N° d'ordre : 133 — 2009 Année 2009

UNIVERSITÉ CLAUDE BERNARD - LYON 1
UFR Informatique – Laboratoire LIRIS, UMR 5205 CNRS, Lyon
École doctorale « Informatique et Mathématiques » de Lyon

THÈSE DE L'UNIVERSITÉ DE LYON

pour l'obtention
du DIPLÔME DE DOCTORAT
(arrêté du 7 août 2006)

présentée par
Clément JAMIN

Algorithmes et structures de données compactes pour la visualisation interactive d'objets 3D volumineux

---

Thèse dirigée par Pierre-Marie Gandoin et Samir Akkouche
soutenue publiquement le 25 septembre 2009 devant le jury composé de :

Olivier Devillers	Directeur de Recherche à l'INRIA Sophia-Antipolis Méditerranée	Rapporteur
Dominique Michelucci	Professeur à l'Université de Bourgogne	Rapporteur
Bruno Lévy	Directeur de Recherche INRIA Nancy Grand Est	Examinateur
Jean Séqueira	Professeur à l'Université de la Méditerranée	Examinateur
Samir Akkouche	Professeur à l'Université Lyon 1	Directeur de thèse
Pierre-Marie Gandoin	Maître de Conférences à l'Université Lyon 2	Co-encadrant

« La qualité d'un homme se calcule à sa démesure ; tentez, essayez, échouez même, ce sera votre réussite. »
Jacques Brel.

Remerciements

En premier lieu, je tiens à remercier Pierre-Marie pour son encadrement scientifique et « littéraire », et pour la confiance qu'il m'a accordée depuis mes premiers pas dans l'équipe GeoMod du LIRIS jusqu'à la rédaction de ce mémoire. Il a su me laisser l'autonomie dont j'avais besoin, tout en étant présent dans les moments de doute. Toujours encourageant, jamais défaitiste, il a eu le don de trouver les mots qu'il fallait lorsque je m'interrogeais sur la pertinence de mon travail. Besoin d'aide ou de soutien moral ? Un mail de Pierre-Marie m'attendait déjà dans ma boîte aux lettres. Je pense en particulier aux quelques refus d'articles qui ont jalonné mon parcours de doctorant, à l'occasion desquels il m'a toujours soutenu pour repartir de plus belle. Tu avais raison, nous avons fini par remporter la bataille !

Je remercie également mon directeur de thèse, Samir, pour son encadrement et sa disponibilité malgré ses lourdes responsabilités, que ce soit lorsque j'avais besoin d'un avis stratégique, d'un conseil avisé ou d'une signature sur un document de la plus haute importance.

Je remercie Olivier Devillers et Dominique Michelucci d'avoir accepté d'être rapporteurs de ce mémoire, ainsi que Bruno Lévy et Jean Séqueira d'avoir donné leur accord pour participer au jury de soutenance.

J'adresse un grand merci également aux personnes que j'ai côtoyées de près ou de loin lors de ces trois années au LIRIS et plus généralement à l'Université Lyon 1. Je pense en particulier à Gabriel, Jamal, Houssam, Rémi, Brett, Nicolas, Adrien, Romain, Anne-Laure, Anthony, Olivier, Lucian, Céline, Jacques, Catarina, Etienne, Raphaëlle, les trois Eric, Thierry, Eliane, Jérôme, Valérie, Julie, Laurence, Sylvain, et j'en oublie sûrement… Merci à Brigitte, Sylvie, Jean-Pierre et Eric pour leur aide administrative ou technique.

Ces remerciements auraient un goût d'inachevé si je ne remerciais pas Nadia — sourires, intelligence et volonté inaltérable, tout en un —, mes parents — tout simplement formidables — parmi lesquels se trouve étonnamment ma mère — c'est la plus belle — et mon père — c'est le plus fort —, mon frère — pour nos nombreuses discussions scientifiques, pour sa coupe de cheveux

et son cerf-volant de 5,9 m^2 — et Hélène — où en est mon diagnostic neuropsychologique ? —, ma grand-mère Bernadette — pour sa gentillesse, sa verve inaltérable et ses petits gâteaux —, et mon grand-père René — pour sa gentillesse, ses fleurs et son sens de l'humour discret mais indéniable. Sans oublier Patricia, Didier, Sabine et Manu pour les bons moments passés ensemble et les discussions où l'ardeur et la véhémence cohabitent avec la ruse et le mysticisme — nous déterminerons qui l'emporte lorsque nous croiserons nos pagaies sur la Drôme. Je remercie aussi les autres membres de la famille : oncles, tantes, cousins, cousines et assimilés, que je ne citerai pas tous nommément afin d'épargner quelque peu la forêt amazonienne.

Enfin, j'adresse mes remerciements aux kayakistes du CKTSV et du CKDM qui m'ont permis d'oublier mes tracas scientifiques durant de longs week-ends. Vous ne trouverez nulle part ailleurs une telle alchimie entre convivialité, art culinaire et musculatures exemplaires. Je remercie en particulier quelques rivières, comme la Bonne, la Pallaresa ou l'Isère, et voue le Gyr aux gémonies pour les nombreux bleus qu'il a occasionnés sur mon tibia.

Cette recherche a été financée par le projet ACI Eros-3D, soutenu par le Centre National de la Recherche Scientifique (CNRS). Les modèles utilisés proviennent de l'Université de Caroline du Nord à Chapel Hill, ainsi que du *Digital Michelangelo Project* et du *3D Scanning Repository* de l'Université de Stanford. Par ailleurs, les maillages *EDF Tx* sont issus du projet « Colonne des danseuses », soutenu par *Electricité de France* au bénéfice de l'Ecole Française d'Athènes.

Table des matières

Table des figures

Liste des tableaux

Introduction

Le travail présenté dans ce mémoire a pris place au sein du projet *Eros 3D*, qui porte sur l'intégration multimodale et la recherche dans des bases de données de modèles 3D d'œuvres d'art. Il vise à exploiter la base de données du C2RMF — Centre de recherche et de restauration des musées de France, dont les sites sont situés au Musée du Louvre et à Versailles —, qui comprend un grand nombre de modèles 3D d'œuvres d'art. L'objectif est de développer une architecture logicielle capable de recaler, stocker, visualiser, retrouver et classer ces données à différents niveaux et pour différentes applications.

Au sein de ce projet, notre travail intervient au niveau des étapes de stockage et de visualisation. Les maillages en question sont très précis et détaillés, donc très volumineux : ils peuvent atteindre plusieurs centaines de millions de triangles. Ils sont ainsi très lourds à stocker, transférer et visualiser.

A la façon de *Google Earth*, nous souhaitons pouvoir observer une statue de Michel-Ange en 3D de façon globale, sans détails superflus — lorsque nous regardons la planète Terre dans son ensemble, la Tour Eiffel n'est guère visible —, puis autoriser l'utilisateur à zoomer à l'envi pour observer précisément le gros orteil du célèbre David. Le tout sans nécessiter plusieurs minutes voire plusieurs heures de lourds calculs.

Nous désirons satisfaire conjointement deux besoins. Le premier concerne le stockage et le transfert ; devant la taille imposante des fichiers originaux, il apparaît indispensable de les comprimer pour les archiver ou les échanger. Le second concerne la consultation des objets ; nous souhaitons les visualiser de façon interactive et fluide, directement à partir du fichier comprimé, sans phase de décompression préalable. Ces deux objectifs sont a priori antagonistes. En effet, le prétraitement des maillages à des fins de visualisation nécessite une réorganisation complète des données afin d'en optimiser l'accès ; réorganisation qui introduit généralement une redondance significative. Or, la redondance est l'ennemie de la compression qui cherche à la réduire *via* une analyse du signal et la mise en œuvre de mécanismes de prédiction complexes. L'évaluation de ces mécanismes est coûteuse et ralentit donc sensiblement l'accès à l'information. Dans ce mémoire, nous cherchons à concilier au mieux ces deux aspects.

Dans notre quête du meilleur compromis, nous ne sommes pas sans arme, puisque le format de compression présenté par Gandoin et Devillers dans [GD02] constitue une solide base pour atteindre nos objectifs. Cette méthode, si elle intègre nativement la notion de progressivité, n'autorise pas le raffinement local d'une zone du maillage — rappelez-vous le gros orteil. Par ailleurs, la phase de compression nécessite le chargement complet du maillage en mémoire — l'algorithme est dit *in-core* —, ce qui n'est pas envisageable dans le cas de maillages très volumineux ; notre méthode se doit donc d'être *out-of-core* — littéralement : « en dehors de la mémoire principale », le principe étant de charger le maillage en mémoire par morceaux —, pour autoriser le traitement d'objets 3D de toute taille.

Le défi qui se pose à nous est le suivant : comment rendre possible le traitement de maillages de toutes tailles ainsi que leur visualisation interactive, tout en conservant un format de fichier compact pour le stockage sur disque ou la transmission réseau ?

Ce mémoire débute par une description détaillée du cadre de l'étude, et se poursuit avec une étude des travaux existants connexes à notre sujet. Les principes de notre méthode sont ensuite exposés, suivis d'une description des particularités de notre implémentation et d'une analyse des résultats obtenus ; avant de conclure et d'évoquer les perspectives les plus prometteuses à nos yeux.

Remarque 1. *Tout au long de ce mémoire, nous nous sommes efforcés d'éviter l'emploi d'anglicismes lorsqu'un équivalent français existait. Cependant, nous avons conservé certains termes anglais, comme in-core, out-of-core, streaming ou vertex/index buffer, en raison de l'absence d'équivalents satisfaisants ou de leur emploi très fréquent en français. D'autre part, lors de la première utilisation d'un terme francisé, nous donnons généralement l'équivalent anglais si celui-ci est communément employé.*

Chapitre 1

Cadre de l'étude : modélisation, compression et visualisation 3D

« Je rêve d'être un dieu avec une centaine de bras, chacun avec une caméra. »
Nobuyoshi Araki.

Les ordinateurs ne manipulent que des chiffres. Plus précisément, deux chiffres : 0 et 1.

A la fois contrainte — au premier abord, les possibilités semblent réduites — et atout — la simplicité est souvent synonyme d'efficacité —, la longévité de ce modèle et les possibilités actuelles de l'informatique nous démontrent sa pertinence. L'histoire de l'informatique est jalonnée de nouveaux modèles de représentation. Elle fut tout d'abord destinée aux calculs mathématiques ; les opérateurs courants — addition, soustraction, multiplication, division, décalages de bits, etc. — furent rapidement développés. La première contrainte à surmonter fut l'absence de continuité : le système binaire ne peut représenter que des entiers (les nombres flottants ne sont rien d'autre que la combinaison de deux entiers : mantisse et exposant). *Exit* domaine réel et notion d'infini. L'informatique travaille exclusivement sur des domaines discrets.

Très vite, les caractères et le texte firent leur entrée, car il était aisé de coder chaque lettre par un nombre.

Puis progressivement, la nature des informations manipulée se diversifia. Les tableaux de nombres à une dimension permirent la représentation de données sonores, l'ajout d'une deuxième dimension ouvrit la porte aux images, puis la troisième dimension apporta vidéo et objets 3D. Les enthousiastes affirment qu'aucune limite n'existe, les sceptiques restent sceptiques. Entre les deux s'ouvre un chemin selon lequel les limites se doivent d'être défiées pour tester leur immuabilité.

Dans ce mémoire, nous nous intéressons à la modélisation tridimensionnelle, qui représente les objets 3D. Alors que les images numériques sont généralement encodées en tant que grilles régulières à deux dimensions, les éléments composant un modèle 3D sont plus souvent répartis de façon irrégulière dans l'espace tridimensionnel que sur une grille régulière 3D. En effet, malgré les progrès rapides des capacités de calcul et de mémoire des stations de travail, l'étendue de l'espace 3D est tel que le stockage sur une grille est très coûteux : si une image 2D représentée par une grille de 1000×1000 points peut être stockée sur environ 4 Mo, l'équivalent 3D sur une grille de $1000 \times 1000 \times 1000$ points occuperait 4 Go ! Par conséquent, de nombreux modèles ont été développés, avec une prédominance des maillages surfaciques (seule la surface est représentée) triangulaires, pris en charge de façon matérielle par les cartes graphiques.

1.1 Structures géométriques

Afin de modéliser les objets 3D, une description *géométrique* est le plus souvent employée. Cette approche se base sur l'utilisation de primitives géométriques telles que les points, les polygones (*e.g.* triangles, quadrilatères, hexagones), les sphères, les tétraèdres, qui sont positionnés et assemblés dans l'espace tridimensionnel euclidien afin de constituer une surface (*e.g.* assemblage de triangles adjacents) ou un volume. Un exemple de maillage surfacique triangulaire est donné sur la figure 1.1. Ce type de modélisation est au centre de notre travail.

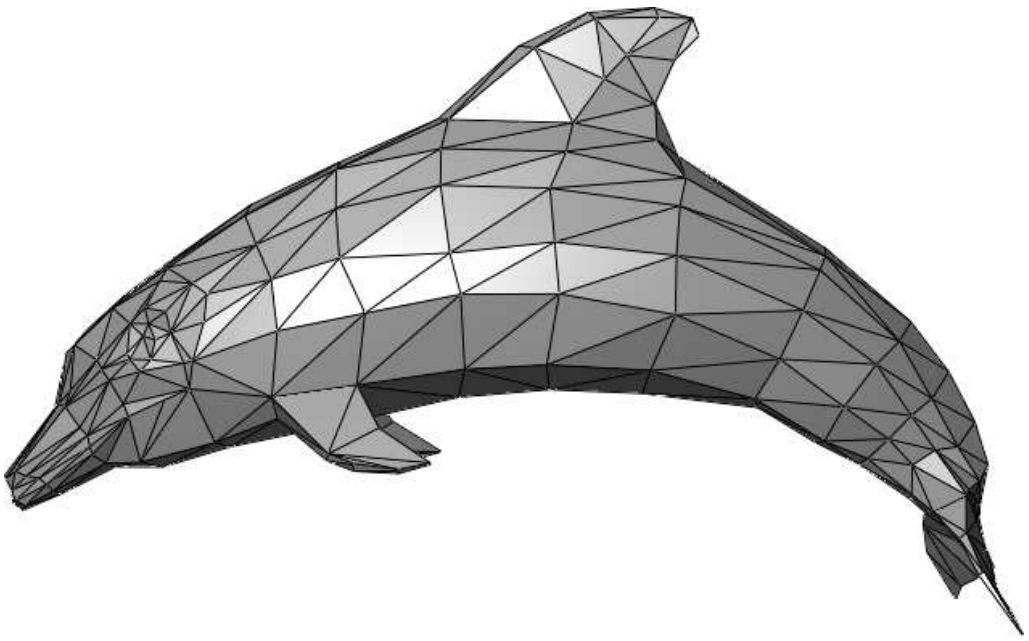


FIGURE 1.1 – Exemple de maillage surfacique triangulaire

1.1.1 Géométrie et connectivité

Ce que nous appelons dans ce mémoire *structure géométrique* est tout d'abord composé d'un noyau dénommé *géométrie*, qui est un ensemble G de n points en dimension d.

Une *structure géométrique* peut se limiter à G uniquement. Cependant, un deuxième ensemble d'information y est souvent associé. Il s'agit d'un graphe C décrivant les relations d'incidence entre les points de G : les arêtes, les triangles, etc. Cet ensemble C est nommé *connectivité*. Dans certains cas, il pourra être complété par les relations d'adjacence explicites entre ses différents éléments.

Un troisième type de données peut être contenue par une structure géométrique, que l'on appelle généralement *attributs*. Ce sont des informations additionnelles comme la couleur, les normales, les matériaux, les textures, les coordonnées de texture, etc.

1.1.2 Principales structures géométriques

Les structures géométriques les plus fréquemment rencontrées sont les suivantes :

Le nuage de points

Le *nuage de points* est une structure géométrique qui ne possède pas de graphe C de connectivité. Elle est uniquement composée d'un ensemble G de points, auxquels peuvent être associés des attributs comme des couleurs ou des normales. Ce type de données est notamment généré par les scanners laser, qui relèvent des points sur la surface de l'objet. Certaines applications utilisent directement ces ensembles de points bruts pour l'affichage, les mesures ou les traitements. L'autre solution consiste à utiliser un algorithme de reconstruction afin de générer plus ou moins automatiquement un graphe de connectivité à partir de ces points.

Le maillage surfacique

Très souvent, la description d'un objet volumique peut se limiter à la description de la surface de celui-ci, pour peu que l'objet soit opaque et que les applications employées ne nécessitent pas d'informations concernant la matière interne. Ainsi, cette surface est décrite par un ensemble de variétés bidimensionnelles plongées dans l'espace euclidien à 3 dimensions. De par le type de données manipulées par un ordinateur, une discrétisation de la surface est

préalablement nécessaire. La surface est découpée en mailles élémentaires qui peuvent être de différentes natures :

- Triangles : il s'agit du cas le plus fréquent, on parle alors de *maillages triangulaires*. C'est généralement le seul cas géré en natif par les cartes graphiques et les *API 3D* telles *OpenGL* ou *DirectX*, en raison de sa simplicité et de sa généralité : coplanarité, interpolation linéaire simple et directe, assemblage aisé, etc.
- Quadrilatères : ils cohabitent souvent avec des triangles.
- Hexagones : rares.
- Autres : tout type de polygones.

Le maillage volumique

Dans certains cas, la description de la surface seule n'est pas suffisante : matériau translucide, composition hétérogènes, besoin de creuser/sculpter dans l'objet, etc. On utilise alors la généralisation en trois dimensions des maillages surfaciques. Les mailles surfaciques se font mailles volumiques, que l'on assemble pour former le volume total de l'objet. A l'instar des maillages surfaciques, les mailles peuvent être de différents types :

- Tétraèdres : il s'agit de la généralisation tridimensionnelle des triangles, avec les mêmes avantages. Il s'agit du cas le plus fréquent pour la représentation de volumes sur grilles irrégulières et pour les calculs scientifiques sur les éléments finis.
- Hexaèdres : en particulier, les hexaèdres réguliers (*i.e.* les cubes) sont utilisés pour les représentations sur grilles régulières. Ils sont alors le pendant tridimensionnel des *pixels* utilisés pour les images $2D$, et sont nommés *voxels*.
- Autres : tout type de polyèdres.

Nos travaux traitent les nuages de points, les maillages surfaciques triangulaires et les maillages volumiques tétraédriques, même si le support de ces derniers reste théorique puisqu'ils n'ont pas fait l'objet d'une implémentation. La majeure partie de nos efforts s'est portée sur les maillages triangulaires, largement majoritaires. C'est pourquoi nous emploierons fréquemment le terme de « triangle » au cours de ce mémoire. Certaines parties, notamment celles traitant d'implémentation et d'optimisation, nécessiteraient quelques adaptations pour être valides en toute dimension.

1.1.3 Maillages surfaciques

Cette section expose quelques définitions essentielles qui concernent le cas particulier des surfaces plongées dans l'espace euclidien à 3 dimensions. Pour plus d'informations, nous invitons le lecteur à consulter l'ouvrage *Differential Geometry of Curves and Surfaces* de M. P. Do Carmo [Car76].

Définition 1. *Une structure géométrique constituée d'un graphe C de sommets, d'arêtes et de faces, et d'un ensemble G de coordonnées de sommets décrivant le plongement de C dans l'espace est appelée un maillage.*

Définition 2. *Soit s un sommet d'un maillage M. Le degré de s est le nombre de voisins de s dans M, ou encore le nombre d'arêtes incidentes à s dans M.*

Définition 3. *Etant donnée une surface triangulée S constituée de s sommets, a arêtes et f faces, on appelle caractéristique d'Euler-Poincaré de S le nombre* $\chi(S) = s - a + f$.

Définition 4. *On appelle genre d'une surface S le nombre* $g = \frac{2-\chi(S)}{2}$. *Ce nombre est indépendant de la manière dont S est triangulée, et correspond au nombre d'anses de S (*0 *pour une surface homéomorphe à une sphère,* 1 *pour une surface homéomorphe à un tore, etc.).*

Remarque 2. *Cette dernière définition est souvent formulée comme suit : toute triangulation d'une surface S de genre g vérifie la relation d'Euler :*

$$s - a + f = 2 - 2g$$

Pour la démonstration de cette relation, le lecteur pourra aussi se reporter aux travaux de Boissonnat et Yvinec [BY95, BY98].

Définition 5. *Un maillage M est dit* manifold *(ou* variété*) si toute arête de G est incidente à deux faces exactement, et les faces incidentes à tout sommet forment un cône simple. Autrement dit, tout sommet de M a un voisinage homéomorphe à un disque.*

Définition 6. *Un maillage est dit* manifold avec bords *s'il vérifie les propriétés d'un maillage* manifold *partout sauf pour un ou plusieurs cycles disjoints d'arêtes incidentes à une seule face chacune. De telles arêtes sont appelées arêtes de bord. Les sommets de bord (i.e. les extrémités des arêtes de bord) ont un voisinage homéomorphe à un demi-disque.*

Définition 7. *Un maillage est dit* simple *s'il est* manifold *et de genre nul, c'est-à-dire topologiquement équivalent à une sphère.*

1.1.4 Complexe simplicial

Notre méthode manipule des complexes simpliciaux, qui sont des espaces topologiques construits *via* la connexion de points, segments, triangles, et leurs homologues *n*-dimensionnels.

Définition 8. *Un* ***simplexe*** *ou n-****simplexe*** *est l'analogue à n dimensions du triangle. Plus précisément, un simplexe est l'enveloppe convexe d'un ensemble de* $n + 1$ *points affinement indépendants dans un espace euclidien de dimension n, ce qui signifie que :*

- *un 0-simplexe est un* ***point****.*
- *un 1-simplexe est un* ***segment****.*
- *un 2-simplexe est un* ***triangle****.*

- *un 3-simplexe est un* ***tétraèdre****.*

Définition 9. *Un* ***complexe simplicial*** $\mathcal{K}$ *est un ensemble de simplexes tel que :*

1. *Toute face d'un simplexe de* $\mathcal{K}$ *est aussi dans* $\mathcal{K}$
2. *L'intersection de deux simplexes* $\sigma_a, \sigma_b \in \mathcal{K}$ *est à la fois une face de* σ_a *et de* σ_b*.*

Un k-complexe simplicial est un complexe simplicial où la dimension du simplexe de plus grande dimension est k. Par exemple, un 2-complexe simplicial doit contenir au moins un triangle, et ne contient pas de tétraèdre ou de simplexe de dimension supérieure.

Définition 10. *Le* ***bord*** *d'un d-simplexe de sommets* $S = \{s_1, ...s_d\}$ *est l'ensemble des* $d+1$ *simplexes de dimension* $d-1$*, de sommets* $S_i = S - \{s_i\}$*, avec* $i = 0, \ldots d$*.*

Définition 11. *La* ***frontière*** *d'un simplexe est l'union de son bord, des bords de ses simplexes bords, et ainsi de suite, par fermeture transitive.*

Définition 12. *Un* ***simplexe maximal*** *est un simplexe qui n'appartient à la frontière d'aucun autre simplexe du complexe.*

1.2 Codage et formats de fichier

1.2.1 Codage naïf

La façon la plus simple d'encoder un maillage consiste à encoder tout d'abord la liste des points en spécifiant leurs 3 coordonnées cartésiennes dans l'espace euclidien. On ajoute ensuite la liste des polygones composant la structure géométrique. Un polygone commence généralement par un code identifiant sa nature (*e.g.* « 3 » pour un triangle) puis par n indices qui se réfèrent à des sommets de la première liste (*e.g.* « 4, 5, 8 » pour un triangle composé du 4^e, 5^e et 8^e sommets de la liste). Ce type d'encodage est simple et intuitif, facile à lire, mais est très redondant, puisque chaque point de G est référencé autant de fois qu'il possède de polygones incidents.

