes de remaillage global ne sont pas adéquats pour traiter des maillages avec plusieurs centaines de milliers de triangles. De plus, les algorithmes locaux utilisent aussi un temps équivalent, voire supérieur, au temps nécessaire pour extraire un maillage avec notre méthode.

# Chapitre 6

# Conclusion et perspectives

Dans cette partie, nous avons étudié les principales méthodes d'extraction de surface à partir de données volumiques. Nous avons montré la complexité liée à ce type d'algorithme et les nombreux facteurs qui doivent être considérés au moment de concevoir une solution, notamment la technique de localisation de sommets sur la surface et la division spatiale. Ces choix ont une incidence directe sur la précision de la surface ou sur la quantité de triangles allongés ou aplatis.

L'exploration de l'état de l'art sur la génération de surfaces nous permet de conclure que la plus grande partie des algorithmes visent à résoudre certains problèmes liés à la reconstruction de surface mais ne résolvent pas tous les problèmes. Cela vient du fait qu'il y a des restrictions fondamentales, et éventuellement contradictoires, entre les techniques utilisées pour résoudre un problème en particulier et les méthodes misant sur l'obtention d'une solution plus générale.

La méthode que nous avons présentée se base principalement sur l'utilisation des composantes connexes d'un objet volumique afin de mieux approximer la géométrie du volume. Nous cherchons à garantir que les surfaces obtenues contiennent un nombre limité ou borné de triangles dégénérés. La contrepartie est le temps de calcul accru par rapport à un algorithme qui ne le fait pas comme AMC. En revanche, une surface produite par AMC doit impérativement passer par une étape de post-traitement qui peut être très coûteuse en temps de calcul afin de pouvoir améliorer la qualité de ses éléments géométriques.

Dans des travaux futurs, nous considérons qu'il est important d'améliorer le temps de calcul de notre solution afin de la rendre interactive. Nous estimons que la seule manière d'arriver à avoir une bonne qualité de surface combinée avec une grande vitesse consiste à appliquer des pré-traitements sur les volumes afin d'extraire l'information que nous devons calculer dans les différentes étapes de notre algorithme : l'estimation de la courbure de la surface discrète, le calcul des champs de distance discrète sur chacun des voxels du volume, etc. Dans ce contexte, nous pouvons profiter de l'apparition récente des techniques plus performantes pour le calcul de ce type d'indicateurs avec des garanties théoriques sur leur complexité et convergence.

Il pourra également être intéressant d'améliorer la représentation des arêtes vives. Pour cela, nous envisageons d'utiliser des fonctions d'erreur quadratique (QEF) dans les cellules qui peuvent contenir ces types d'arêtes.

# Deuxième partie

# Génération “out-of-core “de surfaces adaptatives

# Chapitre 7

# État de l'art des méthodes "out-of-core" pour l'extraction de surfaces

Dans cette section, nous allons présenter les travaux pertinents dans le domaine de la génération de surfaces à partir de grands volumes de données. En règle générale, les méthodes pour traiter des gros volumes se concentrent sur deux aspects : la construction de structures de données de plus en plus compactes et l'optimisation de l'accès aux données sur le disque.

Les méthodes qui visent l'optimisation de la recherche des données appliquent des pré-traitements sur les données afin d'extraire l'information nécessaire pour construire une structure de recherche optimale. La construction de ces structures est étroitement liée aux critères d'intérêt (par exemple, les iso-valeurs) et le résultat final est fortement dépendant des caractéristiques du volume. Ces techniques permettent d'augmenter la capacité des algorithmes à traiter des volumes grandissants sans avoir besoin d'utiliser directement une technique out-of-core.

Néanmoins, lorsque la taille du volume est supérieure à la quantité de mémoire principale disponible, il n'est plus suffisant d'utiliser une structure compacte et il faut chercher des stratégies efficaces pour l'accès aux données sur le disque. Ainsi, les structures de recherche ont été complétées pour être organisées et stockées de manière efficace sur le disque. Cela implique que les pré-traitements doivent tenir compte du coût associé aux accès disque au moment d'organiser les données et de la possibilité de paralléliser les recherches afin d'améliorer la performance générale. Ces approches ne sont pas spécifiques au domaine d'extraction de surfaces mais elles se sont révélées d'une grande utilité dans ce domaine particulier.

## 7.1 Le problème de l'accès aux données sur le disque

Le problème de la complexité de l'accès aux données qui ne résident pas en mémoire a été traité de manière générale dans les travaux d'Aggarwal et Vitter [1]. Ils ont déterminé des bornes inférieures de complexité pour l'accès aux données et identifié l'importance de leur bonne organisation et structuration sur le disque afin d'accélérer leur utilisation pour tout type d'algorithme.

Arge [3, 2] a présenté une technique générale pour transformer une structure de données hiérarchique in-core en une structure hiérarchique out-of-core avec une structure auxiliaire appelée "Buffer tree". Le Buffer tree gère des mémoires tampon dans chaque nœud intérieur de la hiérarchie. Ces mémoires contiennent les opérations (insertion, élimination, etc.) qui vont s'appliquer en bloc sur un ensemble de nœuds au moment où ils sont chargés en mémoire. Ces opérations sont propagées progressivement vers les nœuds inférieurs jusqu'à arriver aux nœuds feuilles qui contiennent l'information à traiter. Cette technique peut être utilisée pour implémenter des algorithmes du type "arbre des segments" afin de faire des requêtes sur un ensemble de segments avec une complexité d'entrée/sortie logarithmique par rapport au nombre de segments. Arge et Vitter [6] ont introduit une première structure de données basée sur des arbres d'intervalles qui peut être utilisée pour organiser un ensemble de points dans un espace n-dimensionnel. Néanmoins, c'est l'apparition des algorithmes de division spatiale de type "Marching Cubes" [72] et "Marching Tetrahedra" [20] qui ont accéléré le développement de structures de données performantes appliquées à l'extraction d'iso-surfaces.

## 7.2 Méthodes de recherche optimale "in-core"

Les méthodes de recherche optimale construisent des structures de données compactes afin de pouvoir répondre efficacement aux questions du type : quelles sont les cellules actives pour une iso-valeur $\lambda$ ? Livnat *et al.* [71] ont introduit un algorithme de complexité quasi optimale pour l'extraction d'iso-surfaces à partir de données volumiques. Cette solution se base sur la notion d'espace de longueur (Span space) défini toujours en $\Re^2$ par rapport aux espaces géométriques en $\Re^n$ ($\Re^3$ dans le cas des volumes). Il doit exister une injection $\phi : C \rightarrow E$ tel que $C$ est l'espace contenant les cellules divisant le volume et $E$ est l'espace de longueur. Les auteurs posent la définition suivante :

**Définition 1. Injection vers l'espace de longueur $\phi$ :** Soit $C$ un ensemble de cellules et P un ensemble de points tels que, $\forall c_i \in C$ il existe un point $p_i \in P$ associé tel que :

$$p_i = (a_i, b_i)$$

avec

$$a_i = min_j\{v_j\} \;\; et \;\; b_i = max_j\{v_j\}$$

et $v_j$ sont les valeurs de voxels du volume aux sommets de la cellule $c_i$.

L'algorithme divise le volume en cellules tétraédriques et affecte la valeur du voxel à chaque sommet de la cellule. Afin d'identifier chaque cellule dans l'espace de longueur $E$, les auteurs utilisent la fonction $\phi : C \rightarrow E$ pour associer à chaque cellule $c_i \in C$ une paire de valeurs $[a_i, b_i] \in E$ définies comme ci-dessus. Il est alors facile de savoir quelle iso-surface passe par une cellule $c_i$. Ensuite, cet algorithme utilise les valeurs minimale et maximale des sommets pour associer un point $(a, b)$ à la cellule et remplacer l'espace de recherche géométrique par un espace de recherche défini comme suit :

**Définition 2. L'espace de recherche :** Soit $C$ un ensemble de cellules, P l'ensemble des points associés dans l'espace de longueur et $v$ une iso-valeur. Le sous-ensemble des cellules $P_v \subset P$ contenant l'iso-valeur $v$ sont les cellules telles que :

$$\forall (x_i, y_i) \in P_v \quad x_i < v \;\; et \;\; y_i > v$$

La définition d'un espace de longueur alternatif permet de réduire l'espace de recherche de $\Re^n$ vers $\Re^2$ avec n'importe quelle valeur de $n$. Géométriquement, la recherche revient à trouver les points qui se trouvent à gauche de la droite $x = v$ et au-dessus de la droite $y = v$ sur le plan Min-Max comme l'illustre la région rayée sur la figure 7.1a.

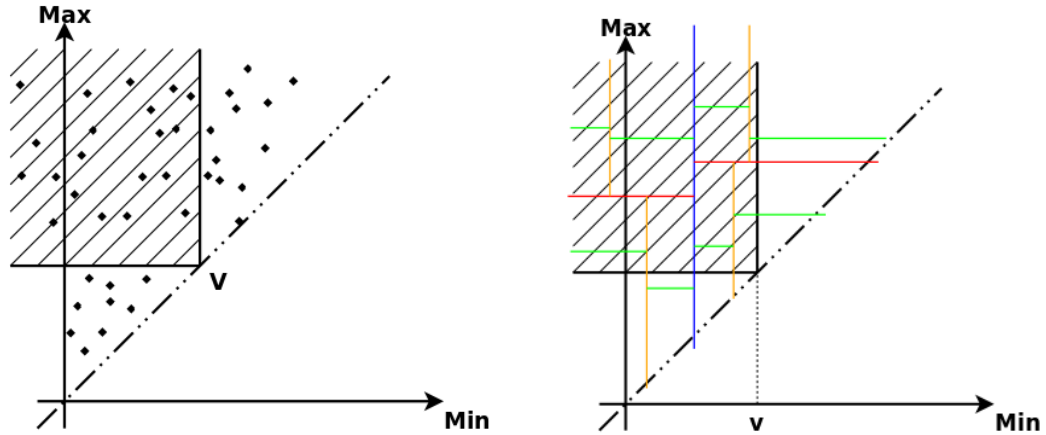


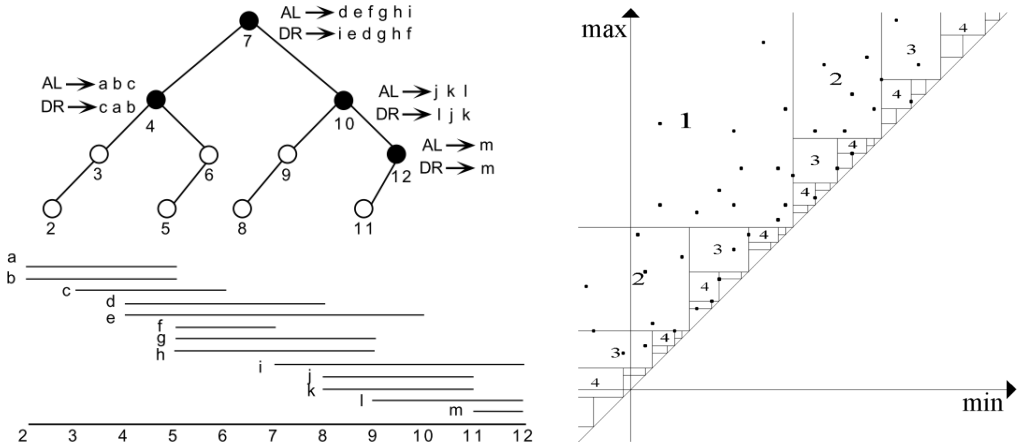

FIGURE 7.1 – (a) Dans un espace de longueur (span space) les extrémités minimale et maximale des intervalles sont utilisées comme les coordonnées $(x, y)$ respectivement d'un point dans un espace bidimensionnel. Les intervalles qui contiennent la valeur $v$ sont à gauche de la droite $x = v$ et au-dessus de la droite $y = v$. (b) L'algorithme divise l'espace des valeurs avec un Kd-tree où chaque niveau de subdivision est affiché avec une couleur différente.

Finalement, afin d'améliorer les temps de recherche, cet algorithme construit un Kd-tree sur l'espace de longueur afin d'accélérer les requêtes comme l'illustre la figure 7.1b. Le Kd-tree peut être construit récursivement lors d'un pré-traitement avec une complexité optimale $O(nlogn)$. Une requête a une complexité de $O(\sqrt{n} + k)$ dans le pire des cas où $n$ est le nombre de cellules et $k$ le nombre de cellules intersectées par la surface. Dans la plus grande partie des cas, $k \gg \sqrt{n}$. Une fois que les cellules contenant la surface sont identifiées, l'approximation polygonale est réalisée avec un algorithme de "Marching Tetrahedra". L'utilisation de la mémoire est proportionnelle à la quantité de cellules, soit $O(n)$. Les mêmes auteurs avec Hansen [99, 98] ont aussi proposé une approche améliorée qui utilise une grille régulière sur l'espace de longueur ; elle obtient des meilleurs résultats empiriques et permet de paralléliser l'extraction de la surface. Le plus grand avantage de cette méthode est qu'elle permet efficacement de trouver toutes les cellules intersectées par la surface pour différentes iso-valeurs.

Les valeurs $[a_i, b_i]$ définies dans l'espace de longueur peuvent être utilisées dans différentes structures de recherche. Dans le cas d'un arbre d'intervalles, pour détecter si une cellule $c_i$ est traversée

par l'iso-surface définie par l'iso-valeur $v$, il faut vérifier que $a_i < v$ et $v < b_i$. Dans l'espace de longueur directement, les cellules intersectées par l'iso-surface $v$, sont celles contenues à l'intérieur du sous-ensemble $P_v \subseteq P$ tel que $\forall(x_i, y_i) \in P_v$, $x_i < v$ et $y_i > v$.

Dans le contexte de la génération de surface, Cignoni *et al.* [32] ont proposé une méthode qui extrait une surface à partir d'un volume $V$ en réalisant une division tétraédrique régulière sur le domaine géométrique. Ils utilisent aussi l'espace de longueur présenté par Livnat *et al.* [71] mais au lieu d'indexer cet espace avec un Kd-tree, ils utilisent un *arbre d'intervalles,* structure initialement proposée par Edelsbrunner [41, 87], afin de pouvoir répondre aux requêtes du type "quels sont les intervalles qui contiennent une valeur particulière $q$ ?". Un arbre d'intervalles est une structure hiérarchique binaire qui stocke dans ses nœuds intérieurs des listes d'intervalles. Un exemple d'un arbre d'intervalles est illustré sur la figure 7.2a. Afin d'optimiser les requêtes, ce type de structures doivent être équilibrées et les intervalles doivent être ordonnés par rapport à leurs extrémités initiales (à gauche). La figure 7.2a montre l'utilisation de ce type de structure sur l'espace de longueur.




(a) (b)

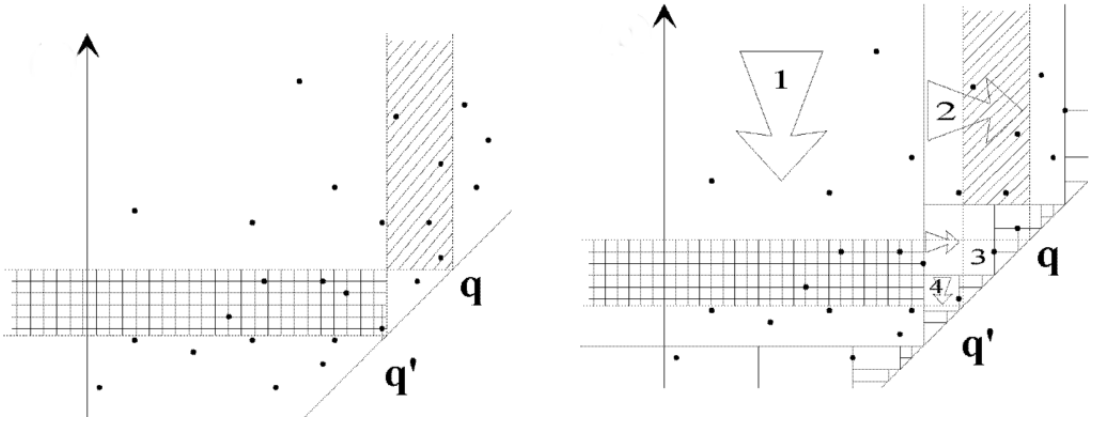
FIGURE 7.2 – (a) Exemple d'un arbre d'intervalles sur 13 intervalles. Chaque nœud contient deux listes d'intervalles ordonnés par rapport aux extrémités dans les deux sens [71]. (b) L'algorithme divise l'espace des valeurs avec un Kd-tree où les régions au même niveau de subdivision sont représentées avec le même nombre [71].

Néanmoins, comme ces algorithmes se concentrent principalement sur l'accélération de l'accès aux données, ils ne considèrent pas les problèmes topologiques et d'approximation liés à l'extraction d'une iso-surface correcte. De plus, ces approches ne sont pas facilement applicables pour l'extraction de surfaces adaptatives sur des jeux de données plus grands que la mémoire disponible.

Bajaj *et al.* [8] ont développé une méthode qui combine plusieurs techniques pour l'extraction d'iso-surfaces. Elle est applicable à n'importe quelle subdivision spatiale et topologique de cellules. Elle comprend deux pré-traitements et deux étapes exécutées en temps réel. Le premier pré-traitement parcourt les cellules et choisit quelques cellules graines $S$ en s'assurant de tenir compte de toutes les composantes connexes du volume. Ensuite, une paire de valeurs minimale et maximale $(a_i, b_i)$ est associée à chaque cellule. Le deuxième pré-traitement utilise ces paires de valeurs pour construire une structure de données de recherche mais, au lieu d'utiliser l'espace de longueur, les auteurs utilisent les intervalles projetés en $1D$ et une structure du type arbre des segments ou un arbre

d'intervalles. Cela leur permet d'avoir une complexité de $O(logn + k)$ pour les requêtes, $n$ étant le nombre de cellules de $S$ et $k$ le nombre de cellules intersectées par la surface. Pendant la première étape exécutée en temps réel, ils utilisent la structure de recherche pour trouver les cellules contenant la surface pour l'iso-valeur $w$ et un algorithme dit "advancing front" pour étendre le front des cellules jusqu'à trouver toutes les cellules intersectées par la surface. Finalement, dans une deuxième étape, les auteurs obtiennent la surface à partir des cellules détectées avec un algorithme de "Marching Cubes".

Cignoni *et al.* [31] ont aussi travaillé avec un arbre d'intervalles afin d'accélérer la détection des cellules intersectées par une iso-surface. L'une des nouveautés de cet algorithme est qu'il peut être utilisé sur des données structurées ou non structurées sans modification mais avec une plus grande utilisation de la mémoire dans ce dernier cas. Une autre caractéristique est qu'il essaie de profiter de la cohérence spatiale dans l'espace de longueur entre deux iso-surfaces correspondant à des iso-valeurs proches. Cette cohérence spatiale peut être appréciée sur la figure 7.3. En (a), nous pouvons voir que les cellules intersectées par l'iso-surface définie par l'iso-valeur $q^{'}$ sont celles intersectées avec l'iso-valeur $q$ dont on a supprimé celles qui sont placées dans la zone hachurée à droite et auxquelles on a ajouté celles qui se trouvent dans la région quadrillée en bas. En (b), nous pouvons voir comment cela affecte les nœuds détectés dans l'arbre d'intervalles. La liste des cellules actives pour $q$ doit être mise à jour pour $q^{'}$ avec un parcours de l'arbre d'intervalles qui étend les cellules contenues dans les nœuds $1, 2, 3$ et $4$ sur la figure.




(a) (b)

FIGURE 7.3 – Utilisation de la cohérence spatiale pour accélérer l'accès aux cellules actives entre des iso-valeurs proches [31]. (a) La différence entre les cellules actives pour $q$ et $q'$ revient à éliminer les cellules de la zone hachurée et à ajouter celles de la zone quadrillée en bas. (b) Au niveau de l'arbre d'intervalles, cela revient à étendre les intervalles contenus dans les nœuds 1 et 2 ou à étendre la recherche sur les nœuds 3 et 4 [31].

## 7.3 Méthodes de recherche optimale out-of-core

L'un des premiers algorithmes vraiment conçu pour traiter des données qui ne peuvent être chargées entièrement en mémoire a été proposé par Chiang et Silva [27]. Ils ont proposé d'utiliser l'arbre d'intervalles en mémoire externe développé par Arge et Vitter et les idées de Cignoni *et al.* afin de construire une structure de recherche efficace dans un espace unidimensionnel par rapport aux iso-valeurs. Cette structure de données est construite une seule fois et stockée sur le disque pour

sa réutilisation. Cela permet d'éliminer le goulot d'étranglement existant dans la recherche des cellules intersectées par une iso-surface. De plus, il permet de charger en mémoire uniquement les cellules intersectées, généralement $O(N^{\frac{2}{3}})$, $N$ étant le nombre total de cellules. De plus, la taille des blocs chargés en mémoire peut être adaptée à l'espace mémoire principal disponible.

