Journées du Groupe de Travail en Modélisation Géométrique 2014, Lyon

Extraction out-of-core de surfaces à partir de volumes de grande taille

Ricardo Uribe Lobello, Florent Dupont et Florence Denis

Université Claude Bernard Lyon 1

Résumé

Les données volumiques de très grande taille sont de plus en plus présentes dans le domaine médical, de la biologie ou encore du calcul scientifique. Les méthodes d'extraction de surface de l'état de l'art se concentrent sur la vitesse d'extraction mais négligent la qualité de la surface ou n'adaptent pas le maillage à la géométrie initiale. Nous présentons dans cet article une nouvelle méthode "out-of-core" d'extraction de surfaces adaptée à des volumes de grande taille. Pour cela, nous utilisons une approche de division spatiale qui commence par diviser le volume original en sous-volumes. Ensuite, nous générons un morceau de surface à partir de chaque sous-volume en utilisant un algorithme de "Dual Contouring". Pour chaque sous-volume, nous gardons l'information nécessaire pour raccorder la surface, une fois que tous les sous-volumes ont été traités. Finalement, nous proposons un algorithme pour connecter les morceaux de surface afin de produire une surface unique. L'approche proposée génère des surfaces adaptées à la géométrie et à la topologie de l'objet original. La méthode peut être utilisée de manière séquentielle et nous l'avons aussi adaptée dans une version qui permet une exécution en parallèle afin de réduire le temps d'extraction de la surface. Par rapport à l'état de l'art, notre approche fournit un bon compromis entre temps d'exécution, qualité de la surface et de l'approximation.

Mots-clés : Extraction de surfaces, Out-of-core, modélisation multi-résolution, données volumiques

1. Introduction

Les données volumiques sont très présentes dans divers domaines de recherche. Elles peuvent être obtenues, par exemple, à partir d'images de résonance magnétique (IRM) ou d'un empilement d'images de coupes physiques de tissus organiques comme dans le cas des scanners digitaux. La représentation volumique est peu adaptée pour certaines opérations comme la visualisation ou la simulation, pour lesquelles il est préférable de disposer de la surface qui sépare l'intérieur de l'objet de l'extérieur. Par ailleurs, l'amélioration des dispositifs de capture d'images permet désormais de produire des images de très haute résolution dont les dimensions peuvent atteindre plusieurs milliers voire dizaines de milliers de voxels dans chaque dimension. Le traitement de ces données massives exige la conception de nouveaux algorithmes ne nécessitant pas le chargement de toutes les données en mémoire, en permettant de les traiter par morceaux sur des machines à ressources limitées. Dans cet article, nous nous sommes intéressés au problème de l'extraction de surfaces adaptatives et topologiquement correctes à partir des données volumiques binaires de grande taille.

2. État de l'art

La plus grande partie des méthodes d'extraction de surfaces à partir des données volumiques, ont été conçues comme des méthodes "in-core". De telles méthodes ont besoin de charger complètement le volume en mémoire afin d'extraire la surface. Jusqu'ici, ces méthodes se sont focalisées sur l'amélioration de la qualité topologique et géométrique de la surface obtenue. L'une des premières d'entre elles est l'algorithme bien connu des "Marching Cubes" (MC) [LC87] qui divise régulièrement le volume en cellules cubiques et utilise une look-up table (LUT) pour approximer la topologie et la géométrie de la surface à l'intérieur de chaque cellule. Il génère ainsi des surfaces régulières mais sa LUT ne contient pas toutes les configurations possibles pour la topologie de la surface à l'intérieur d'une cellule. Différents travaux ont apporté des extensions à cette LUT afin de mieux reproduire la topologie de la

surface [Che95, CEPS13] mais ces modifications continuent à utiliser une grille régulière. L'algorithme des "Marching Tetrahedra" (MT) divise le volume en cellules tétraédriques et utilise une LUT simplifiée pour extraire la surface. Pourtant, cette méthode multiplie le nombre de triangles produits et génère beaucoup de triangles de mauvaise qualité.
Afin de générer des surfaces adaptatives, quelques approches remplacent la grille régulière par un octree et appliquent des critères topologiques pour mieux contrôler la subdivision et obtenir une subdivision adéquate pour la LUT de MC [KBSS01, VKSM04, ZHK04]. D'autres méthodes limitent la différence entre les cellules voisines à un seul niveau ou proposent des structures intermédiaires afin de bien connecter les polygones générés par MC sur les zones de changement de résolution [WKE99, KKDH07]. Néanmoins, ces algorithmes ne résolvent pas les problèmes de qualité de la triangulation de la surface.

Les méthodes dites "duales" proposent de générer une surface duale à celle de MC. La principale caractéristique de ces méthodes est qu'elles génèrent des sommets à l'intérieur de chaque cellule au lieu de le faire sur les arêtes de la cellule. Les premiers algorithmes duaux construisent des surfaces adaptatives mais peuvent faire apparaître des configurations non-variété [JLSW02, Nie04]. Des travaux postérieurs ont permis de surmonter ces problèmes en utilisant des critères topologiques [VKKM03, SJW07, UDD12]. D'autres approches duales proposent de construire une grille duale alignée sur les caractéristiques géométriques du volume et d'extraire la surface en utilisant un algorithme de MC ou de MT [SW04, MS10]. Malheureusement, l'utilisation de grilles duales peut produire des auto-intersections et n'améliore pas la qualité de la triangulation.