Ainsi, un fichier non comprimé comporte au minimum deux parties :

- La *géométrie* : liste L_g de points, dont les coordonnées sont généralement codées sur 32 bits chacune.
- La *connectivité* : liste L_c de primitives de dimension d, chacune étant composé de $d+1$ indices entiers (32 bits) qui se réfèrent chacun à un point de la liste L_g. Par exemple, la séquence 5 15 22 décrit un triangle composé du 5^e, 15^e et 22^e points de la liste L_g.

A cela peuvent s'ajouter des informations complémentaires comme la liste des normales aux sommets, ou encore des coordonnées de texture.

Format texte

Certains formats de fichier, comme OFF (Object File Format), OBJ (développé par *Wavefront Technologies*) ou VRML (Virtual Reality Modeling Language), sont au format *texte*. Les fichiers sont lisibles à l'aide d'un simple éditeur de texte. Voici un exemple de fichier VRML représentant un cube. On distingue aisément l'ensemble des points (géométrie), puis la liste des triangles (connectivité) :

```
#VRML V2.0 utf8
Group {
  children [
    Shape {
      appearance Appearance {
            material  Material {
                  diffuseColor 0.7 0.7 0.7
                  specularColor 0.1 0.1 0.1
                }
          }
      geometry IndexedFaceSet {
        coord Coordinate {
          point [
-3.3 3.3 -3.3
3.3 3.3 -3.3
-3.3 -3.3 -3.3
3.3 -3.3 -3.3
-3.3 3.3 3.3
3.3 3.3 3.3
-3.3 -3.3 3.3
3.3 -3.3 3.3
          ]
        }
        creaseAngle 1.0
        coordIndex [
0 1 2 -1
2 1 3 -1
3 1 5 -1
5 7 3 -1
5 6 7 -1
6 5 4 -1
2 6 4 -1
2 4 0 -1
1 0 4 -1
1 4 5 -1
3 6 2 -1
3 7 6 -1
        ]
      }
    }
  ]
}
```

Format binaire

Les fichiers textes, s'ils présentent l'avantage d'être facile à lire et à comprendre, pâtissent d'un inconvénient majeur : ils sont volumineux. C'est pourquoi furent développés des formats binaires. Parfois, les formats de fichier laissent le choix entre texte et binaire à l'utilisateur. C'est le cas du format PLY (Polygon File Format, développé par *Stanford*) ou du format StL (Stereo-Lithography, développé par *3D Systems*). D'autres formats, comme le 3DS (3D Studio), sont purement binaires.

Le format PLY nous intéresse particulièrement, puisque les scans de statues issus du *Digital Michelangelo Project* emploient ce format [1]. Pour le chargement de ces fichiers, nous utilisons la bibliothèque RPLY [2].

Remarque 3. *Dans un triplet d'indices décrivant un triangle, l'ordre a une importance : les triangles sont dits « orientés ». Ils possèdent donc une face avant et une face arrière. Notre méthode encode l'orientation des triangles afin de garantir un rendu correct des maillages.*

1.2.2 Codage entropique

Un signal peut être décomposé en deux parties : l'information et la redondance. Le codage entropique, aussi appelé codage de source, vise à supprimer l'information redondante. La théorie de l'information définit le concept d'*entropie* d'un message [Sha48], *i.e.* la quantité d'information contenue dans celui-ci. C'est la limite minimale théorique du nombre de bits nécessaires au codage de l'information portée par un signal. Plus l'entropie d'un message est grande, plus l'information est dense, donc plus il est difficile à comprimer. Le but de toute méthode de compression est de supprimer au maximum la redondance, donc d'obtenir un message le plus entropique possible.

Un symbole est un élément d'un ensemble fini appelé alphabet. Très souvent, un symbole est un caractère ou un mot. Selon le théorème du codage de source de Shannon, si l'on considère une suite de symboles issus d'une source, la taille optimale T d'un symbole s vaut

$$T(s) = -\log_b p(s)$$

où b est le nombre de symboles utilisés pour former les codes en sortie (2 dans le cas de l'informatique) et $p(s)$ est la probabilité d'apparition du symbole s en entrée.

Par conséquent, l'entropie H d'un signal composé de n symboles s_i ($i \in$

1. http://graphics.stanford.edu/data/3Dscanrep/
2. http://www.tecgraf.puc-rio.br/~diego/professional/rply/

$\{1, ..., n\}$) encodé en binaire vaut donc

$$H = -\sum_{i=1}^{n} p(s_i)\, T(s_i) = -\sum_{i=1}^{n} p(s_i)\, \log_2 p(s_i)$$

Codage par plages

Le codage par plages, aussi appelé RLE pour *Run-Length Encoding*, consiste à remplacer une donnée qui est répétée plusieurs fois consécutivement par deux indications : le nombre de répétition, et la donnée elle-même. Cette méthode est seulement efficace dans le cas où les données sont très peu aléatoires. Les documents noir et blanc (sans nuances de gris) se prête particulièrement bien à ce type d'encodage : une ligne de 200 pixels noirs sera encodée sur quelques bits au lieu de 200 bits.

Codage par dictionnaire

Introduit par Ziv et Lempel en 1977 [ZL77], le codage par dictionnaire constitue en ligne un dictionnaire adaptatif où figurent, selon les algorithmes, les symboles récemment encodés (« fenêtre glissante »), ou les symboles qui sont répétés dans le message à comprimer. Le symbole courant est recherché dans ce dictionnaire, et, s'il s'y trouve, est encodé par sa position et sa longueur dans ce dernier. Les deux algorithmes de base sont nommés LZ77 [ZL77] et LZ78 [ZL78], et ont donnés naissance à LZSS [SS82] et LZW [Wel84].

Ces algorithmes ont initialement été développés pour l'encodage du texte, et perdent de leur efficacité lorsqu'il s'agit de comprimer d'autres types de données.

Codage de Huffman

Le codage de Huffman est un codage statistique à longueur variable. Introduit en 1952 par Huffman [Huf52], il repose sur une analyse statistique préalable des données en entrée. Chaque symbole se voit attribuer un code dont la longueur est inversement proportionnelle à sa probabilité d'apparition. Ainsi, les codes fréquents sont plus courts que les codes rares. Il est à noter que pour permettre un décodage non ambigu, les codes doivent être séparables, ce qui implique qu'un code donné ne doit pas être le préfixe d'un autre code.

Lors de la phase d'analyse, un arbre binaire est créé. On commence par calculer le nombre d'occurrences de chaque symbole, puis on crée une feuille pour chaque symbole, auquel on associe un poids valant son nombre d'occurrences. Ensuite, l'arbre est créé suivant un principe simple : on associe à chaque fois les

deux nœuds de plus faibles poids pour donner un nœud dont le poids équivaut à la somme des poids de ses fils jusqu'à n'en avoir plus qu'un, la racine. Pour obtenir le code binaire de chaque symbole, on descend l'arbre depuis la racine vers les feuilles en rajoutant à chaque fois à la fin du code un 0 ou un 1 selon la branche suivie (0 pour la gauche, 1 pour la droite).

Par exemple, pour encoder la séquence "ABCA", les codes A B C ont respectivement les probabilités d'apparition 0,5 0,25 0,25. En système binaire, on pourra donc encoder A avec le code 0, B avec le code 10, et C avec le code 11. Un exemple plus complexe d'arbre de Huffman est présenté sur la figure 1.2.

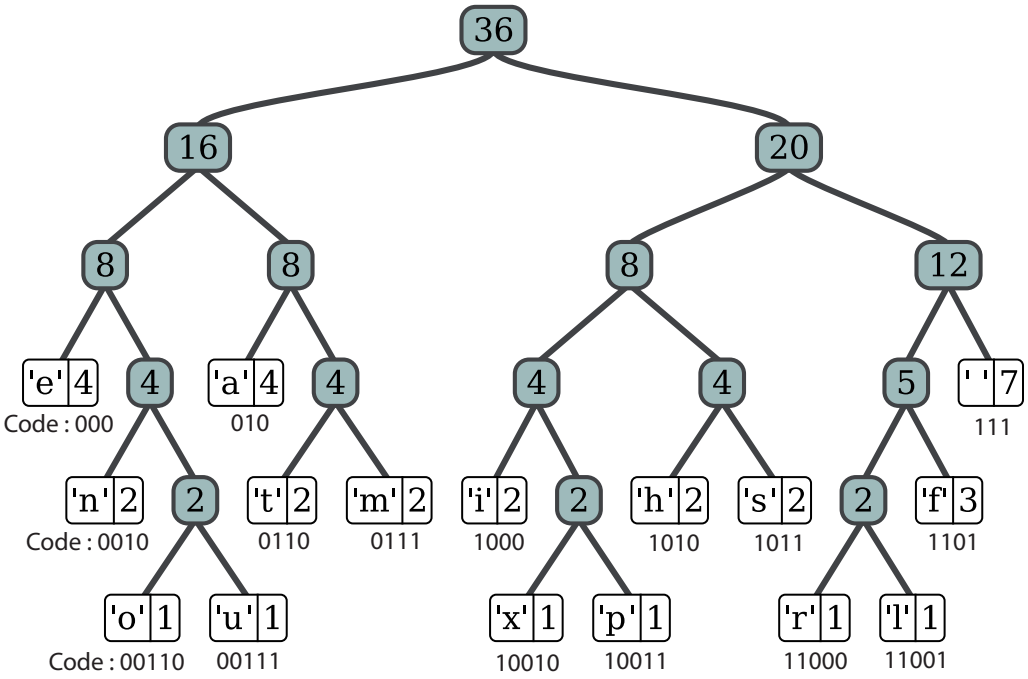


FIGURE 1.2 – Arbre de Huffman construit à partir de la phrase « this is an example of a huffman tree »- Source : Wikipédia

Le codage de Huffman possède un avantage indéniable : le décodage est rapide, ce qui explique son succès. Il est employé comme second étage de compression dans bon nombre de formats de compression actuels comme le JPEG pour les images, le MP3 pour le son ou le MPEG pour les vidéos. Sa principale limitation est due au fait que la longueur des symboles écrits est un nombre entier de bits. Si $T(s)$ est un nombre réel, le nombre de bits employé sera l'arrondi à l'entier supérieur. La taille du message encodé est donc souvent supérieure à la taille optimale énoncée par le théorème de Shannon.

Codage arithmétique

Développé à partir de la fin des années 70 [Ris76, RJ79, Ris83, WNC87], utilisé dans la norme JPEG 2000, le codage arithmétique permet de résoudre

ce problème d'arrondi à l'entier supérieur, et permet de tendre vers l'entropie H du signal. Il permet un gain moyen situé entre 5 et 10% par rapport au codage de Huffman. Fondamentalement, la séquence de symboles complète est encodée à l'aide d'un seul nombre réel de l'intervalle $[0;1[$. Après une phase d'analyse statistique identique à celle de Huffman, chaque symbole est pris en compte séquentiellement et conduit au raffinement progressif de l'intervalle initial $[0;1[$ en fonction de la probabilité du symbole. A la fin du processus, un intervalle de largeur très faible a été construit, et n'importe quel nombre contenu dans cet intervalle code le message complet.

La figure 1.3 illustre le fonctionnement du codage arithmétique statique sur un exemple. On souhaite encoder le message contenant trois symboles distincts « AACBAACACB ». L'algorithme subdivise initialement l'intervalle $[0;1[$ et chaque symbole se voit attribuer un intervalle dont la largeur est proportionnelle à sa probabilité :

- A $\rightarrow [0;0,5[$
- B $\rightarrow [0,5;0,7[$
- C $\rightarrow [0,7;1[$

Nous l'appellerons *partitionnement de référence*. Ici, le premier caractère est A donc l'intervalle courant devient l'intervalle correspondant à A dans le partitionnement de référence : $[0;0,5[$. Puis à chaque étape, on « multiplie » l'intervalle courant par le partitionnement de référence pour obtenir le nouveau partitionnement. Plus précisément, si l'on note $[a;b[$ l'intervalle courant et $[d_i;f_i[$ le i^e intervalle de référence, le i^e intervalle du nouveau partionnement est $[a + d_i(b-a); a + f_i(b-a)[$. Dans notre exemple, on a désormais le partitionnement suivant :

- A $\rightarrow [0;0,25[$
- B $\rightarrow [0,25;0,35[$
- C $\rightarrow [0,35;0,5[$

Le symbole suivant (A) réduit l'intervalle à courant à $[0;0,25[$. On continue ainsi tant qu'il y a des symboles à encoder. L'intervalle final est :

$$[0,21560313\,;\ 0,21563688[$$

La connaissance d'un nombre situé dans cet intervalle et du partitionnement de référence permet de reconstituer le message original. Pour cela, on reprend le même cheminement que lors de l'encodage. Par exemple, si le nombre encodé est $0,21561$, on parcourt les partitionnements successifs de la figure 1.3 et on regarde dans quel intervalle se situe le nombre en question. $0,21561$ est successivement situé dans $[0;0,5[$ (premier caractère : A), puis dans $[0;0,25[$ (second caractère : A), puis dans $[0,175;0,25[$ (troisième caractère : C), et ainsi de suite. On constate aisément que la séquence est reconstruite à l'identique.

A l'instar du codage de Huffman, l'utilisation efficace d'un codeur arithmétique requiert une bonne analyse statistique. Celle-ci peut être effectué de façon statique — *i.e.* avant le début de la compression — ou dynamique — la

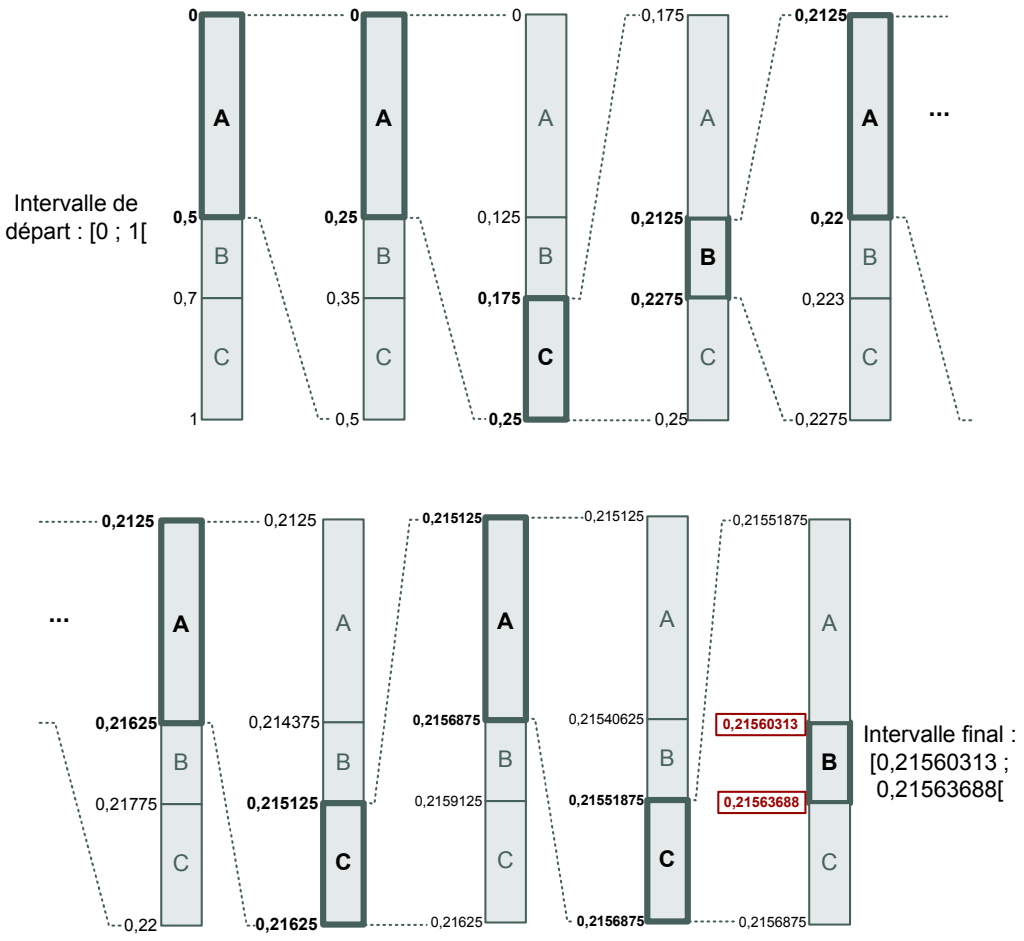


FIGURE 1.3 – Exemple de codage arithmétique de la séquence « AACBAACACB »

table des probabilités est tout d'abord équiprobable, puis est raffinée à chaque symbole en fonction des symboles déjà encodés. La première méthode présente l'avantage d'être immédiatement optimale en termes de coût de stockage, dès le premier symbole. En revanche, elle implique d'encoder préalablement le partitionnement de référence, contrairement au codage dynamique.

Prédiction

La définition de l'entropie donnée précédemment — cette quantité que les méthodes citées ci-dessus cherchent à approcher — considère des probabilités constantes et indépendantes pour chaque symbole. Il n'est donc pas tenu compte de la corrélation entre les symboles : la probabilité du i^e symbole ne tient pas compte de la valeur des symboles précédents. On parle alors d'entropie d'ordre 0.

Dans la pratique, il existe souvent des corrélations entre les symboles et leur voisinage. Corrélations dont il est impératif de tenir compte si l'on souhaite obtenir une compression efficace. C'est ici qu'intervient la notion de *prédiction*, utilisée dans tous les schémas de compression récents. Au moment d'encoder un symbole, l'algorithme analyse son voisinage causal — *i.e.* les symboles qui précèdent, puisque ce sont les seuls qui seront connus lors du décodage —, et module les probabilités initiales. Le nombre de symboles considérés correspond à l'ordre du modeleur statistique. Par exemple, un codeur d'ordre 2 analysera les 2 symboles qui précédent le symbole courant afin de calculer sa probabilité. PPM (*Prediction by Partial Matching*) [CW84] et DMC (*Dynamic Markov Compression*) [CH87, NYC05] sont des exemples d'algorithmes de compression combinant le codage arithmétique et la prédiction à partir des symboles précédents. Dans le cas de la compression de maillage, la prédiction est plus efficace si elle est basée sur des informations spaciales et géométriques, plutôt que sur les symboles encodés précédemment, en raison du caractère multi-dimensionnel des données représentées.

Expression des taux de compression

Il existe plusieurs façons d'exprimer le taux de compression d'un fichier encodé. Nous avons choisi d'utiliser le ratio entre la taille du fichier initial et la taille du fichier comprimé :

Définition 13. *Soient T_i la taille du fichier initial non comprimé, et T_c la taille du fichier comprimé. La taux de compression τ vaut alors :*

$$\tau = \frac{T_i}{T_c}$$

1.3 Discrétisation et quantification

Généralement, les formats de fichier non comprimés stockent les coordonnées géométriques des sommets à l'aide de nombres flottants. Ces derniers offrent une précision de 32 bits — dont 24 bits pour la mantisse avec son signe, selon le standard IEEE [isb85]. Ce type de données n'est absolument pas optimal dans le cadre d'un stockage compact et précis, pour plusieurs raisons :

- Précision limitée : problèmes liés aux arrondis successifs, phénomène de « cancellation », risques d'« overflows » et d'« underflows » sur l'exposant.
- Problèmes de test d'égalité avec 0 ou avec un autre flottant.
- Inconstance de la distance entre deux nombres flottants consécutifs.
- Seule une faible portion de l'intervalle de définition est généralement utilisée pour coder des positions dans l'espace.

C'est pourquoi il n'existe à notre connaissance aucune méthode de compression qui emploie ce format. Les nombres entiers ont la faveur des encodeurs, car ils ne présentent pas les inconvénients cités ci-dessus. La première étape consiste très souvent à calculer la boîte englobante de tous les sommets et à la stocker sous forme de nombres flottants afin de pouvoir ensuite aisément recalculer les coordonnées originales. Une précision k est ensuite choisie, exprimée en nombre de bits. On peut ainsi encoder les coordonnées des points selon chaque axe de la boîte englobante à l'aide d'un entier. Si l'on imagine une grille à trois dimensions remplissant cette boîte et comportant 2^k subdivisions sur chaque axe, chaque triplet d'entiers (chacun encodé sur k bits) indique les coordonnées d'un point dans la grille.

Pour peu que l'on choisisse une valeur de k suffisamment grande, la conversion peut se faire *sans perte d'information*. La boîte englobante permet d'utiliser pleinement l'intervalle de définition — toutes les valeurs possibles sont susceptibles d'être utilisées. Ainsi, $k = 24$ est souvent grandement suffisant pour obtenir un encodage sans perte. Selon la méthode d'obtention du maillage et le nombre de points le composant, il est parfois possible de réduire davantage la valeur de k. Par exemple, un objet 3D issu d'un scanner laser dont la précision est limitée peut autoriser une valeur de k inférieure à 16 bits. Dans la littérature, certains articles de compression de petits maillages se contentent d'une précision de 8 à 12 bits, ce qui est discutable étant donné la perte non négligeable engendrée. Les travaux plus récents, et plus particulièrement ceux traitant de maillages volumineux, utilisent généralement une précision de 16 bits. C'est l'option que nous avons le plus souvent retenue.

Il est à noter que certaines applications, notamment dans le domaine médical ou pour les calculs scientifiques, nécessitent une très haute précision sur les données. Par conséquent, il est important que la précision soit paramétrable par l'utilisateur. C'est le cas d'une grande majorité des algorithmes de compression, ainsi que des travaux exposés dans ce mémoire.

1.4 Création, acquisition et modélisation des maillages

Il existe de nombreuses façons d'obtenir des maillages 3D. Il est bien sûr possible d'écrire directement la liste des points et des polygones dans un fichier texte — en employant le format VRML par exemple —, mais iriez-vous demander à un dessinateur de pratiquer son art en indiquant la liste des coordonnées des pixels à noircir ? Il existe pour cela des logiciels appelés modeleurs 3D, qui sont l'équivalent 3D de Adobe Photoshop ou GIMP.

De même que les appareils photographiques numériques permettent d'automatiser partiellement la prise d'une image, ou que les caméscopes autorisent l'acquisition d'une scène, il existe des appareils capables de produire plus ou moins automatiquement des maillages en numérisant un objet réel.

1.4.1 Création « from scratch » — modeleurs 3D

Blender, Autodesk 3D Studio Max ou encore Pixologic ZBrush sont des logiciels de modélisation 3D permettant à un utilisateur de créer de toutes pièces un objet en trois dimensions par le biais d'une grande variété d'outils tels que :

- Extrusion
- Coupe
- Soudure
- Révolution
- Subdivision de surface
- Simplification de surface
- NURBS, Splines, Patchs
- Courbes de type NURBS

FIGURE 1.4 – Exemples de maillages obtenus à l'aide de modeleurs 3D

Souvent, les artistes commencent avec une forme basique, *e.g.* un cube, qu'ils sculptent, subdivisent, extrudent, tournent, bref : torturent, afin d'arriver à un résultat convaincant (figure 1.4, à gauche) voire stupéfiant (figure 1.4, à droite).

1.4.2 Acquisition

Si vous ne vous sentez pas l'âme d'un artiste option *3D designer*, vous pouvez alors vous tourner vers l'acquisition de maillages à partir d'objets réels. La figure 1.5 présente une chaîne d'acquisition usuelle. On relève trois étapes principales :

1. La numérisation, réalisée à l'aide d'un scanner 3D.
2. Le recalage, qui transforme les données fournies par le scanner en nuage de points.
3. La reconstruction, qui crée la connectivité entre les sommets.

Numérisation

Cette étape implique généralement l'utilisation de scanners 3D — à noter que d'autres méthodes existent, par exemple à l'aide de simples camescopes. Les progrès réalisés par ces scanners laser ont permis la création de maillages très précis et détaillés. Par exemple, le *Digital Michelangelo Project* propose plusieurs scans de statues [3], dont la précision peut atteindre 0,25 mm. Le dispositif utilisé intégrant un scanner est présenté à la figure 1.6. La précision du scan du David de Michel-Ange est de 1 mm, précision que l'on peut apprécier sur la figure 1.7.