Chiang et Silva[28, 29] ont introduit le concept de méta-cellule comme un nouveau niveau de subdivision pour des volumes de grande taille. Ces méta-cellules sont construites lors d'un pré-traitement et contiennent toutes approximativement le même nombre de cellules afin que la charge de chaque méta-cellule ait le même coût d'entrée/sortie. Ensuite, une structure hiérarchique out-of-core, l'arbre binaire d'intervalles bloqué (ABIB) va permettre d'indexer les intervalles définis par les valeurs minimale et maximale de chaque méta-cellule. L'ABIB peut avoir un facteur de branchement supérieur à 2 et, en conséquence, une profondeur de $O(log_B N)$, $N$ étant le nombre d'intervalles et $B$ le facteur de branchement. Chaque nœud feuille de l'arbre ne peut pas contenir plus de $B$ intervalles. S'il en contient déjà plus, ce nœud devient intérieur et les intervalles sont divisés dans au plus $B-1$ sous-nœuds. S'il existe plusieurs intervalles avec les mêmes valeurs maximale et minimale, ils sont ordonnés par rapport au code d'identification de la méta-cellule à laquelle ils appartiennent. Cette structure fournit une complexité de requêtes de $O(log_2 \frac{N}{B} + \frac{K}{B})$ où $K$ est le nombre d'intervalles intersectés par l'iso-surface. Néanmoins, le besoin d'un pré-traitement pour la mise au point de l'ABIB peut prendre beaucoup de temps. La structure générale d'un ABIB est illustrée sur la figure 7.4.

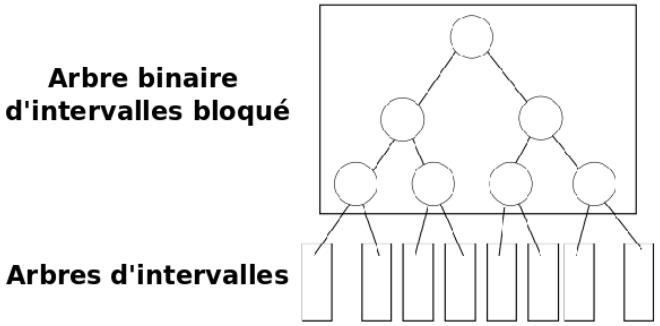



FIGURE 7.4 – Illustration de la structure d'un arbre binaire d'intervalles bloqué (ABIB).

Zhang *et al.* [129] ont développé une méthode pour extraire des iso-surfaces à partir de données volumiques massives. Elle repose sur une utilisation intensive des nouvelles capacités du hardware telles que les processeurs multiples, plusieurs disques et le méta-buffer de la carte graphique pour la visualisation. Cette méthode commence par diviser le volume en méta-cellules en utilisant l'intervalle des iso-valeurs contenues dans chaque partie du volume afin d'équilibrer la charge de chaque processeur et de chaque mémoire associée. Ensuite, un arbre d'intervalle out-of-core est utilisé comme structure de recherche sur les méta-cellules en association avec un arbre d'intervalles pour les cellules à l'intérieur de chaque méta-cellule. Ces opérations sont réalisées lors d'un pré-traitement sur une plateforme avec plusieurs processeurs et les disques correspondants afin d'exécuter les tâches en parallèle. Pour une iso-valeur $q$, l'algorithme réalise la recherche des méta-cellules et cellules actives avec l'arbre d'intervalle, la triangulation est faite avec l'algorithme des "Marching Cubes" et la surface est transmise au processus du méta-buffer pour le rendu graphique. Cette solution supporte le passage à l'échelle avec davantage de processeurs et de disques et permet

de visualiser des surfaces avec des centaines de millions de triangles pour différentes iso-valeurs, et ceci en quelques dizaines de secondes.

En s'inspirant de l'approche proposée par Silva *et al.* [103, 102], Chiang *et al. [26, 43]* ont développé une méthode également basée sur une utilisation intensive des capacités de parallélisation des machines actuelles. Une différence avec les algorithmes précédents est qu'elle est fortement focalisée sur la visualisation. Cette solution divise le volume et indexe les méta-cellules avec un arbre d'intervalles (ABIB). Ensuite, elle utilise une boîte englobante afin de décider quelles sont les méta-cellules visibles par rapport au point de vue de l'observateur. La vue est divisée en tuiles traitées de manière séquentielle par un programme central qui cherche les méta-cellules concernées et distribue les tâches d'extraction de surface entre plusieurs processeurs.

Comme les autres techniques décrites jusqu'ici, ces solutions sont principalement focalisées sur la visualisation et ne génèrent pas de surfaces adaptatives. De plus, elles ne se soucient pas de la topologie ni de la géométrie du volume afin d'améliorer la qualité visuelle et l'approximation de la surface.

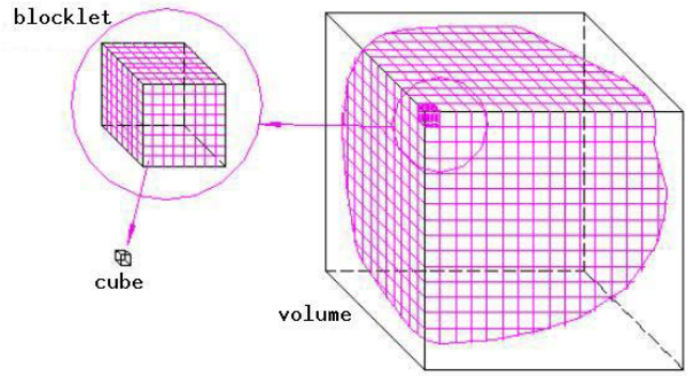
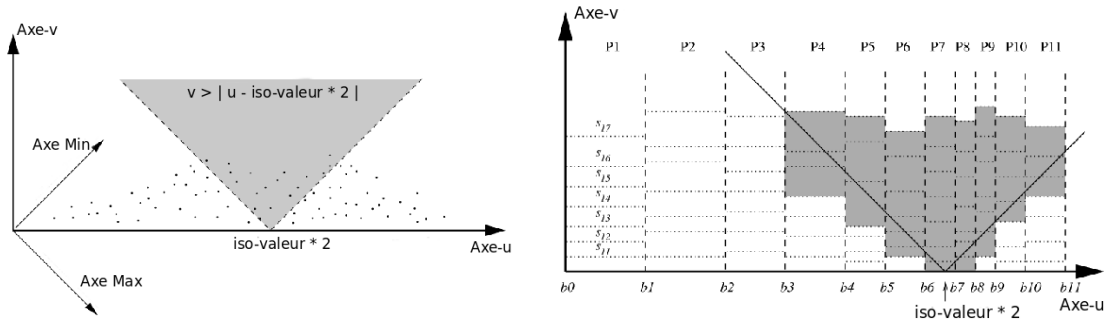



FIGURE 7.5 – Illustration de la division du volume dans l'algorithme de Zhang et Newman [129]. Le volume est divisé en "blocklets" de $8 \times 8 \times 8$ voxels et chaque cube ou voxel est traité par l'algorithme des "Marching Cubes".

Zhang et Newman [129] ont proposé un algorithme focalisé sur la réduction de la mémoire vive nécessaire pour traiter n'importe quelle taille de volume. L'unité fondamentale de subdivision du volume est un ensemble de voxels, généralement isotrope : le "blocklet". Les blocklets peuvent avoir différentes tailles mais les auteurs ont déterminé expérimentalement qu'une taille efficace était $8 \times 8 \times 8$ voxels. Dans cet algorithme, ils se concentrent sur l'extraction d'une iso-surface pour un ensemble d'iso-valeurs qui sont réparties dans des intervalles $T_i$. Ensuite, le volume est divisé en fichiers de partition $P_i$ par rapport à chaque intervalle $T_i$. Ainsi, tous les blocklets actifs (intersectés par la surface) pour un intervalle d'iso-valeurs et adjacents, peuvent être fusionnés en une unique unité et stockés dans le même fichier. Cela donne naissance à des blocklets de taille arbitraire qui sont chargés en mémoire et traités comme une seule entité. Comme cette solution fait une estimation statique du travail à affecter par chaque processeur, la construction des fichiers et la définition des intervalles considérés se fait dans le cadre d'un pré-traitement. Finalement, pour extraire la surface, cet algorithme utilise une version optimisée de l'algorithme "Marching cubes". Comme les méthodes que nous avons vues jusqu'à maintenant, cette solution se focalise sur l'extraction et la visualisation d'iso-surfaces sans prendre en compte les contraintes topologiques

ou géométriques de l'objet original.

Les algorithmes présentés ci-dessus ont besoin de construire des structures auxiliaires (arbre d'intervalles, arbre des segments, etc.) afin d'accélérer l'accès aux cellules et méta-cellules. Une étude des histogrammes de la différence entre les valeurs maximale (max) et minimale (min) définie comme $d = |max - min|$ montre que cette valeur est petite pour la majorité des cellules. Afin de profiter de la similitude entre les valeurs de $d$ pour différentes cellules et de réduire la quantité de mémoire nécessaire pour stocker la structure de recherche, Bordoloi et Shen [12] ont proposé une transformation linéaire des intervalles contenus dans l'espace de longueur vers un nouvel espace appelé "espace UV". Cet espace est construit par une rotation de $45^o$ dans le sens trigonométrique et une mise à l'échelle de $\sqrt{2}$ sur les coordonnées $(u, v)$. Les coordonnées des intervalles dans cet espace sont calculées avec les équations $u = max + min$ et $v = max - min$ qui peuvent être interprétées comme le double de la valeur moyenne et la longueur de l'intervalle respectivement. Finalement, les cellules qui intersectent l'iso-surface pour l'iso-valeur $q$ sont celles pour lesquelles $v > |u - q * 2|$ comme l'illustre la zone grise sur la figure 7.6.




(a) Transformation de l'espace de longueur dans l'espace UV [12].

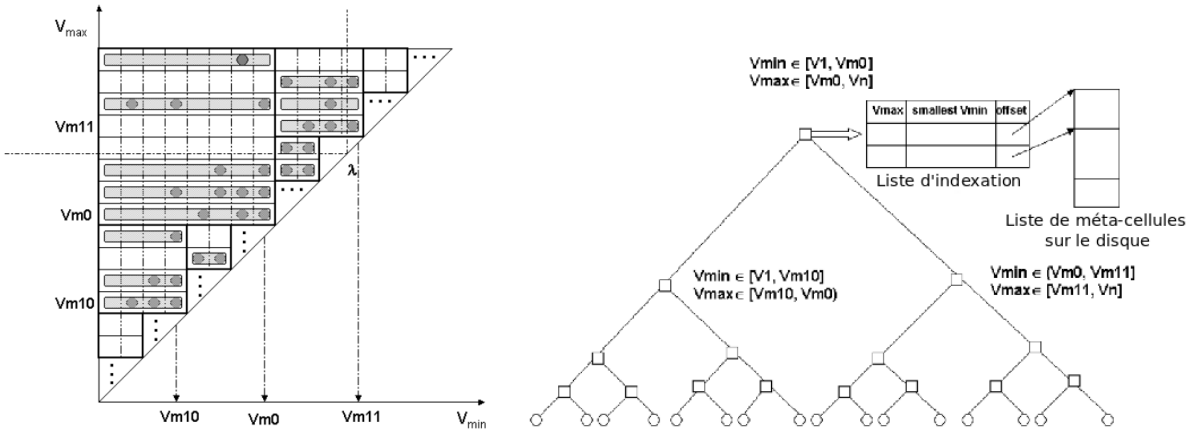
(b) Quantification de l'espace UV [12].

FIGURE 7.6 – (a) L'espace UV est construit par une rotation de $45^o$ dans le sens trigonométrique et une mise à l'échelle de $\sqrt{2}$ de coordonnées $(u, v)$. Les cellules intersectées par l'iso-surface sont dans la zone grise. (b) Quantification des valeurs dans l'espace UV : les axes sont quantifiés de manière indépendante en commençant par l'axe $u$.

Un avantage de cette approche est qu'elle a seulement besoin de stocker une seule liste de cellules ordonnées au lieu de deux listes comme c'est le cas dans la plupart des algorithmes précédents. L'autre avantage est qu'elle peut être combinée avec d'autres algorithmes qui utilisent des méta-cellules et une organisation alternative des cellules. Dans les tests présentés, cet algorithme a réduit la taille de la structure de recherche de 34%. Malheureusement, les auteurs ne proposent pas d'adaptation out-of-core de cette solution mais cela reste envisageable.

La méthode proposée par Wang *et al.* [119] reprend le concept de méta-cellule introduit par Chiang et Silva [28] afin d'optimiser la détection des cellules actives pour une iso-valeur particulière et utilise une représentation compacte d'un arbre d'intervalles défini dans l'espace de longueur. Afin de profiter de la cohérence des cellules avec des valeurs minimale et maximale proches, toutes les méta-cellules qui ont la même valeur maximale de gauche à droite sont regroupées et stockées ensemble en mémoire. Cette technique est illustrée sur la figure 7.7a où les cellules sont représentées avec des points et les regroupements de cellules avec des boîtes allongées appelées "briques". Chaque brique est stockée dans un seul espace consécutif de mémoire pour optimiser son accès. A partir

de cette partition il est possible de construire un arbre d'intervalles compact comme illustré sur la figure 7.7b.




(a) (b)

FIGURE 7.7 – (a) Dans l'espace de longueur, les méta-cellules sont représentées par des points. Les méta-cellules sont regroupées de gauche à droite en briques selon leur valeur maximale. Les briques sont ici représentées comme des boîtes. (b) Arbre d'intervalles correspondant à la partition de la figure (a) : les méta-cellules qui font partie d'une brique sont stockées de manière consécutive en mémoire (Figures extraites de [119]).

L'avantage de cette implémentation est qu'elle divise au moins par deux la taille de l'arbre d'intervalles utilisé parce que chaque nœud n'a pas besoin de stocker les deux listes d'intervalles. Fondamentalement, à chaque niveau de l'arbre il y au plus $\frac{n}{2}$ entrées et la hauteur de l'arbre est $log_2 n$. La taille de l'octree est $O(nlogn)$ ce qui est très inférieur à la taille d'un arbre d'intervalles traditionnel ($O(N)$, $N$ étant le nombre d'intervalles). De plus, cet algorithme peut être adapté pour utiliser une configuration parallèle où chaque nœud a accès à son propre espace mémoire et à son processeur. Cette version parallèle a permis de visualiser des maillages composés de plusieurs centaines de millions de triangles à partir d'un volume de $2048 \times 2048 \times 1920$ voxels. Les temps d'exécution pour ce volume vont de quelques secondes jusqu'à quelques minutes pour différentes iso-valeurs et la mémoire utilisée est de 3.8 Go pour un volume original de 7.5 Go. Néanmoins, il faut aussi souligner que le pré-traitement nécessaire pour construire la structure de recherche requiert plus de 30 minutes d'exécution. De plus, la surface extraite est régulière et l'algorithme n'implémente aucune validation topologique ou optimisation géométrique dans la triangulation finale.

Afin d'optimiser l'utilisation de la mémoire, Petrik et Skala [84] ont développé une technique pour réduire l'espace de recherche à une seule dimension. Pour cela, ils utilisent un nombre $M$ d'intervalles de quantification qui est défini par l'utilisateur et dépend de la quantité d'iso-valeurs possibles dans le volume. Ainsi, pour des volumes codés sur un octet, $M$ peut avoir 256 valeurs afin de reconnaître tous les valeurs maximales possibles. La conversion d'intervalle $(c_{min}, c_{max})$ vers une valeur $t_c$ est calculée avec l'équation $t_c = t_i + t_0$ où :

$$t_i = \left\lfloor \frac{\frac{c_{max} - max_{min}}{max_{max} - min_{min}}}{t_{int}} \right\rfloor * t_{int} \quad , \quad t_0 = \frac{t_{min} - min_{min}}{min_{max} - min_{min}} * t_{int} \quad et \quad t_{int} = 1/M$$

Une fois la transformation réalisée, les cellules du volume sont ordonnées par ordre croissant du paramètre $t_c$. Ensuite, les auteurs construisent une liste indexée par les valeurs de $t_c$ où chaque entrée $q$ contient une liste avec les identifiants des cellules pour lesquelles $t_c = q$. Comme la procédure de construction de cette structure de données nécessite une opération de tri, la construction de la structure de données peut être réalisée en temps $O(N\ log\ N)$ où $N$ est le nombre de cellules.

Pour l'extraction de l'iso-surface pour une iso-valeur $q$, il faut identifier les cellules telles que $c_{min} < q < c_{max}$. Pour cela, il faut convertir la valeur $q$ en une valeur $t_{limit}$ dans l'espace 1D telle que les cellules intersectées par la surface soient contenues dans les entrées $t_c \ll t_{limit}$ avec l'équation suivante :

$$t_{limit} = (n * t_{int}) + \frac{q - min_{min}}{min_{max} - min_{min}} * t_{int}$$

où $n$ est l'intervalle de quantification. Ainsi, l'extraction des cellules actives pour une iso-valeur dévient une opération linéaire en temps avec une structure de données plus compacte que les arbres de recherche présentés jusqu'ici. Cependant, comme seules les valeurs maximales des intervalles sont quantifiées et utilisées pour calculer les valeurs $t_c$, il peut apparaître de légères erreurs de recherche au moment d'extraire l'iso-surface.

## 7.4 Méthodes spécifiques pour des surfaces adaptatives

Gregorski *et al.* [50] ont proposé une méthode pour extraire des surfaces adaptatives à partir de volumes dynamiques. Cette solution utilise une division spatiale tétraédrique conforme inspirée par des hiérarchies de diamants [123, 124] afin de garantir que les surfaces ne contiennent pas de trous. Comme cette solution est fortement orientée vers la visualisation, le raffinement de la grille tétraédrique va être guidé par le point de vue de l'observateur. De plus, les gradients normalisés sont calculés en chaque point du volume et stockés comme une texture. Cela permet d'améliorer l'illumination et le rendu des régions orthogonales à l'observateur. Par rapport à l'accès aux données sur le disque, la méthode emploie un codage des points avec une courbe de remplissage de l'espace (code de Morton). Ce code fournit un ordre avec une forte cohérence spatiale au niveau local, utilisé pour organiser les fichiers sur le disque et améliorer l'accès aux régions du volume qui sont visibles et pertinentes pour le niveau de résolution, le pas de temps et la position de l'observateur.

Cette méthode profite de la cohérence spatiale du volume d'un pas de temps au suivant afin de réutiliser le plus possible la subdivision déjà réalisée. Dans les figures 7.8 A-B, la subdivision existante est réutilisée pour construire le nouveau contour. Par contre, sur les figures 7.8 C-E, la subdivision doit recommencer à partir de la racine parce que le nouveau contour se trouve trop loin de l'ancien.

Enfin, cette méthode utilise des techniques de compression et des cartes graphiques accélératrices pour optimiser les temps de rendu et minimiser la quantité de données qui doivent être chargées en mémoire pendant l'exécution.

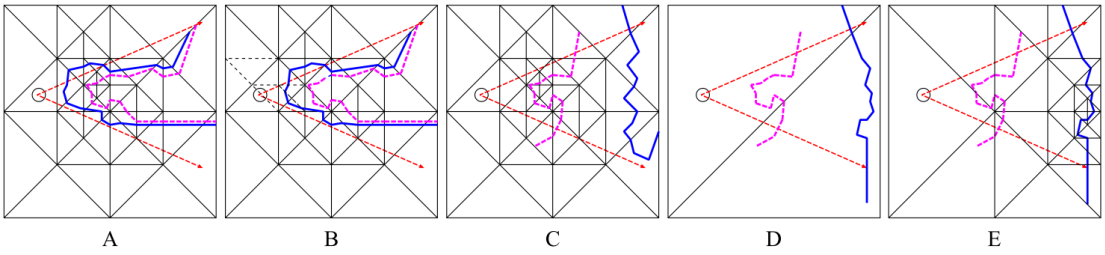



FIGURE 7.8 – Changement de la subdivision du domaine pour des surfaces dynamiques. Sur les figures A-B, le nouveau contour (ligne continue) est près de l'ancien contour (ligne en pointillés) et la subdivision et le contour peuvent être utilisés comme point de départ pour obtenir le nouveau contour. Dans C-E, le nouveau contour (ligne continue) est loin du contour actuel (ligne en pointillés), alors, la subdivision courante est détruite et une nouvelle subdivision est construite à partir de la racine [50].

## 7.5 Discussion

Nous avons décidé de nous intéresser prioritairement au problème d'extraction de surfaces adaptatives avec des ressources limitées en mémoire et nous laisserons de coté pour le moment la gestion de la visualisation. Les approches existantes dans la littérature se sont concentrées sur la construction de structures auxiliaires de recherche pour accélérer l'accès aux données sur le disque dur. De plus, ces structures permettent aussi d'améliorer la détection des cellules actives par rapport aux différentes iso-valeurs. Elles doivent être construites lors d'une coûteuse étape de pré-traitement et deviennent très dépendantes du volume à explorer. Il faut bien souligner que ces méthodes sont très efficaces. Néanmoins, presque toutes, à l'exception de celle proposée par Gregorski *et al.*, utilisent l'algorithme "Marching Cubes" ou l'une de ses extensions pour la génération de l'iso-surface. La surface obtenue n'est donc pas adaptative et les surfaces générées contiennent une grande quantité de triangles et beaucoup de triangles dégénérés.

Pour notre part, nous souhaitons porter une attention particulière sur l'adaptabilité de la surface pour maximiser la qualité de l'approximation tout en limitant le nombre de triangles.

# Chapitre 8

# Génération out-of-core de surfaces adaptatives

Dans cette thèse, nous nous sommes intéressés à la génération de maillages à partir de volumes de grande taille. Cela nous a poussé à analyser les approches dites de "Streaming" et les stratégies out-of-core. Les techniques de "streaming" appliquent des pré-traitements afin d'organiser les données d'entrée de manière à pouvoir les traiter progressivement sans trop affecter l'implémentation de l'algorithme in-core utilisé. Une telle réorganisation permet aussi d'optimiser l'accès aux données pendant l'exécution de l'algorithme. Parmi les stratégies out-of-core, nous nous sommes concentrés sur les approches "Diviser pour régner" qui sont apparues comme une stratégie générale pour attaquer des problèmes trop compliqués ou trop coûteux en ressources pour être résolus directement par une approche globale qui tient compte de toutes les données à la fois. L'essence d'une approche "Diviser pour régner" est le découpage d'un problème complexe en sous-problèmes plus simples qui peuvent être résolus de manière indépendante. Néanmoins, cette stratégie n'est pas nécessairement adaptée à tout type de problèmes puisqu'il faut pouvoir le formuler comme un ensemble équivalent de sous-problèmes plus petits et plus faciles à résoudre. De plus, les données d'entrée doivent avoir une certaine cohérence locale pour pouvoir être divisées et utilisées indépendamment pour chaque sous-problème.