Afin de traiter des volumes de données plus importants, la plus grande partie des algorithmes utilisent des variantes de MC et se focalisent sur l'accès rapide aux données ou le traitement de données issues des séquences temporelles [SCESL02, ZN03, WJV07]. Ces algorithmes projettent les points dans un espace de dimension n dans un espace 2D (espace de longueur) et construisent des structures de division spatiale pour accélérer l'accès aux cellules traversées par la surface comme illustré sur la figure 1.

Par ailleurs, les approches mentionnées jusqu'ici sont des algorithmes "in-core", limités par la quantité de mémoire vive disponible. D'autres approches ont présenté des structures de données mieux adaptées pour les traitements out-of-core. Ainsi, Chiang *et al.* [CS97, CSS98, CS99] ont introduit le concept de *méta-cellule* que nous allons aussi utiliser (voir illustration sur la figure 2). Ces solutions ont été principalement conçues pour la génération rapide de surfaces régulières pour la visualisation ou la simulation [WDS05].

Les algorithmes proposés par Bajaj *et al.* [BPTZ99] et Zhang *et al.* [ZHK04] ont principalement cherché à fournir un accès rapide aux données sans s'intéresser à la qualité de la surface produite ou au nombre de facettes créées. De la même manière, Wang *et al.* [WC09] peut générer des maillages de plusieurs centaines de millions de triangles mais, du fait qu'il utilise une version modifiée de MC, les surfaces produites contiennent une grande quantité de triangles allongés et ne sont pas adaptées à la géométrie du volume original.
La méthode proposé par Gregorsky *et al.* utilise les diamants étudiés par Weiss et De Floriani [WD10] pour construire, par pré-traitement, une hiérarchie de diamants indexée par les iso-valeurs minimale et maximale de chaque cellule tétraédrique. Cette hiérarchie supporte des requêtes du type : "Quelles sont les cellules traversées par une surface à une iso-valeur donnée ?". De plus, elle permet d'extraire des surfaces adaptatives en utilisant la LUT de MT. Cet algorithme

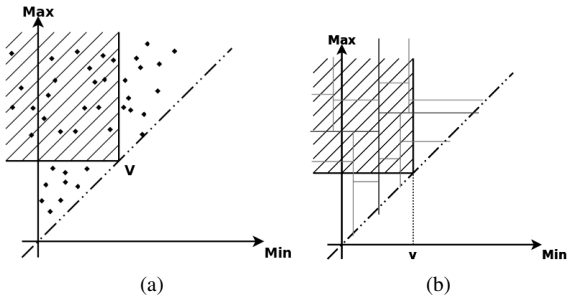


Figure 1: (a) Dans un espace de longueur (span space) les valeurs minimale et maximale de l'iso-surface sont utilisées comme les coordonnées (x,y) d'un point dans un espace bi-dimensionnel. Les intervalles 2D qui contiennent la valeur v sont à gauche de la droite $x = v$ et au-dessus de la droite $y = v$. (b) L'algorithme divise l'espace des valeurs avec un Kd-tree afin d'optimiser le temps d'accès.

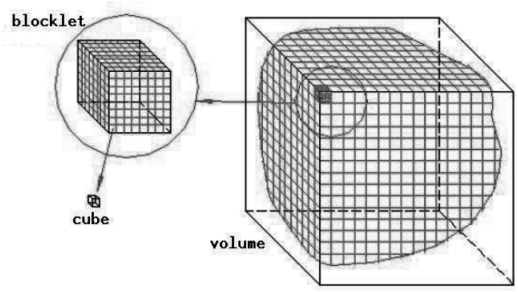


Figure 2: Illustration de la division du volume dans l'algorithme de Zhang et Newman [ZN03] et utilisé aussi par Chiang *et al.* [CS99]. Le volume est divisé en "blocklets" de $8 \times 8 \times 8$ voxels et chaque cube ou voxel est traité par l'algorithme des "Marching Cubes".

est fortement lié à la carte graphique (GPA) afin de fournir un rendu accéléré de volumes qui changent avec le temps. Néanmoins, l'utilisation de MT implique que les surfaces construites vont contenir une grande quantité de triangles (entre 2 et 3 fois ceux de MC) et qu'une grande partie de ces triangles seront allongés ou aplatis. Enfin, Schreiner *et al.* [SSS06, Sch09] ont proposé une méthode pour générer des surfaces massives avec une technique par avancée de fronts (advancing front) afin d'améliorer la qualité générale des triangles sur la surface. Néanmoins, cette méthode peut avoir des temps d'exécution très longs.
La méthode proposée dans cet article s'intéresse au cas de données volumiques massives. Elle cherche à fournir un bon compromis entre le temps d'exécution, les ressources nécessaires et la qualité de la surface obtenue par rapport à la topologie et la géométrie de l'objet volumique original.

3. Notre approche

Notre méthode utilise une approche "Diviser pour mieux régner". Nous commençons par diviser le volume original en sous-volumes que nous appellerons désormais *méta-cellules*. Ensuite, nous utilisons la méthode duale décrite dans [UDD12] pour extraire un morceau de surface à partir de chaque méta-cellule de manière indépendante. Pendant la génération des surfaces, nous gardons l'information nécessaire pour raccorder les morceaux entre eux. Enfin, nous proposons un algorithme pour raccorder les différents morceaux de surface afin d'obtenir une surface unique. Ce pipeline est illustré sur la figure 3.