Le principe du scanner 3D consiste à mesurer la distance à l'objet dans de multiples directions, afin d'obtenir un grand nombre de points situés sur sa surface. L'opération est souvent répétée sous plusieurs angles. Selon les modèles de scanner, il est parfois possible de capturer d'autres informations comme la couleur. On obtient donc plusieurs images de profondeur se chevauchant partiellement.

Recalage

La phase de recalage utilise des algorithmes dont le rôle est de fusionner les différentes images de profondeur de façon cohérente afin d'obtenir un seul nuage de points dans l'espace 3D.

Reconstruction de surface

Enfin, très souvent, cet ensemble de sommets est traité par un algorithme de reconstruction afin de construire automatiquement une surface constituée de

3. http://www-graphics.stanford.edu/dmich-archive/

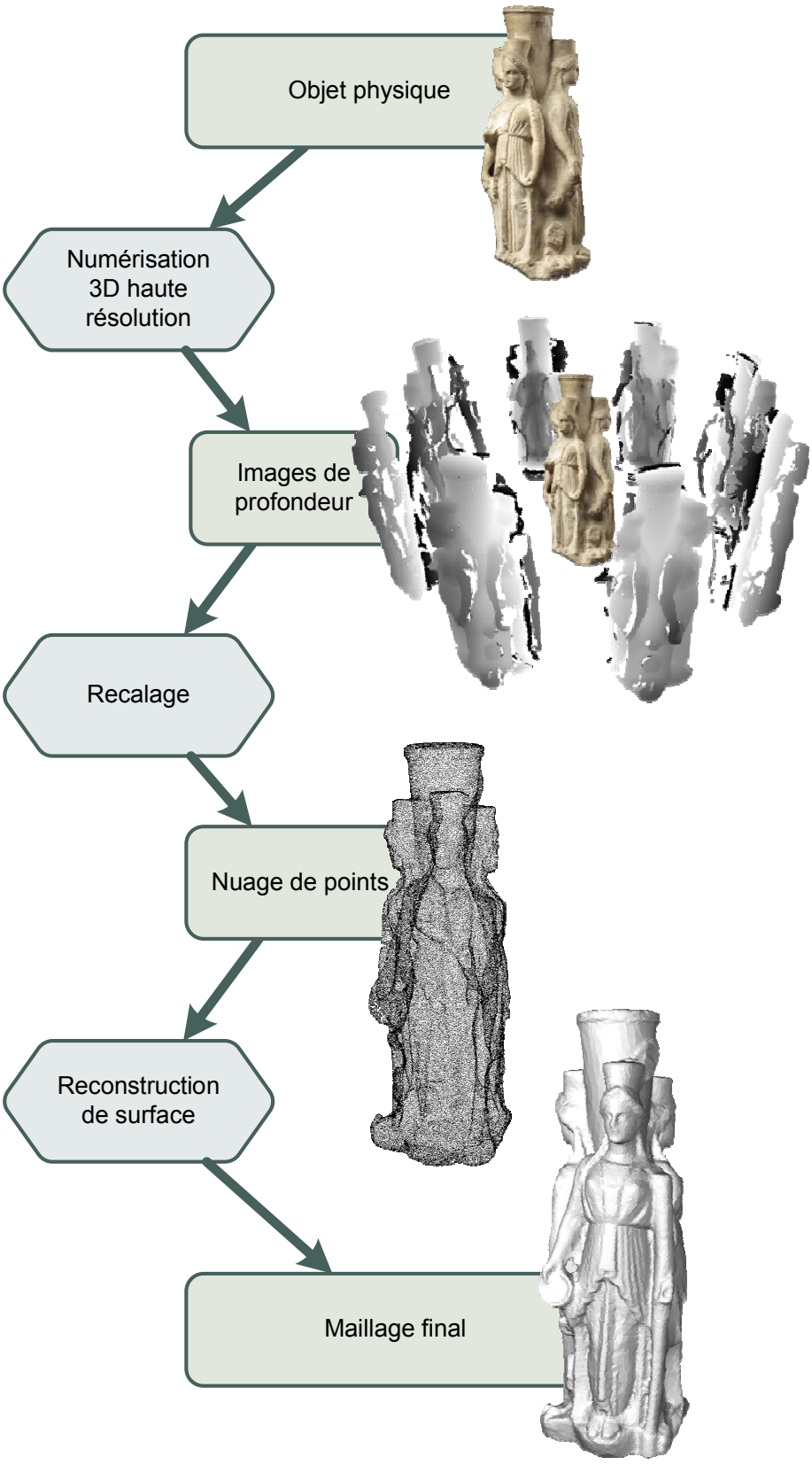


FIGURE 1.5 – Chaîne d'acquisition d'un maillage 3D à partir d'un objet physique

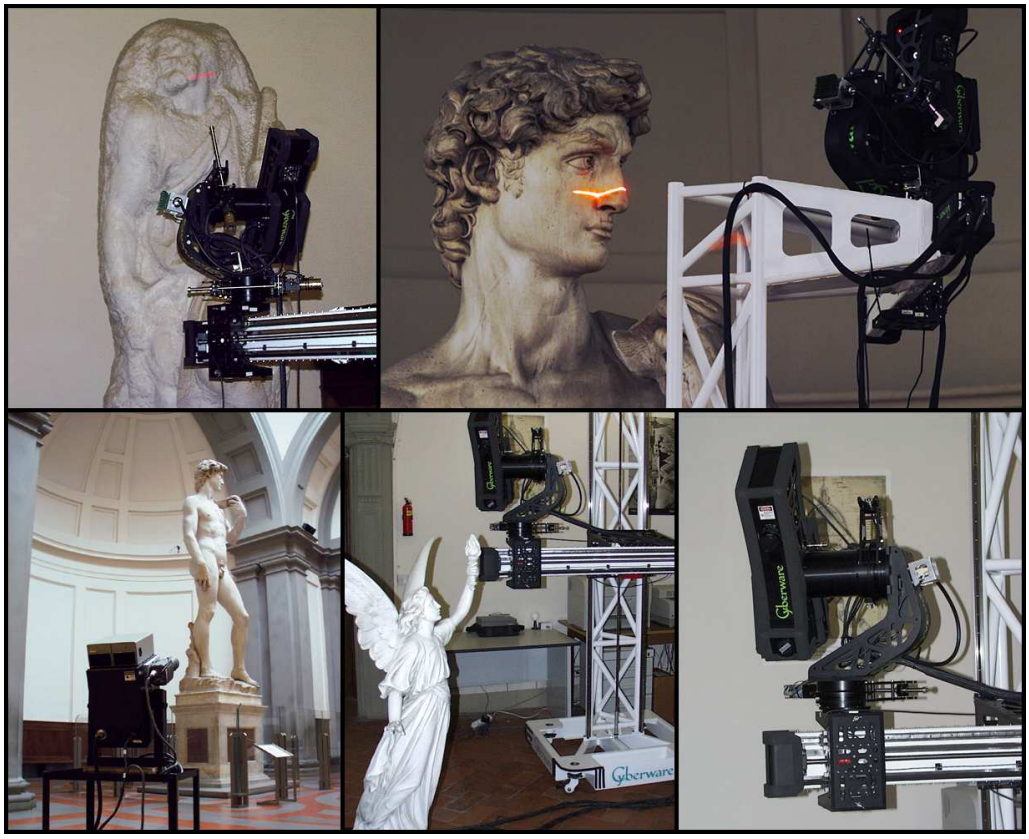


FIGURE 1.6 – Dispositif utilisé par le *Digital Michelangelo Project*

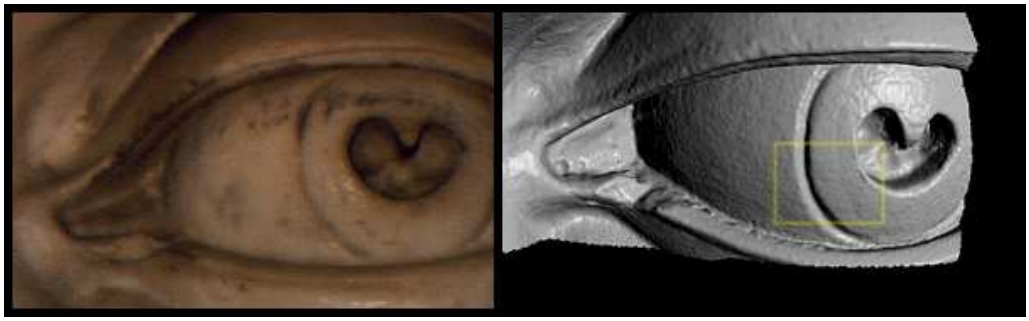


FIGURE 1.7 – Comparaison entre une photographie de la statue du David de Michel-Ange et le rendu obtenu à partir du maillage 3D — source : *Digital Michelangelo Project*

polygones. Ce qui permet par exemple d'obtenir un rendu soigné du maillage (figure 1.8).

FIGURE 1.8 – Exemple de rendu du maillage obtenu par scanner 3D du David de Michel-Ange — source : *Digital Michelangelo Project*

Le scanner 3D montre ses limites lorsque certaines zones de l'objet numérisé sont difficilement accessibles par le rayon du laser. Généralement, ce problème se traduit par des trous polygonaux plus ou moins visibles selon leur localisation et leur taille. La figure 1.9 montre un exemple de trous visibles sur le maillage du David. On peut alors faire appel à des algorithmes de réparation afin de les boucher.

FIGURE 1.9 – Exemple de trous polygonaux sur le maillage du David de Michel-Ange

1.5 Mesures d'erreur

A l'instar des sons, photographies ou vidéos numériques, l'enjeu de la visualisation 3D est de rendre invisible à l'utilisateur la discrétisation du signal. Les organes de perception humains sont imparfaits et relativement faciles à duper. Le CD audio utilise 44 100 échantillons par secondes, chacun encodé sur une précision de 16 bits, pour masquer le découpage introduit. Dans le domaine de la vidéo, la discrétisation sur l'axe du temps est présente depuis les premiers pas du cinéma muet, et s'appuie sur la persistance rétinienne afin de se contenter de 24 images fixes par secondes ; le numérique a introduit un échantillonnage sur les deux dimensions spatiales.

Les maillages 3D n'échappent pas à la règle. Les coordonnées des points se doivent d'être suffisamment précises pour se garder des « effets d'escalier », et les triangles suffisamment petits et nombreux pour créer l'illusion d'une surface lisse. Afin de mesurer la différence entre deux surfaces, de nombreuses mesures ont été développées. Même si aucune d'entre elles n'est totalement adaptée à la mesure de la qualité visuelle d'un maillage par rapport à un modèle de référence, aussi avons-nous retenu la norme la plus satisfaisante. L'erreur L^2 — ou *RMSE* pour *Root Mean Squared Error*) —, communément utilisée pour la comparaison de surfaces dans le domaine de la modélisation géométrique, se définit comme suit :

Définition 14. *L'erreur L^2 est l'erreur géométrique $d(X, Y)$ calculée entre deux surfaces X et Y, indépendamment de leur topologie. Pour la calculer, on cherche tout d'abord à exprimer $d(x, Y)$ représentant la distance euclidienne entre un point x de la surface X et le point le plus proche de la surface Y, pour tous les points de X. L'erreur L^2 est alors exprimée de la façon suivante :*

$$d(X,Y) = \sqrt{\frac{1}{aire(X)} \int_{x \in X} d(x,Y)^2 \, \mathrm{d}x}$$

Il est à noter que la définition de cette erreur n'est pas symétrique : il est fréquent que $d(S_1, S_2) \neq d(S_2, S_1)$.

Cette mesure nous permet notamment d'évaluer le compromis débit-distorsion, le débit étant le nombre de bits lus dans le fichier, et la distorsion l'erreur L^2 entre le maillage courant et le maillage pleine résolution. Pour la calculer, nous avons utilisé l'outil METRO [CRS98] avec échantillonnage de Monte-Carlo. Pour pallier la dissymétrie de l'erreur L^2, nous avons utilisé la moyenne de $d(S_1, S_2)$ et $d(S_2, S_1)$.

Remarque 4. *Dans les chapitres suivants, nous serons fréquemment amenés à parler d'arbres. Afin de nommer les niveaux d'un arbre, nous emploierons la convention suivante : la racine correspond au niveau* 0*, et le niveau augmente lorsque l'on descend dans l'arbre. Par exemple, pour un arbre complet de hauteur* 3*, la racine sera au niveau* 0 *et les feuilles seront au niveau* 3*.*

Chapitre 2

Etat de l'art

« I can't do it all myself. I'm not that kind of artist, I'm the kind of artist who works off other people best. It was like playing with Tinker toys or dominoes, trying to get all the pieces to fit like a jigsaw puzzle so I'd have a whole portrait but it would still have many facets. »
Iggy Pop.

Ce mémoire se situe au carrefour de deux grands domaines de la géométrie algorithmique. Le premier est la compression de structures géométriques, *sans perte* d'information, *out-of-core* et *progressive*. Le second concerne la visualisation interactive de maillages très volumineux.

Ainsi, la diversité des sujets abordés ici implique un état de l'art relativement vaste. Nous avons donc choisi d'adopter le point de vue historique plutôt que de rechercher l'exhaustivité. Dans cette optique, et afin de donner au lecteur une vision d'ensemble, nous ne décrivons pas dans le détail toutes les méthodes citées, mais uniquement celles ayant une forte parenté avec nos travaux.

Les travaux de compression — mono-résolution, puis progressive — ont majoritairement été publiés depuis 1995, avec un objectif principal : obtenir le nombre de bits par sommet le plus faible possible. Les spécificités et les contraintes liées aux maillages volumineux sont traitées par un certain nombre d'articles, dont un sous-ensemble se concentre sur leur visualisation interactive. Sans oublier les rares travaux qui traitent de la visualisation à partir de fichiers comprimés.

Remarque 5. *Le vaste domaine de la compression* ***avec perte****, dont le principe général consiste en une analyse fréquentielle du maillage, n'apparaît pas dans cette étude, car nous avons choisi de nous restreindre au champ de la compression* ***sans perte*** *d'infor-*

mation.

2.1 Compression de maillages

La compression de maillages géométriques est un domaine théorique situé entre la géométrie algorithmique et la compression de données standard. Une grande partie des articles étudiés concerne la compression de maillages surfaciques triangulaires, domaine qui a été beaucoup plus étudié que le cas général dans lequel nous situons notre travail. Dans tous les cas, il s'agit de coupler un codage de la géométrie (coordonnées des points) et un codage de la connectivité (connections entre les points). Alors que ces deux informations sont encodées séparément dans les fichiers non comprimés, elles sont généralement entrelacées dans les fichiers comprimés. On distingue les algorithmes de compression mono-résolution, qui nécessitent d'attendre la décompression complète avant de visualiser l'objet, des méthodes de compression progressive, qui permettent de voir le modèle se raffiner au fur et à mesure de la décompression.

2.1.1 Compression mono-résolution

La relation d'Euler (*cf.* section 1.1.3) nous indique que dans le cas d'un maillage triangulaire, le nombre de triangles est environ le double du nombre de points. Dans un fichier binaire représentant un maillage triangulaire, la connectivité occupe généralement deux fois plus d'espace que la géométrie. C'est pourquoi la plupart des méthodes de compression donnent la priorité à l'efficacité de la compression de la connectivité. Leur idée commune est d'énumérer les sommets dans un ordre déterministe qui permettra la reconstruction de la connectivité sans que celle-ci soit encodée de façon explicite. La séquence obtenue est constituée de chaque sommet accompagné d'un code indiquant la façon dont il est relié aux sommets précédents. La connectivité est donc déterminée par ces codes, mais aussi par l'ordre des sommets. La compression géométrique, quant à elle, plutôt que de stocker les coordonnées absolues des sommets, utilise souvent le codage différentiel et la prédiction de position géométrique.

Codage de la connectivité

Les différentes méthodes de codage de la connectivité visent à se rapprocher de la borne inférieure théorique de 3,24 bits par sommet, obtenue par Tutte [Tut62] en énumérant les différents maillages constructibles à partir d'un ensemble de points.

Une première catégorie d'articles [Cho97, Dee95, ESV96, XHM99] utilise le

principe des bandes de triangles, largement utilisées par les cartes graphiques, et par conséquent par les bibliothèques graphiques telles OpenGL ou DirectX. Chaque sommet lu est connecté aux deux sommets précédents, formant ainsi un nouveau triangle, et réduisant d'autant le nombre de références à un même sommet. Le cas le plus simple consiste à construire cette bande en zigzag : le nouveau sommet est connecté au dernier segment, alternativement par la droite et par la gauche. Un cas plus général propose de lever cette contrainte en associant un code binaire à chaque nouveau sommet qui indique si celui-ci doit être connecté par la droite ou par la gauche. Cette technique est très efficace pour coder un maillage décomposable en un faible nombre de bandes de triangles. Ce qui n'est malheureusement que très rarement le cas : les sommets partagés par plusieurs bandes doivent être dupliqués. Selon Evans, Skiena et Varshney [ESV96], dans cette représentation, chaque sommet est référencé en moyenne deux fois dans la séquence finale. Des techniques telles que le stockage en mémoire tampon des k derniers sommets permettent de réduire cette redondance. Les meilleurs résultats en termes de compression de la connectivité sont obtenus par l'algorithme utilisé par Deering [Dee95] et Chow [Cho97] : $7,5 + \log_2 n/8$ bits par sommet (où n est le nombre total de sommets).

Une seconde série d'articles [GS98, KR99a, TR98, TG98] présente des façons similaires de construire la séquence de sommets. Le parcours débute par une face ou une arête et procède par conquête. A chaque sommet est associée une commande permettant de déduire la connectivité de ce nouveau sommet aux précédents. La méthode de Touma et Gotsman [TG98] encode le degré des sommets et parvient ainsi à limiter à deux le nombre de commandes nécessaires. Ils obtiennent de très bons taux de compression pour des maillages réguliers (1,4 bits par sommet en moyenne), mais qui se dégradent pour des maillages irréguliers. King et Rossignac [KR99a] proposent un algorithme dont les performances en moyenne en sont proches, et qui sont bornées à 3,67 bits par sommet dans le pire des cas.

Codage de la géométrie

Seule une partie de ces méthodes traitent de l'encodage de la géométrie. L'ordre d'apparition des sommets dans le fichier comprimé est optimisé pour réduire le coût de stockage de la connectivité. Cependant, il implique souvent une bonne proximité de deux sommets successifs, ce qui est favorable à la compression de la géométrie. Cette dernière fait très souvent appel à deux notions :

- La prédiction de position : à partir de la position géométrique des n sommets précédents, un prédicteur P donne une prédiction de la position du sommet à venir :

$$P(\mathbf{C}, s_{i-1}, ..., s_{i-n}) = \sum_{k=1}^{n} C_k \, s_{i-k}, \quad avec \;\; \mathbf{C} = (C_1, ..., C_n)$$

On distingue généralement trois types de prédicteurs, illustrés par la figure 2.1 :
- le prédicteur différentiel ($n = 1$, $\mathbf{C} = (1)$)
- le prédicteur linéaire direct ($n = 2$, $\mathbf{C} = (2, -1)$) et
- le prédicteur de type parallélogramme ($n = 3$, $\mathbf{C} = (1, 1, -1)$).

- Le codage différentiel : seule la différence entre la position géométrique réelle du sommet et la position prédite est écrite par codage entropique.

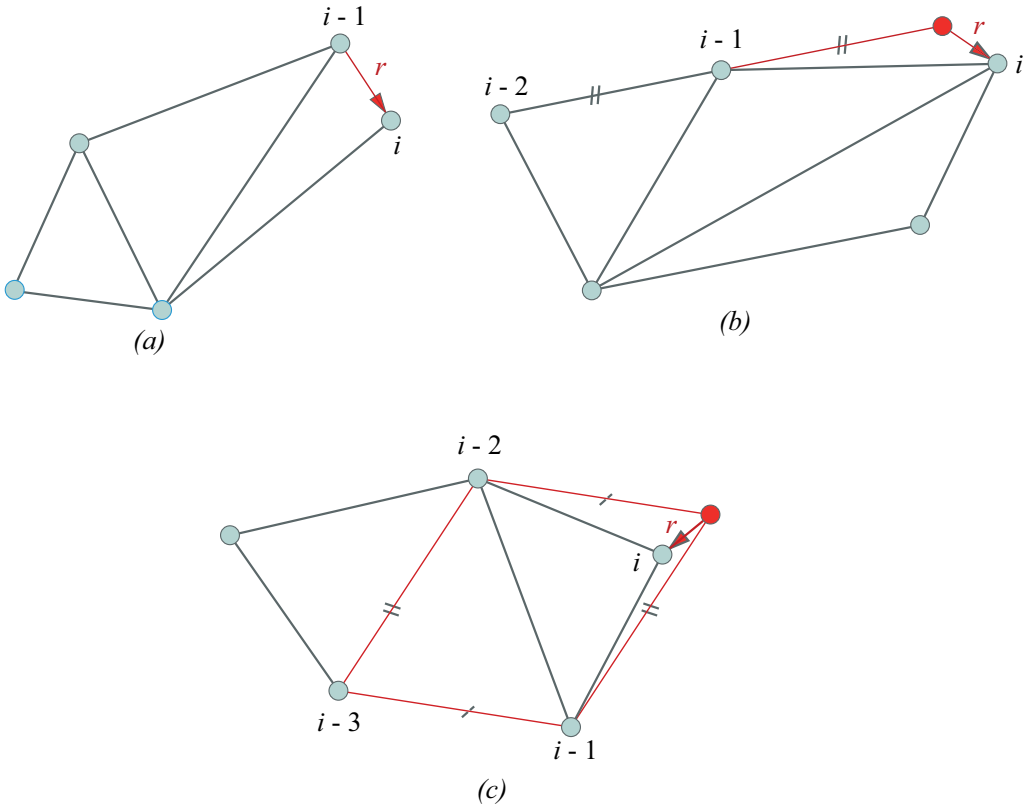


FIGURE 2.1 – Techniques de prédiction pour le codage des positions : (a) prédicteur différentiel, (b) prédicteur linéaire direct, (c) prédicteur de type parallélogramme

On constate que même si les articles dissocient souvent les coûts de codage de la géométrie et de la connectivité, ceux-ci sont fortement liés, et une méthode est à évaluer en prenant en compte ces deux paramètres simultanément.

Maillages volumineux

En 2003, Isenburg et Gumhold [IG03] ont introduit une méthode *out-of-core* mono-résolution, permettant ainsi la compression de maillages très volumineux, algorithme que nous détaillerons dans la partie 2.2.

Une revue plus détaillée de ces méthodes peut être trouvée dans les études de Gotsman *et al.* [GGK02] et de Alliez *et al.* [AG03].

2.1.2 Compression progressive

La compression progressive est basée sur la notion de raffinement. A tout moment de la décompression, il est possible d'obtenir une approximation du modèle original d'autant plus précise que la quantité de données lues est importante. Il n'est donc pas nécessaire de lire l'intégralité du fichier pour obtenir un premier aperçu de l'objet. Cela peut notamment s'avérer utile lors d'une transmission réseau ou lors de la lecture d'un fichier volumineux. L'enjeu consiste à obtenir une approximation du maillage final la plus fidèle possible tout au long du processus de raffinement. La mesure du ratio débit/distorsion permet d'en apprécier la qualité. Ce domaine de recherche a été particulièrement actif ces dix dernières années.

Au commencement étaient les maillages progressifs...

Les premières techniques de visualisation progressive ne s'intéressaient pas à la compression, et induisaient souvent une augmentation non négligeable de la taille du fichier. La raison en est que le stockage d'une structure hiérarchique pour coder la géométrie et/ou la connectivité tend à accroître la taille de la structure. Les premiers travaux introduisant la notion de *maillages progressifs* sont dus à Hoppe [Hop96]. Le processus de simplification consiste en une série de contractions d'arêtes, processus durant lequel chacune d'entre elles se voit attribuer un code afin de permettre le déroulement inverse lors du raffinement, *via* une suite de séparations de sommet. Ce code indique le sommet qui doit être séparé (coût $= log_2(n)$, si n est le nombre total de sommets) ainsi que les deux arêtes adjacentes à séparer. Ainsi, le coût de codage de cette méthode, en $O(n.log_2(n))$, est non linéaire, ce qui ne permet pas l'encodage de maillages volumineux.