Concernant le temps d'exécution, les approches out-of-core doivent tenir compte de plusieurs facteurs. En premier lieu, le coût d'organiser et de diviser l'entrée ne doit pas dépasser le coût de son traitement par un algorithme global. Ensuite, il faut considérer que diviser un problème en sous-problèmes implique de rassembler les résultats des calculs et de les combiner afin de produire la solution globale. En conséquence, il est important que la stratégie choisie tienne compte du coût en mémoire et de la complexité d'assembler les sorties de chaque sous-problème.

Dans le contexte présent, notre entrée consiste en des volumes de grande taille. Le premier inconvénient pour traiter ces volumes est l'impossibilité de les charger entièrement en mémoire. Néanmoins, de par la cohérence spatiale naturelle des données volumiques qui nous intéressent, une subdivision des données d'entrée peut être une bonne réponse à cette problématique. Ainsi, nous proposons de combiner la subdivision avec une approche basée sur l'idée de "Streaming" afin de traiter progressivement les données d'entrée. Ensuite, la même organisation des données d'entrée peut être utilisée pour rassembler progressivement les morceaux de surface afin d'obtenir un mail-

lage unique. Cet aspect est particulièrement important dans un contexte d'extraction de surfaces parce que le résultat doit vérifier certaines propriétés globales.

Nous avons besoin de définir les conditions pour que la méthode puisse générer les morceaux de la surface de manière indépendante tout en fournissant l'information nécessaire pour les assembler ultérieurement. Les méthodes de division spatiale semblant les plus adéquates, nous avons décidé d'utiliser une extension de la méthode de génération de surface présentée dans les chapitres précédents. La chaîne de traitement de notre méthode d'extraction de surface est illustrée sur la figure 8.1.

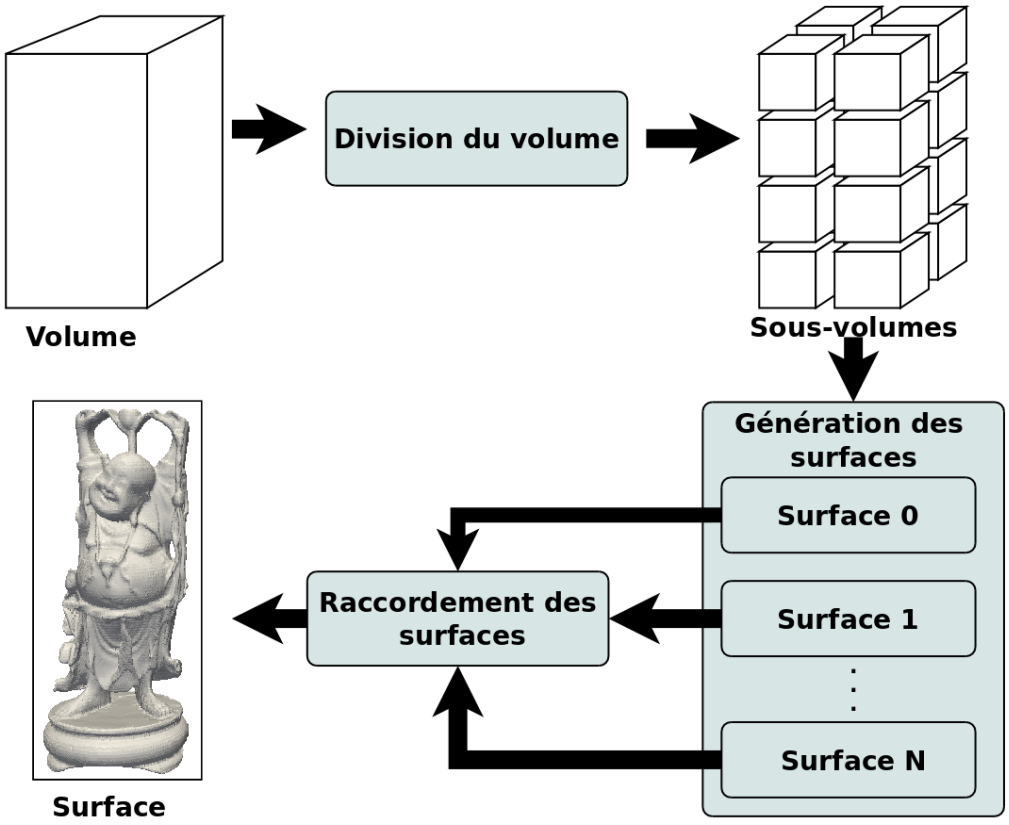



FIGURE 8.1 – Chaîne de traitement de notre algorithme out-of-core. Une fois le volume divisé en méta-cellules, celles-ci peuvent être traitées de manière indépendante pour générer des morceaux de surface. Ensuite, les morceaux sont connectés pour former la surface finale.

Comme l'illustre la figure 8.1, notre solution commence par la division du volume en sous-volumes plus petits, qui tiennent chacun en mémoire. Ensuite, nous chargeons les volumes un par un et nous générons la surface contenue à l'intérieur de chacun d'eux en stockant l'information locale nécessaire pour l'assemblage. Finalement, nous utilisons un parcours prédéfini pour explorer les surfaces générées et construire les polygones qui vont les connecter. Dans les sections suivantes nous allons expliquer plus en détail les étapes de notre méthode.

## 8.1 Considérations sur la division des données

Notre méthode commence par diviser le volume $V$ en sous-volumes que nous appellerons *méta-cellules* et que nous définirons de la manière suivante :

**Définition 1. Méta-cellule :** *un ensemble de voxels organisés de manière structurée dans un parallélépipède aligné sur les axes principaux (voir figure 8.2a). Une méta-cellule contient six méta-facettes et chacune est composée des voxels contenus sur les facettes de la méta-cellule (figure 8.2b). Une méta-arête est l'ensemble de voxels contenus dans une arête de la méta-cellule comme l'illustre la figure 8.2c) et il existe douze méta-arêtes dans une méta-cellule. Un méta-sommet est le voxel contenu dans un sommet extrême de la méta-cellule (voir figure 8.2d).*

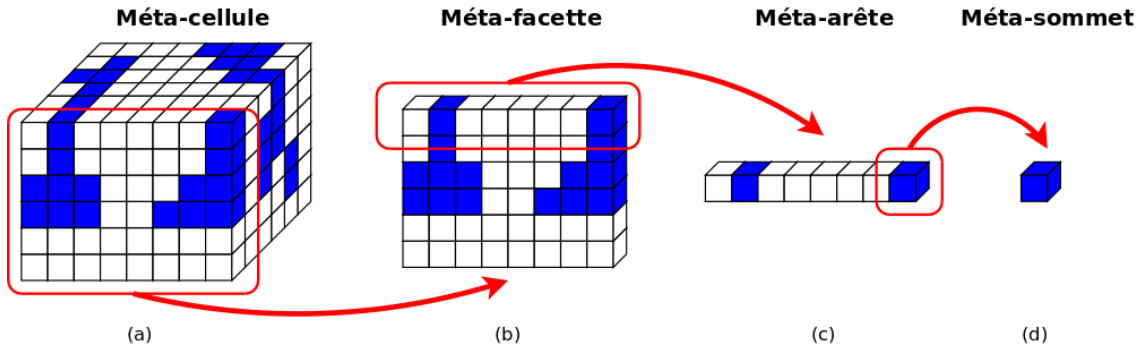



FIGURE 8.2 – (a) Description d'une méta-cellule comme un ensemble de voxels organisés de manière structurée dans un parallélépipède . Une méta-cellule est composée des 6 méta-facettes (b), 12 méta-arêtes (c) et 8 méta-sommets (d).

Les dimensions de chaque méta-cellule doivent garantir un compromis entre une taille qui peut tenir en mémoire et un nombre réduit de divisions. Cet équilibre permettra de minimiser l'impact de l'algorithme de raccordement de la surface sur l'ensemble de la solution.

Le volume peut être divisé en autant des méta-cellules que nécessaire. Celles-ci ne doivent pas obligatoirement avoir la même taille et des méta-cellules voisines peuvent avoir des dimensions très différentes. Néanmoins, les méta-cellules doivent remplir plusieurs conditions afin de permettre à notre algorithme d'avoir une information complète pendant la génération de la surface.

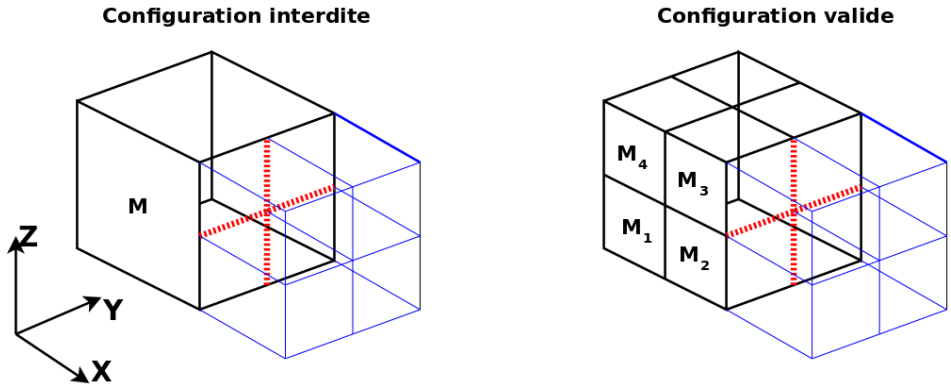



FIGURE 8.3 – Division d'un volume en méta-cellules. À gauche, la subdivision n'est pas conforme parce que l'intersection de la méta-cellule $M$ et des méta-cellules plus fines n'est pas une méta-facette pour toutes les méta-cellules. À droite, la division de la cellule $M$ en méta-cellules $M_1, M_2, ..., M_8$ fait que, même si toutes les méta-cellules n'ont pas nécessairement la même taille sur l'axe $X$, la subdivision est conforme.

La première de ces conditions est que la subdivision produite doit être conforme selon la définition suivante :

**Définition 2. Division conforme :** *Une division conforme produit un ensemble de méta-cellules $C$ tel que pour toute paire de méta-cellules $c_1$ et $c_2$ appartenant à $C$, son intersection est toujours :*
*– une méta-cellule pour le cas $c_1 = c_2$ ;*
*– une méta-facette, méta-arête ou méta-sommet si $c_1$ et $c_2$ sont adjacentes ;*
*– vide si $c_1$ et $c_2$ ne sont pas adjacentes.*

Comme l'illustre la figure 8.3, nous ne pouvons pas avoir de méta-cellules fines qui partagent seulement une partie de la méta-facette d'une méta-cellule plus grande (voir la figure 8.3 à gauche). Cependant, il est possible d'avoir des méta-cellules avec des tailles différentes sur un des axes mais qui partagent une méta-facette dans les deux autres dimensions comme l'illustre la figure 8.3 à droite.

La deuxième condition est que les méta-cellules ne doivent pas former une division disjointe des voxels mais elles doivent partager une partie de leur information avec les cellules voisines. Ainsi, deux méta-cellules adjacentes par une méta-facette doivent partager les voxels contenus dans cette méta-facette, les quatre méta-cellules adjacentes par une méta-arête doivent partager les voxels de la méta-arête et les huit méta-cellules partageant un méta-sommet doivent partager le voxel contenu dans le méta-sommet. Ce recouvrement entre méta-cellules pendant la division nous permettra ultérieurement de connecter les morceaux de surface construits séparément. La figure 8.4 illustre une méta-facette partagée par deux méta-cellules.

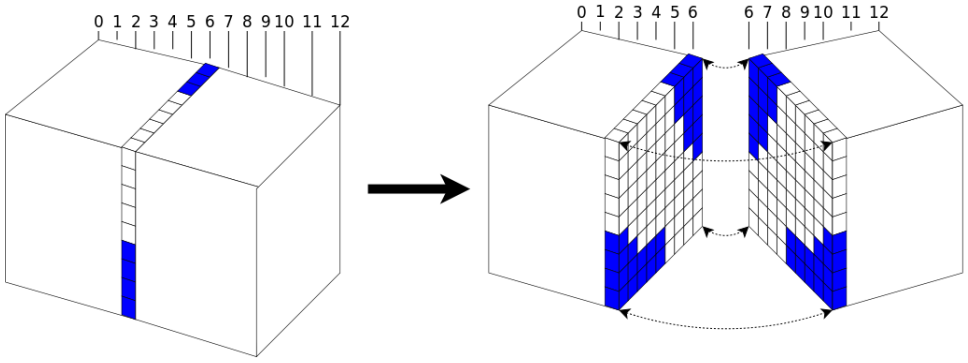



FIGURE 8.4 – Illustration d'une méta-facette dans la position 6 partagée par deux méta-cellules (à gauche). Seuls les voxels sur la facette partagée sont visibles.

Une autre caractéristique souhaitée mais pas nécessaire est que la subdivision produise des méta-cellules les plus isotropiques possible. Ainsi, les surfaces approcheront mieux les caractéristiques fines du volume à l'intérieur de la méta-cellule sans introduire d'artefact visuel. Une autre alternative consiste à favoriser la direction dans laquelle la surface présente le plus de détails avec une subdivision plus fine dans cette direction.

## 8.2 Algorithme modifié pour l'extraction des morceaux de surface

L'un des principaux problèmes pour un algorithme d'extraction de surface out-of-core est l'assemblage des morceaux de surface générés de manière indépendante. Pour cela, l'algorithme doit garantir à la fois que la surface produite à l'intérieur de la méta-cellule est correcte et que les surfaces

peuvent être connectées sans avoir besoin de charger toutes les surfaces ou les sous-volumes en mémoire. Dans ce but, nous avons modifié l'algorithme d'extraction de surfaces présenté dans la première partie de cette thèse. Une raison pour utiliser cet algorithme est que son approche de division spatiale basée sur un octree nous permet de générer une surface adaptative à l'intérieur de chaque méta-cellule. De plus, comme notre algorithme est capable de détecter les configurations complexes sur les facettes des cellules, nous pouvons utiliser cette caractéristique pour nous assurer que les cellules de l'octree qui touchent les méta-facettes ne sont pas complexes (voir page 40). Si cette condition est remplie, nous pourrons utiliser l'information contenue dans les octrees afin de bien connecter les sommets duaux entre les cellules de différentes méta-cellules. Toute l'information nécessaire pour connecter les surfaces entre elles est contenue dans les cellules adjacentes aux méta-facettes des méta-cellules.

Afin d'extraire toutes les cellules adjacentes aux méta-facettes, nous avons défini une orientation et une numérotation en lien avec le code de Morton sur toutes les cellules des octrees et des facettes des méta-cellules. Cette numérotation est illustrée sur la figure 8.5.

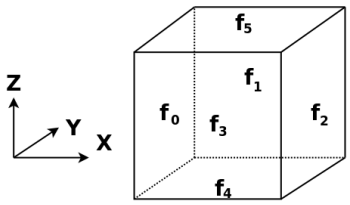



FIGURE 8.5 – Indice des méta-facettes des méta-cellules par rapport aux coordonnées spatiales.

Une procédure pour obtenir les cellules adjacentes à chaque facette d'une méta-cellule à été déjà présentée dans la page 48. Le tableau 8.1 montre la position et la valeur du bit dans le code de Morton qui permet d'identifier les cellules adjacentes aux facettes de la méta-cellule. En utilisant ce tableau, nous pourrons trouver et stocker toutes les cellules de l'octree qui seront nécessaires au moment de faire le raccordement de la surface. Nous considérons $b_3^3 b_2^3 b_1^3\ b_3^2 b_2^2 b_1^2\ b_3^1 b_2^1 b_1^1$ comme le code de Morton en 3D d'une cellule avec le bit le plus à droite comme le moins significatif. L'exposant représente la profondeur et l'indice la position du bit à chaque niveau du code. Ainsi, le code $b_i^n$ représente la position de la cellule à la profondeur $n$ et par rapport à l'axe $i$ où $i = 1$ pour l'axe $X$, , $i = 2$ pour l'axe $Y$ et $i = 3$ pour l'axe $Z$. Par exemple, $b_1^2$ est la valeur (1 ou 0) du bit pour un code à la profondeur 2 et la position relative de la cellule par rapport à l'axe $X$.

| **Facette** | **Direction** | **Position du bit (Axe de référence)** | $b_i^n$ |
|---|---|---|---|
| $f_0$ | $X$ négatif | 1 | 0 |
| $f_2$ | $X$ positif | 1 | 1 |
| $f_3$ | $Y$ négatif | 2 | 0 |
| $f_1$ | $Y$ positif | 2 | 1 |
| $f_4$ | $Z$ négatif | 3 | 0 |
| $f_5$ | $Z$ positif | 3 | 1 |

Tableau 8.1 – Direction, position et valeur du bit $b_i^n$ dans le code de Morton pour les cellules adjacentes à chacune des facettes d'une méta-cellule par rapport à l'index de la facette.

Pour obtenir les cellules adjacentes à une méta-facette, nous devons parcourir l'octree récursivement de manière à traverser seulement les cellules dont le bit $b_i^n$, $(n = 0, ...N$, $N$ étant la profondeur de

l'octree) à la position $i$, possède la valeur indiquée sur le tableau 8.1 par rapport à la méta-facette qui nous intéresse. Par exemple, dans le cas de la méta-facette avec l'index $f_2$ (vers le sens positif de l'axe $X$), nous devons uniquement tenir compte des cellules avec un code de Morton dans le premier bit à 1 à chaque niveau ; par exemple, la cellule dont le code est 00**1** 01**1** 11**1** est illustrée en orange sur la figure 8.6.

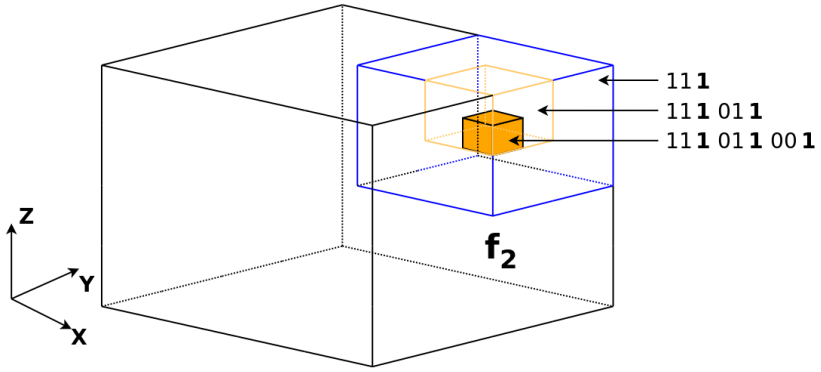



FIGURE 8.6 – Cellules explorées pour trouver la cellule avec le code de Morton 11**1** 01**1** 00**1** et adjacente à la facette $f_2$ dans la direction positive de l'axe $X$.

Il est nécessaire de stocker toutes les cellules qui touchent les facettes de la méta-cellule. Néanmoins, le nombre de ces cellules reste très limité par rapport au nombre total de cellules contenues dans l'octree. Cette procédure est décrite plus en détail dans l'algorithme *1*.

**Algorithme 1:** Algorithme pour l'extraction des cellules adjacentes.

**Entrées** : Cellule de l'octree $R$ (initialement la cellule racine). Indice de la facette $i$ (obtenu comme indiqué dans le tableau 8.1).
**Sorties** : Liste des cellules $L$ adjacentes à la facette $i$.
Initialisation : positionBit ← `PositionBit` ($i$)
Initialisation : valeurBit ← `ValeurBit` ($i$)
**tant que** `CellulePasFeuille` ($c$) **faire**
  S ← `SousCellules` ($c$)
  **pour chaque** $s \in S$ **faire**
    bits ← `TroisDernierBits` ($s$)
    bit ← `BitParPosition` (bits,positionBit)
    **si** *bit* = *valeurBit* **alors**
      **si** `CelluleEstFeuille` ($s$) **alors**
        L ← `AjouterCellule` ($s$)
      **fin**
      **sinon**
        `DetecterCellulesParFacette` ($s$)
      **fin**
    **fin**
  **fin**
**fin**

La numérotation et l'algorithme présenté sont appliqués sur chacune des méta-facettes de chaque méta-cellule et les cellules obtenues doivent être stockées afin d'être utilisées dans le raccordement final de la surface qui sera décrit dans la section suivante.

## 8.3 Raccordement de la surface

### 8.3.1 Parcours de raccordement

Une fois que nous avons généré les morceaux de surface à l'intérieur de chaque méta-cellule, nous devons les rassembler afin de former une surface unique. Pour cela, nous devons tenir compte de la localisation spatiale de chaque méta-cellule par rapport aux méta-cellules voisines. La localisation de chaque méta-cellule est indexée avec les coordonnées de sa position dans la grille utilisée pour diviser le volume. Cela va nous permettre de définir un ordre simple de parcours que nous illustrons sur la figure 8.7 et qui commence avec la méta-cellule placée à l'origine (position $(0, 0, 0)$) et balaye le volume selon la direction $X$, puis selon la direction $Y$ et finalement selon la direction $Z$.