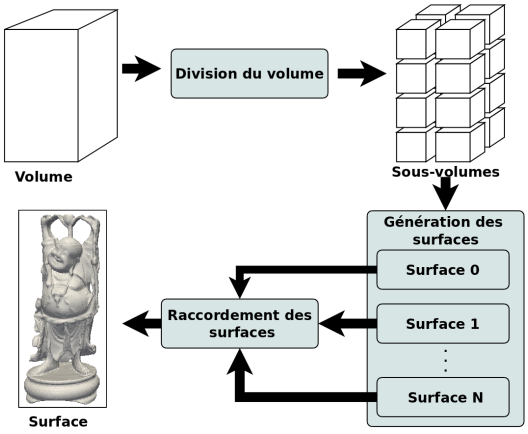


Figure 3: Chaîne de traitement de notre algorithme out-of-core. Une fois le volume divisé en méta-cellules, celles-ci peuvent être traitées de manière indépendante pour générer des morceaux de surface. Ensuite, les morceaux sont connectés pour former la surface finale.

3.1. Division du volume

Chaque méta-cellule contient des méta-facettes, méta-arêtes et méta-sommets composés des voxels contenus respectivement dans les facettes, les arêtes et les sommets d'une méta-cellule. Diverses considérations doivent être prises en compte au moment de construire les méta-cellules. La première est que le volume peut seulement être subdivisé de manière conforme, selon la définition suivante :

Définition 1 ***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.

Nous ne pouvons pas avoir de méta-cellules fines qui partagent seulement une partie de la méta-facette d'une méta-cellule plus grande comme illustré sur la figure 4 à gauche. Cependant, comme dans l'exemple sur la figure 4 à droite, il est possible d'avoir des méta-cellules de tailles différentes dans une direction (axe X) si elles partagent une méta-facette dans les deux autres dimensions (Y et Z).

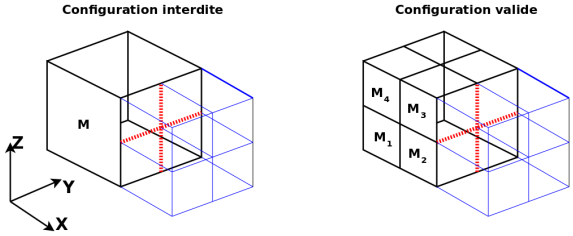


Figure 4: Division d'un volume en méta-cellules. À gauche, la subdivision n'est pas conforme parce que l'intersection de la méta-cellule M et des méta-cellules plus fines n'est pas une méta-facette pour toutes les méta-cellules. À droite, la division de la cellule M en méta-cellules $M_1, M_2, ..., M_8$ est conforme.

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 5 illustre une méta-facette partagée par deux méta-cellules.

Le volume original peut être divisé en utilisant un prétraitement qui charge les coupes $x-y$ du volume original et le parcourt afin de le découper et de l'enregistrer dans plusieurs fichiers selon la quantité de subdivisions à réaliser sur

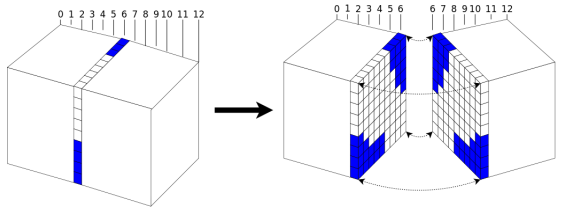


Figure 5: Illustration d'une méta-facette dans la position 6 partagée par deux méta-cellules (à gauche). Seuls les voxels sur la facette partagée sont visibles.

le plan $x-y$. Cette étape est réalisée pour chaque coupe ou ensemble de coupes en fonction de la mémoire disponible comme l'illustre partiellement la figure 6 qui présente les coupes avec des couleurs différentes.

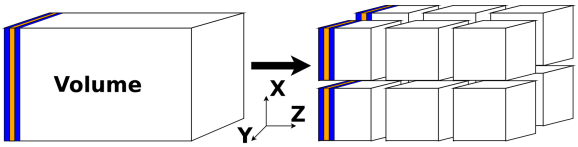


Figure 6: Subdivision anisotropique du volume en 12 méta-cellules.

Cette procédure nous permet de pré-traiter des volumes de taille quelconque afin de générer les méta-cellules les plus appropriées en fonction des ressources (mémoire et puissance de calcul) disponibles.

3.2. Génération de surfaces

Notre algorithme construit une surface en utilisant l'algorithme dual "Manifold Dual Marching Cubes" (MDMC) que nous avons proposé [UDD12]. Cette méthode a pour caractéristique de générer des surfaces duales de celles générées par l'algorithme de MC. En conséquence, pour chaque patch de surface produit par MC, MDMC génère un sommet dual et, inversement, pour chaque sommet placé par MC sur une arête, MDMC génère un patch de surface qui traverse cette arête.
Les détails de cet algorithme sont en dehors du champ de cet article mais quelques points sont importants à souligner pour la solution présentée ici. MDMC permet de détecter les configurations topologiquement complexes sur les arêtes et les facettes des cellules. Ces configurations sont éliminées à l'aide d'une subdivision progressive des cellules. Ce processus s'arrête quand aucune des cellules traversées par la surface ne contient de configuration complexe.
Ce test s'applique aussi sur les cellules touchant les méta-facettes de méta-cellules et, combiné avec le fait que deux méta-cellules adjacentes partagent le même ensemble de voxels, il permet d'éviter l'apparition de problèmes de couplage entre les morceaux des surfaces générés dans des méta-cellules adjacentes. Cette caractéristique de la subdivision permet également d'éviter la propagation des problèmes de couplage entre les deux subdivisions à l'intérieur des méta-cellules.