... comprimés...

Par la suite, dans le but d'obtenir un coût de codage le plus faible possible, plusieurs méthodes de compression mono-résolution parmi celles présentées ci-dessus ont pu être étendues à la compression progressive.

Dans certains cas, seule la connectivité est transmise de façon progressive : lorsqu'un point est créé, ses coordonnées sont précises et définitives. A l'inverse, d'autres méthodes offrent un codage progressif des coordonnées des points, alors que la connectivité est comprimée en mono-résolution. Enfin, une troisième catégorie de travaux proposent un encodage totalement progressif, tant au niveau de la géométrie que de la connectivité. Dans ce dernier cas, il s'agit de trouver le meilleur compromis entre nombre de sommets et précision sur leur position géométrique, problème qui a été étudié de façon détaillée par

King et Rossignac [KR99b].

Taubin *et al.* [TGHL98] ont adapté la méthode de compression mono-résolution proposée par Taubin et Rossignac [TR98] afin de rendre progressif le codage de la connectivité. Cohen-Or *et al.* [COLR99] alternent entre les 2- et 4-colorations du graphe des triangles du maillage pour localiser un système indépendant de sommets, sur lesquels des techniques de simplification séquentielle par suppression de sommet sont appliquées. Les trous sont rebouchés grâce à une retriangulation locale et déterministe qui ne nécessite aucun surcoût d'encodage. La connectivité est ainsi encodée progressivement, par un coût moyen de 6 bits par sommet, et la géométrie est comprimée à l'aide de la prédiction de positions. Li et Kuo [LK98] introduisent la progressivité de la géométrie, qui est ainsi raffinée en même temps que la connectivité : l'insertion progressive de sommets est accompagnée d'une augmentation de la précision sur les coordonnées. Pajarola et Rossignac [PR00], obtiennent une moyenne de 7,2 bits par sommet pour le codage de la connectivité ; plusieurs niveaux de résolution sont construits, chacun d'entre eux regroupant une série indépendante de contractions d'arête. Alliez et Desbrun [AD01] ont proposé un algorithme basé sur la suppression progressive des sommets indépendants, comme Cohen-Or *et al.* [COLR99]. Ils ajoutent des phases de retriangulation et régularisation afin de combler les trous polygonaux engendrés par la suppression de sommets, sous la contrainte de maintenir les degrés des sommets autour de 6, afin d'assurer une meilleure régularité du maillage tout au long du processus de décimation. Ils améliorent par ailleurs le codage des positions en décomposant l'information en deux composantes normale et tangentielle. Il en résulte un coût moyen de 3,7 bits par sommet. Karni *et al.* [KBG02] ordonnent les sommets d'une façon particulière à l'aide d'une chaîne d'arêtes optimisée et minimisent le nombre de sauts entre deux sommets non incidents. Ils obtiennent un coût moyen de 4,5 bits par sommet, ainsi qu'un accès optimisé aux données afin de permettre un rendu rapide.

. . . finalement couronnés de succès

Les taux de compression se sont peu à peu rapprochés de ceux obtenus en mono-résolution, pour finir par quasiment les égaler. Gandoin et Devillers [GD02] obtiennent 19,3 bits par sommet en moyenne au total (géométrie et connectivité réunies), contre 18,5 pour la méthode mono-résolution de Touma et Gotsman [TG98]. Le format progressif présenté dans cet article [GD02] constitue la base de ce travail de thèse. Contrairement à la majorité des méthodes de compression, la priorité est donnée à la compression géométrique. La méthode se fonde sur un *kd-arbre* dans lequel les points sont répartis. Afin de mémoriser la connectivité entre les points, les feuilles du *kd-arbre* sont fusionnées deux à deux, avec stockage des « codes de connectivité » indiquant les changements induits sur le maillage (« contraction d'arête » ou « fusion de sommets »). L'al-

gorithme est très général, puisqu'il n'est pas spécialisé pour une certaine dimension : il peut encoder des maillages 2D jusqu'aux maillages tétraédriques, et rien ne s'oppose à son utilisation dans des dimensions encore supérieures. De même, il est capable d'appréhender tout type de topologie, depuis les maillages réguliers et manifold jusqu'aux soupes de polygones. Cette méthode est décrite en détail dans la section 3.2. Peng et Kuo [PK05] se sont basés sur cet article pour améliorer les taux de compression (entre -10% et -20% de réduction de taille), au détriment du caractère généraliste de l'algorithme : seuls des modèles triangulaires 3D sont utilisables. Pour cela, la prédiction est largement employée, et le *kd-arbre* est remplacé par un *octree*, qu'ils construisent selon une file de priorité afin d'améliorer le ratio débit-distorsion.

2.2 Traitement de maillages volumineux

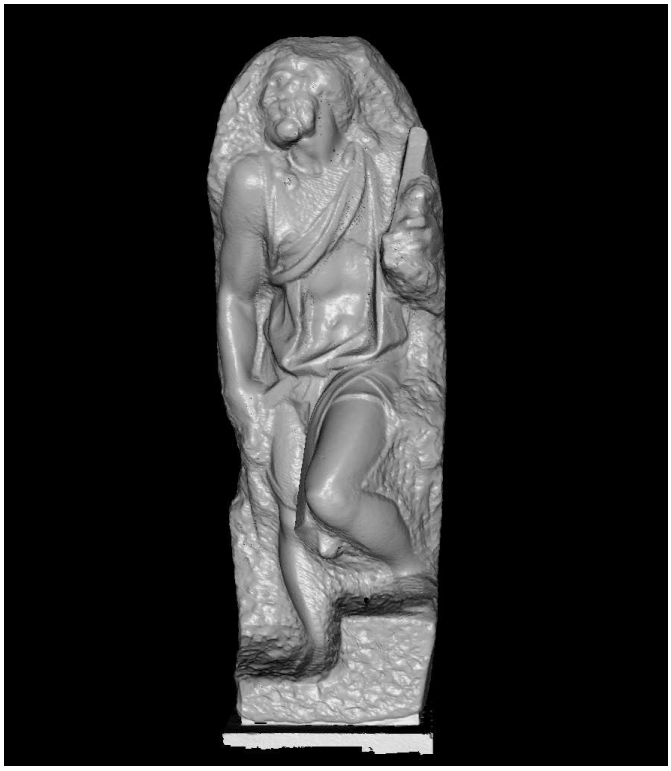


FIGURE 2.2 – Modèle du Saint Matthieu de Michel-Ange, numérisé dans le cadre du *Digital Michelangelo Project*

Ces dernières années, le développement rapide des capacités de stockage a conduit à une forte augmentation du volume des données. Cela est particulièrement vrai dans le domaine des données géométriques, où les maillages, qu'ils soient issus de numérisations par laser ou de modélisations de scènes complexes, peuvent atteindre plusieurs giga-octets ; par exemple, le modèle numérisé du Saint Matthieu de Michel-Ange présenté sur la figure 2.2, numérisé par laser à une précision de 0,25 mm dans le cadre du *Digital Michelan-*

gelo Project, contient 372 millions de triangles et pèse 7 giga-octets au format PLY. Malheureusement, la courbe de croissance de la mémoire externe n'est pas accompagnée d'une amélioration proportionnelle des temps d'accès et des capacités des mémoires centrales. Les modèles 3D de plusieurs giga-octets ne peuvent plus être chargés entièrement en mémoire vive sur une station de travail standard, alors même que les formats de fichier n'ont pas été conçus dans cette éventualité. Par exemple, simplifier le maillage du Saint Matthieu avec une méthode *in-core* emploierait plus de 55 Go de mémoire centrale (source : [CMRS03]). Il s'ensuit une perte grandissante d'efficacité dans le chargement et l'accès aux données, et par conséquent une diminution drastique des performances de tout traitement ultérieur, voire dans certains cas l'impossibilité de les effectuer : compression, parcours ordonné des éléments, calculs tels que l'enveloppe convexe ou l'iso-contour (sur un modèle de terrain, ou bien à partir d'une fonction scalaire définie sur la surface d'un maillage), et même affichage.

C'est pourquoi les années 2000 ont vu apparaître de nombreux travaux proposant de nouveaux formats et algorithmes afin de gérer, comprimer, traiter ou éditer les maillages dont la taille implique une gestion *out-of-core* des maillages. L'idée principale de ces algorithmes est d'organiser les opérations à effectuer afin de pouvoir les exécuter sur une partie seulement du fichier. Ce dernier est donc vu comme un ensemble de blocs, chacun d'eux étant d'une taille telle qu'il peut être chargé sans difficulté en mémoire centrale. Dans certains cas, un prétraitement — souvent basé sur un tri externe — est effectué afin de réorganiser et optimiser la position des données dans le fichier.

Ainsi, Lindstrom [Lin00] introduit un algorithme capable d'effectuer une simplification *out-of-core* des ensembles volumineux de données polygonales. Pour cela, il étend les travaux de Rossignac et Borrel [RB93] en remplaçant la classification de sommet originale par l'utilisation de l'erreur quadratique afin de déterminer le sommet approchant chaque groupe. Au delà de l'amélioration du taux de distorsion, l'espace disque requis est diminué, et une seule passe sur le modèle complet est nécessaire, au lieu de deux auparavant. La complexité de l'algorithme devient ainsi linéaire, ce qui autorise le traitement de maillages de toute taille. Cependant, si elle s'affranchit de la taille des données en entrée, la complexité mémoire reste dépendante de la taille des données en sortie, ce qui limite la taille du modèle simplifié. Un an plus tard, Lindstrom et Silva [LS01] se basent sur ces travaux afin de développer une méthode de simplification dont la complexité mémoire est indépendante de la taille des données en entrée et de celles en sortie.

Nous avons déjà évoqué la méthode proposée par Isenburg et Gumhold [IG03] dans le paragraphe consacré à la compression mono-résolution (2.1.1). Pour réaliser la compression *out-of-core*, l'encodeur accède au maillage de la façon la moins fréquente et la plus cohérente possible. Une structure de données adaptée à la gestion en mémoire externe, basée sur le partitionnement spatial, offre un accès transparent au maillage volumineux dont la représentation utilise

les demi-arêtes. La compression est faite en une seule passe, et à tout instant, seule une partie du maillage doit être gardée en mémoire. Le fichier résultant est fortement comprimé, et permet une décompression en continu (*streaming*) avec une faible empreinte mémoire, dont un exemple est donnée par la figure 2.3.

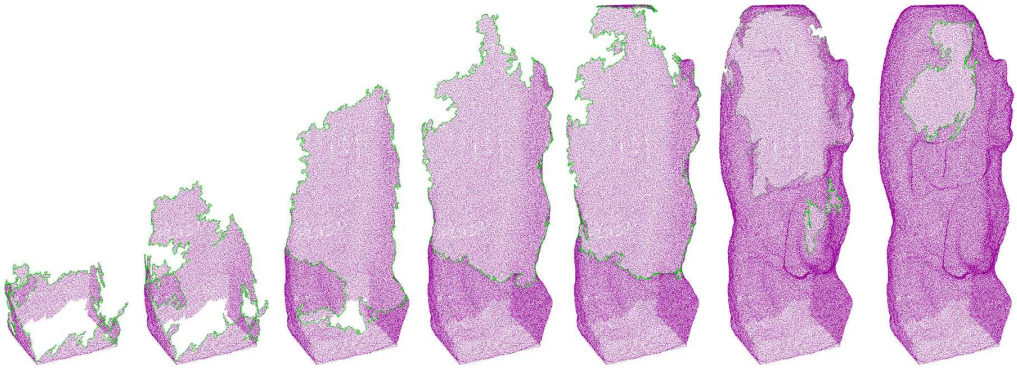


FIGURE 2.3 – Etapes de décompression du Saint Matthieu par la méthode présentée dans [IG03]

Cignoni *et al.* [CMRS03] ont introduit une structure de données nommée *Octree-based External Memory Mesh (OEMM)*, qui permet de gérer en mémoire externe de très gros maillages et d'implémenter tous les algorithmes géométriques appliqués au maillage *via* une édition locale. Pour cela, seule une partie du maillage est chargée à chaque instant en mémoire centrale. Une représentation globale indexée permet d'accéder aux données, et plusieurs mécanismes permettent le chargement et le traitement de portions de l'objet : partitionnement de l'espace intégré dans l'OEMM, détection des frontières de la région courante grâce à un système de *tags*, et ré-indexation automatique des parties chargées en mémoire. Cette structure peut être utilisée pour l'édition, la visualisation mono-résolution, ou encore la simplification de maillages très volumineux, sur des architectures matérielles à faible coût, grâce à un paramétrage par l'utilisateur de la taille des portions chargées.

Isenburg et Lindstrom [IL05] sont partis du constat qu'historiquement, les formats de fichier 3D usuels (OFF, OBJ, PLY, VRML, etc.) n'ont pas été développés dans l'optique de gérer des fichiers dont la taille interdit leur chargement complet en mémoire vive. Notamment, les formats indexés n'imposent aucune contrainte sur la position dans le fichier des sommets composant un polygone. Avec l'augmentation de la taille des fichiers se pose le problème de la cohérence et de l'ordre des éléments. Un segment peut être composé du premier et du dernier sommet de la liste complètes des points. La figure 2.4 fait apparaître en rouge les triangles dont les indices des sommets présentent une grande disparité ; les trois maillages de droite sont issus de scans laser, visiblement effectués bloc par bloc. Partant de ce constat, les auteurs ont développé un nouveau format de fichier adapté au *streaming* de maillages polygonaux, quelle que soit leur taille, et dont l'objectif est de remplacer les formats actuels. La

caractéristique principale d'un format de *streaming* est qu'il indique de façon explicite la première et la dernière fois qu'un sommet est utilisé par un polygone. Les sommets et les faces sont donc entrelacés dans le fichier, un sommet étant introduit uniquement lorsqu'il va être utilisé ; lorsqu'un triangle fait référence à un sommet pour la dernière fois, il l'indique (par exemple *via* un indice négatif). Ainsi, le nombre de sommets à conserver en mémoire à un instant donné est réduit, sous réserve que l'organisation du fichier est cohérente.

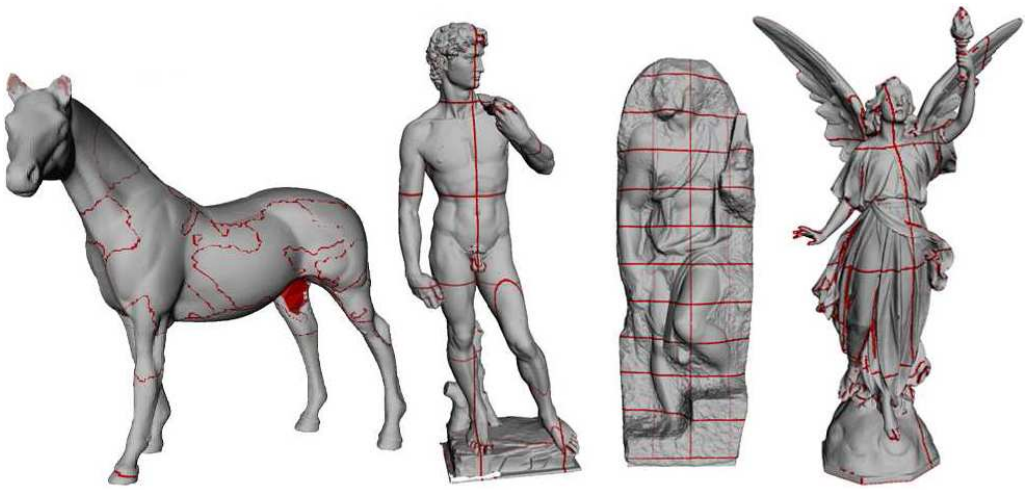


FIGURE 2.4 – La coloration en rouge des triangles dont la plage d'indices des sommets est très étendue met en évidence la façon dont ils ont été construits - source : [IL05]

Isenburg, Lindstrom, et Snoeyink [ILS05] se sont démarqués des approches antérieures de la compression de maillages, qui nécessitent généralement le maillage complet avant de débuter le traitement. Ils ont proposé un algorithme de compression non-progressive en *streaming* qui emploie très peu de ressources mémoire et qui permet ainsi l'encodage et le décodage de maillages de toutes tailles. Le processus de compression peut commencer son travail incrémentalement dès lors qu'il dispose de quelques polygones, lus à partir d'un fichier non comprimé ou reçus *via* une connexion réseau. Il n'est donc pas nécessaire de créer une structure de données sur disque en préalable. Il en résulte des temps de compression très réduits : alors que l'encodeur de Isenburg et Gumhold [IG03] prenaient environ 7 heures pour créer une structure sur disque de 11 Go puis 4 heures pour comprimer les données avec une empreinte mémoire de près de 400 Mo, les auteurs obtiennent ici un fichier comprimé en 15 minutes, sans créer de fichier temporaire, avec une occupation mémoire de 6 Mo.

En 2006, Cai *et al.* [CLW+06] ont proposé le premier algorithme de compression progressive *out-of-core*, permettant ainsi la compression de maillages volumineux. Leur technique permet d'adapter la plupart des algorithmes de compression progressive *in-core* basés sur un *octree* en algorithmes *out-of-core*. L'idée est de remplacer l'*octree* initial par un *multi-level adaptive octree*. Il s'agit

d'un *octree* composé de deux niveaux : un *adaptive octa-block tree (BlockTree)* qui divise l'espace du modèle en blocs, et dans chaque feuille-bloc non vide du BlockTree, un *local adaptive octree (LocalTree)* contenant le sous-maillage localisé dans le bloc. Il est ainsi possible de parcourir cet arbre multi-niveau de la même façon que l'arbre de la méthode *in-core* à adapter. Les blocs peuvent donc être chargés un à un afin d'en comprimer le sous-maillage interne, avec une empreinte-mémoire restreinte.

2.3 Visualisation interactive d'objets volumineux

Un sous-ensemble de ces méthodes travaillant avec des maillages volumineux mérite une attention particulière de notre part. Il s'agit de la visualisation rapide et interactive de maillages de grande taille. La multiplication de ces objets, qu'ils soient issus de scanners 3D haute résolution, de modèle issus de la CAO (Conception Assistée par Ordinateur) ou d'isosurfaces, s'est confrontée au besoin le plus élémentaire lorsque l'on dispose d'un tel maillage : sa visualisation. Les cartes graphiques ne disposaient et ne disposent toujours pas d'une mémoire suffisante pour charger dynamiquement ces maillages, rendant impossible le chargement de ces données *via* les méthodes habituelles. A l'heure du multimédia et du haut débit, attendre plusieurs heures pour obtenir un rendu n'était pas satisfaisant. La communauté scientifique s'est donc attelée au développement de techniques permettant de s'affranchir des contraintes liées à la taille des maillages en question.

L'idée commune de ces travaux repose sur la sélection des informations pertinentes à afficher. Pourquoi afficher 370 millions de triangles et des sommets avec une précision de 16 bits par coordonnée quand quelques dizaines de milliers de triangles et 8 bits de précision suffisent à obtenir une précision au pixel près après projection de la vue globale du Saint Matthieu sur l'écran ? Pourquoi charger et afficher les données hors du champ de la caméra lorsque l'utilisateur souhaite seulement visualiser le gros orteil du David de Michel-Ange ? En réponse à ces deux questions, les algorithmes sont *out-of-core*, dans la mesure où seules les données nécessaires et suffisantes à un affichage précis mais concis sont chargées en mémoire à un instant donné. Ainsi, ils présentent systématiquement deux caractéristiques principales :

- Progressivité : elle permet un niveau de détail adapté au cadre de visualisation, *i.e.* à la position de la caméra par rapport à l'objet 3D.
- Partitionnement du modèle : il autorise l'affichage des seules parties visibles du point de vue de la caméra.

En complément, un arbre ou un graphe est très souvent construit. Il indique les différents raffinements/simplifications possibles au niveau des éléments du maillage (points, segments, triangles...). L'élimination des faces cachées ou hors-champ est fréquemment utilisée afin de réduire le flux de données vers la

carte graphique.

A l'aube du nouveau millénaire, Rusinkiewicz et Levoy [RL00] ont introduit QSplat, premier système de rendu *out-of-core* à base de points (et non de maillages polygonaux) répartis dans une structure hiérarchique de sphères englobantes. Ces dernières permettent de gérer les niveaux de détail pour le rendu et d'effectuer des tests de visibilité. Les nœuds de l'arbre contiennent les informations suivantes : le centre de la sphère, son rayon, un cône de visibilité et éventuellement une couleur. La taille d'une sphère englobante est choisie de façon à ce qu'elle englobe les sphères du niveau inférieur, afin d'éviter les trous. Les données (points, sphères et hiérarchies) sont toutes quantifiées et représentées sur un nombre optimal de bits de façon à réduire au maximum l'espace nécessaire en mémoire. Notamment, les positions des points contenus dans une sphère sont relatives à la sphère de niveau supérieur. Lors du rendu, l'arbre est parcouru soit jusqu'aux feuilles, représentant un point isolé, soit jusqu'à ce que l'aire de la sphère d'un nœud projetée sur l'écran soit inférieure à un certain seuil. Trois types de primitive sont utilisés pour l'affichage des points : un point OpenGL (*i.e.* un carré), un cercle (*i.e.* un polygone avec une texture représentant un disque) ou un « point flou » dont l'opacité baisse radialement selon une Gaussienne. Ainsi, il est possible d'afficher entre 1,5 et 2,5 millions de points par seconde sur des machines de milieu de gamme (processeur de 1 Ghz). La méthode utilise un affichage adaptatif : l'utilisateur navigue autour de l'objet avec un affichage rapide et approximatif puis l'affichage se raffine automatiquement quand il arrête de se déplacer.

El-Sana et Chiang [ESjC00] se sont basés sur la méthode *in-core* de simplification en fonction du point de vue présentée dans [ESV99] pour en proposer une version *out-of-core*, dans l'objectif d'afficher interactivement des objets de plusieurs millions de triangles. Durant la phase de prétraitement, ils construisent un arbre de sous-maillages en prenant soin d'optimiser les accès au disque, et ordonnent les contractions d'arête afin d'assurer une bonne qualité d'image. Lors de la navigation, seuls les nœuds de l'arbre actifs, *i.e.* nécessaires à l'affichage courant ou susceptibles d'être utilisés dans un futur proche (pré-chargement), sont conservés en mémoire centrale, les autres étant laissés sur disque. Pour des maillages dont l'arbre tient entièrement en mémoire, la vitesse d'affichage obtenue est très légèrement plus lente que celle de [ESV99], mais pour des maillages dont la taille dépasse celle de la mémoire vive, la méthode est en moyenne de 4,5 fois plus performante.

Par la suite, Lindstrom [Lin03] a développé une méthode basée sur un *octree*, qui est utilisé pour répartir le maillage dans des blocs et construire une hiérarchie multi-résolution. Le processus présente trois étapes principales :

- Simplification du maillage donné en entrée. Par conséquent, le processus implique une perte sur le maillage d'origine.
- Construction de la hiérarchie de niveaux de détail
- Navigation : raffinement et rendu dépendant de la caméra, avec système

de cache.