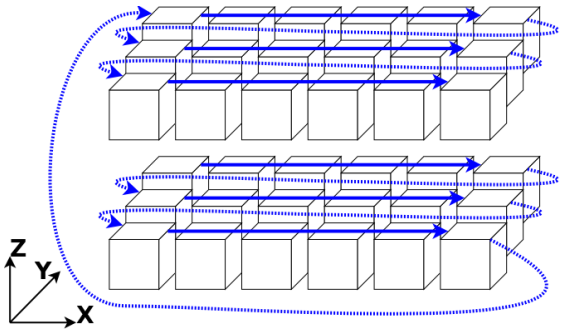



FIGURE 8.7 – Ordre de parcours des méta-cellules par rapport à l'orientation du volume.

Afin de pouvoir relier les surfaces, il est nécessaire d'identifier les endroits où la surface coupe les facettes des méta-cellules. C'est possible si nous exploitons le fait que notre algorithme (décrit dans le chapitre 4) génère des polygones seulement à l'intérieur des cellules qui partagent une arête minimale traversant la surface $\partial V$. Par conséquent, nous devons considérer uniquement les arêtes contenues sur la méta-facette d'une méta-cellule. Pour cela, nous n'avons pas besoin d'explorer toutes les cellules de l'octree mais seulement les cellules adjacentes à la méta-facette considérée. Nous disposons de ces cellules parce qu'elles ont été stockées pour chaque méta-cellule dans l'étape de génération de surface.

Néanmoins, il ne suffit pas d'identifier les arêtes pour compléter la surface parce que pour générer des polygones, notre algorithme a besoin de toutes les cellules qui partagent l'arête. Ces cellules se trouvent dans des méta-cellules adjacentes, il est donc nécessaire de construire des structures intermédiaires afin de pouvoir connecter les surfaces.

### 8.3.2 Face-octrees pour des volumes divisés dans une direction

Dans le cas où le volume a été divisé dans une seule direction, les arêtes à considérer sont contenues entièrement à l'intérieur des méta-facettes. Nous avons donc seulement besoin de l'information des deux méta-cellules adjacentes à la méta-facette concernée. Pour construire la structure des données intermédiaire appelée *Face-Octree*, nous utilisons les cellules adjacentes à la méta-facette. Cette octree est implémenté de telle manière que nous n'avons besoin de stocker que des cellules feuilles et pas les cellules intermédiaires (octree linéaire). Cette caractéristique permet de minimiser la quantité de mémoire utilisée. Néanmoins, il ne suffit pas de rassembler les cellules dans un nouvel

octree, il est aussi nécessaire de rétablir la localisation spatiale et hiérarchique existante entre toutes les cellules. Pour chaque cellule, cette localisation est encodée implicitement dans les codes de Morton mais le fait de combiner les cellules provenant des méta-cellules différentes ne respecte plus ce codage. Pour résoudre cela, nous devons changer les codes de Morton de toutes les cellules afin de refléter leur localisation spatiale dans la nouvelle structure. Cela revient à ajouter un niveau supplémentaire de profondeur dans le code de Morton pour toutes les cellules.

| **Préfixes des codes de Morton** | | |
|---|---|---|
| **Cas de parcours** | **Méta-cellules** | **Code de Morton** |
| Facette perpendiculaire à $X$ | $B_{i,j,k}$ | 00**0** |
| | $B_{i+1,j,k}$ | 00**1** |
| Facette perpendiculaire à $Y$ | $B_{i,j,k}$ | 0**0**0 |
| | $B_{i,j+1,k}$ | 0**1**0 |
| Facette perpendiculaire à $Z$ | $B_{i,j,k}$ | **0**00 |
| | $B_{i,j,k+1}$ | **1**00 |

Tableau 8.2 – Préfixes à ajouter aux cellules dans chaque méta-cellule afin de construire les face-trees par rapport aux orientations des méta-facettes.

Comme nous travaillons en trois dimensions, trois bits doivent être ajoutés au début des codes de chaque cellule. Les trois bits à ajouter vont dépendre de la méta-cellule à laquelle elles appartiennent. Comme dans notre solution nous utilisons le parcours illustré sur la figure 8.9, nous allons toujours considérer la méta-cellule placée dans la position $(i, j, k)$ comme le repère pour la localisation spatiale à appliquer sur les cellules du Face-octree. Ainsi, le tableau 8.2 montre les configurations de bits qu'il faut ajouter afin d'établir la localisation de chaque cellule dans la nouvelle structure.

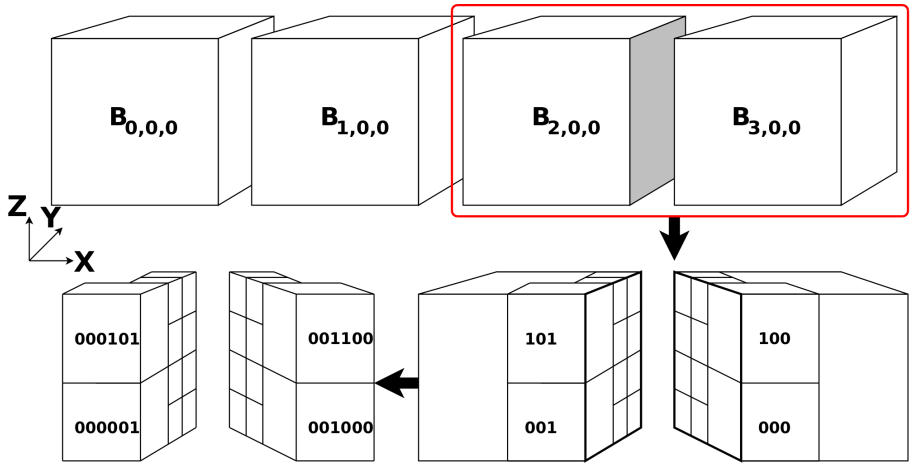



FIGURE 8.8 – Un volume allongé divisé en quatre méta-cellules ($B_{0,0,0}$ à $B_{3,0,0}$) construit des Face-octrees entre les méta-cellules qui partagent une méta-facette (en gris entre $B_{2,0,0}$ et $B_{3,0,0}$) comme illustré ci-dessus. Les codes de Morton des cellules sont modifiés afin de refléter leur localisation dans la nouvelle structure.

Les Face-octrees sont des structures relatives et au fur et à mesure que nous avançons dans le parcours du volume, la cellule courante sera toujours considérée comme étant le repère d'origine. La figure 8.8 illustre la construction d'un Face-octree à partir de deux méta-cellules $B_{2,0,0}$ et $B_{3,0,0}$. En premier lieu, les cellules des octrees sont rassemblées (en bas à droite) et finalement les codes de Morton sont modifiés en ajoutant les préfixes listés dans le tableau 8.2 pour retrouver une

cohérence spatiale et hiérarchique des cellules à l'intérieur du Face-octree.

Dans le cas illustré sur la figure 8.8, seules les cellules les plus grandes sont prises comme exemple. Une cellule avec un code de Morton de 101 sur la méta-cellule $B_{2,0,0}$ appartient à la cellule repère $(B_{2,0,0})$ et son code de Morton changera en **000** 101 afin de refléter sa nouvelle position spatiale dans le Face-octree. Par contre, la cellule avec le code 100 dans la méta-cellule $B_{3,0,0}$ deviendra **001** 100 pour que toutes les cellules à l'intérieur de $B_{3,0,0}$ soient à droite des cellules à l'intérieur de $B_{2,0,0}$.

### 8.3.3 Edge-octrees pour des volumes génériques

Dans le cas où le volume est divisé dans plus d'une direction, les arêtes intersectant la surface peuvent être contenues dans une méta-arête. Nous avons donc besoin de l'information de quatre méta-cellules afin de pouvoir connecter entièrement la surface. Dans le but de couvrir toutes les configurations possibles, nous avons identifié l'ensemble de méta-arêtes et des méta-facettes que nous devons explorer afin de pouvoir connecter complètement la surface. Ces configurations sont illustrées sur la figure 8.9.

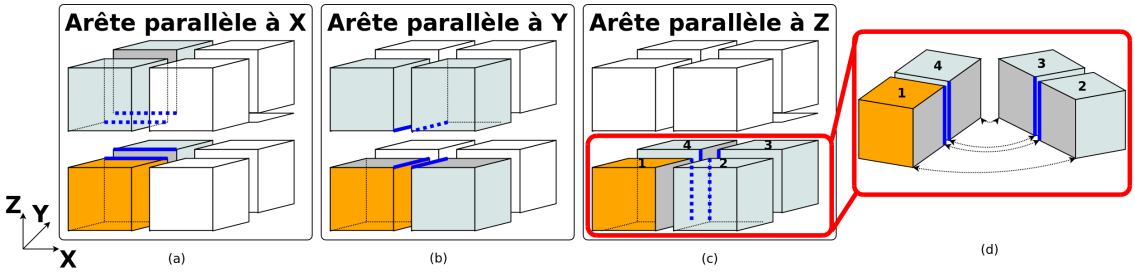



FIGURE 8.9 – Configurations des méta-arêtes (en gras) à utiliser pour connecter les surfaces. La méta-cellule plus foncée est utilisée comme repère pour construire l'edge-octree. Les méta-facettes à considérer sont celles sur le plan $XZ$ pour la méta-arête alignée sur $X$ (a), sur le plan $XY$ pour la méta-arête alignée sur $Y$ (b) et sur le plan $YZ$ pour la méta-arête alignée sur $Z$ (c). Un zoom sur la méta-arête alignée sur $Z$ partagée par les quatre méta-cellules (d).

La figure 8.9 montre que nous devons considérer trois méta-arêtes, selon $X$, $Y$ et $Z$, et les méta-facettes qui la partagent sur les plans $XZ$, $XY$ et $YZ$ respectivement. Nous devons pouvoir accéder à toutes les cellules adjacentes aux méta-facettes concernées. Pour le faire d'une manière efficace, nous avons conçu une autre structure intermédiaire que nous appellerons *Edge-octree*. Il s'agit d'un octree composé de toutes les cellules des quatre méta-cellules qui partagent une méta-arête. De la même manière que le Face-octree, les Edge-octrees contiennent seulement des cellules feuilles. Une fois que nous avons rassemblé les cellules dans l'edge-octree, nous devons appliquer une extension de la technique utilisée avec les Face-octrees. Les préfixes à ajouter aux cellules sont listés dans le tableau 8.3 en prenant, à nouveau, la cellule dans la position $(i, j, k)$ comme repère.

Par exemple, à partir d'une méta-cellule $B_{i,j,k}$ et dans le cas d'une méta-arête orientée selon l'axe $Z$, nous devons utiliser les méta-cellules $B_{i,j,k}$, $B_{i+1,j,k}$, $B_{i,j+1,k}$ et $B_{i+1,j+1,k}$. L'addition des préfixes du tableau 8.3 aura pour effet de remplacer les méta-cellules par des cellules et d'augmenter de 1 la profondeur représentée par ses codes de Morton. Pour les deux méta-cellules $B_{i,j,k}$ et $B_{i+1,j,k}$ , les bits qu'il faut ajouter au code de Morton de chaque cellule sont 00**0** et 00**1** respectivement. Géométriquement, cela implique que toutes les cellules en $B_{i,j,k}$ seront à l'intérieur d'une cellule

| Préfixes des codes de Morton | | |
|---|---|---|
| **Cas de parcours** | **Méta-cellules** | **Code de Morton** |
| Arête parallèle à $X$ | $B_{i,j,k}$ | 000 |
| | $B_{i,j+1,k}$ | 0**1**0 |
| | $B_{i,j,k+1}$ | **1**00 |
| | $B_{i,j+1,k+1}$ | **11**0 |
| Arête parallèle à $Y$ | $B_{i,j,k}$ | 000 |
| | $B_{i+1,j,k}$ | 00**1** |
| | $B_{i,j,k+1}$ | **1**00 |
| | $B_{i+1,j,k+1}$ | **1**0**1** |
| Arête parallèle à $Z$ | $B_{i,j,k}$ | 000 |
| | $B_{i+1,j,k}$ | 00**1** |
| | $B_{i,j+1,k}$ | 0**1**0 |
| | $B_{i+1,j+1,k}$ | 0**11** |

Tableau 8.3 – Préfixes à ajouter aux cellules dans chaque méta-cellule afin de construire les edge-trees par rapport aux orientations des méta-arêtes.

placée relativement dans la position $(0, 0, 0)$ et les cellules en $B_{i+1,j,k}$ seront à l'intérieur d'une cellule placée dans la position $(1, 0, 0)$. En conséquence, pour une cellule $c$ en $B_{i+1,j,k}$ avec un code de Morton 1 110 000 100 110 nous obtenons un code 1 **001** 110 000 100 110.

L'utilisation de cette technique est illustrée en $3D$ sur la figure 8.10 où la méta-arête centrale (en gras) est utilisée comme référence pour construire un edge-octree. En (a), les cellules adjacentes à la méta-facette sont stockées indépendamment dans chaque méta-cellule. En (b), les cellules sont combinées dans un nouvel octree et ses codes de Morton sont modifiés pour refléter sa position dans la nouvelle structure.

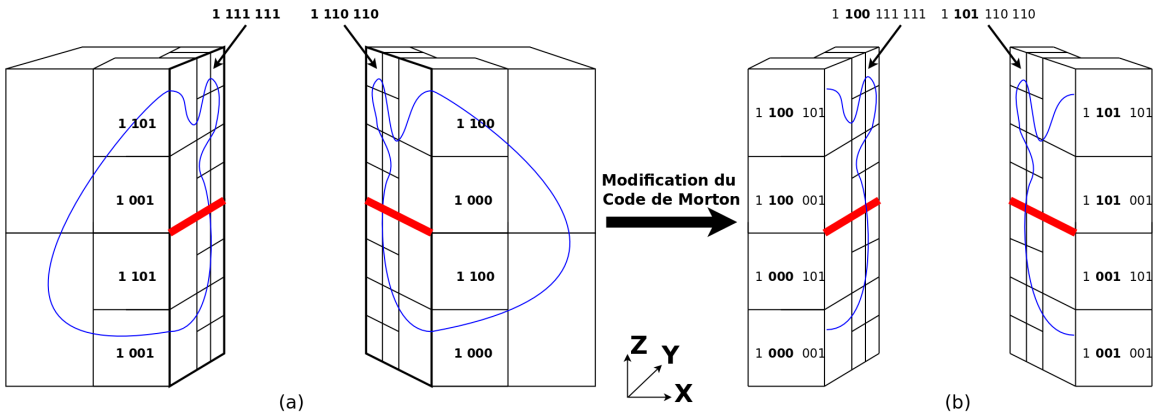



FIGURE 8.10 – Illustration de la modification des codes de Morton dans la construction d'un edge-octree. En (a), la méta-arête épaisse en rouge est utilisée pour identifier les cellules qui feront partie de l'edge-octree. En (b), les codes de Morton des cellules adjacentes à la méta-arête sont modifiés (en gras) pour refléter leur position spatiale et hiérarchique dans le nouveau edge-octree.

Il existe un cas particulier dans la construction de ces cellules intermédiaires, il concerne les méta-arêtes localisées sur les limites du volume original. Elle ne sont pas considérées parce qu'elles ne peuvent pas être traitées du fait qu'il doit exister au moins trois cellules pour générer un polygone et que cette condition n'est pas remplie.

### 8.3.4 Génération de polygones

Une fois le Face-octree ou l'edge-octree construit, nous devons connecter les sommets duaux contenus à l'intérieur des cellules. L'algorithme proposé par Dual Contouring a, en principe, besoin de toute la hiérarchie de l'octree mais, dans le but de minimiser la quantité de mémoire nécessaire pour construire les structures intermédiaires, nous avons fait le choix de stocker seulement les cellules feuilles. Il y a donc fallu développer un algorithme alternatif qui n'a pas besoin de parcourir toute la hiérarchie de l'octree et utilise seulement les cellules feuilles pour générer les polygones de la surface.

Un parcours la structure intermédiaire permet d'identifier les cellules qui sont intersectées par la surface $\partial V$. Pour chacune, nous parcourons toutes les arêtes qui coupent la surface et identifions les cellules qui partagent cette arête. Pour avoir un accès rapide aux cellules voisines par leur code de Morton, nous devons identifier l'orientation de chaque arête par rapport au repère de la cellule. Nous avons attribué un indice aux arêtes des cellules afin de savoir quel est l'offset qu'il faut utiliser au moment d'obtenir les cellules voisines, voir illustration sur la figure 8.11.

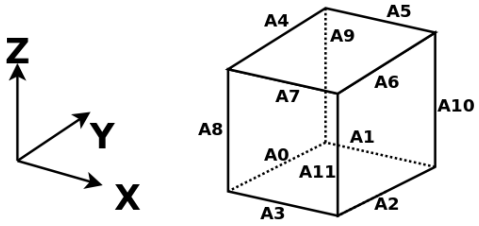



FIGURE 8.11 – Numérotation des arêtes de la cellule par rapport au repère du volume.

A chaque arête de la cellule, correspond un offset (tableau 8.4) pour obtenir les cellules voisines. Comme nous l'avons fait dans l'algorithme de génération de surface, nous devons convertir le code de Morton de la cellule courante vers ses coordonnées spatiales, ajouter l'offset correspondant puis en déduire le code de Morton de la cellule voisine. Les offsets dans le tableau 8.4 sont présentés dans un ordre cohérent afin d'assurer que les polygones ont la bonne orientation par rapport à l'intérieur et l'extérieur de la surface finale.

| **Arête** | **Offset de cellules** | | |
|---|---|---|---|
| A0 | $(0,0,-1)$ | $(-1,0,-1)$ | $(-1,0,0)$ |
| A1 | $(0,-1,0)$ | $(0,-1,-1)$ | $(0,0,-1)$ |
| A2 | $(1,0,0)$ | $(1,0,-1)$ | $(0,0,-1)$ |
| A3 | $(0,0,-1)$ | $(0,1,-1)$ | $(0,1,0)$ |
| A4 | $(-1,0,0)$ | $(-1,0,1)$ | $(0,0,1)$ |
| A5 | $(0,0,1)$ | $(0,-1,1)$ | $(0,-1,0)$ |
| A6 | $(0,0,1)$ | $(1,0,1)$ | $(1,0,0)$ |
| A7 | $(0,1,0)$ | $(0,1,1)$ | $(0,0,1)$ |
| A8 | $(-1,0,0)$ | $(-1,-1,0)$ | $(0,-1,0)$ |
| A9 | $(0,-1,0)$ | $(1,-1,0)$ | $(1,0,0)$ |
| A10 | $(1,0,0)$ | $(1,1,0)$ | $(0,1,0)$ |
| A11 | $(0,1,0)$ | $(-1,1,0)$ | $(-1,0,0)$ |

Tableau 8.4 – Offsets à utiliser pour déterminer les cellules voisines d'une cellule et qui partagent l'arête correspondante. Les offsets sont listés toujours dans le même ordre pour assurer la bonne orientation de la surface finale.

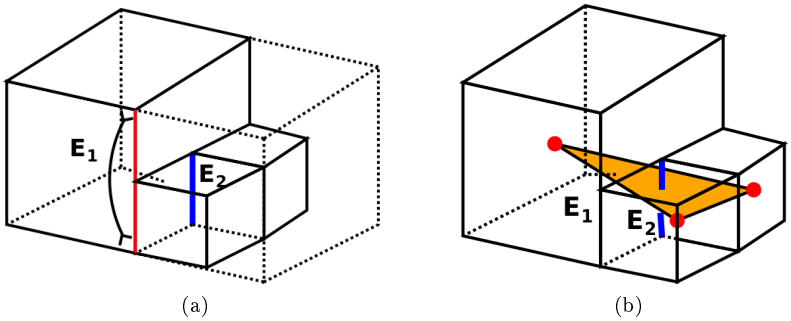
Une fois que les cellules partageant une arête ont été obtenues, nous devons chercher le sommet dual lié à l'arête. Cette étape a déjà été expliquée dans la section 4.3 et ne pose pas de problème. La procédure générale pour générer les polygones à partir des structures intermédiaires est décrite dans l'algorithme 2.

**Algorithme 2:** Algorithme pour la génération des polygones à partir des structures intermédiaires.

**Entrées** : Liste des cellules de l'octree $L$.
**Sorties** : Polygones entre les sommets duaux des cellules dans $L$.

```
pour chaque c ∈ L faire
    si IntersecteLaSurface (c) alors
        pour chaque i ∈ 0 1 2 3 4 5 ... 11 faire
            a ← FacetteParIndex (i) ;
            si IntersecteLaSurface (a) alors
                indexes ← IndexesParArete (i) ;
                listeCellulesAdjacentes ← AjouterCellule (c) ;
                pour chaque j ∈ 0 1 2 faire
                    listeCellulesAdjacentes ← AjouterCellule (CelluleVoisineParIndex (c,j)) ;
                fin
                GenererPolygones (listeCellulesAdjacentes) ;
            fin
        fin
    fin
fin
```

Il faut noter que l'on ne peut générer des polygones que lorsque toutes les cellules adjacentes sont des cellules feuilles. Par ailleurs, pour pouvoir obtenir toutes les cellules adjacentes à une arête, nous devons nous trouver sur l'arête intersectée et minimale. En effet, à partir d'une cellule au niveau $n$, il est plus efficace de détecter des cellules aux niveaux supérieurs (en réalisant une opération de décalage de 3 bits) de la hiérarchie qu'aux niveaux inférieurs où il faut choisir entre plusieurs cellules filles partageant une partie de l'arête en question (voir figure 8.12a).




(a) (b)

FIGURE 8.12 – (a) La recherche des cellules adjacentes qui partagent une arête à partir d'une arête non minimale oblige à parcourir toutes les cellules plus petites qui partagent l'arête de départ. (b) En utilisant une arête minimale, il est possible d'utiliser le code de Morton de la cellule adjacente (même si elle n'existe pas). Des décalages de trois bits sur ce code permettent de monter dans la hiérarchie et de trouver la cellule qui partage l'arête de départ. Les sommets duaux (cercles) peuvent alors être connectés.