Afin de pouvoir raccorder effectivement les surfaces entre elles, il faut conserver une certaine quantité d'information sur la subdivision qui a permis de construire la surface à l'intérieur de chaque méta-cellule. Nous avons identifié que les seules informations nécessaires étaient les cellules de l'octree adjacentes à chaque méta-facette. Pour cette étape, nous utilisons la cohérence spatiale du code de Morton (expliqué en [UDD12]) afin d'établir quelle sont les cellules à stocker. 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, le bit le plus à droite étant 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, la valeur (1 ou 0) de b_1^2 donne la position relative de la cellule par rapport à l'axe X, à la profondeur 2. Ainsi, pour obtenir les cellules d'un octree adjacentes à une facette f perpendiculaire à l'axe X dans la direction négative, nous devons considérer seulement les cellules feuilles de l'octree dont le bit le moins significatif à chaque niveau vaut zéro. Par exemple, le code 1 10**0** 11**0** 00**0** représente une cellule au troisième niveau de profondeur, adjacente à f. Les valeurs des bits à prendre en compte pour chaque orientation sont indiquées dans le tableau 1.

Direction	**Position du bit (Axe de référence)**	b_i^n
X négatif	1	0
X positif	1	1
Y négatif	2	0
Y positif	2	1
Z négatif	3	0
Z positif	3	1

Tableau 1: Direction, position et valeur du bit b_i^n du code de Morton pour les cellules adjacentes à chacune des facettes d'une méta-cellule par rapport à l'index de la facette.

Nous avons vérifié que les cellules adjacentes aux méta-facettes représentent une très petite fraction des cellules intérieures de l'octree. L'algorithme général pour l'extraction de ces cellules est décrit en 1.

4. Raccordement de la surface

La phase de raccordement utilise uniquement les informations des cellules gardées dans l'étape de génération des

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 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

morceaux de surface. Les données volumiques ne sont ensuite plus nécessaires et peuvent être entièrement déchargées de la mémoire.

4.1. 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 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.

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 génère des polygones seulement à l'intérieur des cellules qui partagent une arête minimale traversant la surface ∂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

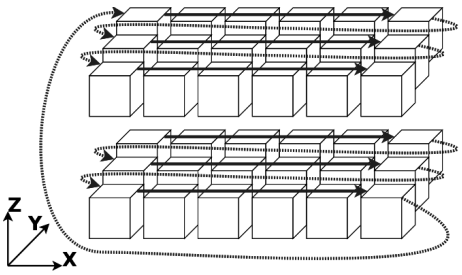


Figure 7: Ordre de parcours des méta-cellules par rapport à l'orientation du volume.

dans l'étape de génération de surface.
Néanmoins, il ne suffit pas d'identifier les arêtes pour compléter la surface car, pour générer des polygones, notre algorithme a besoin de toutes les cellules qui partagent l'arête. Ces cellules se trouvant dans des méta-cellules adjacentes, il est nécessaire de construire des structures intermédiaires.

4.1.1. Construction des Edge-octrees

Les arêtes intersectant la surface peuvent être contenues dans une méta-facette ou sur 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 des méta-arêtes et des méta-facettes que nous devons explorer. Ces configurations sont illustrées sur la figure 8.

La figure 8 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. Il est aussi nécessaire de mémoriser 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. Les préfixes à ajouter aux cellules sont listés dans le tableau 2 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 $B_{i+1,j+1,k}$.

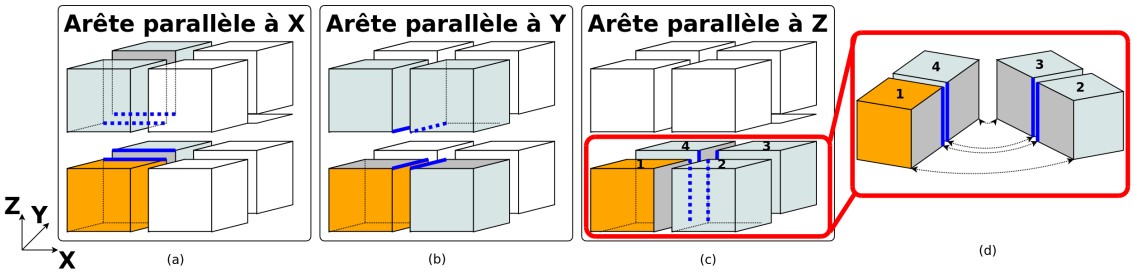


Figure 8: Configurations des méta-arêtes (en gras) à utiliser pour connecter les surfaces. La méta-cellule orange est utilisée comme repère pour construire l'edge-octree. Les méta-facettes à considérer (en gris) sont celles parallèles au plan XZ pour la méta-arête alignée sur X (a), au plan XY pour la méta-arête alignée sur Y (b) et au 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).