Les deux premières phases sont exécutées une seule fois par modèle. Elles créent une représentation du modèle sur disque dur, à partir de laquelle la troisième phase en permet la visualisation interactive. Une métrique d'erreur quadratique permet de choisir les positions des points représentants pour chaque niveau de détail, et le raffinement est basé sur la visibilité et l'erreur dans « l'espace-écran ». Les feuilles de l'*octree* correspondent chacune à un point du maillage lorsque celui-ci est raffiné à sa précision maximale ; chaque point est accompagné de sa normale. Les nœuds internes de l'arbre contiennent quand à eux : un point représentatif, des informations d'erreur, une sphère englobante, les triangles contenus, des informations de courbure. Le rendu est asynchrone, exécuté dans un *thread* indépendant du *thread* qui est chargé de mettre à jour le sous-ensemble de l'*octree* conservé en mémoire et affiché à l'écran. Ainsi, l'utilisateur peut se déplacer librement autour de l'objet. S'il souhaite afficher une zone qui n'est pas encore raffinée, une version peu précise apparaîtra avant de se raffiner progressivement. La figure 2.5 présente deux captures d'écran illustrant ces principes.

Avec *Quick-VDR*, Yoon *et al.* [YSGM04] ont proposé un algorithme du même type, mais cette fois-ci sans perte, avec la particularité suivante : une hiérarchie de blocs (appelés *clusters*) est construite. L'arbre qui en résulte est appelé *CHPM (Clustered Hierarchy of Progressive Meshes)*. Chaque bloc contient une région localisée du maillage constituée de quelques milliers de triangles, ainsi qu'un maillage progressif construit à partir de ce sous-maillage. Lors de l'affichage, le maillage est raffiné en deux étapes :

1. Tout d'abord, un raffinement « grossier » est effectué en sélectionnant dans le CHPM les blocs actifs.
2. Puis un raffinement plus fin est effectué en fonction du choix du niveau de détail auquel seront affichés les maillages progressifs contenus dans les blocs actifs.

Cette approche mixte permet de lisser la transition lors du passage d'un niveau de détail à un autre. Les blocs sont aussi utilisés pour les calculs de visibilité (*culling* d'occlusion et de *frustum*). Afin d'éviter les saccades liées aux délais de chargement depuis le disque, les auteurs introduisent une image (*frame*) de latence lors du rendu. Il est à noter que l'empreinte mémoire lors du rendu est bornée, ce qui constitue une garantie importante pour certaines applications.

Cignoni *et al.* [CGG+04] ont introduit en 2004 les *Adaptive TetraPuzzles*, méthode de visualisation interactive d'objets polygonaux volumineux dépendante du point de vue. Une des particularités intéressantes de ce travail est l'utilisation d'une hiérarchie basée sur la subdivision récursive de tétraèdres pour le partitionnement de l'espace, qui facilite la gestion des frontières entre les sous-maillages — issus du partitionnement — de niveaux de détail différents : ces frontières se doivent de coïncider parfaitement afin de ne pas introduire d'artefacts lors du rendu. Un cube englobant est tout d'abord calculé, puis la

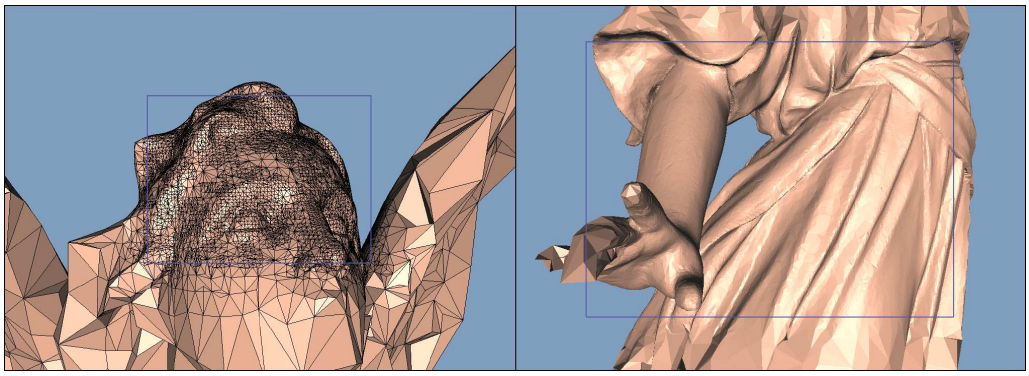


FIGURE 2.5 – Images obtenues par Lindstrom [Lin03] sur le maillage Lucy — A gauche : le rectangle violet encadre ce qui est visible à l'écran, ce qui est hors-champ est très grossier — A droite : la main vient d'apparaître dans le champ de la caméra et n'a pas encore été totalement raffinée

subdivision de l'espace débute par la création de six tétraèdres autour d'une de ses diagonales ; par la suite, si l'on nomme a_T la plus grande arête d'un tétraèdre T, chaque étape de subdivision (*i.e.* la création d'un nouveau niveau dans l'arbre de partitionnement) est générée par la division binaire de chaque tétraèdre T par le triangle composé du milieu de a_T et de l'arête opposée ; la division s'arrête lorsque le tétraèdre possède quelques milliers de triangles. Un exemple d'une telle structure est donnée sur la figure 2.6. Un ensemble de tétraèdres partageant leur plus longue arête est appelée un *diamant* ; si l'on s'assure que les tétraèdres composant un même diamant sont toujours divisés/fusionnés simultanément, et que l'on fixe les points et arêtes composant la frontière entre deux sous-maillages de cellules qui n'appartiennent pas au même diamant, alors les frontières entre deux sous-maillages voisins coïncideront avec *brio*, sans former de déchirure ni de chevauchement. Les frontières entre les niveaux $n-1$ et n de l'arbre de partitionnement sont distinctes des frontières entre les niveaux n et $n+1$. Ainsi, aucun point n'est fixé sur plusieurs niveaux de détail successifs et la précision de tous les points affichés est correcte. La phase de construction initiale est intrinsèquement parallélisable.

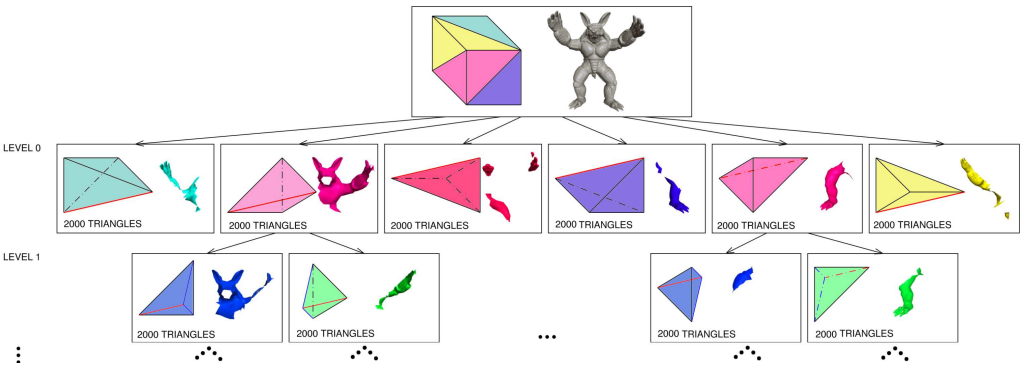


FIGURE 2.6 – Exemple d'arbre utilisé par l'algorithme de Cignoni *et al.* [CGG+04] sur le maillage Armadillo

Afin d'optimiser le rendu, chaque cellule tétraédrique de l'arbre de partitionnement contient notamment les informations suivantes :

- Sphère englobante
- Cône englobant les normales
- Erreur saturée dans l'espace du modèle (*i.e.* valeur maximale des erreurs)

Les deux premières sont utilisées pour déterminer si les faces contenues sont visibles, afin de déterminer si un sous-maillage doit être affiché ou non. Pour cela, sa sphère englobante doit couper le cône de vision, et au moins une normale doit être tournée vers la caméra. Il est à noter que cette élimination des faces non visibles est effectuée via un parcours préfixe de l'arbre ; ainsi, de larges zones hors-champ sont rapidement éliminées, ce qui évite de parcourir tous les nœuds de l'arbre. La dernière information est utilisée pour calculer si le sous-maillage présent dans la cellule a une précision suffisante : les auteurs calculent la taille en pixels à l'écran de la sphère dont le diamètre vaut l'erreur saturée et dont le centre est le point de la sphère englobante le plus proche de la caméra. Si celle-ci est inférieure ou égale à la précision d'affichage demandée (exprimée en pixels), la précision est satisfaisante et le sous-maillage est affiché. Afin de masquer la latence introduite par le chargement des données depuis le disque, une prédiction des futurs mouvements de caméra est effectuée afin de pré-charger les sous-maillages susceptibles d'être prochainement nécessaires. La structure de l'arbre est stockée en mémoire vive, les sous-maillages utilisés ou pré-chargés sont stockés en mémoire vidéo, tandis que les autres sous-maillages sont conservés sur disque. Le système de sous-maillages est adapté à une utilisation efficace du GPU puisqu'à chaque nouvelle image, il est possible de n'envoyer que les nouveaux sous-maillages à la carte graphique. Le point faible de la méthode est la lenteur de la phase de construction. Afin de la pallier, les auteurs calculent leur structure sur un réseau d'ordinateurs composé d'une station maître, et entre 1 et 16 travailleurs. Même avec 1+16 ordinateurs, la construction du St. Matthieu dure plus de 7 heures, alors qu'elle dure plus de 25 heures avec 1+1 ordinateurs. Cela s'explique par la lourdeur des calculs liés à la multi-résolution : construction de bandes de triangles optimisant la gestion du cache, calcul de volumes englobants, simplification, etc. Il est néanmoins à noter que cette phase n'est exécutée qu'une seule fois pour chaque modèle, contrairement à la visualisation. Cette dernière atteste d'une moyenne située autour de 70 millions de triangles et 45 images affichés par seconde, ce qui est nettement au dessus des performances de [Lin03] (3 millions de triangles par seconde).

Les deux méthodes décrites précédemment sont particulièrement adaptées aux maillages présentant une surface relativement lisse et topologiquement simple, afin d'obtenir des surfaces simplifiées visuellement correctes dans les niveaux intermédiaires. Gobbetti et Marton [GM05] ont développé les *Far Voxels*, dont la figure 2.7 donne un aperçu, capables d'afficher aussi bien des maillages réguliers que des « soupes de polygones » complexes, même si de l'aveu même des auteurs, la priorité est donnée à la vitesse de rendu plutôt qu'à la

qualité d'image. Le principe central reste de diviser l'espace à l'aide d'un arbre de partitionnement, ici un arbre BSP (*Binary Space Partitioning*). En revanche, si les feuilles de l'arbre sont des blocs de taille fixe contenant chacun une région du maillage triangulaire original sous forme de bandes de triangles généralisées, les nœuds internes contiennent quant à eux des primitives volumiques plus complexes, adaptées au rendu. Plus précisément, la zone de l'espace associée à un nœud du BSP est divisée en voxels ; sur chaque voxel, un processus d'échantillonnage basé sur un lancer de rayons depuis de nombreuses positions de caméra est effectué. Les auteurs s'appuient sur le fait qu'un voxel sera toujours visualisé depuis une distance telle que l'angle de vue sera très faible. L'occlusion de la part de l'environnement est prise en compte, afin que les voxels toujours masqués ne soient pas stockés et que seules les surfaces effectivement visibles soient échantillonnées. En sortie de ce calcul, pour chaque voxel, on dispose d'informations de rendu approximatives dépendant du point de vue. Afin de générer une représentation plus compacte et efficace, ces échantillons sont transformés en ce que les auteurs appellent des *shaders*, *i.e.* des fonctions retournant une couleur suivant les paramètres suivants :

- ses paramètres internes (déterminés par le lancer de rayons)
- la direction de visualisation v
- la direction de la lumière l

Au lieu de créer un unique *shader* généraliste, trois classes de fonctions sont définies :

1. K1a : *shader* représentant une surface plate, selon la loi de Lambert : normale au plan, matériaux recto, verso. Ces valeurs sont calculées par la moyenne des normales et des couleurs échantillonnées.

2. K1b : Même chose que K1a, à l'exception du calcul de la normale, qui utilise ici l'analyse en composante principale.

3. K2 : *shader* lisse, pour les autres cas, paramétré par 6 valeurs de réflectances et 6 points de contrôle de la normale, associés aux directions de visualisation principales ($\pm x$, $\pm y$, $\pm z$). Lors du rendu, une interpolation est effectuée à partir de la direction de visualisation v.

Lors du rendu, le raffinement du maillage est exécuté du premier plan vers l'arrière-plan. Il est paramétré par le seuil de projection des voxels (généralement 1 pixel). Les lectures depuis le disque ou le réseau sont asynchrones et emploient une file de priorité, dans le but de réduire la latence. Comme dans [CGG+04], afin de partager les tâches entre CPU et carte graphique, les *shaders* sont calculés sur la carte graphique, et les requêtes d'occlusion matérielles sont utilisées pour déterminer la visibilité des sous-arbres. Les auteurs utilisent un réseau de 16 PCs pour la construction *out-of-core* du modèle, qui dure par exemple environ 5h20 pour le St Matthieu. La phase la plus coûteuse est le calcul des *shaders* qui nécessite de nombreux lancers de rayons. La vitesse de rendu se situe entre 40 et 50 millions de triangles par seconde, ce qui est légèrement en deçà des performances des *TetraPuzzles* [CGG+04].

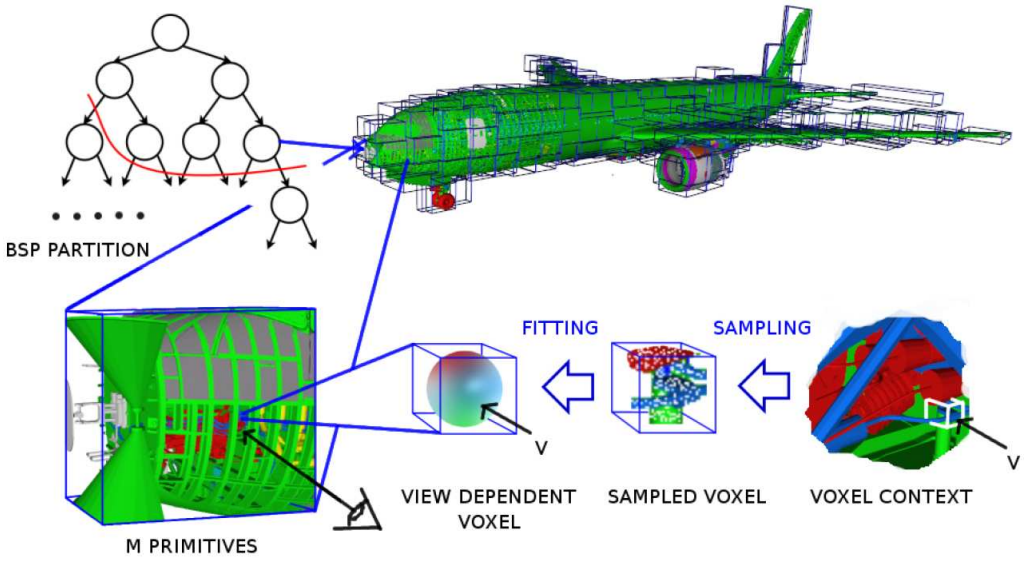


FIGURE 2.7 – Structure multi-résolution employée par Gobbetti et Marton [GM05]

L'article de Cignoni *et al.* [CGG+05] n'a pas apporté de nouvelle méthode à proprement parler, mais plutôt un cadre formel général qui englobe toutes les méthodes de visualisation par lots (*batched rendering*). Pour cela, ils proposent une adaptation du modèle *Multi Triangulation (MT)* de Puppo [Pup96]. Ce dernier est une approche générale destinée à la gestion de maillages multi-résolution, dont l'idée principale est de créer un graphe acyclique orienté (en anglais *Directed Acyclic Graph* ou *DAG*) où chaque nœud contient un fragment de maillage à une certaine résolution qui varie selon les nœuds. Actuellement, cette approche interdit le rendu temps-réel car le coût de parcours du graphe est très élevé par rapport au temps de rendu. C'est pourquoi Cignoni *et al.* [CGG+05] proposent un nouveau paradigme appelé *Batched Multi Triangulation* qui favorise l'utilisation de la puissance de calcul des GPU actuels. L'idée fondamentale consiste à déplacer la granularité de la multi-résolution, située au niveau des triangles dans les travaux de Puppo [Pup96], au niveau de sous-ensembles (appelés *patchs*) de triangles. Ces sous-ensembles sont pré-calculés, optimisés sous forme de bandes de triangles, et stockés en mémoire secondaire, prêts à être chargés à la demande et envoyés tels quels à la carte graphique. Seule la structure du *DAG* est conservée en mémoire centrale, ce qui autorise la gestion de très gros volumes de données. On constate que ce formalisme englobe une grande partie des méthodes de visualisation existantes comme les *TetraPuzzles*, proposée par les mêmes auteurs. Les auteurs définissent donc une façon « robuste et élégante » de construire des graphes *MT* ayant un bon comportement pour la multi-résolution, grâce au concept de $\mathscr{V}$*-partition* illustré par la figure 2.8. Les sous-maillages utilisés pour la multi-résolution sont ceux des partitions $\mathscr{V}_i^*$. Le processus de simplification se déroule en plusieurs étapes discrètes : un niveau de détail est généré pour chaque partition $\mathscr{V}_i^*$. Pour que l'assemblage de sous-maillages issus de $\mathscr{V}_i^*$ et $\mathscr{V}_{i+1}^*$ ne crée pas de discontinuité,

certaines conditions doivent être respectées : lors de la simplification de l'étape i vers l'étape $i+1$, les sous-maillages de $\mathcal{V}_i^*$ sont groupés pour former les sous-maillages de $\mathcal{V}_i$. La surface est alors simplifiée, à l'exception des frontières des sous-maillages $\mathcal{V}_i$, qui restent fixées. Le résultat de la simplification est divisé selon la partition $\mathcal{V}_{i+1}^*$. Il est ainsi possible d'accoler un sous-maillage de la partition $\mathcal{V}_i^*$ à un sous-maillage de $\mathcal{V}_{i+1}^*$ sans créer de discontinuité — ils sont alors adjacents par leurs arêtes vertes sur la figure 2.8. Le processus est répété récursivement. Les *TetraPuzzles* [CGG+04], avec leur structure hiérarchique basée sur des tétraèdres, sont un cas particulier de ce schéma de division de l'espace.

Pour chaque image, un budget temps et mémoire est alloué, ce qui implique que l'algorithme de raffinement soit interruptible. Une *coupe* sur le graphe *MT* est maintenue, correspondant aux sous-maillages actuellement affichés. La *coupe* est avancée/reculée en fonction des raffinements/simplifications nécessaires pour satisfaire un critère de précision basé sur la longueur moyenne des arêtes contenues dans le sous-maillage considéré. Si le budget temps et mémoire n'est pas atteint, les sous-maillages susceptibles d'être affichés prochainement sont pré-chargés. Au contraire, si le budget est atteint alors que le processus de raffinement n'est pas terminé, le maillage est affiché dans son état courant, et le processus est poursuivi lors des pas de temps suivants. Ainsi, à chaque pas de temps, un nombre relativement faible de sous-maillages est envoyé vers la carte graphique, ce qui minimise la quantité de données à transférer, permettant ainsi d'assurer un rendu temps-réel. Il est à noter que ce système de rendu est multitâche afin d'exploiter pleinement les processeurs multi-cœurs.

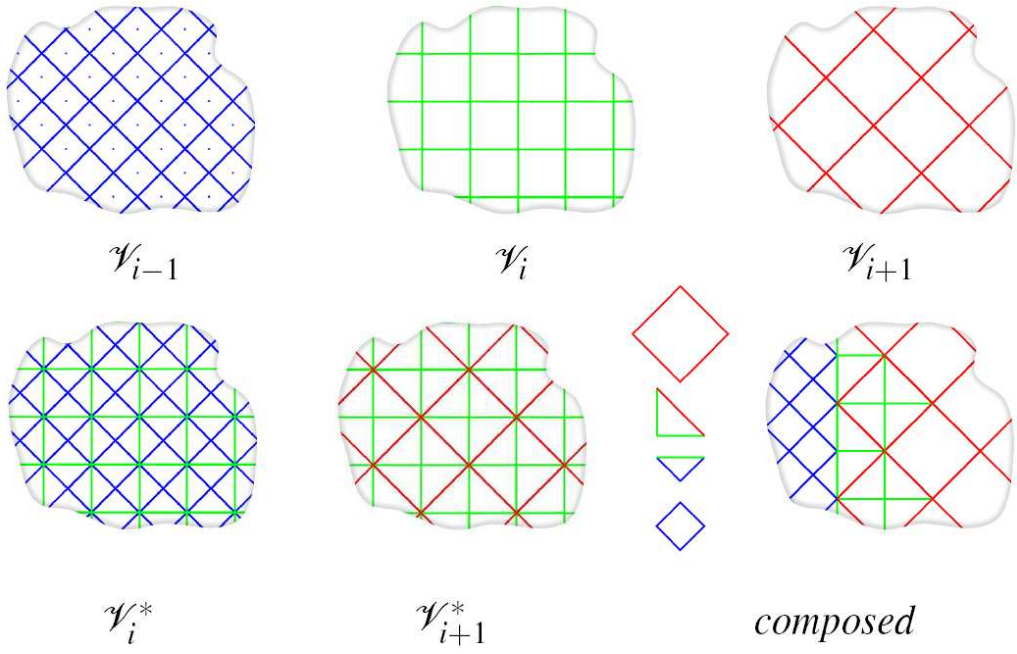


FIGURE 2.8 – Exemple de séquence de $\mathcal{V}$*-partition* dans le cas 2D [CGG+05]

Callieri *et al.* [CPCS08] présentent l'outil Virtual Inspector, qui permet à un

utilisateur non initié d'inspecter interactivement un maillage volumineux sur un PC standard. Leur format permet d'attacher tout type de données multimédia aux points du maillage, et l'interface est très flexible afin que chacun puisse adapter le logiciel à ses besoins. Cet article ne propose pas de nouvel algorithme de visualisation, mais il se concentre sur la mise en œuvre de plusieurs méthodes « state of the art » pour obtenir un logiciel performant et simple d'utilisation.

Récemment, Hu *et al.* [HSH09] ont introduit le premier algorithme de visualisation fortement parallélisé, entièrement implémenté sur carte graphique. Les auteurs utilisent la hiérarchie de sommets des *maillages progressifs* de Hoppe [Hop96], et proposent une nouvelle structure de données pour l'encodage des niveaux de détail. Alors que les récentes méthodes de visualisation avaient déplacé la granularité au niveau de sous-ensembles du maillage (ou *patchs*) — afin d'éviter l'envoi à la carte graphique du maillage actif complet pour chaque image —, au détriment de la finesse de raffinement, l'implémentation du processus de raffinement sur le GPU permet aux auteurs de revenir à une granularité au niveau des sommets et de bénéficier d'une progressivité plus fine. En revanche, la méthode n'est pas *out-of-core*, car la structure de données statique, qui nécessite 57% de mémoire supplémentaire par rapport à une liste indexée de triangles, doit être stockée entièrement sur la mémoire de la carte graphique. Par conséquent, le plus gros maillage présenté dans les résultats contient 10 millions de triangles et occupe 329 Mo de mémoire. Cette taille peut être facilement calculée à l'aide de la formule suivante : $69n + 56m$, où n et m sont respectivement les nombres de sommets du maillage d'origine et du maillage affiché. L'algorithme de visualisation est décomposé en trois étapes successives :

1. Mise à jour de l'état des sommets contenus dans l'arbre : *fusionnés*, *divisés*, ou *en cours de division*.

2. Mise à jour de la liste qui contient les sommets utilisés dans l'état courant du maillage.

3. Mise à jour de l'*index buffer*, *i.e.* de la liste des triangles, chacun d'entre eux pointant sur 3 sommets de la liste précédente.

A l'instar de la *Batched Multi Triangulation* [CGG+05], chaque pas de temps se voit allouer un budget, ce qui conduit à un amortissement sur plusieurs images des éventuelles phases de raffinement coûteuses. L'encodage des normales et des coordonnées de texture est possible et constitue un avantage sur bon nombre de méthodes de visualisation. Le temps de mise à jour du maillage varie linéairement en fonction du nombre de faces affichées : par exemple, la durée de mise à jour d'un maillage d'environ 300000 triangles se situe autour de $30ms$.