Dans la procédure expliquée ci-dessus nous prenons toujours la cellule courante, de coordonnées $(i, j, k)$, comme le point d'origine du repère. La méthode pour connecter la surface doit progres-

sivement parcourir toutes les méta-cellules, explorer toutes les méta-arêtes dans les trois directions et construire les structures intermédiaires pour chacune, générer les polygones avec les sommets duaux et puis les stocker avant de décharger les structures de la mémoire et continuer avec la méta-cellule suivante. Le mécanisme est illustré sur la figure

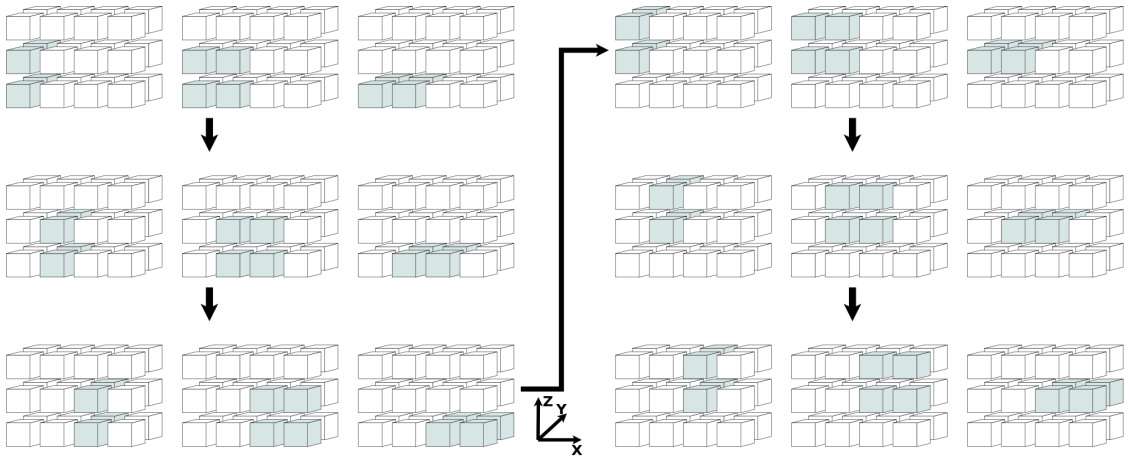



FIGURE 8.13 – Parcours de raccordement suivi par notre algorithme. Seules les structures intermédiaires (Edge-octree ou Face-octree) des méta-cellules concernées (en gris) à chaque raccordement sont chargées en mémoire.

Enfin, quand toutes les facettes ont été générées, elles sont rassemblées dans un fichier unique pour construire la surface.

## 8.4 Extensions et perspectives

### 8.4.1 Parallélisation

L'algorithme présenté ci-dessus a toutes les caractéristiques nécessaires pour être parallélisé. Les données d'entrée peuvent être stockées de manière séparée, l'algorithme d'extraction de surface a uniquement besoin de l'information à l'intérieur de la méta-cellule et les informations nécessaires pour connecter les morceaux de surface peuvent être obtenues et stockées de manière indépendante. Seule l'étape de raccordement n'est pas entièrement parallélisable puisque, pour raccorder deux morceaux de surface, nous avons besoin de charger les informations des quatre méta-cellules adjacentes en mémoire. Mais, selon la subdivision choisie pour le volume, la phase de raccordement peut être légère en consommation mémoire et très rapide.

Une alternative de pseudo-parallélisation est d'utiliser des processus léger ("thread") pour profiter des multiples cœurs fournis par les processeurs actuels. Cette alternative a l'avantage d'une implémentation facile et presque directe à partir d'une implémentation séquentielle. De plus, comme les processus légers partagent un même espace mémoire, la communication des données entre eux est automatique et facile à gérer, ce qui peut grandement accélérer l'étape de raccordement. Malheureusement, cette facilité d'échange des données peut faire apparaître des problèmes dits de "faux partage" ("false sharing"). Ils se produisent quand plusieurs processus essaient d'accéder, même en lecture, au même bloc de mémoire. Ces accès concurrents peuvent faire qu'un processus reste en

attente tandis que l'autre n'a pas fini son accès. Par conséquent, une telle implémentation peut n'améliorer la vitesse d'exécution que dans un pourcentage très inférieur à ce qui serait attendu.

Une autre alternative est l'utilisation de processeurs de la plateforme de calcul parallèle de type CUDA pour utiliser les processeurs GPU en carte graphique. Cette plateforme est devenue très populaire pour certaines applications mais son utilisation imposerait une forte modification du code en bonne partie parce que le standard C++ n'est pas entièrement supporté. De plus, il serait impossible dans certains cas de garder toutes les surfaces produites en mémoire et elles devraient être stockées de manière temporaire dans des fichiers sur le disque, ce qui ralentirait fortement la construction de la surface finale.

### 8.4.2 Structure de recherche

Une manière d'incorporer les structures de recherche optimale dans notre solution peut être l'utilisation des niveaux minimal et maximal de subdivision de nos octrees. Le niveau minimal de subdivision est la profondeur jusqu'à laquelle nous allons diviser le volume de manière régulière afin de détecter ses principales caractéristiques. Le niveau maximal est strictement plus grand que le niveau minimal et dénote la profondeur maximale des cellules dans l'octree. L'adaptabilité de notre algorithme est contrainte entre ces deux niveaux dans le sens où les critères topologiques et géométriques de subdivision s'appliquent sur les cellules filles qui se trouvent entre ces deux niveaux.

Ainsi, l'utilisation du niveau minimal peut permettre d'accélérer l'accès aux cellules situées sur la surface pour une iso-valeur particulière. Un pré-traitement peut partir des cellules produites par une subdivision régulière jusqu'au niveau minimal et les organiser par rapport aux iso-valeurs dans une structure de recherche optimale définie sur l'espace de longueur à l'image de ce qui a été proposé dans la littérature. Nous pouvons alors implémenter des requêtes du type : "quelles sont les cellules intersectées par une iso-valeur donnée ?". Ce schéma peut être complété par des patrons de découpage pré-calculés pour accélérer la subdivision adaptative des cellules et un guidage par des critères topologiques ou de courbure de la surface.

Cette section a décrit les principales étapes de notre algorithme de génération de surfaces à partir de données volumiques massives. Dans la section suivante, nous allons présenter quelques résultats de son application sur un ensemble de volumes de grande taille et nous analyserons aussi ses caractéristiques et possibles applications.

# Chapitre 9

# Résultats

Cette section décrit les principaux résultats de notre solution out-of-core sur un ensemble de données volumiques. Nous avons mesuré ses performances avec plusieurs critères tels que l'approximation géométrique, la qualité des triangles produits, le temps d'exécution et la quantité de mémoire nécessaire par rapport à la taille du volume d'entrée. De plus, nous avons évalué les effets de la subdivision et du recouvrement entre méta-cellules sur la qualité générale de la surface obtenue. Finalement, nous avons mis en œuvre une stratégie de parallélisation et mesuré sa performance.

L'ensemble des données de test ont été obtenues de deux manières différentes. Une première partie des volumes ont été construites par discrétisation de surfaces. Ainsi, pour une surface $\partial V$ et un domaine $\Omega$ avec une résolution de $512^3$ voxels, cette méthode va nous fournir une fonction indicatrice discrète $F(x, y, z)$ définie sur lune grille régulière de $512^3$ voxels sur $\Omega$ telle que $F(x, y, z) = 1$ si $(x, y, z)$ (le centre du voxel) est à l'intérieur de $\partial V$ et 0 sinon. Ce type de modèle va nous permettre d'évaluer la qualité de l'approximation obtenue par rapport à la surface originale de référence.

Le deuxième ensemble de données sera constitué principalement d'empilements d'images de diverses modalités. Ces volumes vont nous servir principalement pour identifier les limitations théoriques et techniques de notre méthode et tester sa performance par rapport au temps d'exécution et à la mémoire vive utilisée.

## 9.1 Subdivision et qualité de l'approximation

Dans cette partie nous mesurons l'impact de la subdivision d'un volume et la redondance introduite dans la qualité de l'approximation obtenue. Comme notre solution génère une surface adaptative à l'intérieur de chacune des méta-cellules, la méta-facette de voxels partagée entre deux méta-cellules adjacentes va nous permettre de construire les octrees de manière indépendante en nous assurant que les zones de contact entre les deux surfaces construites et adjacentes pourront être connectées ultérieurement. Le premier de nos exemples est une surface extraite à partir d'un volume de $2048 \times 2048 \times 1024$ voxels discrétisé à partir de la surface "Happy Buddha". Le volume a été divisé en 32 méta-cellules formant une grille de $4 \times 4 \times 2$. La surface de la figure 9.3 a été obtenue en utilisant des octrees de profondeur maximale 7 et un seuil de courbure $\lambda = 0.9$. Le niveau de profondeur maximale a été choisi afin de représenter des détails ayant au moins 4 voxels d'épaisseur. Le seuil de courbure utilisé va simplifier considérablement la surface dans les zones lisses du modèle.

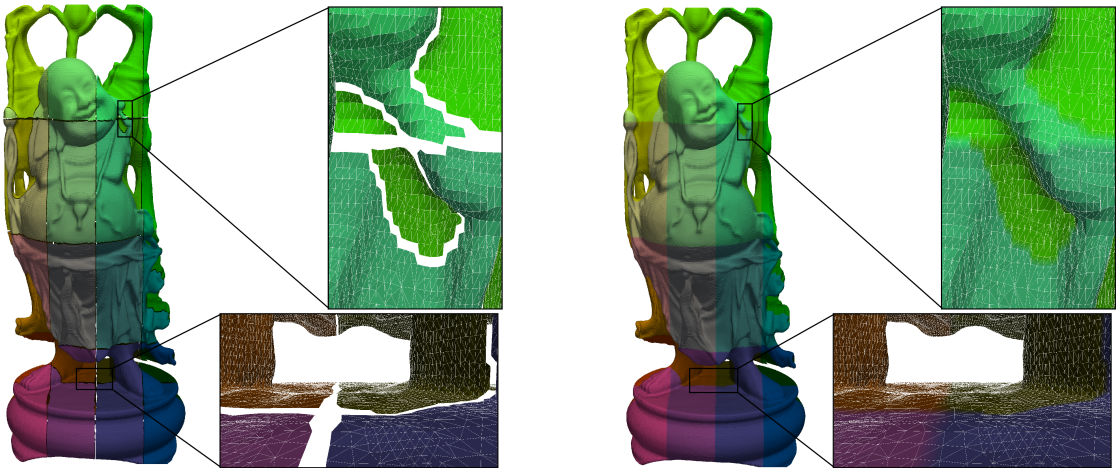


FIGURE 9.1 – Reconstruction d'un volume discrétisé de $2048 \times 2048 \times 1024$ voxels. Le volume a été divisé en 32 méta-cellules sur une grille de taille $4 \times 4 \times 2$. À gauche nous pouvons voir les morceaux de surface générés de manière indépendante et encore déconnectés. À droite nous voyons la surface après raccordement.

À gauche de la figure 9.1 nous pouvons voir les surfaces extraites à partir des méta-cellules et qui n'ont pas encore été connectées. Ces surfaces contiennent 1.916.547 facettes et 971.068 sommets et occupent 74 Mo sur le disque. À droite, nous voyons la surface connectée avec 1.969.405 facettes et 971.068 sommets pour une taille de 76 Mo sur le disque. Le nombre de sommets reste inchangé après l'étape de raccordement qui ne produit que des facettes reliant les bords des surfaces. Dans le cas de la figure 9.3, les facettes de connexion sont au nombre de 52.858 ce qui représente 2.68% des triangles construits au total. Les zooms de la figure montrent aussi que la taille des triangles est adaptée à la courbure estimée de la surface, notamment sur les triangles générés à la base du modèle. Les différentes tailles de triangles sur une zone relativement plane sont liées à l'existence de bruit dans la représentation volumique de la surface originale.

Nous avons aussi réalisé des tests sur un volume provenant du modèle "Asian Dragon" de $2048 \times 512 \times 512$ voxels. Afin d'obtenir une surface adaptative, nous avons utilisé un octree de profondeur maximale 8 et un seuil de courbure $\lambda = 0.9$. A cause de l'anisotropie de ce volume, nous l'avons divisé selon $X$ seulement. Pour l'étape de connexion de la surface entre deux méta-cellules, il a donc été nécessaire d'utiliser uniquement les facettes partagées par deux méta-cellules et de construire des *Face-octrees* comme expliqué dans la section 8.3.2. Sur la figure 9.2, nous pouvons observer les différentes surfaces obtenues à partir de chaque méta-cellule avec des couleurs différentes afin de bien identifier les zones de connexion. Les zooms sur la surface montrent des zones de contact entre les surfaces produites à l'intérieur de chaque méta-cellule avant et après les avoir connectées. Cela nous permet de vérifier que la qualité de la triangulation n'est pas affectée dans les zones de connexion. De plus, l'existence de triangles de tailles différentes confirme que les zones entre méta-cellules n'ont pas besoin de contenir des cellules de la même taille mais seulement des cellules qui ne sont pas complexes.

La figure 9.2 permet aussi de confirmer que nous n'avons pas besoin d'avoir le même niveau de résolution maximale pour les octrees sur toutes les méta-cellules mais seulement d'assurer que la topologie de la surface est simple sur toutes les méta-facettes.

Pour deux méta-cellules voisines, notre méthode peut générer des surfaces ouvertes avec des fron-

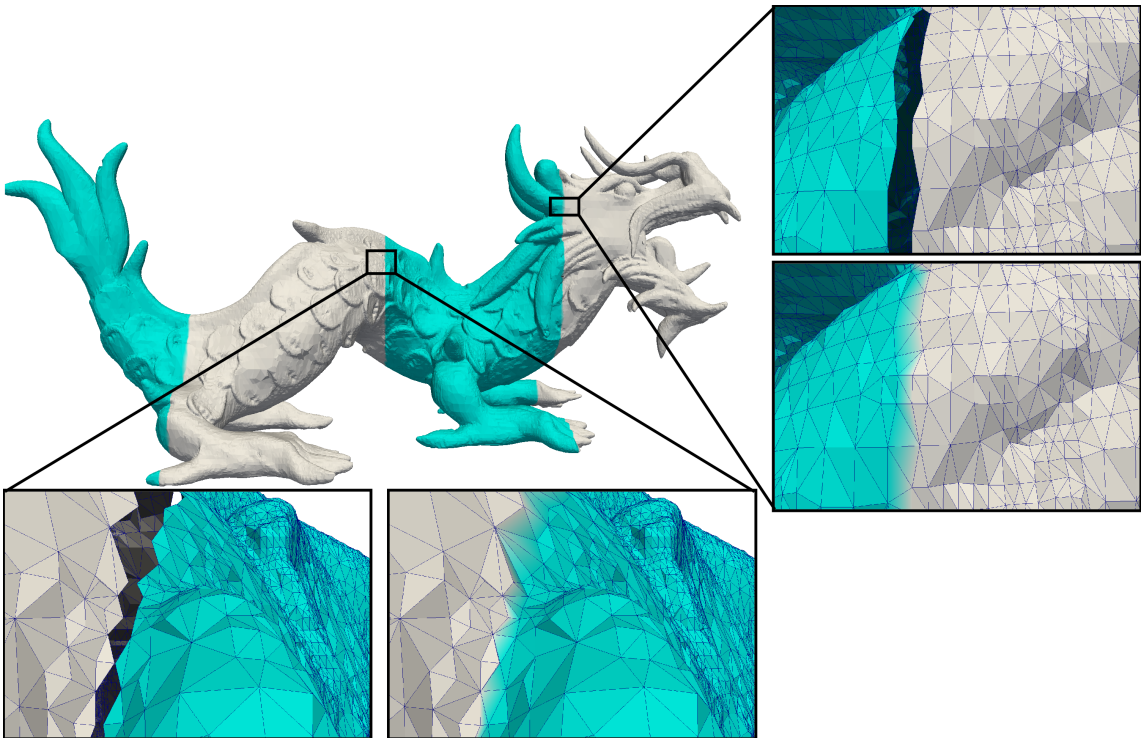


FIGURE 9.2 – Génération de la surface à partir d'un volume $2048 \times 512 \times 512$ voxels. Le volume a été divisé en 4 méta-cellules selon $X$. Les surfaces ont été extraites avec un octree de profondeur 8 et un seuil de courbure de 0.9. Les zooms permettent d'observer la qualité de la triangulation dans les zones de connexion de la surface.

tières topologiquement simples pour chaque cellule de l'octree parce que chaque méta-cellule partage la même méta-facette de voxels. Néanmoins, pour valider que notre approche n'affecte pas fortement la qualité de l'approximation, nous avons mesuré les erreurs géométriques obtenues par rapport aux modèles originaux avec plusieurs niveaux de subdivision. Les mesures utilisées sont le carré de la distance moyenne (RMS pour Root Mean Square) et la distance de Hausdorff que nous avons estimée avec la librairie METRO [33]. Les quatre modèles utilisés contiennent $1024 \times 1024 \times 1024$ voxels et les résultats des erreurs pour quatre découpages du volume ($M = 1,\ 8,\ 64$ et 512 méta-cellules) sont présentés dans le tableau 9.1. Toutes les surfaces ont été extraites à partir d'octrees de profondeur 7 pour un seuil de courbure $\lambda = 0.9$. Dans ce type de mesure il faut tenir compte de l'erreur géométrique introduite par la discrétisation de chaque surface. Cette erreur est liée au pas de discrétisation de la fonction $F(x, y, z)$ qui correspond à la taille des voxels. Cette information est aussi contenue dans le tableau 9.1.

Les valeurs d'erreur montrent à quel point la procédure de discrétisation se répercute sur l'approximation de la surface. Cela permet de mettre en perspective les erreurs d'approximation obtenues avec notre méthode. Dans le cas de l'erreur RMS, nous pouvons constater que les variations de l'erreur d'approximation selon la taille des méta-cellules reste presque toujours à l'intérieur de l'intervalle d'erreur défini par le pas de discrétisation de chaque volume. Le tableau 9.1 confirme que, même si l'erreur géométrique est légèrement dépendante de la taille des méta-cellules, elle reste dans la plus grande partie des cas presque constante. De plus, l'erreur potentiellement introduite par une augmentation de M, peut être compensée par l'utilisation du même niveau de profondeur des octrees à l'intérieur de cellules de plus en plus petites, ce qui va finalement perme-

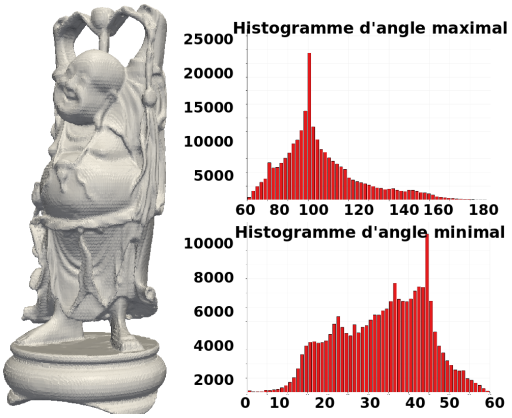
| Modèles | Pas de discrétisation | Root-Mean-Square (RMS) | | | | Hausdorff | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Nombre de méta-cellules | | | | | | | |
| | | 1 | 8 | 64 | 512 | 1 | 8 | 64 | 512 |
| **Dragon** | .00027 | .00274 | .00266 | .00265 | .00257 | .00616 | .00614 | .00590 | .00590 |
| **Filigree** | .00139 | .00118 | .00117 | .00119 | .00141 | .01563 | .01559 | .02153 | .02153 |
| **Buddha** | .00022 | .00019 | .00020 | .00021 | .00021 | .00798 | .00793 | .00815 | .00825 |
| **Vase Lion** | .00135 | .00121 | .00121 | .00128 | .00128 | .00485 | .00446 | .00446 | .00457 |

Tableau 9.1 – Erreurs géométriques des surfaces extraites par rapport aux surfaces originales. Les surfaces ont été extraites à partir de volumes de $1024 \times 1024 \times 1024$ voxels avec le pas de discrétisation indiqué dans la deuxième colonne. Les octrees utilisés ont un profondeur maximale 7 et un seuil de courbure 0.9. Les erreurs mesurées sont la distance RMS et la distance de Hausdorff et elles ont été estimées pour quatre découpages avec $M = 1,\ 8,\ 64$ et 512 méta-cellules.

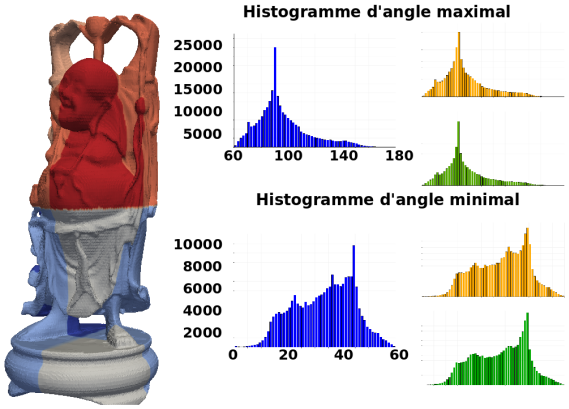
ttre d'obtenir des surfaces plus détaillées. Finalement, il faut toujours trouver un équilibre entre la taille des méta-cellules et la profondeur maximale de l'octree à l'intérieur de chacune d'entre elles afin d'améliorer la qualité de l'approximation, éviter une augmentation excessive de la taille de la surface et raccourcir le temps de raccordement des surfaces.