Préfixes des codes de Morton		
Arête	**Méta-cellules**	**Code de Morton**
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
Parallèle à Y	$B_{i,j,k}$	000
	$B_{i+1,j,k}$	00**1**
	$B_{i,j,k+1}$	**1**00
	$B_{i+1,j,k+1}$	**101**
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 2: Préfixes à ajouter aux cellules dans chaque méta-cellule afin de construire les edge-octrees par rapport aux orientations des méta-arêtes.

L'addition des préfixes du tableau 2 aura pour effet de remplacer les méta-cellules par des cellules et d'augmenter de 1 la profondeur représentée par les 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 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 9 où la méta-arête centrale (en rouge) 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 leur code de Morton est modifié pour refléter leur position dans la nouvelle structure.

Il existe un cas particulier dans la construction de ces cellules intermédiaires, pour les méta-arêtes localisées sur les limites externes du volume original mais elles ne sont pas

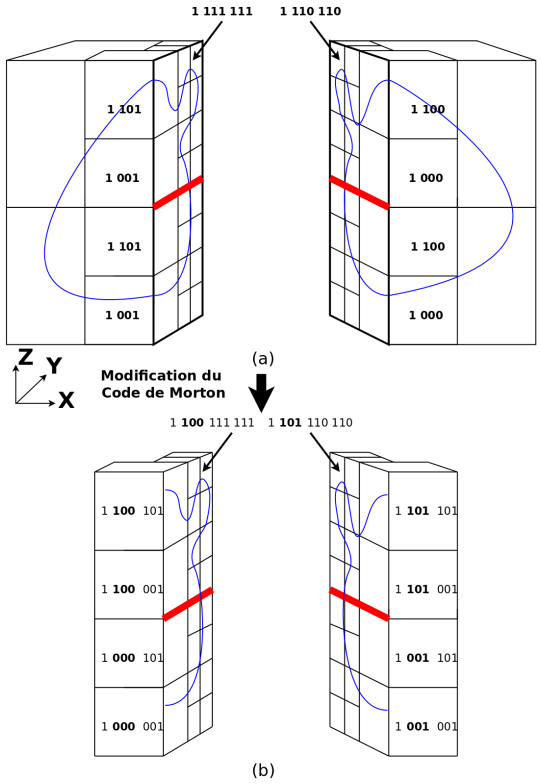


Figure 9: Illustration de la modification des codes de Morton dans la construction d'un edge-octree. En (a), la méta-arête en rouge est utilisée pour identifier les cellules qui feront partie de l'edge-octree. En (b), les codes de Morton des cellules adjacentes à la méta-arête sont modifiés (en gras) pour refléter leur position spatiale et hiérarchique dans le nouveau edge-octree.

traitées du fait qu'un objet volumique touchant les bords du domaine contient par définition des bords.

4.2. Génération de polygones

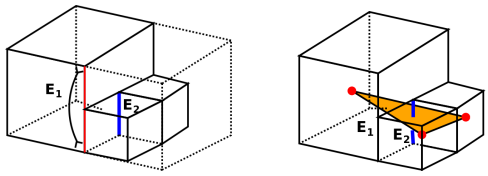
Une fois l'edge-octree construit, nous devons connecter les sommets duaux contenus à l'intérieur des cellules. Dans ce but, nous avons développé un algorithme qui n'a pas besoin de parcourir toute la hiérarchie de l'octree et utilise seulement les cellules feuilles pour générer les polygones de la surface. Un parcours de la structure intermédiaire permet de détecter les cellules qui sont intersectées par la surface ∂V. Pour chacune, nous parcourons toutes les arêtes qui coupent la surface et identifions les cellules qui les partagent. Pour avoir un accès rapide aux cellules voisines par leur code de Morton, nous devons considérer l'orientation de chaque arête par rapport au repère de la cellule.

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 les 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 de la hiérarchie (en réalisant une opération de décalage de 3 bits dans les code de Morton) qu'aux niveaux inférieurs où il faut choisir entre plusieurs cellules filles partageant une partie de l'arête en question (voir figure **??**a).



(a) (b)

Figure 10: (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 (points rouges) peuvent alors être connectés.

Notre algorithme ne génère des sommets qu'à l'intérieur des cellules. En conséquence, aucun sommet n'est produit pendant la phase de raccordement.

5. Résultats

Cette section décrit les principaux résultats de notre méthode sur un ensemble de données volumiques. Nous avons mesuré ses performances avec plusieurs critères tels que l'approximation géométrique, le temps d'exécution et la quantité de mémoire nécessaire par rapport à la taille du volume d'entrée. Finalement, nous avons mis en œuvre une stratégie de parallélisation et mesuré sa performance.
L'ensemble des données de test a été obtenu de deux manières différentes. Une première partie des volumes a été construite par discrétisation de surfaces. Ainsi, pour une surface ∂V et un domaine Ω cette méthode va nous fournir des volumes binaires de voxels avec une résolution donnée. 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 acquises selon diverses modalités. Après segmentation, ces données 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.

5.1. Caractéristiques des surfaces

Nous avons 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 $\lambda = 0.9$.