2.4 Compression et visualisation combinées

Les méthodes de compression progressive sont arrivées à maturité (le taux de compression obtenu est proche du taux théorique) et la visualisation interactive de maillages volumineux donne depuis quelques années des résultats tout à fait exploitables. Cependant, même si beaucoup d'auteurs citent, parmi les perspectives de recherche, la combinaison entre compression et visualisation, peu d'articles traitent de cette problématique. Les fichiers générés par les méthodes de visualisation sont bien souvent de taille nettement supérieure à la taille du fichier original, qui est lui-même déjà très volumineux. En fait, la compression favorise une petite taille de fichier au détriment de la rapidité de décompression et donc d'accès aux données, tandis que la visualisation se concentre sur la vitesse d'affichage. Les deux objectifs s'opposent et se trouvent donc en concurrence. Cependant, quelques recherches récentes visent à introduire la compression dans les méthodes de visualisation.

Notamment, Namane *et al.* ont développé un format comprimé de la méthode de rendu à base de points QSplat [NBB04]. Pour cela, ils utilisent le codage de Huffman ainsi que le codage différentiel afin d'encoder les normales, les positions et les rayons des sphères. Le schéma de compression est donc très simple, aussi la compression est relativement faible : le ratio de compression moyen obtenu est de $1,8$ par rapport aux fichiers obtenus par l'algorithme QSplat.

Parmi les articles traitant de la visualisation de maillages volumineux se dessine un sous-ensemble de travaux qui abordent le cas particulier des terrains. Leur particularité est due au type des maillages les représentant : il s'agit de modèles souvent appelés $2,5D$ (ou pseudo $3D$) de par le fait qu'ils sont projetables sur une surface $2D$, ce qui est le cas des terrains sans surplombs. Ils sont très souvent représentés par une grille régulière en deux dimensions dont les sommets sont pourvus d'une hauteur ou altitude. Ainsi, un certain nombre de contraintes habituellement traitées par les algorithmes de rendu en trois dimensions disparaissent, et des méthodes spécifiques sont développées afin d'en tirer parti. Ces dernières constituant un domaine de recherche à part entière, nous nous limiterons ici à deux articles récents et représentatifs des performances atteintes.

Lossasso et Hoppe [LH04] sont partis du constat que les méthodes antérieures de gestion de niveaux de détail pour les terrains adaptaient très souvent la densité de triangles à l'escarpement local, et conduisaient ainsi à gérer des critères de raffinement complexes, à utiliser le mode de rendu *immédiat*, et à parcourir la structure de données multi-résolution sans exploiter pleinement la mémoire cache. Ces dernières années ayant vu les performances des cartes graphiques augmenter jusqu'à plusieurs centaines de millions de triangles affichés par seconde, les auteurs affirment qu'il n'est plus nécessaire de réduire au

maximum le nombre de triangles affichés, mais plutôt de veiller à minimiser la quantité de données transférés à chaque image vers la mémoire de la carte. Dans leurs *Geometry Clipmaps*, ils choisissent donc d'afficher des triangles dont la taille projetée à l'écran est proche du pixel, et de concentrer leurs efforts afin d'être capables d'envoyer les données au *pipeline* graphique de façon soutenue. A la manière du *MIP mapping* pour les textures, la structure géométrique utilisée pour l'affichage courant est une pyramide de grilles régulières imbriquées. Chaque grille est constituée d'un nombre $n \times n$ de sommets, avec n de même valeur pour toutes les grilles ; la précision d'une grille l à une grille $l+1$ est doublée, donc la longueur du côté de la grille $l+1$ vaut la moitié de celle de l. Chaque grille est stockée sous la forme d'un *vertex buffer* de taille fixe en mémoire vidéo. Les grilles sont centrées autour de la position de la caméra sur le terrain, comme illustré sur la figure 2.9 ; ainsi les triangles les plus proches sont très précis, puis lorsque l'on s'éloigne du point de vue, la densité de polygones diminue. Lorsque la caméra se déplace, les *vertex buffers* sont mis à jour progressivement et efficacement grâce à l'accès toroïdal (« en escargot ») aux *buffers* : lors du glissement de la grille sur le terrain, seule une partie en forme de L doit être transférée depuis la mémoire centrale vers la carte graphique, et l'accès toroïdal évite le déplacement du reste des données, et permet donc d'exploiter la cohérence temporelle entre deux images successives. Avec 11 niveaux de pyramide de taille 255^2, les auteurs obtiennent 120 images et 59 millions de triangles par seconde. A l'instar de Hu *et al.* [HSH09], les auteurs n'utilisent pas de gestion *out-of-core* des données, mais exploitent la régularité de leur structure pour comprimer les données (avec perte), ce qui permet de stocker leur carte d'élévation des Etats-Unis contenant 20 milliards d'échantillons sur 355 Mo.

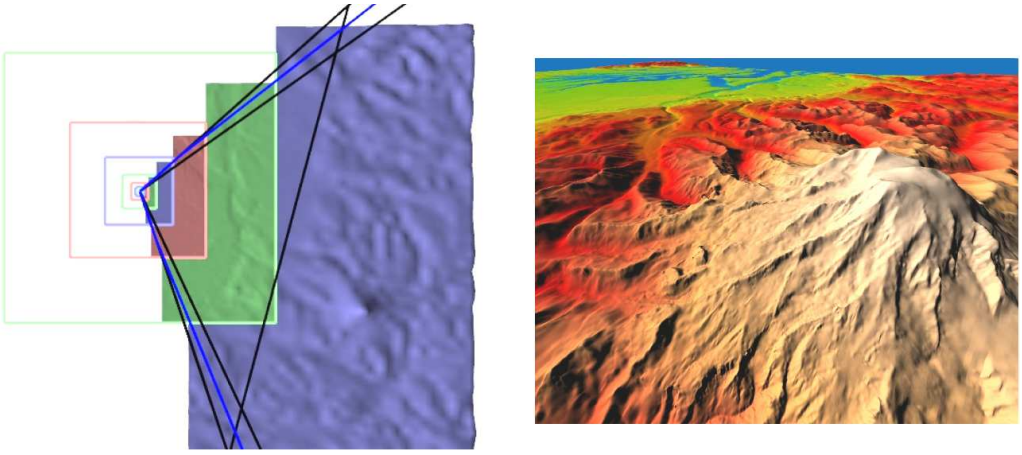


FIGURE 2.9 – A gauche : Résultat du *frustum culling* utilisé par [LH04] - A droite : rendu obtenu - source :[LH04]

En 2006, Gobetti *et al.* [GMC+06] ont introduit les C-BDAM (pour *Compressed Batched Dynamic Adaptive Meshes*), extension des BDAM [CGG+03a] et des P-BDAM (P pour *Planet-Sized*) [CGG+03b] de Cignoni *et al.*. La méthode vise

à renouer avec l'adaptabilité locale des BDAM, abandonnée par les *Geometry Clipmaps* [LH04], tout en bénéficiant des taux de compression de celles-ci. Les BDAM — sur lequels se fondent les C-BDAM — ont une approche similaire aux *TetraPuzzles* [CGG+04] (*cf.* section 2.3), excepté qu'il s'agit de $2,5D$: la hiérarchie de tétraèdres est remplacée par une hiérarchie $2D$ de triangles rectangles, regroupés en diamants carrés formés par le regroupement de deux triangles partageant leur hypothénuse. Les cellules triangulaires d'un niveau l sont obtenues en divisant les cellules du niveau $l-1$ en deux, par un segment partant du milieu de l'hypoténuse vers le sommet opposé. Chaque cellule contient un sous-maillage constitué de polygones, et les sous-maillages sont assemblés à la manière d'un puzzle pour constituer la surface à afficher. Les C-BDAM conservent cette hiérarchie, mais contrairement au BDAM dont les sommets des polygones contenus dans les sous-maillages peuvent avoir une position variable, les sommets sont placés sur une grille régulière identique pour tous les sous-maillages, quelle que soit leur position dans l'arbre multi-résolution. Il en découle deux avantages principaux :

- La connectivité est identique pour tous les sous-maillages, il est donc seulement nécessaire de stocker un *index buffer* générique sur le GPU.
- Seules les hauteurs des points — et éventuellement les attributs associés — doivent être envoyés à la carte graphique.

Les larges sous-maillages stockés en haut de l'arbre possèdent donc des sommets plus éloignés les uns des autres que les sous-maillages situés en bas, plus petits donc plus denses en sommets. Dans les BDAM, les sous-maillages étaient stockés un à un dans le fichier, au format brut. Les C-BDAM ne tiennent pas uniquement leur nom — « compressed »— de l'économie du stockage des indices de triangle et des coordonnées horizontales des sommets, mais aussi de l'exploitation de la cohérence entre deux niveaux de détail successifs pour comprimer les données à l'aide d'ondelettes. Ce qui permet aux auteurs de ne pas proposer de gestion *out-of-core* des données, car toutes les données sont chargées en mémoire vive dans un dépôt, à partir duquel elles sont envoyées au GPU en fonction des déplacements de la caméra. Les performances atteignent une moyenne de 90 images et 130 millions de triangles par seconde. La figure 2.10 présente quelques rendus de Paris obtenus par cette méthode. Les deux principaux inconvénients tiennent d'une part de la compression à base d'ondelette qui introduit des pertes et impose que la densité spectrale de puissance (carré du module de la transformée de Fourier) soit bornée, et d'autre part, même si le nombre de triangles est plus faible que dans les *Geometry Clipmaps*, il reste supérieur à celui des modèles BDAM et des P-BDAM.

Dans un domaine proche de la visualisation, Yoon et Lindstrom [YL07] ont introduit un algorithme de compression *out-of-core* de maillages triangulaires qui permet un accès aléatoire optimisé au maillage, directement à partir du fichier comprimé. Leur format n'est pas progressif et n'est donc pas approprié à la visualisation globale d'un modèle, puisqu'il nécessiterait le chargement complet de celui-ci en mémoire. Même s'il peut être employé pour visualiser

FIGURE 2.10 – Images virtuelles de la ville de Paris obtenues par les C-BDAM - source :[GMC+06]

des petites parties de gros maillages à pleine résolution, il se destine davantage à être utilisé dans le cadre d'algorithmes de parcours, comme la réorganisation de maillage ou le calcul de l'iso-contour par exemple. Néanmoins, plusieurs fonctionnalités intéressantes sont proposées :

- accès aléatoire,
- possibilité de requêtes d'adjacence et d'incidence,
- réduction des entrées-sorties sur disque, amélioration des performances jusqu'à un facteur 6.

Le problème abordé par les auteurs est similaire au nôtre de par leur recherche du compromis entre compression et accès rapide. Comme dans notre cas, les méthodes antérieures qui proposent accès aléatoire et requêtes de connectivité produisent des fichiers dont la taille est supérieure à celle des fichiers indexés d'origine. Les techniques utilisées sont par ailleurs proches de celles employées par la visualisation. Le maillage est subdivisé spatialement dans des blocs, à l'instar des *Streaming Meshes* d'Isenburg *et al.* [ILS05]. Chaque sous-maillage est comprimé indépendamment tout en conservant la disposition des éléments. Lors de la phase de relecture et d'accès au maillage, seuls les blocs contenant les données recherchées sont chargés et décomprimés à la volée, et la connectivité est reconstruite dynamiquement à partir des données partiellement chargées.

2.5 Conclusion

Combiner compression et visualisation vise à combiner taille réduite sur disque et rapidité de visualisation. Comme nous venons de le voir, aucune technique existante ne propose simultanément :

- compression sans perte de maillages quelconques (complexes simpliciaux),
- capacité *out-of-core* afin de traiter des objets de toute taille,
- visualisation interactive avec décompression à la volée (sans phase de pré-calcul) des données pertinentes.

Cette recherche de compromis est au cœur du travail exposé dans ce mémoire.

Chapitre 3

CHuMI Viewer — Principes et théorie

« Cinema is a matter of what's in the frame and what's out. »
Martin Scorsese.

La compression et la visualisation de maillages sont deux domaines de l'informatique graphique dont les contraintes et les objectifs sont généralement incompatibles, sinon contradictoires. La réduction de la redondance implique souvent une complexification des données encodées, du fait des mécanismes de prédiction — mécanismes dont l'efficacité est directement liée à la profondeur de l'analyse. Cette couche supplémentaire ralentit l'accès aux données et entre en conflit avec les performances requises pour la visualisation en temps réel. A l'inverse, la navigation dynamique au sein d'un maillage, corrélée avec les mouvements de caméra, requiert que les données soient soigneusement préparées et hiérarchisées. Cette organisation introduit redondance, et parfois même perte de données lorsque les polyèdres originaux sont approchés par des primitives géométriques simplifiées. Au delà des problèmes de stockage et de transmission *via* un réseau, l'augmentation de la taille des données générée par ce type de prétraitement pourrait à moyen terme être pénalisant pour la vitesse de visualisation elle-même, au vu de l'écart grandissant entre puissance de calcul des processeurs et temps d'accès à la mémoire externe.

Ce mémoire traite de la recherche d'un compromis entre compression et visualisation interactive. Nous avons choisi comme point de départ l'algorithme de compression *in-core* et sans perte de Gandoin et Devillers [GD02]. A ses taux de compression compétitifs, nous avons ajouté des capacités de traitement *out-of-core* — afin de permettre la manipulation de maillages sans limitation de taille —, ainsi qu'un partitionnement hiérarchique de l'espace autorisant le raffinement local à la demande : la visualisation interactive effectue le chargement des seules données nécessaires et suffisantes à la requête de l'utilisateur. Pour atteindre ces objectifs, l'idée fondamentale consiste à diviser l'objet ori-

ginal au sein d'un arbre de partitionnement de l'espace. Cette subdivision est effectuée en introduisant une structure hiérarchique principale — appelée *nSP-arbre*— dans laquelle sont plongées les structures de données originales (un *kd-arbre* couplé à un complexe simplicial). Autre caractéristique importante de la méthode, la distribution des données de géométrie et de connectivité au sein de cette structure est optimisée en faveur du compromis débit-distorsion pour supprimer l'effet de bloc inhérent aux approches de type *kd-arbre*. Nous avons baptisé l'algorithme présenté ici *CHuMI Viewer*, pour *Compressive Huge Mesh Interactive Viewer*.

Après une présentation des motivations de ce travail (section 3.1) suivi d'un résumé de la méthode choisie comme point de départ (section 3.2), notre contribution est introduite par un exemple (section 3.3), puis détaillée dans deux sections complémentaires. Tout d'abord, les algorithmes et structures de données relatifs à la construction et à la compression *out-of-core* sont présentés (section 3.4 et 3.5), puis la description est complétée du point de vue de la visualisation (section 3.6). Enfin, l'amélioration du compromis débit-distorsion est abordée dans la section 3.7.

3.1 Motivations

Le lecteur sera inévitablement amené à se poser la question suivante : au vu des capacités grandissantes des disques durs — le téraoctet est sur le point de devenir l'unité courante — et du prix dérisoire du gigaoctet de mémoire vive, pourquoi s'obstiner à comprimer, et plus encore à combiner compression et visualisation ?

Tout d'abord, l'histoire de l'informatique nous enseigne que la taille des données à traiter tend à augmenter parallèlement à la capacité des supports de stockage. Qui aurait imaginé il y a dix ans que les films vendus en 2009 occuperaient jusqu'à 50 Go (Blu-Ray double couche) et que certains systèmes d'exploitations requerraient près de 10 Go pour leur seule installation (Windows Vista) ? Le maillage le plus volumineux que nous ayons manipulé dans cette thèse contient 372 millions de triangles et occupe la bagatelle de 7,3 Go.

Il est certes légitime de s'interroger sur le bien-fondé de cette course à la qualité qui dépasse parfois les facultés de nos cinq sens voire de notre cerveau. Cependant, au-delà de cette augmentation constante de la précision et du volume des données numérisées mises à notre disposition, il existe une raison très objective de chercher à réduire la taille des données : si la capacité des media de stockage croît rapidement, il n'en est pas de même de la rapidité d'accès aux données. Temps de latence, temps de recherche et temps de transfert, sans oublier le temps de réponse du contrôleur du disque, toutes ces valeurs qu'il faut additionner pour obtenir le temps de transfert total n'ont pas connu

d'amélioration significative au cours de ces dernières années. En revanche, les performances des processeurs n'ont de cesse de s'améliorer, et l'avènement des unités multi-cœurs ne fait qu'amplifier les capacités de calcul des ordinateurs, pour peu que ces nouveaux processeurs soient efficacement exploités.

Si l'on ajoute à cela la position centrale dans nos vies quotidiennes acquises en quelques années par les réseaux de télécommunication :

1. explosion des échanges de données multimédia (e-mails, Windows Live Messenger, *peer-to-peer*),

2. développement du *streaming* (*YouTube*, *Deezer* pour ne citer que les plus médiatiques),

3. mise en réseau croissante d'un grand nombre d'applications (Google Documents, Microsoft Office Online, etc.),

on comprend mieux en quoi l'accès aux données constitue le maillon faible de la chaîne de traitement de l'information, et pourquoi la compression occupe aujourd'hui dans cette chaîne une place fondamentale.

On assiste donc à un écart grandissant entre puissance de calcul et temps d'accès aux données. Si notre méthode de visualisation n'est pas tout à fait aussi réactive que les algorithmes de visualisation pure, il est probable qu'elle les égale ou les surpasse à court terme, du seul fait de l'envol des capacités de calcul. Le domaine de la vidéo numérique grand public, du fait de la taille déraisonnable des données vidéo non comprimées, a intégré la compression en son sein dès ses débuts (création du *Moving Picture Experts Group* ou *MPEG* en 1988). Et dans le domaine de la photographie, les appareils grand public se contentent d'une compression avec perte (format JPG) tandis que les équipements (semi-)professionnels proposent de la compression sans perte (format RAW). La musique, quant à elle, voit peu à peu le disque compact — non comprimé — perdre du terrain en faveur du MP3, dont le format de compression s'accompagne pourtant d'une perte de qualité non négligeable.

Les utilisateurs de maillages (scientifiques, industriels, architectes, graphistes, designers…) se sont longtemps contentés de méthodes de compression mono-résolution — voire, le plus souvent, d'un simple *gzip* sur un fichier VRML ! — car la taille raisonnable des objets autorisait l'attente de leur décompression complète avant affichage, et cet affichage était toujours possible « d'un seul tenant » dans un visualiseur 3D ordinaire. La donne est en train de changer : l'avènement de maillages volumineux constitués de plusieurs dizaines, voire centaines de millions de primitives, notamment issus de scanner laser, nécessite de développer de nouveaux formats de fichier permettant la compression **et/ou** la visualisation sélective et interactive de ces objets. Nous avons fait le choix du « **et** ».

3.2 La compression progressive comme point de départ

Dans cette section, nous présentons brièvement la méthode proposée par Gandoin et Devillers [GD02], sur laquelle nos travaux sont basés. L'algorithme permet la compression de complexes simpliciaux de toutes dimensions. Il est *in-core*, c'est-à-dire qu'il ne permet pas de la prise en charge de maillage dont la taille interdit leur chargement complet en mémoire centrale — la limite est de l'ordre du million de triangles. L'encodage est sans perte et progressif, mais il interdit le raffinement localisé : chaque niveau de détail doit être décodé entièrement avant d'accéder au niveau de détail supérieur. Il est impossible d'obtenir une forte précision dans une zone sans raffiner à cette même précision le reste du maillage.

La méthode est basée sur la construction d'un *kd-arbre*, par subdivision des cellules en leur milieu — contrairement à d'autres méthodes basées sur un *kd-arbre*, l'obtention d'un nombre similaire de points de part et d'autre de la frontière n'est pas recherchée. La cellule racine est la boîte englobante de l'ensemble de points, et la première donnée écrite dans le fichier est le nombre total de sommets. Puis, pour chaque subdivision binaire de cellule (ou *kd-cellule*), seul le nombre de points de la première demi-cellule est encodé, le contenu de l'autre demi-cellule pouvant être déduit par soustraction. Au fur et à mesure que l'algorithme progresse, la taille des cellules diminue et les données encodées fournissent une localisation plus précise des points. Le processus de subdivision itère jusqu'à ce qu'il n'y ait plus de cellules non vides dont le côté est plus grand que 1 unité, de façon à ce que chaque point soit localisé avec la précision maximale. La figure 3.1 montre un exemple d'un tel processus.

Une fois les points séparés, chaque feuille du *kd-arbre* correspond à un sommet du maillage : il s'agit du centre de la cellule. La connectivité du modèle est intégrée et le processus de subdivision est inversé : les cellules sœurs sont fusionnées deux à deux depuis les feuilles du *kd-arbre* jusqu'à sa racine, et leur connectivité est mise à jour. La connectivité évolue, entre deux versions successives du modèle : ces changements sont encodés par des symboles insérés parmi les codes géométriques. La fusion de deux sommets est effectuée par l'un des deux opérateurs de décimation suivants :

- La contraction d'arête, définie par Hoppe *et al.* [HDD+93] et largement utilisée en simplification de surface, fusionne deux sommets adjacents (*i.e.* deux *kd-cellules* de notre arbre) sous certaines hypothèses. Les deux sommets délimitant l'arête sont fusionnés, ce qui conduit à la suppression des deux triangles adjacents (un seul si l'arête appartient à la bordure du maillage) (cf. figure 3.2a).
- La fusion de sommets, telle que définie par Popović et Hoppe [PH97], est une opération plus générale qui fusionne deux sommets même si elles ne sont pas adjacentes dans la connectivité courante. Le résultat n'est en gé-

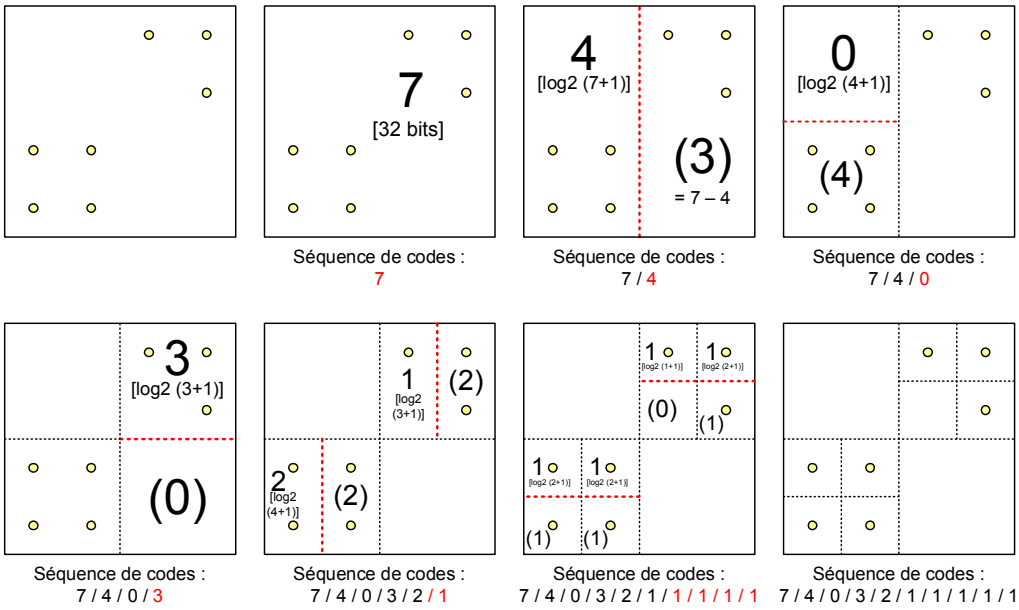
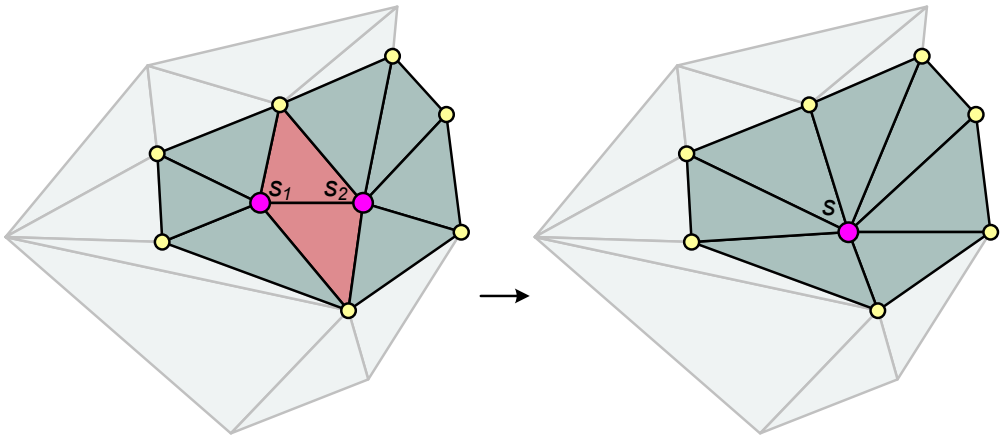


FIGURE 3.1 – Exemple de subdivision de l'espace employée par Gandoin et Devillers [GD02], selon un *kd-arbre*, dans un espace bidimensionnel avec une précision de 2 bits par coordonnée

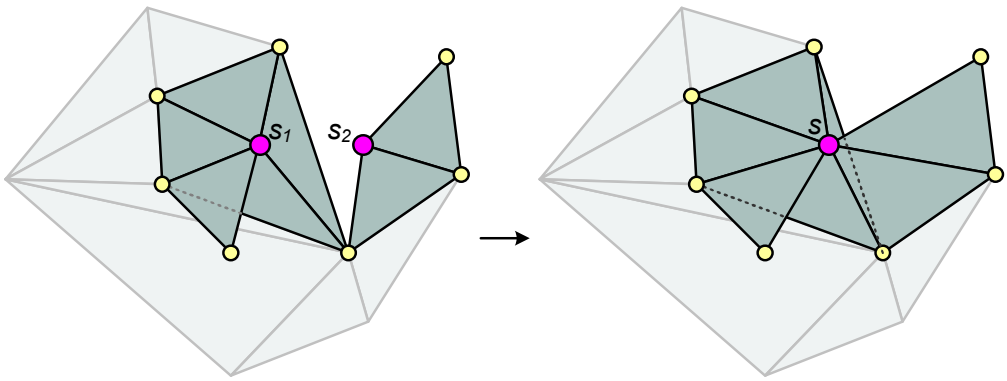
néral pas *manifold* (cf. figure 3.2b) et la séquence de codes correspondante est approximativement deux fois plus longue que la séquence de codes d'une contraction d'arête.