## 9.2 Subdivision et qualité des éléments géométriques de la surface

Un autre facteur important à tester est l'impact du découpage du volume sur la qualité des triangles produits par la reconstruction. Dans ce but, nous avons comparé les caractéristiques des éléments géométriques entre modèles compacts (reconstruits sur le volume complet) et modèles subdivisés avec différentes tailles de méta-cellules. Sur la figure 9.3a, nous avons appliqué notre algorithme d'extraction de surface sur un volume "Happy Buddha" de $1024^3$ voxels.




(a) Modèle compact de $1024^3$ voxels.




(b) Modèle subdivisé en 8 méta-cellules.

FIGURE 9.3 – Estimation de la qualité des éléments géométriques de la surface en fonction du découpage du volume. (a) Une surface obtenue à partir d'un volume compact. (b) Surface extraite à partir de 8 méta-cellules. Les histogrammes illustrent les distributions d'angle maximal et minimal pour les surfaces extraites avec de $M = 8$ (en bleu), $M = 64$ (en jaune) et $M = 512$ (en vert).

Les histogrammes montrent la distribution des valeurs des angles minimal et maximal pour les facettes triangulaires qui forment la surface. Afin de tester l'impact du découpage du volume

sur ces distributions, nous avons divisé le volume "Buddha " original en $M = 8$, 64 et 512 méta-cellules de $512^3$, $64^3$ et $8^3$ voxels respectivement. Dans le but d'obtenir une surface de taille presque équivalente, nous avons utilisé différentes profondeurs pour les octrees à l'intérieur de chaque méta-cellule selon la valeur de $M$. Ainsi, les volumes divisés en 8, 64 et 512 méta-cellules utilisent des octrees de profondeur 8, 7 et 6 respectivement. La figure 9.3b montre les histogrammes d'angle maximal et minimal pour les surfaces extraites avec $M = 8$ méta-cellules (en bleu), $m = 64$ méta-cellules (en jaune) et $M = 512$ méta-cellules (en vert). Il est clair que ces distributions ne sont pas fortement affectées par le nombre de méta-cellules utilisées, ce qui paraît normal, puisque la surface est essentiellement générée à l'intérieur des méta-cellules avec le même algorithme que pour la génération de la surface à partir d'un volume compact. Sur les zones de connexion entre méta-cellules, les triangles produits conservent aussi des bonnes qualités parce que les sommets utilisés pour les construire ont aussi été générés à l'intérieur des méta-cellules.

## 9.3 Temps d'exécution et utilisation de la mémoire

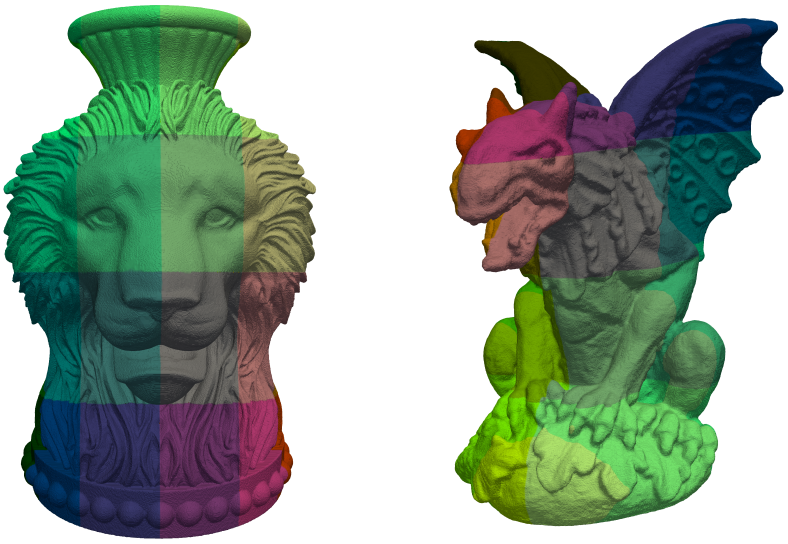
Le temps d'exécution de notre solution va fortement dépendre de la topologie et de la géométrie du volume qui vont conditionner la structure de l'octree à construire. Nous avons évalué les différentes étapes, la construction de l'octree et l'extraction de la surface à partir de cet octree pour chaque méta-cellule, et, au niveau global, le parcours des méta-cellules pour générer les surfaces et le raccordement de ces différentes surfaces entre elles. Pour l'étape de construction de l'octree, nous avons voulu quantifier le temps de calcul des différents tests topologiques et géométriques qui sont appliqués directement sur les données volumiques.

Le tableau 9.2 montre les caractéristiques des surfaces générées (nombre de sommets et de facettes) et les temps d'exécution de notre algorithme sur un ensemble des données volumiques de $2048 \times 2048 \times 1024$ voxels ($4Go$) divisés en 32 méta-cellules. Les temps affichés correspondent à l'ensemble des méta-cellules. Dans la colonne "Volume" est indiqué le temps nécessaire pour toutes les opérations appliquées sur les données volumiques comme les tests topologiques, l'estimation de la courbure et l'extraction des composantes connexes. Dans la colonne "Octree" nous listons les temps de construction de l'octree en incluant le temps de traitement du volume affiché dans la colonne précédente. La colonne suivante contient le temps nécessaire pour générer les sommets du maillage et sa connectivité. La colonne "Racc." (Raccordement) montre le temps de raccordement de toutes les surfaces et la dernière colonne affiche le temps total de génération de la surface complète.

| Critères / Modèles | Taille de la Surface | | Temps d'exécution (min : sec) | | | | |
|---|---|---|---|---|---|---|---|
| | #Points | #Triangles | Volume | Octree | Surface | Racc. | Total |
| Dragon | 842.069 | 1.684.096 | 0 :57 | 2 :30 | 1 :13 | 0 :02 | 4 :42 |
| Asian Dragon | 558.268 | 1.097.417 | 0 :54 | 1 :45 | 0 :45 | 0 :02 | 3 :26 |
| Wales Dragon | 994.292 | 1.988.619 | 0 :56 | 2 :43 | 1 :40 | 0 :03 | 5 :22 |
| Armadillo | 538.594 | 1.077.259 | 0 :36 | 1 :45 | 0 :45 | 0 :01 | 3 :07 |
| Filigree | 1.042.428 | 2.084.817 | 0 :54 | 3 :00 | 1 :40 | 0 :03 | 5 :37 |
| Buddha | 971.068 | 1.942.056 | 0 :57 | 2 :49 | 1 :28 | 0 :02 | 5 :16 |
| Gargoyle | 860.073 | 1.720.102 | 0 :53 | 2 :28 | 1 :09 | 0 :01 | 4 :31 |
| Vase Lion | 866.962 | 1.733.898 | 0 :48 | 2 :27 | 1 :16 | 0 :01 | 4 :32 |

Tableau 9.2 – Temps d'exécution de notre algorithme sur un ensemble de volumes de $2048 \times 2048 \times 1024$ voxels ($4Go$) divisés en 32 méta-cellules de $512^3$ voxels chacune.

Le tableau 9.2 confirme que la majeure partie du temps est consacrée à l'application des critères sur les données volumiques ainsi qu'à la construction des octrees. Le temps de raccordement des surfaces est vraiment court et représente moins de 2% du temps total d'exécution. Quelques exemples des surfaces obtenues peuvent être observés sur la figure 9.4.



(a) Vase Lion (b) Gargoyle

FIGURE 9.4 – Surfaces générées avec notre algorithme out-of-core à partir de volumes de $2048 \times 2048 \times 1024$ voxels ($4Go$) divisés en 32 méta-cellules, chaque couleur représente la surface d'une méta-cellule.

L'analyse de la quantité de mémoire vive nécessaire pour notre algorithme requiert de considérer plusieurs facteurs, dont, en premier lieu, la structure de données utilisée pour construire l'octree. Dans cette structure, la représentation de la cellule est la plus fondamentale parce qu'elle va contenir la plus grande partie de l'information pour construire la surface. Les cellules de l'octree contiennent les valeurs de la fonction scalaire $F(x, y, z)$ aux huit sommets de la cellule, la position initiale et finale de la cellule (parce que les cellules ne sont pas toujours des cubes mais des hexaèdres), les informations sur les normales à chaque intersection avec une arête, la localisation et la connexion des sommets duaux avec les arêtes de la cellule. Ces données correspondent à 64 octets par cellule environ. Le deuxième facteur dépend de la méthode utilisée pour traiter les différentes méta-cellules. Dans notre implémentation, nous chargeons en mémoire une seule méta-cellule à la fois. Ensuite, nous utilisons notre algorithme pour extraire la surface à partir de cette méta-cellule et nous gardons en mémoire les cellules adjacentes aux méta-facettes. Dans l'implémentation actuelle, toutes les cellules adjacentes aux méta-facettes pour toutes les méta-cellules sont conservées en mémoire. Par contre, la surface extraite à partir de chaque méta-cellule est stockée indépendamment dans un fichier intermédiaire sur le disque. Comme nous traitons les méta-cellules de manière séquentielle, nous allons nous intéresser aux pics de mémoire vive pendant l'exécution de notre algorithme. La quantité de mémoire utilisée va dépendre principalement de la profondeur des octrees et du seuil de courbure utilisé. Pour une plus grande profondeur, nous aurons potentiellement besoin de garder plus de cellules intermédiaires. Avec un seuil de courbure plus élevé, plus de cellules seront divisées et le nombre de cellules augmentera, affectant en même temps la taille de la surface produite à partir de la méta-cellule.

| Critères / Volumes | Estimation de la mémoire utilisée (Moctets) | | | | |
|---|---|---|---|---|---|
| | Pics d'utilisation | | | | Allouée |
| | Octree | Edge-octrees | Surface | Totale | Totale |
| **Dragon** | 61 | 43 | 2 | 106 | 2106 |
| **Asian Dragon** | 53 | 38 | 2 | 93 | 1760 |
| **Wales Dragon** | 76 | 51 | 3 | 130 | 2560 |
| **Armadillo** | 43 | 31 | 1 | 75 | 1440 |
| **Filigree** | 71 | 45 | 2 | 118 | 2373 |
| **Buddha** | 60 | 45 | 2 | 107 | 2054 |
| **Gargoyle** | 56 | 41 | 2 | 99 | 1889 |
| **Vase lion** | 57 | 38 | 2 | 97 | 1984 |

Tableau 9.3 – Utilisation mémoire pour un ensemble de volumes de $2048 \times 2048 \times 1024$ voxels ($4Go$). Les volumes ont été divisés en 32 méta-cellules de $512^3$ voxels chacune.

Dans le tableau 9.3, nous pouvons observer les quantités estimées de mémoire vive nécessaires sur un ensemble de modèles volumiques de $2048 \times 2048 \times 1024$ voxels ($4Go$). Ces volumes ont été divisés en 32 méta-cellules et les octrees construits ont une profondeur maximale de 7 avec un seuil de courbure $\lambda = 0.9$. Même si toutes les cellules intermédiaires sont gardées en mémoire, ce qui correspond à $32 \times 6 = 192$ méta-facettes, elles représentent moins que la quantité de mémoire requise par la construction d'un octree sur une seule méta-cellule. Dans ces exemples, cette quantité est inférieure à 3% de la mémoire vive totale utilisée indiquée dans la dernière colonne. Ces ordres de grandeur permettent d'estimer la limite pratique, en termes de mémoire, de l'implémentation actuelle. Nous devons éviter que la mémoire utilisée pour les cellules intermédiaires ainsi que pour l'octree de la méta-cellule courante ne dépasse la taille de la mémoire vive disponible. Une manière de surmonter cette limitation serait de stocker les informations des Edge-octrees sur le disque et de ne les charger en mémoire qu'au moment du raccordement. Néanmoins, chaque méta-cellule doit pouvoir être entièrement chargée dans la mémoire vive. La taille des méta-cellules devra donc être fixée en s'assurant que les méta-cellules et les octrees correspondants n'occupent pas trop de place en mémoire.

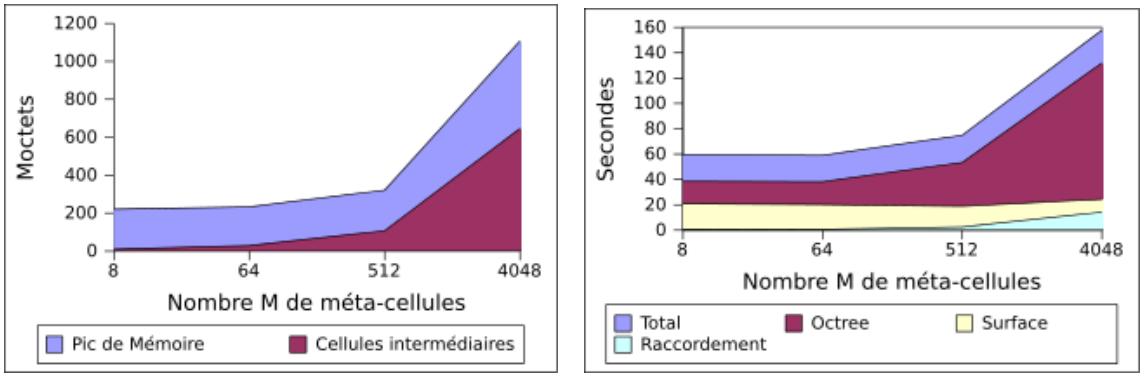
Pour illustrer la relation entre la taille des méta-cellules, le temps d'exécution et la taille de la surface produite, le volume "Happy Buddha" de $1024 \times 1024 \times 1024$ voxels a été traité avec $M = 8$, 64, 512 et 4048 méta-cellules et des octrees de profondeur maximale de 7, 6, 5 et 4 respectivement, fournissant quatre surfaces de qualité géométrique équivalente.

Le tableau 9.4 reflète l'impact du découpage en méta-cellules sur la taille de la surface, la mémoire utilisée et les temps d'exécution afin d'obtenir une qualité d'approximation équivalente. Il montre qu'un plus grand nombre de méta-cellules va fortement augmenter la quantité de cellules intermédiaires qu'il faut garder en mémoire afin de pouvoir connecter les surfaces. Dans cet exemple, la mémoire nécessaire passe de moins de 5% avec 8 méta-cellules à 55% pour un volume divisé en 4048 méta-cellules comme le montre la figure 9.5a. Quant aux temps d'exécution, il apparaît qu'avec des valeurs plus élevées de $M$, il faut construire plus d'octrees et conserver plus de cellules intermédiaires, ce qui augmente grandement le temps dédié à la construction des octrees. Par contre, le temps de génération de la surface n'est pas fortement affecté parce qu'il dépend surtout du parcours des cellules feuilles des octrees, qui n'augmente pas beaucoup avec $M$. Par comparaison, le temps de raccordement est directement lié à $M$ et, même si sa contribution dans le temps total

| Critères | | M (#Méta-cellules) | | | |
|---|---|---|---|---|---|
| | | 8 | 64 | 512 | 4048 |
| **Caractéristiques de la surface** | **#Points** | 393292 | 196802 | 205924 | 248847 |
| | **#Triangles** | 786598 | 393703 | 408769 | 497673 |
| **Mémoire utilisée (Moctets)** | **Pic de Mémoire** | 221 | 233 | 321 | 1108 |
| | **Cellules intermédiaires** | 10 | 29 | 107 | 646 |
| **Qualité de l'approximation** | **RMS** | 0.005921 | 0.005909 | 0.005909 | 0.005850 |
| | **Hausdorff** | 0.127038 | 0.127038 | 0.127038 | 0.127038 |
| **Temps d'exécution (millisecondes)** | **Octree** | 38589 | 38185 | 53351 | 131916 |
| | **Surface** | 20965 | 20226 | 18940 | 24390 |
| | **Raccordement** | 228 | 646 | 2559 | 14231 |
| | **Total** | 59782 | 59057 | 74850 | 157729 |

Tableau 9.4 – Résultats sur un volume de $1024 \times 1024 \times 1024$ voxels divisé en $M = 8$, 64, 512 et 4048 méta-cellules. Les profondeurs des octrees (7, 6, 5 et 4 respectivement), ont été ajustées afin d'obtenir des approximations géométriques équivalentes par rapport à la distance RMS et la distance de Hausdorff. Le tableau présente le nombre de sommets et de facettes des surfaces, les pics maximaux de mémoire utilisée et les temps d'exécution pour extraire chaque surface.

de traitement reste très réduite, il augmente de manière significative avec $M$. La contribution de chaque étape de l'algorithme dans le temps total d'exécution est présenté sur la figure 9.5b.




(a) Pics d'utilisation de la mémoire.

(b) Temps d'exécution.

FIGURE 9.5 – (a) Utilisation de la mémoire en fonction du nombre de méta-cellules. (b) Contribution de chaque étape de l'algorithme dans les temps d'exécution (en secondes).

Les résultats obtenus montrent qu'il est possible d'obtenir une bonne approximation en utilisant plusieurs subdivisions et en utilisant un découpage en méta-cellules, la profondeur et le seuil de courbure étant les facteurs d'ajustement de la qualité de l'approximation voulue. Néanmoins, il est clair qu'un plus grand nombre de méta-cellules a un fort impact sur la quantité de mémoire utilisée et le temps d'exécution sans nécessairement garantir l'obtention d'une meilleure approximation. Il faut donc essayer de minimiser ce nombre et de profiter le plus possible de critères de profondeur et de courbure liés à l'octree afin d'assurer l'obtention d'une bonne approximation sans une empreinte additionnelle sur la mémoire et le temps de traitement nécessaires.

## 9.4 Temps d'exécution et utilisation de la mémoire avec parallélisation

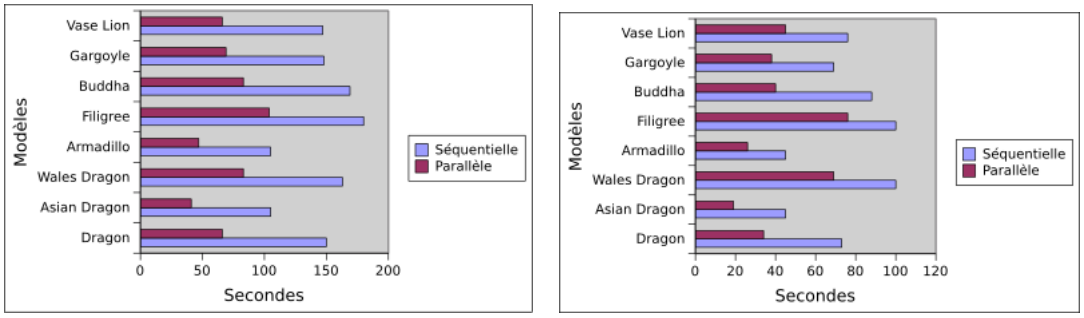
Nous avons présenté ci-dessus les résultats de notre approche out-of-core pour traiter des volumes de grande taille de manière séquentielle. Cette approche peut aussi être parallélisée pour réduire les temps d'exécution et profiter des nouvelles technologies de processeurs à multiples cœurs. Nous avons donc implémenté une version parallèle qui profite de la programmation "multi-thread" afin de paralléliser l'étape d'extraction des surfaces à partir de plusieurs méta-cellules. L'avantage des processus légers est qu'ils partagent le même espace mémoire alloué pour tout le programme. Cela évite d'utiliser des fichier intermédiaires stockés sur le disque pour les informations à partager entre les différentes tâches et permet d'accélérer l'accès aux données provenant des différentes méta-cellules (cellules intermédiaires) nécessaires pour faire le raccordement de la surface. L'étape de raccordement est aussi parallélisable mais son implémentation n'est pas aussi directe que dans le cas des méta-cellules. De plus, comme elle ne représente qu'une très petite partie du temps total d'exécution, nous n'avons pas développé de version parallèle de cette étape. Notre algorithme parallèle a été testé sur le même ensemble de volumes que précédemment (tableau 9.2) afin de le comparer à l'implémentation séquentielle. Dans cette expérience, nous avons utilisé seulement 4 "threads" sur quatre processeurs "dual core" de 2.4 GHz chacun. Les temps mesurés dans le tableau 9.5 correspondent aux pics de temps utilisé par le plus lent des quatre processeurs fonctionnant de manière concurrente.

| Critères / Volumes | Mesures de temps - Pics de mesures par thread (min :sec) | | | | |
|---|---|---|---|---|---|
| | **Volume** | **Octree** | **Surface** | **Raccordement** | **Total** |
| Dragon | 0 :28 | 1 :06 | 0 :34 | 0 :02 | 2 :10 |
| Asian Dragon | 0 :27 | 0 :41 | 0 :19 | 0 :01 | 1 :28 |
| Wales Dragon | 0 :26 | 1 :23 | 1 :09 | 0 :02 | 3 :00 |
| Armadillo | 0 :26 | 0 :47 | 0 :26 | 0 :01 | 1 :40 |
| Filigree | 0 :26 | 1 :44 | 1 :16 | 0 :03 | 3 :29 |
| Buddha | 0 :28 | 1 :23 | 0 :40 | 0 :02 | 2 :33 |
| Gargoyle | 0 :27 | 1 :09 | 0 :38 | 0 :01 | 2 :15 |
| Vase Lion | 0 :18 | 1 :06 | 0 :45 | 0 :01 | 2 :10 |

Tableau 9.5 – Temps d'exécution de la version "multi-thread" de notre algorithme avec quatre "threads" concurrents sur quatre processeurs "dual core" de 2.4 GHz chacun. L'algorithme a été appliqué sur les volumes du tableau 9.2.