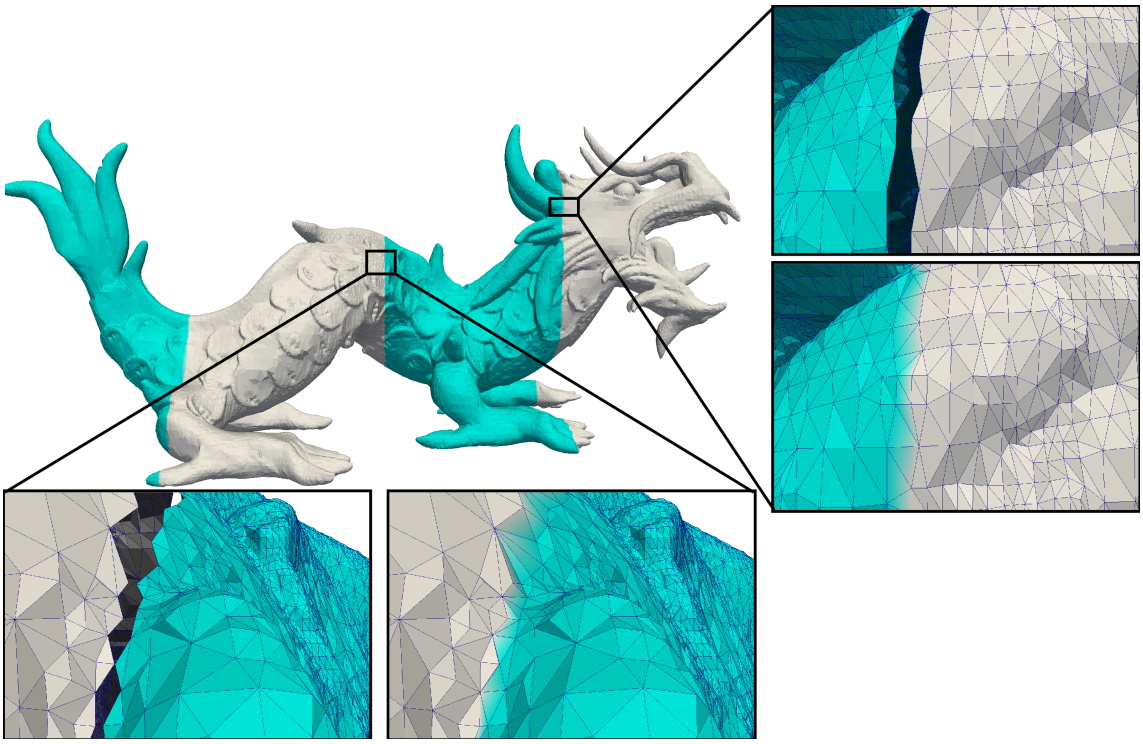


Figure 11: Génération de la surface à partir d'un volume $2048 \times 512 \times 512$ voxels. Le volume a été divisé en 4 méta-cellules selon X. Les surfaces ont été extraites avec un octree de profondeur 8 et un seuil de courbure de 0.9. Les zooms permettent d'observer la qualité de la triangulation dans les zones de connexion de la surface.

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. Sur la figure 11, 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 peuvent contenir des cellules de tailles différentes pourvu que la topologie de la subdivision induite par les cellules sur les méta-facettes ne soit pas complexe.

Pour deux méta-cellules voisines, notre méthode peut générer des surfaces ouvertes avec des frontières topologiquement simples pour chaque cellule de l'octree parce que chaque méta-cellule partage avec sa voisine une méta-facette de voxels. Néanmoins, pour valider le fait 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 dis-

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

Tableau 3: Erreurs géométriques des surfaces extraites par rapport aux surfaces originales. Les surfaces ont été extraites à partir de volumes de 1024^3 voxels. Les octrees utilisés ont un profondeur maximale 7 et un seuil de courbure 0.9. Nous avons mesurés la distance RMS et la distance de Hausdorff pour quatre découpages avec $M = 1, 8, 64$ et 512 méta-cellules.

tance de Hausdorff que nous avons estimée avec la librairie METRO [CRS98]. 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 3. Toutes les surfaces ont été extraites à partir d'octrees de profondeur 7 pour un seuil de courbure $\lambda = 0.9$.
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, 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 permettre 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.

6. 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 4 montre les caractéristiques des surfaces générées (nombre de sommets et de facettes) et les temps d'exécution de notre algorithme sur un ensemble de données volumiques de $2048 \times 2048 \times 1024$ voxels ($4Go$) divisé en 32 méta-cellules. Les temps affichés correspondent à l'ensemble des méta-cellules. Dans la colonne "Volume" est indiqué le temps nécessaire pour charger le volume en mémoire. Dans la colonne "Octree" nous listons les temps de construction de l'octree en incluant les tests topologiques sur le volume. La colonne suivante "Surface" 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.