Ces opérations conduisent à la destruction de certains éléments de la connectivité : arêtes, triangles, etc. La façon dont la connectivité évolue pendant ces opérations de décimation est encodée par une séquence qui permet une reconstruction sans perte grâce aux opérateurs inverses (expansion d'arête et subdivision de sommet). Ces derniers permettront de reconstruire progressivement la connectivité lors de la décompression. Il est à noter que ces opérations encodent explicitement les triangles, à l'inverse de certaines méthodes comme [PK05] qui génèrent automatiquement un triangle (ABC) dès qu'il existe trois arêtes (AB), (BC) et (AC), ce qui favorise la compression mais peut introduire des pertes sur la connectivité.

Si cette méthode de compression sans perte obtient des taux compétitifs et peut traiter tout type de complexe simplicial, elle n'est pas adaptée à la navigation interactive au sein d'un maillage volumineux. Non seulement son empreinte mémoire la rend incompatible avec les maillages dépassant le million de sommets, mais, plus contraignant, la conception intrinsèque de la structure de données impose un codage hiérarchique des relations de voisinage qui interdit tout raffinement localisé. En effet, étant donné 2 sommets v et w situés dans un niveau intermédiaire du *kd-arbre*, il est possible de trouver un descendant v_i de v qui est connecté à un descendant w_j de w. Ainsi, en termes de connecti-



(a) Contraction d'arête



(b) Fusion de sommets

FIGURE 3.2 – Les deux opérateurs de fusion employés par Gandoin et Devillers [GD02] : (a) Contraction d'arête, (b) Fusion de sommets

vité, la cellule contenant v ne peut pas être raffinée sans raffiner également la cellule contenant w. Par conséquent, l'accès direct et le chargement sélectif de sous-parties du maillage sont impossibles : pour visualiser un seul sommet et ses voisins à un certain niveau de détail, le niveau de détail précédent doit être entièrement décodé.

Cette base offre donc un schéma de compression progressif pour les maillages qui peuvent être chargés en mémoire. Notre contribution y ajoute des capacités de traitement *out-of-core*, afin de pouvoir manipuler des maillages sans limitation de taille. Par ailleurs, alors que la progressivité de l'algorithme [GD02] impose de raffiner le maillage entier d'un niveau de détail $n-1$ à un niveau n avant de pouvoir évoluer vers le niveau $n+1$, la structure de données décrite dans ce mémoire autorise le raffinement local sur une zone partielle du maillage. En plus de cette propriété indispensable à la visualisation interactive, nous proposons un certain nombre d'optimisations et de choix algorithmiques qui permettent d'atteindre des conditions de navigation satisfaisantes — la parallélisation des processus de compression et décompression pour exploiter les processeurs multi-cœurs en est un exemple. Nous montrons enfin comment une redistribution des données au sein de la structure permet de s'affranchir des effets de bloc et conduit à une amélioration du compromis débit-distorsion *via* un encodage anticipé de la géométrie.

3.3 Vue d'ensemble de la méthode

Notre méthode prend en entrée un maillage non comprimé, constitué d'une liste de sommets suivie d'une liste de triangles, par exemple au format VRML, OFF ou PLY.

Le processus d'encodage de *CHuMI* se déroule en deux phases distinctes : la construction de l'arbre de partitionnement, et la compression proprement dite. Tout d'abord, le partitionnement de l'espace est effectué, *via* la création d'une structure hiérarchique appelée *SP-arbre* — ou *SP-tree*, pour *space-partitionning tree* — dans laquelle les simplexes sont répartis. La hauteur de l'arbre est bornée et dépend de plusieurs paramètres, dont les détails seront donnés dans la section 3.4. Afin d'éviter les *nSP-cellules* (*i.e.* les cellules du *SP-arbre*) ne contenant qu'un faible nombre de simplexes, une *nSP-cellule* doit contenir au moins N_{min} simplexes (valeur définie par l'utilisateur) pour être divisée. De plus, les simplexes qui appartiennent à plusieurs *nSP-cellules* sont dupliqués dans chacune d'elles, afin de rendre chaque *nSP-cellule* indépendante — tous les simplexes contenus entièrement ou partiellement dans celle-ci y sont encodés. La figure 3.3 présente un exemple de construction d'un tel arbre, dans le cas 2D, avec $N_{min} = 10$. On peut y lire le déroulement suivant : les simplexes du maillage original, dont la précision est alors maximale, sont insérés un à un dans le *SP-arbre* : l'algorithme les fait « descendre » dans l'arbre. Dès qu'une feuille de

ce dernier atteint N_{min} éléments, ses enfants sont créés et les éléments y sont répartis.

A ce stade, chaque feuille du *SP-arbre* contient un ensemble de simplexes issus du maillage original. Afin d'assurer le traitement de maillages de toute taille, un mécanisme de mémoire virtuelle a été développé pour éviter de saturer la mémoire centrale. Le *SP-arbre* est parcouru en ordre postfixe (une cellule est traitée après que tous ses fils ont été traités) afin de réaliser la compression — autrement dit, le calcul des codes de géométrie et de connectivité. Ainsi, dans chaque *nSP-cellule*, un complexe simplicial et le *kd-arbre* associé sont construits à partir de la liste de simplexes. Puis les *kd-cellules* sont progressivement fusionnées comme dans la méthode [GD02] (cf. section 3.2), et la séquence de codes correspondante est construite, où les codes de géométrie et de connectivité sont entrelacés, et écrite dans un fichier.

Alors que dans la méthode [GD02], ce processus de fusion itérait depuis les feuilles du *kd-arbre* jusqu'à la *kd-cellule* racine, ici, les fusions à l'intérieur d'une *nSP-cellule* s'arrêtent lorsque la précision minimale de la *nSP-cellule* est atteinte. Dans l'exemple de la figure 3.4, cela correspond à 1 fusion des *kd-cellules* pour chaque dimension pour une *nSP-cellule* non racine — on passe ainsi d'un précision p à une précision $p-1$ sur chaque coordonnée —, et à 3 fusions dans chaque dimension pour la *nSP-cellule* racine — on passe de 3 bits par coordonnée à 0. Quand tous les enfants d'une *nSP-cellule* s ont été traités, les sous-maillages qu'ils contiennent sont fusionnés (les simplexes dupliqués disparaissent) pour former le sous-maillage associé à s, qui est alors prêt à être traité. Ce processus est effectué jusqu'à ce qu'un seul sommet demeure dans la *nSP-cellule* racine.

Une fois l'encodage terminé, nous disposons d'un fichier comprimé adapté à la décompression progressive et sélective.

Le processus de raffinement s'effectue naturellement dans le sens inverse. La figure 3.4 illustre son parcours : les étapes 1 à 6 sont similaires à [GD02] : nous commençons avec une précision nulle et un seul point au centre. Puis les codes de géométrie et de connectivité sont lus séquentiellement dans le fichier comprimé, ce qui raffine progressivement l'ensemble du maillage. Dans cet exemple, quand la précision atteint 3 bits (en pratique, on attend généralement 7 ou 8 bits), l'espace est divisé en 4 *nSP-cellules* (étape 7), et le maillage est subdivisé dans ces sous-espaces (étape 8). On note que les triangles situés sur la frontière et les *kd-cellules* associées ont été dupliqués durant la phase de compression. Les 4 sous-maillages obtenus deviennent indépendants, chacun correspondant à une *nSP-cellule* dont le contenu est directement accessible *via* un indice pointant dans le fichier comprimé. Ainsi, nous pouvons sélectionner une ou plusieurs *nSP-cellules* et lire dans le fichier comprimé les codes de raffinement qui y sont associés, afin de poursuivre localement le processus de raffinement, en fonction de la position de la caméra (étape 9 : ici, seule la *nSP-*

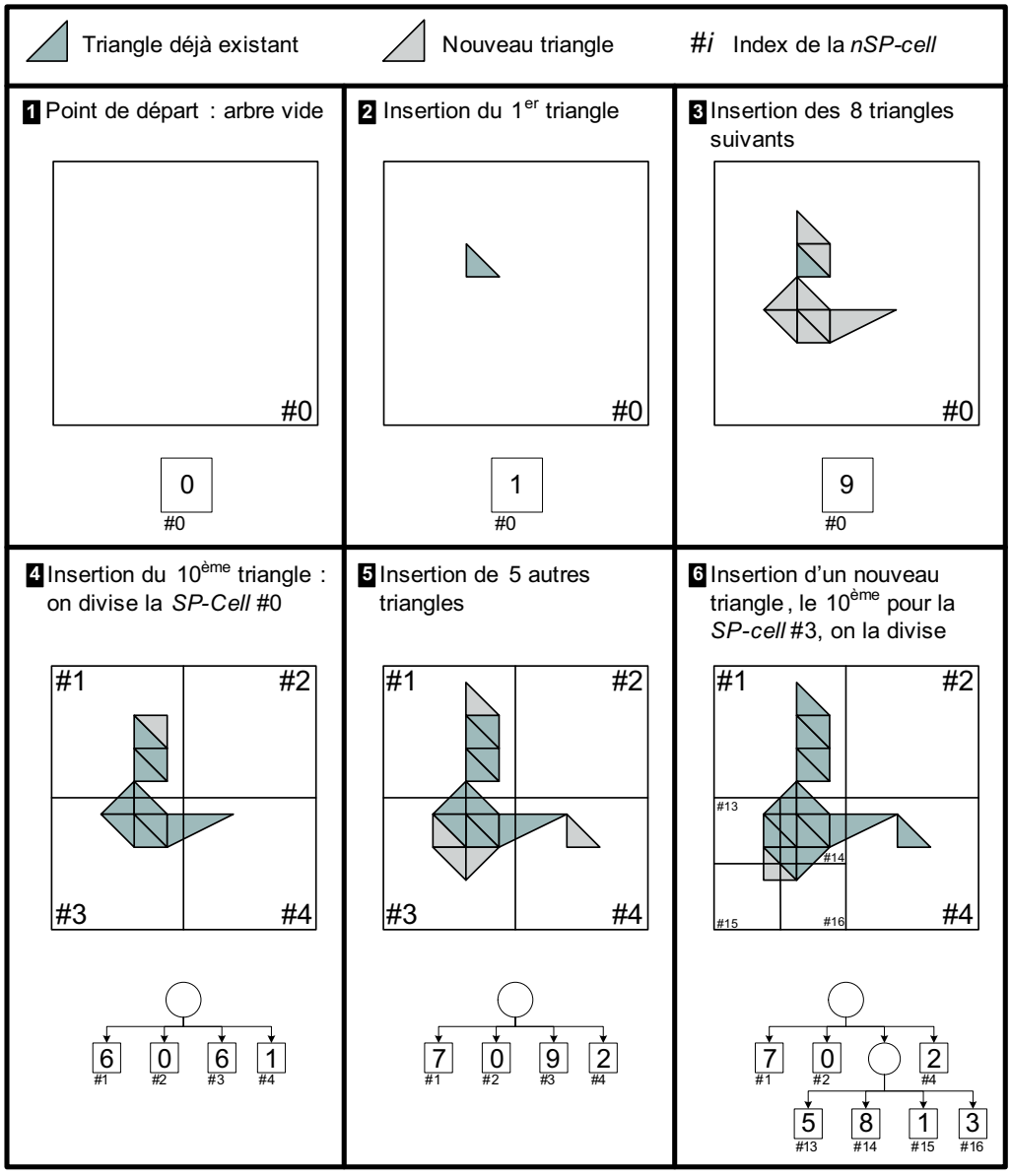


FIGURE 3.3 – Exemple de construction du *SP-arbre*, dans le cas bidimensionnel, avec $N_{min} = 10$

cellule #4 se trouve dans le champ de la caméra). Quand une des *nSP-cellules* atteint sa précision maximale (4 bits dans notre exemple), elle est divisée à nouveau (étape 10). Ce processus est répété récursivement jusqu'à ce que chaque *nSP-cellule* dans le cône de vision ait atteint un niveau de précision déterminé par sa distance à la caméra. Pour finir, on connecte les *nSP-cellules* ensemble, en gérant les différences de résolution éventuelles pour éviter que les simplexes de frontière ne se chevauchent, et les sous-maillages de ces *nSP-cellules* est envoyé au *pipeline* graphique.

3.4 Construction out-of-core

La phase de construction est la première étape de notre algorithme. Elle correspond au partitionnement de l'espace en fonction du maillage original, et conduit à la création du *SP-arbre*. Avant de décrire en détail le processus de construction, quelques définitions sont introduites ici :

Définition 15. *Un n**SP-arbre** est un arbre de partitionnement de l'espace avec n subdivisions par axe. Chaque nœud interne délimite un sous-espace cubique appelé nSP-cellule. Puisque nos maillages sont plongés dans un espace de dimension d, une nSP-cellule interne possède n^d enfants — quelle que soit sa position dans l'arbre —, et dans notre cas, les nSP-cellules sont subdivisées en parts égales, i.e. les n^d enfants sont de même taille. La structure et l'utilisation de cet arbre sont détaillées dans la section 3.4.2.*

Définition 16. *Chaque nSP-cellule c contient un sous-maillage dont la précision des sommets peut varier, lors de la visualisation, en fonction de la distance entre c et la caméra. Les bornes de la précision des sommets dans c sont appelées **précisions minimale** et **maximale** de c.*

3.4.1 Quantification des coordonnées des points

Tout d'abord, les points contenus dans le fichier d'entrée sont parcourus. Une boîte englobante cubique est extraite et les points sont quantifiés afin d'obtenir des coordonnées entières relatives à la boîte, en accord avec le niveau de précision p (nombre de bits par coordonnée) demandé par l'utilisateur. Les points sont écrits dans un fichier brut (RWP pour ***R**aW **P**oints*) avec un encodage binaire standard utilisant des codes de taille constante (p bits pour chaque coordonnée) pour garantir un accès direct dans le fichier : pour accéder au i^e sommet, il suffit de se déplacer de $3 \times p \times (i-1)$ bits dans le fichier RWP.

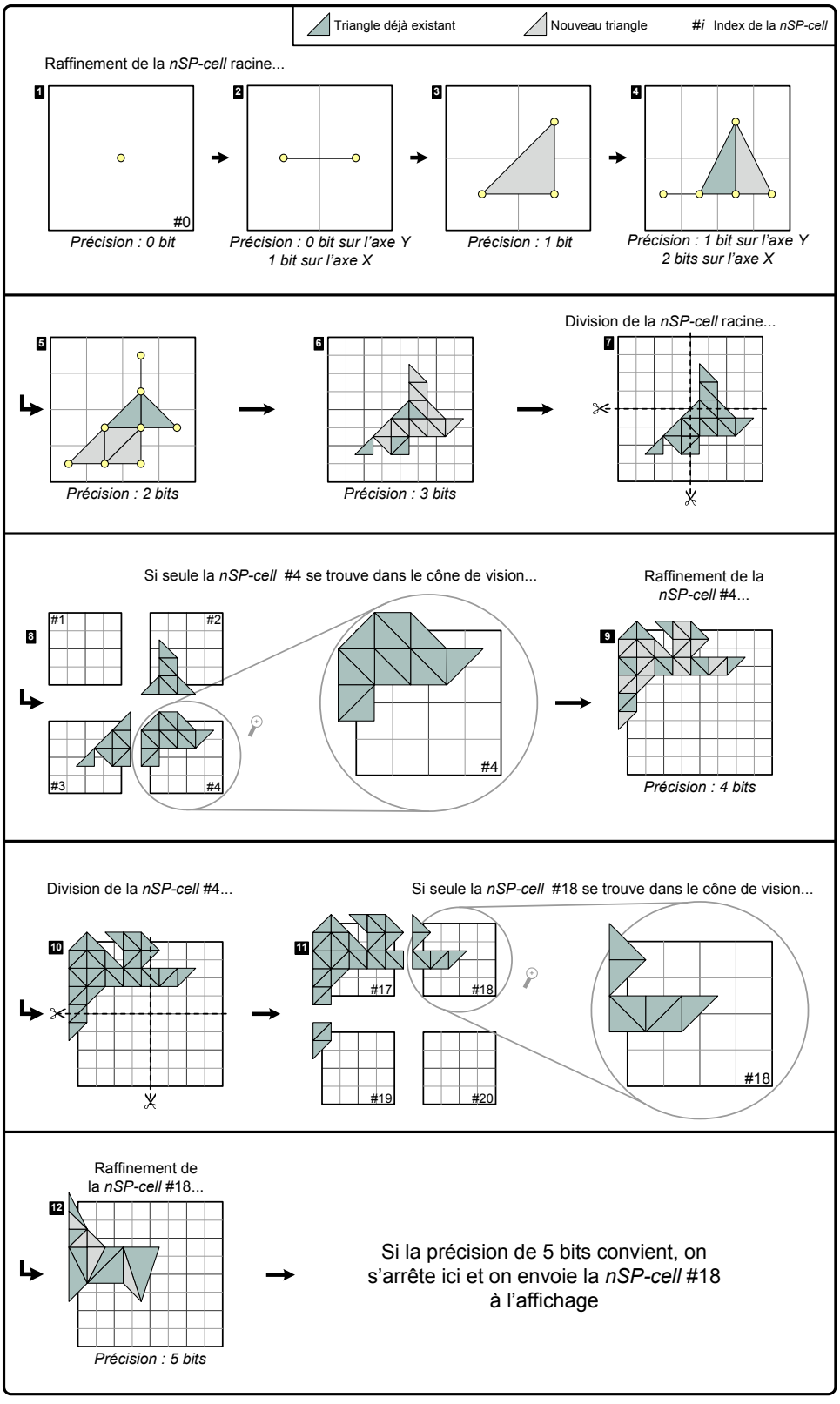


FIGURE 3.4 – Vue d'ensemble de la méthode

3.4.2 Construction du *nSP-arbre*

Comme dans un grand nombre de méthodes traitant d'objets volumineux, l'espace est partitionné à l'aide d'un arbre pour plusieurs raisons :

- permettre un parcours et un accès facilité aux différentes zones du maillage, sur la base de leur localisation dans l'espace,
- permettre une sélection rapide des informations pertinentes et une élimination facile des autres données,
- pouvoir utiliser la structure intrinsèquement hiérarchique des arbres pour gérer les niveaux de détail,
- bénéficier des nombreux algorithmes de parcours, d'insertion et de suppression associés aux arbres, qui sont à la fois simples et efficaces.

Un exemple d'arbre de partitionnement de l'espace est donné à la figure 3.5. Ici, chaque niveau de l'arbre voit la division de toutes les cellules en 4 cellules filles.

L'arbre que nous utilisons, appelé *nSP-arbre*, présente plusieurs particularités.

Tout d'abord, il est caractérisé par un paramètre n. Il s'agit du nombre de subdivision par axe effectuées lors du passage d'un niveau à l'autre de l'arbre. Dans l'exemple de la figure 3.5, $n = 2$. Dans le cas $n = 4$, la division d'une cellule occasionne la création de 16 cellules filles. Si on généralise, avec d la dimension de l'espace, le nombre de cellules filles générées par la division d'une cellule d'un *nSP-arbre* vaut n^d.