Les résultats du tableau 9.5 confirment que l'implémentation parallèle améliore la performance de notre algorithme sur tous les exemples. Néanmoins, l'amélioration n'est pas proportionnelle au nombre de cœurs utilisés. Nous aurons pu nous attendre à ce que l'utilisation de 4 cœurs divise presque par 4 le temps d'exécution : ce n'est pas le cas. L'accélération la plus marquée est constatée sur l'étape de construction de l'octree et de génération de la surface, ce qui est clairement dû au fait que ces étapes ont été explicitement parallélisées. L'implémentation idéale d'un algorithme parallèle sur quatre "threads" devrait produire une réduction du temps d'exécution de l'ordre de 75 %. Cependant, les temps obtenus illustrent une réduction de seulement 45 % en moyenne comme l'illustrent les diagrammes de la figure 9.6.

Une explication de la dégradation de la performance est qu'une implémentation basée sur des "threads" n'est pas entièrement parallèle puisqu'ils peuvent partager des espaces mémoire. Ceci




(a) Temps de construction de l'octree.

(b) Temps de génération de la surface.

FIGURE 9.6 – Comparaison des temps d'exécution entre l'implémentation séquentielle et l'implémentation parallèle. (a) Temps de construction de l'octree et (b) temps de génération de la surface.

est utile au moment de coordonner les tâches entre différents "threads" mais pose aussi des problèmes dans, le contrôle de l'accès aux espaces mémoire, notamment le faux partage ("false-sharing") déjà mentionné dans la section 8.4.1. Dans notre cas, ce problème se présente quand deux threads essaient d'accéder à un même bloc mémoire contenant, soit des informations partagées, soit des informations déconnectées mais stockées complètement ou partiellement sur le même bloc de mémoire physique, réduisant aussi le gain d'une implémentation parallèle.

Une solution serait d'utiliser des outils spécialisés pour la parallélisation de telle manière que les espaces mémoire ne soient pas partagés entre plusieurs "threads" en exploitant une communication efficace entre les processus.

## 9.5 Application sur de volumes de données de grand taille

Nous avons testé notre application sur des données de grande taille. Dans le premier exemple, nous avons utilisé un volume de $4096 \times 4096 \times 2048$ voxels équivalent à 34Go d'espace sur le disque. Il ne peut pas être traité sur un ordinateur conventionnel avec un algorithme de génération de surface in-core. Nous avons divisé le volume en 256 méta-cellules d'entre $512^3$ et $513^3$ voxels du au fait du recouvrement partiel entre méta-cellules. L'algorithme a été exécuté sur un ordinateur avec 4 double-cœurs de 2.4 GHz chacun. Les résultats sont présentés dans le tableau 9.6.

L'algorithme parallèle a pu produire une surface de plus de 10 millions de facettes avec un gain de plus de 60% sur le temps d'exécution par rapport à l'algorithme séquentiel. Dans l'exemple précédent, nous avons extrait une surface de taille moyenne en considérant les caractéristiques topologiques et géométriques d'un volume beaucoup plus grand. Avec des paramètres différents, nous pourrions obtenir des surfaces à différents niveaux de résolution à partir du même volume.

### 9.5.1 Génération d'une surface massive

Jusqu'à maintenant, nous avons traité des volumes de grandes dimensions pour extraire des maillages adaptatifs de quelques millions de triangles. Néanmoins, il existe certains volumes qui contiennent une grande quantité de détails de haute résolution pour lesquelles il est nécessaire d'extraire une surface plus grande encore. Notre algorithme peut aussi traiter ces volumes afin de générer des

| **Étapes de l'algorithme** | **Pics de mémoire (Moctets)** |
|---|---|
| Construction de l'octree | 182 |
| Génération de la surface | 381 |
| Taille de la surface | 5 |
| Totale | 563 |
| Taille de la surface | 284 |

| **Éléments géométriques** | |
|---|---|
| Nombre de sommets | 5.378.030 |
| Nombre de facettes | 10.566.037 |

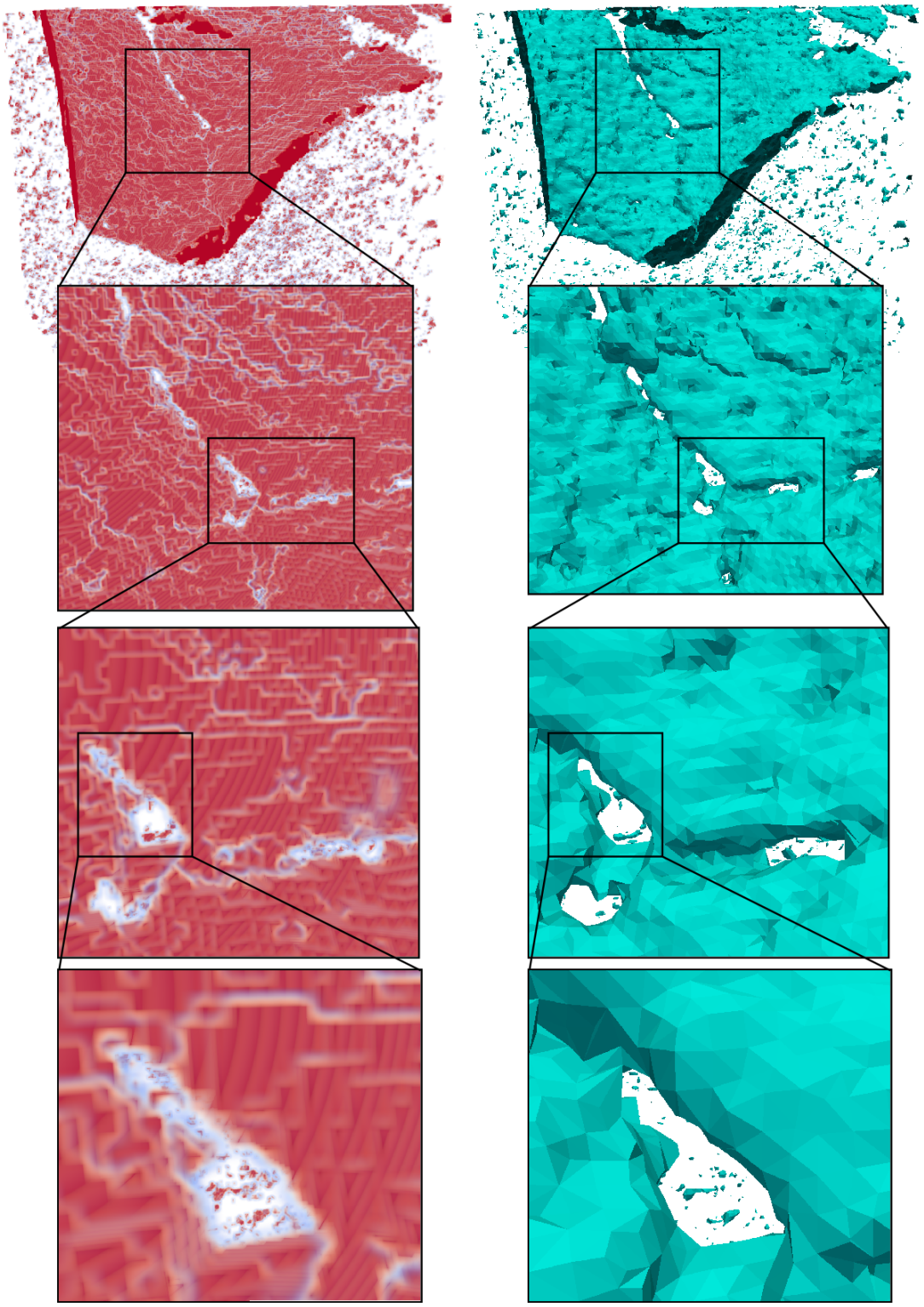
| **Temps d'exécution** | | |
|---|---|---|
| **Étapes** | **(min : sec)** | |
| | **Séquentielle** | **Parallèle** |
| Chargement des volumes | 12 :51 | 4 :59 |
| Construction de l'octree | 20 :40 | 8 :55 |
| Génération de la surface | 13 :05 | 5 :45 |
| Raccordement des surfaces | 0 :50 | 0 :56 |
| Total | 47 :26 | 19 :35 |

Tableau 9.6 – Comparaison de nos algorithmes séquentiel et parallèle sur un volume de $4096 \times 4096 \times 2048$ voxels (34Go) divisé en 256 méta-cellules. Les octrees utilisés ont une profondeur maximale de 7 et le seuil de courbure est de 0.9.

surfaces composées de centaines de millions de triangles. Pour tester cela, nous avons utilisé des données résultant de la segmentation d'un volume de genou de $2048 \times 2048 \times 2048$ voxels équivalent à 8 Go. Pour le traiter, il a été divisé en 512 cellules de $256^3$ et $257^3$ voxels. Nous avons utilisé des octrees d'une profondeur minimale de 6 et maximale de 7 à l'intérieur de chaque méta-cellule ainsi qu'un seuil de courbure de 0.9. La surface a été extraite avec un ordinateur conventionnel avec quatre cœurs de 2.4GHz chacun, 8 Go de mémoire principale et un disque de 7200 révolutions par minute.

Moins de 8 heures ont été nécessaires pour extraire une surface avec environ 610 millions de triangles et 315 millions de points qui occupe 10 Go sur le disque. En moyenne, nous avons traité une méta-cellule toutes les 50 secondes avec plus de 1.1 million de facettes par méta-cellule. Il a fallu moins de 10 minutes pour les raccordements. Notre solution a enregistré un pic de mémoire vive de 1.5 Go pour chaque méta-cellule et approximativement la même quantité de mémoire pour stocker les cellules adjacentes aux facettes des méta-cellules. La plus grande partie de la mémoire est utilisée par la structure de données de l'octree. Les accès disque sont également importants parce que chaque morceau de la surface doit être stocké sur le disque et exploré pendant l'étape de raccordement.

La figure 9.7 présente la surface obtenue avec notre méthode. En (a), nous pouvons observer le volume contenu à l'intérieur d'une méta-cellule et en (b) la surface extraite à partir de ce volume. Chaque figure montre plus en détail une région du volume et la partie de la surface correspondante. Notre méthode peut être comparée avec la solution de génération de surfaces et de remaillage présentée par Schreiner *et al.* [95, 96]. Cette méthode extrait une surface à partir d'un volume en utilisant une approche d'avancement de front ("Advancing front") à partir d'une subdivision initiale du volume d'intérêt. Elle produit des surfaces lisses de très bonne qualité géométrique et visuelle. Néanmoins, comme la méthode d'avancement de front a une forte composante algorithmique variationnelle, elle est beaucoup plus lente que notre solution.



(a) Volume à l'intérieur d'une méta-cellule de $256 \times 256 \times 256$ voxels.

(b) Surface extraite à partir du volume de la méta-cellule en (a). Ce patch contient 170000 triangles.

FIGURE 9.7 – Partie de la surface construite à partir d'un volume de $2048 \times 2048 \times 2048$ voxels (8 Go). Il faut moins de 8 heures pour générer une surface d'environ 610 millions de triangles et 315 millions de points.

Par comparaison, pour produire un maillage d'environ 390 millions de triangles, la méthode de Schreiner nécessite 123 heures. La nôtre génère une surface 50% plus grande en 8 heures seulement. Force est de reconnaître que les surfaces générées par notre algorithme n'ont pas la même qualité de celles produites par Schreiner mais elles fournissent un compromis entre des méthodes rapides produisant des maillages de mauvaise qualité telles que les "Marching Cubes" et des bonnes triangulations générées en des temps prohibitifs.

## 9.6 Discussion

Notre méthode de génération de surfaces adaptatives a été élaborée pour garantir de bonnes caractéristiques géométriques et topologiques de la surface. Nos efforts ont ensuite porté sur son adaptation à des données volumiques massives ou la construction de surfaces elles-mêmes massives. Il n'y a pas, a priori, de limitation théorique et technique pour traiter avec notre solution de très gros volumes et obtenir des maillages très volumineux.

Dans nos tests, nous avons utilisé des volumes acquis expérimentalement ainsi que des volumes discrétisés à partir de surfaces compactes, c'est à dire comportant peu de composantes connexes, quelques régions lisses et un genre peu élevé. Pour ce type de surfaces, utiliser un algorithme d'extraction de surfaces adaptatives a du sens parce qu'il existe de nombreuses régions lisses qui peuvent être représentées avec peu de triangles.

La possibilité de s'adapter aux caractéristiques topologiques et géométriques des volumes représente un vrai avantage dans une grande quantité de domaines où les représentations volumiques sont de plus en plus utilisées. Les nouvelles techniques de capture et de segmentation d'images permettent d'obtenir des modèles compacts qui sont mieux représentés par une surface adaptative et multi-résolution. Cela permet de distinguer et de valoriser notre approche par rapport à la grande majorité d'algorithmes de l'état de l'art.

# Chapitre 10

# Conclusions et perspectives

Les sujets abordés dans ce mémoire vont de l'étude des caractéristiques des objets volumiques jusqu'à la génération de surfaces à partir de volumes massifs. Nous avons étudié les méthodes de génération de surfaces basées sur des divisions spatiales comme une étape intermédiaire afin de pouvoir ensuite proposer un algorithme de génération de surfaces à partir de données volumiques de grande taille. Nous présentons dans ce chapitre nos principales contributions et dégagerons quelques perspectives.

## 10.1 Génération de surfaces adaptatives

Tout au long de ce mémoire, nous avons étudié les caractéristiques des objets volumiques, la manière de les utiliser pour générer des surfaces adaptatives. Pour cela, nous avons proposé un pipeline complet pour l'extraction de surfaces adaptatives à partir de données volumiques. Nous avons détaillé les différentes étapes de notre solution et nos contributions dans chacune d'entre elles.

Dans l'étape de construction de l'octree, nous avons proposé et implémenté un critère de cellule complexe adapté aux données volumiques pour garantir que toutes les cellules de l'octree contiennent des morceaux de la surface topologiquement équivalents à un disque. Cela permet d'assurer que la surface produite est 2-variété. De plus, si le volume utilisé est compact, la surface sera aussi fermée.

Par rapport à la connectivité du maillage, nous avons modifié la technique de génération des sommets duaux proposé par "Dual Marching Cubes" afin d'éliminer les potentiels arêtes et sommets non-variété générés. Nous avons vérifié la topologie des cellules adjacentes pour assurer un bon raccordement de la surface.

Concernant la qualité de l'approximation, notre contribution a consisté en une nouvelle méthode de localisation qui permet de toujours placer les sommets sur la surface discrète pour améliorer l'approximation géométrique et de maximiser la moyenne de la distance entre les sommets duaux des cellules adjacentes pour améliorer la qualité géométrique des triangles produits.

Nous avons appliqué notre pipeline sur un ensemble de volumes et évalué les résultats avec plusieurs critères tels que la qualité de l'approximation, la quantité de triangles générés et la performance en

temps et l'utilisation de la mémoire. Nous avons aussi comparé nos résultats avec les algorithmes de l'état de l'art. Cela nous a permis de conclure que, même si notre méthode ne résout pas tous les problèmes, elle améliore la qualité générale des triangles sur la surface par rapport aux méthodes duales et produit beaucoup moins de triangles aplatis que les méthode primales, tout en conservant la même qualité d'approximation.

### 10.1.1 Perspectives

Concernant la qualité de l'approximation, même si notre algorithme approxime correctement la majorité des surfaces, il reste perfectible en présence d'arêtes vives. Les arêtes vives sont toujours un défi, que la plus grande partie des méthodes abordent grâce à des fonctions d'erreur quadratique (QEF). Nous envisageons d'implémenter également cette solution sur les régions contenant des arêtes vives. Cela devrait permettre d'améliorer la qualité des surfaces reconstruites pour des objets qui comportent des arêtes vives comme les pièces mécaniques manufacturées.

Notre objectif initial était de construire un outil pour obtenir des surfaces avec une certaine qualité, mais pas pour être interactif ni permettre la visualisation et l'exploration en temps réel. Néanmoins, si nous envisageons de l'étendre à ces usages, nous aurons besoin de structures de recherche optimales ou de pré-traitements sur le volume afin d'accélérer la méthode. Un pré-traitement peut organiser les données par rapport aux valeurs telles que la courbure de la surface ou la complexité topologique de la région. Une telle information peut aussi être utilisée pour guider le raffinement et la topologie de la triangulation sur des régions spécifiques de la surface.

## 10.2 Génération de surfaces à partir de données volumiques de grande taille

Une deuxième partie de notre travail a été dédiée à l'extraction des surfaces à partir de données volumiques massives. Dans ce contexte, nous avons confirmé que les méthodes de l'état de l'art ont été principalement focalisées sur la visualisation et l'exploration des surfaces. Pour cela, elles ont privilégié la vitesse d'accès aux données par rapport à la qualité générale de la surface obtenue. Les méthodes que nous avons décrites utilisent des pré-traitements sur les volumes pour construire des structures de recherche optimisées. Afin de traiter des volumes de grande taille, une petite partie de ces structures de recherche ont été adaptées afin de pouvoir être stockées sur le disque. Une caractéristique de presque tous les algorithmes de l'état de l'art est qu'ils utilisent l'algorithme "Marching Cubes" ou une des ses versions améliorées pour générer la triangulation. Cela implique que les surfaces construites ne sont pas adaptatives et contiennent une grande quantité de triangles dégénérés.

Nous avons proposé un algorithme d'extraction de surfaces adaptatives à partir de données volumiques massives qui utilise une approche diviser pour régner. Dans ce but, nous avons découpé le volume en sous-volumes appelés méta-cellules et modifié l'algorithme de génération de surface présenté dans la première partie de ce manuscrit pour générer les surfaces à l'intérieur de chaque méta-cellule. Les surfaces sont ensuite connectées en une unique composante. Notre algorithme a l'avantage de générer des surfaces adaptatives sans limite théorique sur la taille de la surface à générer. En dernier lieu, nous avons proposé une implémentation parallèle de notre solution.

Les surfaces obtenues dans la phase de test confirment que notre solution crée des surfaces adaptées aux caractéristiques du volume et que la subdivision du volume n'affecte pas fortement la qualité de l'approximation. La méthode proposée pour raccorder les surfaces n'impose pas un coût significatif sur le temps d'exécution et l'empreinte mémoire par rapport aux autres étapes de la reconstruction.

### 10.2.1 Perspectives

La large diffusion des systèmes d'imagerie produisant des données volumiques montrent l'intérêt des méthodes de visualisation et d'exploration de gros volumes. Il est donc fondamental de travailler sur l'aspect vitesse et la capacité d'adaptation de la méthode au différents critères tels que le point de vue de l'observateur.

Dans nos travaux futurs, nous envisageons d'utiliser notre algorithme comme un outil de prétraitement sur le volume qui va nous permettre de construire une subdivision en cellules à partir desquelles nous obtiendrons la surface. Différentes subdivisions peuvent être construites à plusieurs niveaux de résolution de manière à produire une hiérarchie de subdivisions qui puissent être combinées avec des structures de recherche optimales en un outil de visualisation de surfaces adaptatives et multi-résolution. Une telle application peut être complétée avec des techniques de compression et de quantification pour la transmission et le stockage de modèles.

# Bibliographie


[1] A. Aggarwal and J.S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9) :1116–1127, 1988.

[2] L. Arge. The buffer tree : A new technique for optimal I/O-algorithms. In *Proceedings of the 4th International Workshop on Algorithms and Data Structures*, number August, pages 334–345. Springer-Verlag, 1995.

[3] L. Arge. The I/O-complexity of ordered binary-decision diagram manipulation. In *Algorithms and Computations, Lecture Notes in Computer Science*, number August, pages 82–91. 1995.

[4] L. Arge, M. Knudsen, and K. Larsen. A general lower bound on the I/O-complexity of comparison-based algorithms. In *Proceedings of the Third Workshop on Algorithms and Data Structures*, pages 83–94, London, UK, UK, 1993. Springer-Verlag.

[5] L. Arge, DE. Vengroff, and J.S. Vitter. External-memory algorithms for processing line segments in geographic information systems. *Algorithmica*, 47(1) :1–25, 2007.

[6] L. Arge and J.S. Vitter. Optimal dynamic interval management in external memory. *In Proceedings of IEEE Symposium on foundations of computer science*, pages 560–569, 1996.

[7] S. Azernikov and A. Miropolsky. Surface reconstruction of freeform objects based on multiresolution volumetric method. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, volume 4, pages 115–126, Seattle, Washington, USA, 2003. ACM.

[8] C.L. Bajaj, V. Pascucci, and DR. Schikore. Fast isocontouring for improved interactivity. *Proceedings of the symposium on Volume visualization*, pages 39–46, 1996.

[9] C.L. Bajaj, V. Pascucci, and D. Thompson. Parallel accelerated isocontouring for out-of-core visualization. *roceedings of IEEE symposium on Parallel visualization and graphics*, pages 97–104, 1999.

[10] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. *Computing in Euclidean geometry*, 1 :23–90, 1992.

[11] M. Bolitho, M. Kazhdan, R. Burns, and H. Hoppe. Multilevel streaming for out-of-core surface reconstruction. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 69–78, Barcelona, Spain, 2007. Eurographics Association.

[12] U.D. Bordoloi and HW. Shen. Space efficient fast isosurface extraction for large datasets. In *Proceedings of the 14th IEEE Visualization*, pages 201–208, Seattle, Washington, USA, 2003. IEEE Computer Society.

[13] T. Boubekeur, W. Heidrich, and X. Granier. Volume Surface Trees. *Computer Graphics Forum*, 25(3) :399–406, 2006.

[14] P.S. Bradley and O.L. Mangasarian. Massive data discrimination via linear support vector machines. *Optimization Methods and Software*, 13(1) :1–10, 2000.