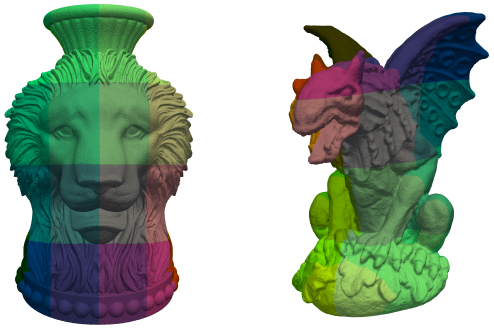
Ce tableau confirme que la majeure partie du temps est consacrée à l'application des critères sur les données volumiques ainsi qu'à la construction des octrees. Le temps de raccordement des surfaces est vraiment court et représente

Modèles \ Critères	Taille de la Surface	
	#Points	**#Triangles**
Dragon	842.069	1.684.096
Asian Dragon	558.268	1.097.417
Wales Dragon	994.292	1.988.619
Armadillo	538.594	1.077.259
Filigree	1.042.428	2.084.817
Buddha	971.068	1.942.056
Gargoyle	860.073	1.720.102
Vase Lion	866.962	1.733.898

Modèles	**Temps d'exécution (min : sec)**				
	Volume	**Octree**	**Surface**	**Racc.**	**Total**
Dragon	0 :57	2 :30	1 :13	0 :02	4 :42
Asian Dragon	0 :54	1 :45	0 :45	0 :02	3 :26
Wales Dragon	0 :56	2 :43	1 :40	0 :03	5 :22
Armadillo	0 :36	1 :45	0 :45	0 :01	3 :07
Filigree	0 :54	3 :00	1 :40	0 :03	5 :37
Buddha	0 :57	2 :49	1 :28	0 :02	5 :16
Gargoyle	0 :53	2 :28	1 :09	0 :01	4 :31
Vase Lion	0 :48	2 :27	1 :16	0 :01	4 :32

Tableau 4: Taille de la surface (haut) et temps d'exécution (bas) 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.

moins de 2% du temps total d'exécution. Quelques exemples des surfaces obtenues peuvent être observés sur la figure 12.



(a) Vase Lion (b) Gargoyle

Figure 12: 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.

Dans le tableau 5, nous pouvons observer les quantités estimées de mémoire vive nécessaire 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 besoin 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.

Volumes	**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 5: 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.

Notre algorithme permet à l'utilisateur de trouver un bon équilibre entre la quantité et la taille des méta-cellules afin d'optimiser l'utilisation de la mémoire et de la puissance de calcul disponibles.

6.1. Temps d'exécution avec pallélisation

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 "multithread" afin de paralléliser l'étape d'extraction de surfaces à partir de plusieurs méta-cellules. L'avantage des "threads" est qu'ils partagent le même espace mémoire alloué pour tout le programme. Cela évite d'utiliser des fichiers 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 (tableaux 4 et 5) 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 6 correspondent aux pics de temps utilisé par le plus lent des quatre processeurs fonctionnant de manière concurrente.

Volumes	**Temps - Pics par thread (min :sec)**				
	Volume	**Octree**	**Surface**	**Racc.**	**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 6: Temps d'exécution de la version "multithread" 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 4.

Les résultats du tableau 6 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 aurions 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. Cela résulte probablement, soit des effets de "False sharing" qui peuvent bloquer les processus entre eux s'ils essaient d'accéder aux mêmes zones de la mémoire, soit d'un goulot d'étranglement dans l'accès sur le disque. Cette parallélisation peut être clairement améliorée et nous y travaillerons dans le futur.
Pour le moment, 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.

6.2. Application sur des volumes de données de grande 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 de 512^3 voxels. L'algorithme a été exécuté sur un ordinateur à 4 double-cœurs de 2.4 GHz chacun. Les résultats sont présentés dans le tableau 7.

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 7: 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.

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 ou utiliser un niveau de résolution plus élevé afin d'obtenir une surface massive.

7. Conclusions et perspectives

Nous avons proposé un algorithme d'extraction de surfaces adaptatives à partir de données volumiques massives qui utilise une approche "diviser pour mieux régner". L'algorithme proposé a l'avantage de générer des surfaces adaptatives sans limite théorique sur la taille de la surface à générer. 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. Par rapport aux méthodes de l'état de l'art, qui presque toutes utilisent l'algorithme "Marching Cubes" ou une de ses versions améliorées pour générer la triangulation et dont les surfaces construites ne sont pas adaptatives et contiennent une grande quantité de triangles dégénérés, notre méthode produit des surfaces avec une résolution adaptée à la courbure locale et présentant des triangles de bonne qualité. Et même si des algorithmes qui construisent des surfaces adaptatives existent, ils utilisent MT pour extraire la surface, ce qui multiplie la quantité de triangles produits et provoque l'apparition de beaucoup de triangles allongés. Enfin, nous avons proposé une implémentation parallèle de notre solution qui permet de réduire le temps d'extraction de la surface.

Dans nos travaux futurs, nous envisageons d'utiliser notre algorithme comme un outil de pré-traitement 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 et seront combinées avec des structures de recherche optimales afin de produire un outil interactif de visualisation de surfaces adaptatives et multi-résolution qui permette de manipuler des données de grandes dimensions. Une telle application peut être complétée avec des techniques de compression et de quantification pour la transmission et le stockage des modèles.
Finalement, nous envisageons aussi de développer une version complètement parallèle de notre algorithme de manière à exploiter le parallélisme grandissant des processeurs existants.

Références

[BPTZ99] BAJAJ C. L., PASCUCCI V., THOMPSON D., ZHANG X. Y. : Parallel accelerated isocontouring for out-of-core visualization. In *IEEE symposium on Parallel visualization and graphics* (Washington, DC, USA, 1999), PVGS '99, IEEE Computer Society, pp. 97–104.