[15] L. Buatois and G. Caumon. GPU accelerated isosurface extraction on tetrahedral grids. *Proceedings of the Second international conference on Advances in Visual Computing - Volume Part I*, 1 :383–392, 2006.

[16] B.P. Carneiro, C.T. Silva, and A.E. Kaufman. Tetra-cubes : an algorithm to generate 3D isosurfaces based upon tetrahedra. In *Proceedings of IX Brazilian symposium on computer, graphics, image processing and vision (SIBGRAPI)*, pages 205–210, 1996.

[17] H. Carr. (No) More Marching Cubes. In *Proceedings of the Sixth Eurographics / IEEE VGTC conference on Volume Graphics*, pages 81–90, Prague, Czech Republic, 2007. Eurographics Association.

[18] H. Carr, T. Moller, and J. Snoeyink. Simplicial subdivisions and sampling artifacts. In *Proceedings of the conference on Visualization*, pages 99–106, San Diego, California, 2001. IEEE Computer Society.

[19] H. Carr, T. Möller, and J. Snoeyink. Artifacts caused by simplicial subdivision. *IEEE transactions on visualization and computer graphics*, 12(2) :231–242, 2006.

[20] S.L. Chan. A new tetrahedral tesselation scheme for isosurface generation. *Computers & Graphics*, 22(1) :83–90, February 1998.

[21] L-S. Che, G.T. Herman, R.A. Reynolds, and J.K. Udupa. Surface Shading in the Cuberille Environment. *IEEE Computer Graphics and Applications*, 5(12) :33–43, 1985.

[22] E.V. Chernyaev. Marching cubes 33 : Construction of topologically correct isosurfaces. Technical report, Institute for High Energy Physics, Moscow, Russia, 1995.

[23] Y-J. Chiang. *Dynamic and I/O-Efficient Algorithms for Computational Geometry and Graph Problems : Theoretical and Experimental Results.* PhD thesis, Brown University, Providence, RI, USA, 1995.

[24] Y-J. Chiang. Experiments on the Practical I/O Efficiency of Geometric Algorithms : Distribution Sweep vs. Plane Sweep. Technical report, Brown University, Providence, RI, USA, 1995.

[25] Y-J. Chiang, J. El-Sana, and P. Lindstrom. Out-of-core algorithms for scientific visualization and computer graphics. *Proceedings of the conference on Visualization, Course Notes*, 2002.

[26] Y-J. Chiang, R. Farias, C.T. Silva, and B. Wei. A unified infrastructure for parallel out-of-core isosurface extraction and volume rendering of unstructured grids. *Symposium on Parallel and Large-Data Visualization and Graphics*, pages 59–66, 2001.

[27] Y-J. Chiang and C.T. Silva. I/O optimal isosurface extraction. In *Proceedings of the 8th conference on Visualization*, pages 293–300, Phoenix, Arizona, USA, 1997. IEEE.

[28] Y-J. Chiang, C.T. Silva, and W.J. Schroeder. Interactive out-of-core isosurface extraction. In *Proceedings of the conference on Visualization*, pages 167–174, Research Triangle Park, North Carolina, USA, 1998. IEEE Computer Society Press.

[29] YJ. Chiang and C.T. Silva. External memory techniques for isosurface extraction in scientific visualization. *Memory and Visualization*, pages 247–277, 1999.

[30] P. Cignoni, F. Ganovelli, C. Montani, and R. Scopigno. Reconstruction of topologically correct and adaptive trilinear isosurfaces. *Computers & Graphics*, 24 :399–418, 2000.

[31] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2) :158–170, 1997.

[32] P. Cignoni and C. Montani. Optimal isosurface extraction from irregular volume data. In *Symposium on Volume Visualization*, volume D, pages 31–38, San Francisco, California, USA, 1996. IEEE Press.

[33] P. Cignoni, C. Rocchini, and R. Scopigno. Metro : measuring error on simplified surfaces. In *Computer Graphics Forum*, volume 17, pages 167–174. Wiley Online Library, 1998.

[34] D. Coeurjolly, A. Montanvert, and J-M. Chassery. Géométrie discrète et images numériques, September 2007.

[35] T.H. Cormen, C. Stein, R.L. Rivest, and C.E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[36] L. Custodio, T. Etiene, S. Pesco, and C. Silva. Practical considerations on Marching Cubes 33 topological correctness. *Computers & Graphics*, 37(7) :840–850, April 2013.

[37] T.K. Dey and S. Goswami. Tight cocone : a water-tight surface reconstructor. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, volume 3, pages 127–134, Seattle, Washington, USA, 2003. ACM.

[38] C.A. Dietrich, C.E. Scheidegger, J.L.D. Comba, L.P. Nedel, and C.T. Silva. Marching cubes without skinny triangles. *Computing in Science & Engineering*, 11(2) :82–87, 2009.

[39] M.B. Dillencourt, H. Samet, and M. Tamminen. A general approach to connected-component labeling for arbitrary image representations. *ACM*, 39(2) :253–280, April 1992.

[40] C. Dyken, G. Ziegler, C. Theobalt, and H-P. Seidel. High-speed Marching Cubes using HistoPyramids. *Computer Graphics Forum*, 27(8) :2028–2039, December 2008.

[41] H. Edelsbrunner. *Dynamic Data Structures for orthogonal intersection queries*. Forschungsberichte : Institut f{ü}r Informationsverarbeitung. Inst. f. Informationsverarbeitung, TU Graz, Graz, 1980.

[42] T. Etiene, L.G. Nonato, C. Scheidegger, J. Tierny, T.J. Peters, V. Pascucci, R.M. Kirby, and Others. Topology verification for isosurface extraction. *IEEE Transactions on Visualization and Computer Graphics*, 18(6) :952–965, 2010.

[43] R. Farias and C.T. Silva. Out-of-core rendering of large, unstructured grids. *Computer Graphics and Applications, IEEE*, 21(4) :42–50, 2001.

[44] V. Fiorin, P. Cignoni, and R. Scopigno. Out-of-core MLS reconstruction. *IASTED International Conference on Computer Graphics and Imaging*, pages 27–34, 2007.

[45] F. Flin. Adaptative estimation of normals and surface area for discrete 3d objects. *IEEE Transactions on Image Processing*, 14(5) :585–596, 2005.

[46] PJ. Frey. Génération et adaptation de maillages de surfaces à partir de données anatomiques discrètes. Technical report, INRIA, 2003.

[47] M. Garland. *Quadric-based polygonal surface simplification*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1999.

[48] M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, volume 9731, pages 209–216. ACM Press/Addison-Wesley Publishing Co., 1997.

[49] T. Gerstner and M. Rumpf. Multiresolutional Parallel Isosurface Extraction based on Tetrahedral Bisection. *Symp. Volume Visualization*, pages 267–278, 1999.

[50] B. Gregorski, J. Senecal, M.A. Duchaineau, and K.I. Joy. Adaptive extraction of time-varying isosurfaces. *IEEE transactions on visualization and computer graphics*, 10(6) :683–694, 2004.

[51] Y. Han and R.A. Wagner. An efficient and fast parallel-connected component algorithm. *ACM*, 37(3) :626–642, July 1990.

[52] H. Hoppe, T. DeRose, and T. Duchamp. Mesh optimization. *Proceedings of the annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, pages 19–26, 1993.

[53] C.T. Howic and E.H. Blake. The Mesh Propagation Algorithm for Isosurface Construction. *Computer Graphics Forum*, 13(3) :65–74, August 1994.

[54] Ch. Icking, R. Klein, and Th. Ottmann. Priority search trees in secondary memory. In *International Workshop on Graph-theoretic concepts in computer science*, pages 84–93, 1988.

[55] M. Isenburg and S. Gumhold. Out-of-core compression for gigantic polygon meshes. *ACM Transactions on Graphics (TOG)*, 22(3) :935–942, 2003.

[56] M. Isenburg and P. Lindstrom. Streaming Meshes. In *IEEE Visualization*, pages 231–238. IEEE Computer Society Press, 2005.

[57] M. Isenburg, P. Lindstrom, S. Gumhold, and J. Snoeyink. Large mesh simplification using processing sequences. In *Proceedings of the 14th IEEE Visualization*, pages 465–472. IEEE Computer Society, 2003.

[58] J. Jansson and A. Logg. Algorithms and Data Structures for Multi-Adaptive Time-Stepping. *ACM Transactions on Mathematical Software*, 35(3) :1–24, October 2008.

[59] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, volume pages, pages 339–346, San Antonio, Texas, 2002. ACM New York, NY, USA.

[60] T. Ju and T. Udeshi. Intersection-free contouring on an octree grid. *Pacific Conference on Computer Graphics and Applications (Pacific Graphics)*, 2006.

[61] M. Kazhdan, A. Klein, K. Dalal, and H. Hoppe. Unconstrained isosurface extraction on arbitrary octrees. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 125–133, Barcelona, Spain, 2007. Eurographics Association.

[62] L.P. Kobbelt, M. Botsch, U. Schwanecke, and H-P. Seidel. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, pages 57–66, New York, New York, USA, 2001. ACM Press.

[63] V.A. Kovalevsky. Finite Topology as Applied to Image Analysis. In *Computer Vision, Graphics, and Image Processing*, volume 161, pages 141–161, 1989.

[64] L. Latecki. 3D Well-Composed Pictures. *Graphical Models and Image Processing*, 59(3) :164–172, May 1997.

[65] S. Lefebvre and H. Hoppe. Perfect spatial hashing. *ACM Transactions on Graphics (TOG)*, 25(3) :579–588, 2006.

[66] A. Lenoir. Fast estimation of mean curvature on the surface of a 3d discrete object. *Discrete Geometry for Computer Imagery*, 1347 :175–186, 1997.

[67] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3) :29–37, May 1988.

[68] T. Lewiner, V. Mello, A. Peixoto, S. PESCO, and H. Lopes. Fast generation of pointerless octree duals. *Computer Graphics Forum*, 29(5) :1661–1669, 2009.

[69] H.R. Lewis and C.H. Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19(2) :161–187, 1982.

[70] P. Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, pages 259–262. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 2000.

[71] Y. Livnat, HW. Shen, and CR. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1) :73–84, 1996.

[72] W.E. Lorensen and H.E. Cline. Marching cubes : A high resolution 3D surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, pages 163–169. ACM, 1987.

[73] J. Manson and S. Schaefer. Isosurfaces Over Simplicial Partitions of Multiresolution Grids. *Computer Graphics Forum*, 29(2) :377–385, 2010.

[74] J. Manson, J. Smith, and S. Schaefer. Contouring Discrete Indicator Functions. *Computer Graphics Forum*, 30(2) :385–393, 2011.

[75] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. Laplacian mesh optimization. In *GRAPHITE*, pages 381–389, New York, New York, USA, 2006. ACM Press.

[76] T. Newman. Approaches that exploit vector-parallelism for three rendering and volume visualization techniques. *Computers & Graphics*, 24(5) :755–774, October 2000.

[77] T.S. Newman and H. Yi. A survey of the marching cubes algorithm. *Computers \& Graphics*, 30(5) :854–879, 2006.

[78] M.B. Nielsen, O. Nilsson, A. Söderström, and K. Museth. Out-of-core and compressed level set methods. *ACM Transactions on Graphics*, 26(4) :16–56, October 2007.

[79] G.M. Nielson. On marching cubes. *IEEE Transactions on visualization and computer*, 9(3) :283–297, July 2003.

[80] G.M. Nielson. Dual marching cubes. In *Proceedings of the conference on Visualization*, pages 489–496, 2004.

[81] G.M. Nielson. Dual Marching Tetrahedra : Contouring in the Tetrahedronal Environment. In *Advances in Visual Computing*, pages 183–194, 2008.

[82] G.M. Nielson and B. Hamann. The asymptotic decider : resolving the ambiguity in marching cubes. In *Proceedings of the 2nd conference on Visualization*, pages 83–91, San Diego, California, 1991. IEEE Computer Society Press.

[83] J. Patera and V. Skala. A comparison of fundamental methods for iso surface extraction. *Machine Graphics and Vision*, 13(4) :329–343, 2004.

[84] S. Petrik and V. Skala. Space and time efficient isosurface extraction. *Computers & Graphics*, 32(6) :704—-710, December 2008.

[85] H. Pottmann, J. Wallner, Q-X. Huang, and Y-L. Yang. Integral invariants for robust geometry processing. *Computer Aided Geometric Design*, 26(1) :37–60, 2009.

[86] H. Pottmann, J. Wallner, Y-L. Yang, Y-K. Lai, and S-M. Hu. Principal curvatures from the integral invariant viewpoint. *Computer Aided Geometric Design*, 24(8-9) :428–442, November 2007.

[87] F. Preparata and M. Shamos. *Computational Geometry : an Introduction*. Springer-Verlag, 1985.

[88] E. Rabani and S. Toledo. Out-of-core SVD and QR decompositions. In *SIAM Conference on Parallel Processing for Scientific Computing*, number 572, pages 1–9, 2001.

[89] S. Raman and R. Wenger. Quality isosurface mesh generation using an extended marching cubes lookup table. In *Proceedings of the 10th Joint Eurographics / IEEE - VGTC conference on Visualization*, volume 27, pages 791–798, Eindhoven, The Netherlands, May 2008. Eurographics Association.

[90] J. Salmon and M.S. Warren. Parallel, out-of-core methods for n-body simulation. In *SIAM Conference on Parallel Processing for Scientific Computing*, pages 1–11, 1997.

[91] H. Samet and M. Tamminen. Efficient Component Labeling of Images of Arbitrary Dimension Represented by Linear Bintrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4) :579–586, July 1988.

[92] S. Schaefer, T. Ju, and J. Warren. Dual Contouring : "The Secret Sauce". Technical report, Department of Computer Science, Rice University, 2002.

[93] S. Schaefer, T. Ju, and J. Warren. Manifold dual contouring. *IEEE transactions on visualization and computer graphics*, 13(3) :610–619, 2007.

[94] S Schaefer and J Warren. Dual Marching Cubes : primal contouring of dual grids. *Computer Graphics Forum*, 24(2) :195–201, 2005.

[95] J. Schreiner, C.E. Scheidegger, and C.T. Silva. High-quality extraction of isosurfaces from regular and irregular grids. *IEEE transactions on visualization and computer graphics*, 12(5) :1205–1212, 2006.

[96] J.M. Schreiner. *Uniform and Adaptive (Re) Meshing*. PhD thesis, University of Utah, 2009.

[97] R. Shekhar, E. Fayyad, R. Yagel, and J.F. Cornhill. Octree-based decimation of marching cubes surfaces. In *Proceedings of the 7th conference on Visualization*, pages 335–342, San Francisco, California, USA, 1996. IEEE Computer Society Press.

[98] H. Shen and C.R. Johnson. Sweeping simplices : a fast iso-surface extraction algorithm for unstructured grids. In *IEEE Conference on Visualization*, pages 143–150. IEEE Comput. Soc. Press, 1995.

[99] HW. Shen and C.D. Hansen. Isosurfacing in span space with utmost efficiency (ISSUE). In *Proceedings of the 7th conference on Visualization*, volume 3, pages 287–294. ACM Press, 1996.

[100] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. In *Workshop on Volume visualization*, pages 63–70, San Diego, California, USA, 1990. ACM Press.

[101] R. Shu, C. Zhou, and M.S. Kankanhalli. Adaptive marching cubes. *The Visual Computer*, 11(4) :202–217, 1995.

[102] C. Silva, Y-J. Chiang, J. El-Sana, and P. Lindstrom. Out-of-core algorithms for scientific visualization and computer graphics. In *Visualization Course Notes*, number October, 2002.

[103] C.T. Silva, E. Faufman, and C. Pavlakos. PVR : High-performance volume rendering. *IEEE Computing in Science and Engineering*, 3(4) :18–28, 1996.

[104] P. Stelldinger. Topologically correct 3d surface reconstruction and segmentation from noisy samples. In *Proceedings of the 12th international conference on Combinatorial image analysis*, volume 4958, pages 274–285, Buffalo, NY, USA, 2008. Springer-Verlag.

[105] P. Stelldinger and L.J. Latecki. 3D Object Digitization : Topology Preserving Reconstruction. In *IEEE International Conference on Image Analysis, ICPR*, pages 693–696. IEEE, 2006.

[106] P. Stelldinger, L.J. Latecki, and M. Siqueira. Topological equivalence between a 3D object and the reconstruction of its digital image. *IEEE transactions on pattern analysis and machine intelligence*, 29(1) :126–140, January 2007.

[107] G. Stockman and L.G. Shapiro. *Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.

[108] A. Szymczak and J. Vanderhyde. Extraction of topologically simple isosurfaces from volume datasets. In *Proceedings of the 14th IEEE Visualization*, pages 67–74, Seattle, WA, USA, 2003. IEEE Computer Society.

[109] G.M. Treece, R.W. Prager, and A.H. Gee. Regularised marching tetrahedra : improved iso-surface extraction. *Computers & Graphics*, 23(September) :583–598, 1998.

[110] T. Tu, D.R. Ohallaron, and J.C. Lopez. Etree : A database-oriented method for generating large octree meshes. *International Meshing Roundtable*, 20(2) :127–138, 2002.

[111] G. Varadhan, S. Krishnan, Y.J. Kim, and D. Manocha. Feature-sensitive subdivision and isosurface reconstruction. In *Proceedings of the 14th IEEE Visualization*, pages 99–106. IEEE, 2003.

[112] G. Varadhan, S. Krishnan, T. Sriram, and D. Manocha. Topology preserving surface extraction using adaptive subdivision. In *Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 235–244. Eurographics Association, 2004.

[113] V. Vidal, C. Wolf, and F. Dupont. Combinatorial mesh optimization. *The Visual Computer*, 28(5) :511–525, November 2011.

[114] J.S. Vitter. External-memory algorithms and data structures : dealing with massive data. *ACM Comput. Surv.*, 33(2) :209–271, 2010.

[115] B. Vrolijk and F. Post. Interactive out-of-core isosurface visualisation in time-varying data sets. *Computers & Graphics*, 30(2) :265–276, April 2006.

[116] I. Wald, A. Dietrich, and P. Slusallek. An interactive out-of-core rendering framework for visualizing massively complex models. In *Courses on SIGGRAPH*, page 17, Los Angeles, California, 2005. ACM Press.

[117] C. Wang and Y-J. Chiang. Isosurface extraction and view-dependent filtering from time-varying fields using Persistent Time-Octree (PTOT). *IEEE transactions on visualization and computer graphics*, 15(6) :1367–1374, 2009.

[118] C.C.L. Wang and Y. Chen. Layered depth-normal images for complex geometries-part two : manifold-preserved adaptive contouring. In *ASME International Design Engineering Technical Conferences*, volume 3, pages 729–740, New York, New York, USA, 2008. American Society of Mechanical Engineers.

[119] Q. Wang, J. JaJa, and A. Varshney. An efficient and scalable parallel algorithm for out-of-core isosurface extraction and rendering. *Parallel and Distributed Computing*, 67(5) :592–603, 2007.

[120] KW. Waters, CS. Co, and KI. Joy. Isosurface extraction using fixed-sized buckets. In *Proceedings of the Seventh Joint Eurographics / IEEE VGTC conference on Visualization*, pages 207–214, Leeds, United Kingdom, 2005. Eurographics Association.

[121] M. Weiler, R.P. Botchen, S. Stegmeier, T. Ertl, J. Huang, Y. Jang, D.S. Ebert, and K.P. Gaither. Hardware-Assisted Feature Analysis of Procedurally Encoded Multifield Volumetric Data. *IEEE Computer Graphics and Applications*, 25(5) :72–81, 2005.

[122] K. Weiss. *Diamond-based models for scientific visualization.* PhD thesis, University of Maryland, College Park, Md., 2011.

[123] K. Weiss and L. De Floriani. Diamond hierarchies of arbitrary dimension. In *Proceedings of the Symposium on Geometry Processing*, volume 28, pages 1289–1300, Berlin, Germany, July 2009. Eurographics Association.

[124] K. Weiss and L. De Floriani. Isodiamond hierarchies : An efficient multiresolution representation for isosurfaces and interval volumes. *IEEE Transactions on Visualization and Computer Graphics*, 16(4) :583–598, 2010.

[125] K. Weiss and L. De Floriani. Modeling Multiresolution 3D Scalar Fields through Regular Simplex Bisection. In *Scientific Visualization : Interactions, Features, Metaphors*, pages 360–377, 2011.

[126] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 15(2) :100–111, April 1999.

[127] J. Wilhelms and A. Van Gelder. Octrees for Faster Isosurface Generation. *ACM Transactions on Graphics (TOG)*, 11(3) :201–227, 1992.

[128] F. Yncia. Reconstruction directe d'isosurfaces sur un maillage à résolution adaptative de type octree. Technical Report 77, Université Paris-Est Marne la Vallée, Paris, France, 2008.

[129] H. Zhang and T.S. Newman. Efficient parallel out-of-core isosurface extraction. In *Parallel and Large-Data Visualization and Graphics*, pages 9–16. IEEE, 2003.

[130] N. Zhang, W. Hong, and A. Kaufman. Dual contouring with topology-preserving simplification using enhanced cell representation. In *Proceedings of the conference on Visualization*, pages 505–512. IEEE Computer Society, 2004.

[131] X. Zhang, C. Bajaj, and W. Blanke. Scalable Isosurface Visualization of Massive Datasets on COTS Clusters. In *Symposium on Parallel and Large-Data Visualization and Graphics*, pages 51–150, San Diego, CA, USA, 2001. IEEE Computer Society Press.

[132] K. Zhou, M. Gong, X. Huang, and B. Guo. Data-Parallel Octrees for Surface Reconstruction. *IEEE transactions on visualization and computer graphics*, 17(5) :669–681, May 2010.