[CEPS13] CUSTODIO L., ETIENE T., PESCO S., SILVA C. : Practical considerations on marching cubes 33 topological correctness. *Computers & Graphics. Vol. 37*, Num. 7 (2013), 840 – 850.

[Che95] CHERNYAEV E. : Marching cubes 33 : Construction of topologically correct isosurfaces. *Institute for High Energy Physics, Moscow, Russia* (1995), 1–8.

[CRS98] CIGNONI P., ROCCHINI C., SCOPIGNO R. : Metro : measuring error on simplified surfaces.

[CS97] CHIANG Y.-J., SILVA C. : I/o optimal isosurface extraction. In *Visualization '97* (oct. 1997), pp. 293 –300.

[CS99] CHIANG Y.-J., SILVA C. T. : External memory algorithms. American Mathematical Society, Boston, MA, USA, 1999, ch. External memory techniques for isosurface extraction in scientific visualization, pp. 247–277.

[CSS98] CHIANG Y.-J., SILVA C. T., SCHROEDER W. J. : Interactive out-of-core isosurface extraction. In *Visualization* (Los Alamitos, CA, USA, 1998), IEEE Computer Society Press, pp. 167–174.

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J. : Dual contouring of hermite data. SIGGRAPH, ACM, pp. 339–346.

[KBSS01] KOBBELT L. P., BOTSCH M., SCHWANECKE U., SEIDEL H.-P. : Feature sensitive surface extraction from volume data. SIGGRAPH, ACM, pp. 57–66.

[KKDH07] KAZHDAN M., KLEIN A., DALAL K., HOPPE H. : Unconstrained isosurface extraction on arbitrary octrees. In *Eurographics symposium on Geometry processing* (2007), Eurographics Association, pp. 125–133.

[LC87] LORENSEN W., CLINE H. : Marching cubes : A high resolution 3D surface construction algorithm. In *SIGGRAPH* (1987), vol. 87, ACM, pp. 163–169.

[MS10] MANSON J., SCHAEFER S. : Isosurfaces over simplicial partitions of multiresolution grids. *Computer Graphics Forum. Vol. 29*, Num. 2 (2010), 377–385.

[Nie04] NIELSON G. M. : Dual marching cubes. In *Visualization* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 489–496.

[SCESL02] SILVA C., CHIANG Y., EL-SANA J., LINDSTROM P. : Out-of-core algorithms for scientific visualization and computer graphics. In *Visualization Course Notes* (2002).

[Sch09] SCHREINER J. M. : *Uniform and adaptive (re)meshing of surfaces*. PhD thesis, Salt Lake City, UT, USA, 2009.

[SJW07] SCHAEFER S., JU T., WARREN J. : Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics. Vol. 13* (May 2007), 610–619.

[SSS06] SCHREINER J., SCHEIDEGGER C. E., SILVA C. T. : High-quality extraction of isosurfaces from regular and irregular grids. *IEEE transactions on visualization and computer graphics. Vol. 12*, Num. 5 (2006), 1205–12.

[SW04] SCHAEFER S., WARREN J. : Dual marching cubes : Primal contouring of dual grids. Pacific Graphics, IEEE Computer Society, pp. 70–76.

[UDD12] URIBE LOBELLO R., DUPONT F., DENIS F. : Multi-Resolution dual contouring from volumetric data. In *International Conference on Computer Graphics Theory and Applications* (février 2012).

[VKKM03] VARADHAN G., KRISHNAN S., KIM Y., MANOCHA D. : Feature-sensitive subdivision and isosurface reconstruction. In *Visualization, IEEE* (2003), IEEE, pp. 99–106.

[VKSM04] VARADHAN G., KRISHNAN S., SRIRAM T., MANOCHA D. : Topology preserving surface extraction using adaptive subdivision. *Eurographics Symposium on Geometry processing* (2004), 235–244.

[WC09] WANG C., CHIANG Y.-J. : Isosurface extraction and view-dependent filtering from time-varying fields using persistent time-octree (ptot). *IEEE Transactions on Visualization and Computer Graphics. Vol. 15*, Num. 6 (novembre 2009), 1367–1374.

[WD10] WEISS K., DE FLORIANI L. : Isodiamond hierarchies : An efficient multiresolution representation for isosurfaces and interval volumes. *IEEE Transactions on Visualization and Computer Graphics. Vol. 16*, Num. 4 (2010), 583–598.

[WDS05] WALD I., DIETRICH A., SLUSALLEK P. : An interactive out-of-core rendering framework for visualizing massively complex models. *ACM Courses on SIGGRAPH* (2005), 17.

[WJV07] WANG Q., JAJA J., VARSHNEY A. : An efficient and scalable parallel algorithm for out-of-core isosurface extraction and rendering. *J. Parallel Distrib. Comput.. Vol. 67*, Num. 5 (mai 2007), 592–603.

[WKE99] WESTERMANN R., KOBBELT L., ERTL T. : Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer. Vol. 15*, Num. 2 (avril 1999), 100–111.

[ZHK04] ZHANG N., HONG W., KAUFMAN A. : Dual contouring with topology-preserving simplification using enhanced cell representation. In *Visualization* (2004), pp. 505–512.

[ZN03] ZHANG H., NEWMAN T. : Efficient parallel out-of-core isosurface extraction. In *Parallel and Large-Data Visualization and Graphics* (2003), IEEE, pp. 9–16.