De plus, dans notre cas, chaque niveau de l'arbre se voit associer une plage de précision :

- La cellule racine se voit attribuer une plage de p_r bits par coordonnée, choisie par l'utilisateur. p_r est donc la précision que le maillage atteindra avant d'être subdivisé la première fois. En d'autres termes, tant que la visualisation ne nécessitera pas une précision supérieure à p_r bits par coordonnées, le maillage ne sera pas divisé et l'algorithme de décompression sera très proche de celui de [GD02].
- Les cellules des niveaux suivants de l'arbre se voient attribuer automatiquement une précision supplémentaire p_l, qui est calculée à partir de n :

  $$p_l = log_2\ n$$

  Ce calcul de la valeur de p_l s'explique par la nécessité d'avoir une relation linéaire entre le facteur de zoom de la caméra et la division des cellules. Par exemple, pour $n = 2$, si l'on part du principe que p_r correspond à la précision requise pour afficher correctement la cellule racine lorsqu'elle occupe toute l'image, alors la précision maximale des cellules du premier niveau de l'arbre doit valoir $p_r + 1$, car pour que l'une d'elle occupe tout l'écran, il faut multiplier le facteur de zoom par 2, et donc ajouter

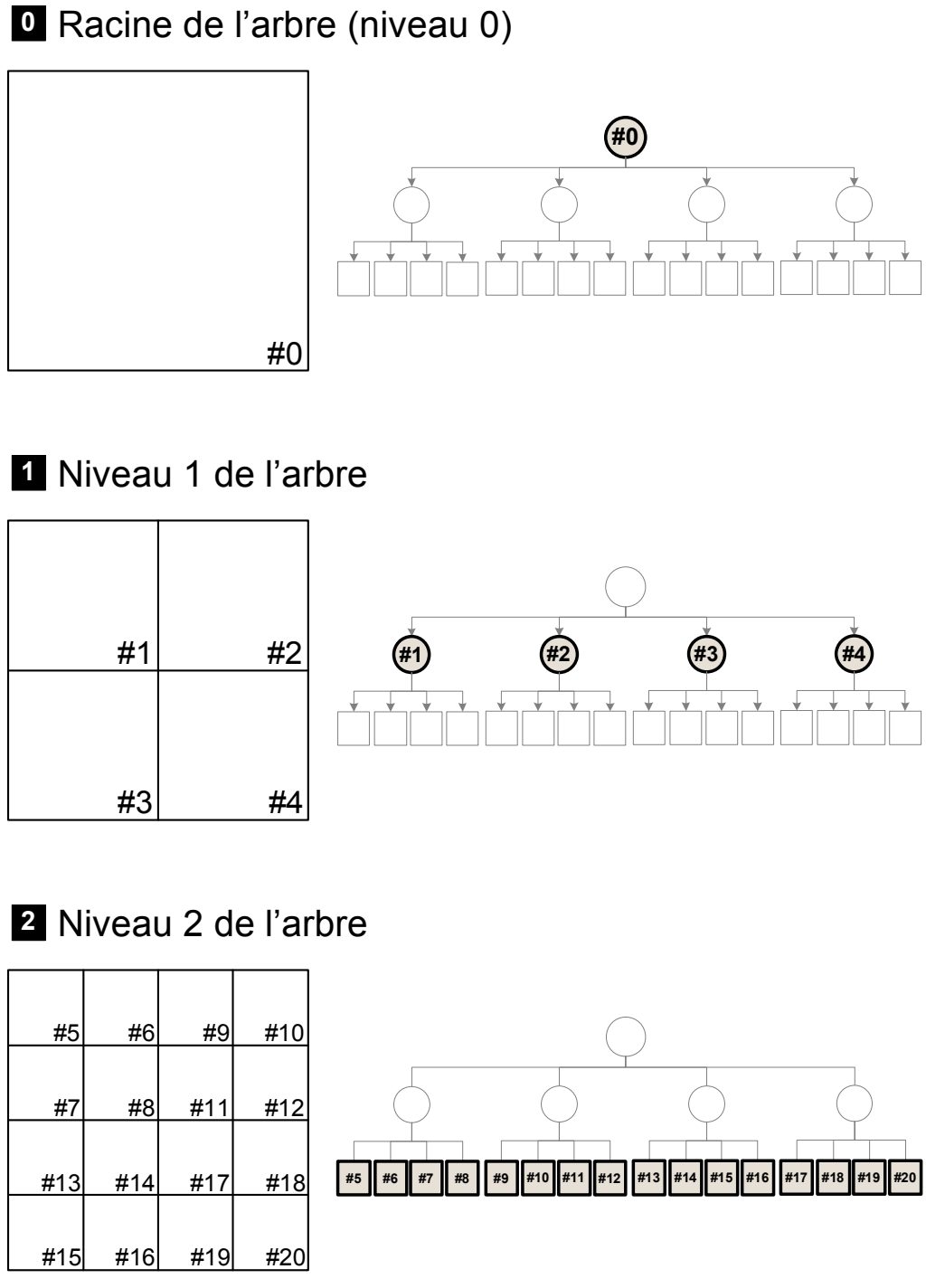


FIGURE 3.5 – Exemple d'arbre de partitionnement de l'espace à 3 niveaux, dans le cas bidimensionnel, avec 1 division selon chaque axe entre chaque niveau (la division d'une cellule génère 4 cellules filles)

$log_2\ 2 = 1$ bit de précision. De même, pour $n = 4$, il faut multiplier le facteur de zoom par 4 pour qu'une cellule du second niveau occupe toute l'image, donc multiplier la précision par 4 , *i.e.* donner $log_2\ 4 = 2$ bits supplémentaires.

Ainsi, la hauteur maximale du *nSP-arbre* dépend de p_r, de p_l — *i.e.* de n — et de la précision p totale choisie pour le maillage, *i.e.* le nombre de bits par coordonnée. Par exemple, pour $p = 12$ bits, $p_r = 6$ bits et $n = 4$, la hauteur du *nSP-arbre* est 4.

Pour finir, une dernière condition est appliquée : une *nSP-cellule* ne peut être divisée que si le nombre de simplexes maximaux qu'elle contient est au moins égal au seuil N_{min}. Ainsi, le *nSP-arbre* prend en compte la densité locale du maillage : les régions éparses sont très peu subdivisées, tandis que les zones denses sont fortement divisées. Ce principe simple combine adaptabilité et faible coût de codage. Conséquence pratique de cette dernière condition, la précision minimale des **feuilles** de l'arbre n'a pas de valeur fixée, tandis que leur précision maximale vaut toujours p.

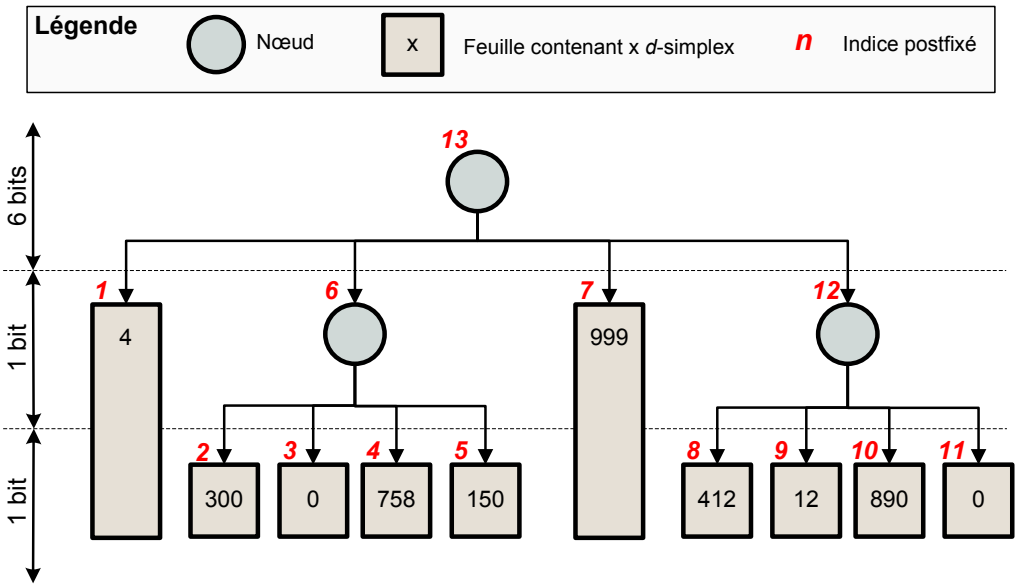


FIGURE 3.6 – Exemple 2D d'un *nSP-arbre* avec $p = 8$ bits, $p_r = 6$ bits, $n = 2$ et $N_{min} = 1000$

Une fois le *nSP-arbre* construit, chaque feuille contient l'ensemble des simplexes maximaux situés dans son sous-espace correspondant. La figure 3.6 montre un exemple 2D d'un tel *nSP-arbre*. Ici, la précision totale demandée est de 8 bits, la précision p_r de la racine est de 6 bits, et on calcule $p_l = log_2\ 2 = 1$, donc chaque niveau suivant du *nSP-arbre* contient les codes pour raffiner son sous-maillage de 1 bit supplémentaire. La figure 3.3, quant à elle, présente le déroulement de la construction d'un *nSP-arbre*.

Le sous-maillage des feuilles est stocké dans un fichier d'indices (RWI, pour ***RaW I**ndex*) où les $d+1$ indices qui composent chaque d-simplexe sont écrits dans un format binaire brut pour réduire l'empreinte mémoire sur disque tout en permettant une lecture rapide.

3.4.3 Gestion des fichiers RWP et RWI

Le processus de construction produit en sortie deux fichiers :

- Le fichier RWP, pour ***RaW P**oints*, qui contient la liste des sommets quantifiés du maillage, dans l'ordre du fichier original. Il s'agit d'une suite de d-uplets d'entiers, chacun stocké sur p bits.
- Le fichier RWI, pour ***RaW I**ndex*, qui stocke les listes des simplexes maximaux (n-uplets d'indices de sommets) contenus dans les *nSP-cellules*.

Fichier RWP

Même si les points et simplexes ne sont a priori pas ordonnés selon une quelconque logique dans le fichier en entrée, il est très fréquent que l'ordre dans lequel ils se trouvent ait une cohérence, généralement spatiale — les sommets et simplexes géométriquement proches se retrouvent très souvent encodés dans le fichier à une faible distance l'un de l'autre. Il est donc probable que les différents accès à un même point du fichier RWP soient proches dans le temps. Nous avons donc mis en place un « cache » stocké en mémoire vive, basé sur une *hash table*, afin de réduire le nombre d'accès au disque lors de la lecture des coordonnées des sommets. Ainsi, les points accédés récemment ont une forte probabilité de se trouver dans le cache, et la *hash table* (dont la *hash key* est un modulo de l'indice du point) garantit un accès en temps constant aux données mises en cache.

Fichier RWI

Puisque le nombre de simplexes maximaux contenus dans une *nSP-cellule* n'est pas connu à l'avance, l'emplacement pour son contenu ne peut pas être réservé dans le fichier RWI. Par conséquent, l'algorithme utilise une gestion basée sur des listes de blocs : le fichier RWI est divisé en blocs de taille constante (*e.g.* contenant 100 simplexes maximaux chacun), et le contenu d'une *nSP-cellule* est stocké dans une liste chaînée de blocs. Comme indiqué sur la figure 3.7, chaque *nSP-cellule* garde en mémoire vive trois informations :

1. la position de son premier bloc dans le fichier
2. la position courante dans le fichier (où on va écrire le prochain simplexe maximal)

3. le nombre de simplexes maximaux dans le bloc courant (afin de savoir quand on doit allouer un nouveau bloc)

Remarque 6. *Sur la figure 3.7, un bloc contient une liste de* 100 *triangles ; si l'on souhaite traiter les maillages composés d'éléments de dimension variable (par exemple des arêtes et des triangles), il est nécessaire d'ajouter la dimension des éléments dans le fichier RWI.*

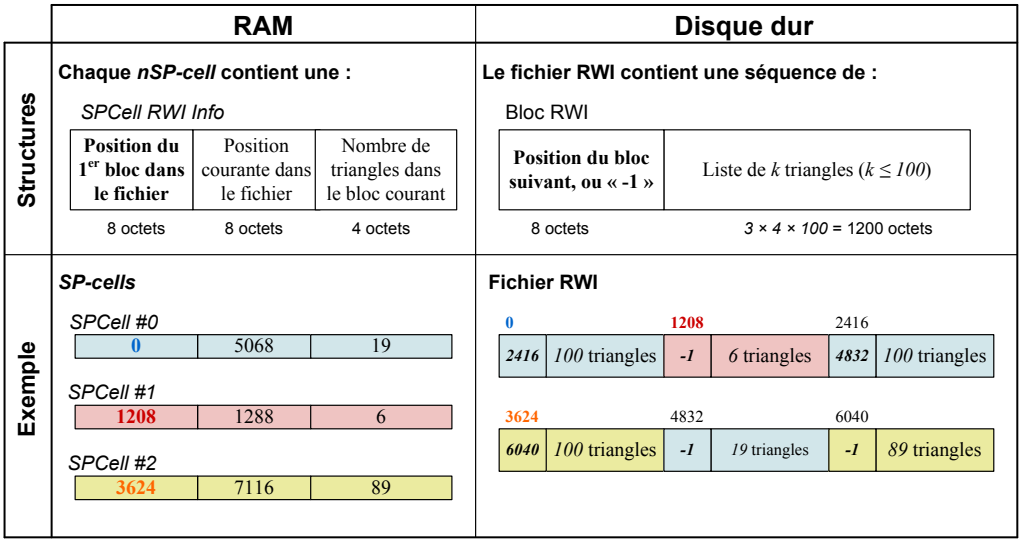


FIGURE 3.7 – Structure du fichier RWI (100 simplexes maximaux par bloc)

Chaque bloc dans le fichier RWI est composé de la position dans le fichier du prochain bloc de la liste chaînée (ou -1 si c'est le dernier bloc) suivi de la liste des simplexes maximaux (indices pointant un sommet dans le fichier RWP). Chaque simplexe maximal lu dans le fichier d'entrée est inséré dans le *nSP-arbre*. Plutôt que d'écrire dans le fichier RWI à chaque insertion d'un nouveau simplexe maximal dans le *nSP-arbre*, chaque *nSP-cellule* maintient un tampon contenant les derniers simplexes maximaux qui y ont été insérés. Lorsque la taille cumulée de ces tampons dépasse une valeur maximale B_{max}, les tampons sont écrits dans le fichier RWI, en commençant par le tampon le plus gros et ainsi de suite, jusqu'à ce que leur taille cumulée soit ramenée en dessous de $B_{max}/2$. Ainsi, l'écriture séquentielle sur disque est favorisée. Lorsque le nombre de simplexes maximaux contenus dans une *nSP-cellule* c atteint N_{min}, c est subdivisée, et ses simplexes maximaux — présents dans le tampon de c ou déjà écrits dans le fichier RWI — sont répartis dans les nouvelles *nSP-cellules* créées. Les blocs du fichier RWI ainsi libérés sont ajoutés dans une liste de blocs « libres » afin d'être réutilisés pour les futures allocations de bloc.

3.4.4 Choix de n

Afin d'obtenir un entier pour p_l, n doit être une puissance de 2. Cela garantit en outre que les *kd-cellules* ne chevauchent pas deux *nSP-cellules*, et par conséquent que tout descendant d'une *kd-cellule* contenue dans une *nSP-cellule* c reste à l'intérieur de c.

3.4.5 Duplication de simplexes

Une première façon de déterminer si un simplexe appartient à une *nSP-cellule* c consiste à vérifier si au moins un de ses sommets incidents se trouve à l'intérieur de c. Ce test est rapide et adapté à la plupart des cas, mais est imparfait car une cellule peut être traversée par un simplexe sans contenir aucun de ses sommets. Ce cas de figure n'a lieu que rarement et la probabilité qu'il soit visuellement perçu est très faible. Il peut cependant constituer un problème pour certaines applications spécifiques, auquel cas un test d'intersection plus précis — simplexe *versus axis-aligned box* — peut être effectué. La figure 3.8 donne un exemple des résultats obtenus par les deux méthodes : un triangle traverse la cellule supérieure gauche sans qu'aucun de ses sommets n'y soit contenu.

Quel que soit le test choisi, un simplexe du modèle original peut simultanément appartenir à plusieurs *nSP-cellules*. Par conséquent, certains simplexes et leurs *kd-cellules* associées sont dupliqués durant la distribution dans le *nSP-arbre*, afin d'éviter que le rendu ne présente des trous : il est indispensable qu'une *nSP-cellule* soit indépendante et contienne tous les simplexes qui la traverse, pour permettre l'élimination rapide de toutes les *nSP-cellules* non contenues dans le cône de vision lors du rendu. Les dessins 7—8 et 10—11 de la figure 3.4 montrent le partitionnement spatial d'une *nSP-cellule*. Le nombre de duplications peut paraître rédhibitoire sur ces schémas, mais la taille des simplexes par rapport à leur *nSP-cellule* y est volontairement exagérée par souci de clarté. En réalité, une très faible proportion de simplexes est dupliquée. Dans le cas des maillages triangulaires, si l'on cumule le nombre de triangles créés par duplication sur chaque niveau de l'arbre, cela représente de 5 à 7% du nombre total de triangles. Bien que cette technique réduise légèrement les taux de compression, elle a été retenue pour des raisons d'efficacité.

3.5 Compression out-of-core

Après cette étape de construction, nous disposons des informations suivantes :

- en mémoire vive : un *nSP-arbre* dont les feuilles contiennent chacune une

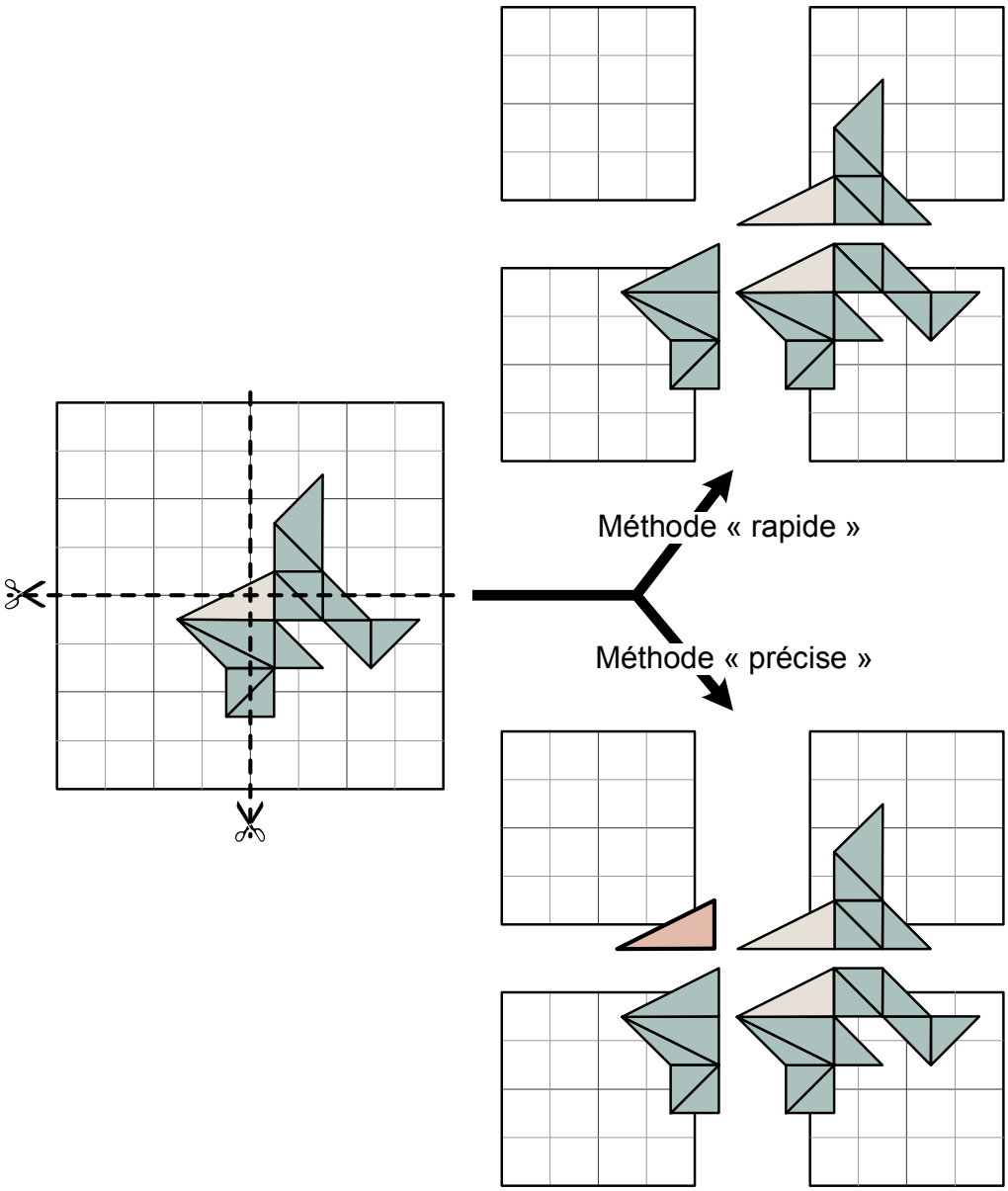


FIGURE 3.8 – Comparaison des deux méthodes possibles de duplication de simplexes

structure appelée « SPCell RWI Info » (cf. figure 3.7)
- sur disque dur :
  - un fichier RWP contenant les points dont les coordonnées sont quantifiées
  - un fichier RWI contenant les simplexes maximaux (n-uplets d'indices) localisés dans les feuilles du *nSP-arbre* (cf. figure 3.7)

Le maillage est donc divisé en plusieurs sous-maillages indépendants, en concordance avec le *nSP-arbre*. Dans cette section, nous détaillons les étapes successives du processus de compression.

3.5.1 Une compression en deux passes

Afin d'atteindre des taux de compression compétitifs, la méthode utilise un encodeur arithmétique statique (cf. 1.2.2). Pour s'approcher de la taille du code minimal théorique, il est nécessaire de recueillir des données statistiques sur l'ensemble du fichier. Données qu'il est impossible de connaître avant d'avoir terminé les calculs complets de compression. Le caractère *out-of-core* de notre algorithme interdit de stocker en mémoire vive l'ensemble des résultats de ces calculs : les codes de géométrie et de connectivité doivent être régulièrement écrits sur disque, à un moment où les données statistiques ne sont que partiellement disponibles.

C'est pourquoi notre méthode requiert un encodage en deux passes :
- La première passe génère un fichier temporaire semi-comprimé (dénommé « fichier SCT », pour *Semi-Compressed Temporary*) dont la compression n'est pas optimale.
- La seconde passe crée le fichier final utilisant un codage entropique (dénommé « fichier FEC », pour *Final Entropy-Coded*) dont la compression est effectuée de façon optimale grâce aux statistiques récoltées durant la première passe.

3.5.2 Ecriture de l'en-tête du fichier FEC

Certains paramètres caractérisant le modèle original et le *nSP-arbre* peuvent être écrit immédiatement, sans attendre que la collecte des données statistiques nécessaires à la seconde passe de compression soit terminée. L'en-tête du fichier contient toutes les données générales requises par le décodeur : le système de coordonnées (origine et résolution de la grille afin de reconstituer les positions originales des sommets), la dimension d du maillage, le nombre p de bits par coordonnée, le nombre n de subdivisions par axe pour le partitionnement de l'espace, ainsi que la précision p_r de la *nSP-cellule* racine.

3.5.3 Traitement de chaque cellule du *nSP-arbre*

Il est enfin temps d'entrer dans le vif du sujet, à savoir la compression proprement dite.

Un parcours postfixe du *nSP-arbre* (nombres rouges sur la figure 3.6) est effectué et les opérations suivantes sont appliquées à chaque *nSP-cellule c* (l'ensemble du processus est illustré à la figure 3.9).

(a) Construction du *kd-arbre* et du complexe simplicial : Si c est une feuille du *nSP-arbre* original, les simplexes maximaux appartenant à c sont lus dans le fichier RWI et les sommets associés dans le fichier RWP ; sinon, les simplexes maximaux et les sommets contenus dans c sont déjà chargés en mémoire (cf. paragraphe (d) ci-après) ; puis dans tous les cas, un *kd-arbre* qui sépare les sommets (processus descendant) identique à [GD02] est construit. Enfin, le complexe simplicial dont les sommets sont les feuilles du *kd-arbre* est généré.

(b) Calcul des codes de géométrie et de connectivité : Les feuilles du *kd-arbre* sont fusionnées séquentiellement en suivant un processus ascendant analogue à celui de [GD02] (cf. section 3.2). Pour chaque fusion de deux feuilles, une séquence de codes combinant information de géométrie et de connectivité est calculée. Il est à noter que pour les besoins du rendu, cette séquence est enrichie par l'orientation des triangles qui disparaissent lors de la fusion. Il ne s'agit pas d'un encodage complet des normales, mais d'une description binaire de l'orientation de chaque triangle. Les normales aux sommets sont calculées en temps réel lors du rendu, par moyennage des normales aux triangles adjacents. Grâce à une prédiction basée sur l'orientation des triangles voisins, le surcoût dû à ce code est rendu négligeable (environ $0,01$ bits par sommet). Le processus de fusion est stoppé lorsque la précision minimale de c est atteinte.

Nous nous sommes inspiré des idées de la méthode originale de Gandoin et Devillers [GD02] pour calculer les codes de géométrie et de connectivité, avec quelques adaptations ou simplifications pour garantir un rendu rapide. Du côté de la géométrie, chaque subdivision de *kd-cellule* est encodée avec l'un des trois codes suivants :

- 0 : la première demi-cellule est non-vide
- 1 : la seconde demi-cellule est non-vide
- 2 : les deux demi-cellules sont non-vides

Le coût théorique de $log_2\ 3 = 1,58$ bits par division est réduit à une moyenne de $1,2$ bits grâce à l'utilisation de données statistiques par niveau pour le codage entropique — la partie haute du *kd-arbre* (*i.e.* les *kd-cellules* proches de la racine) est essentiellement composée de codes 2, tandis que les niveaux plus bas contiennent principalement des codes 0 et 1. La figure 3.10 montre un exemple d'encodage de la géométrie.

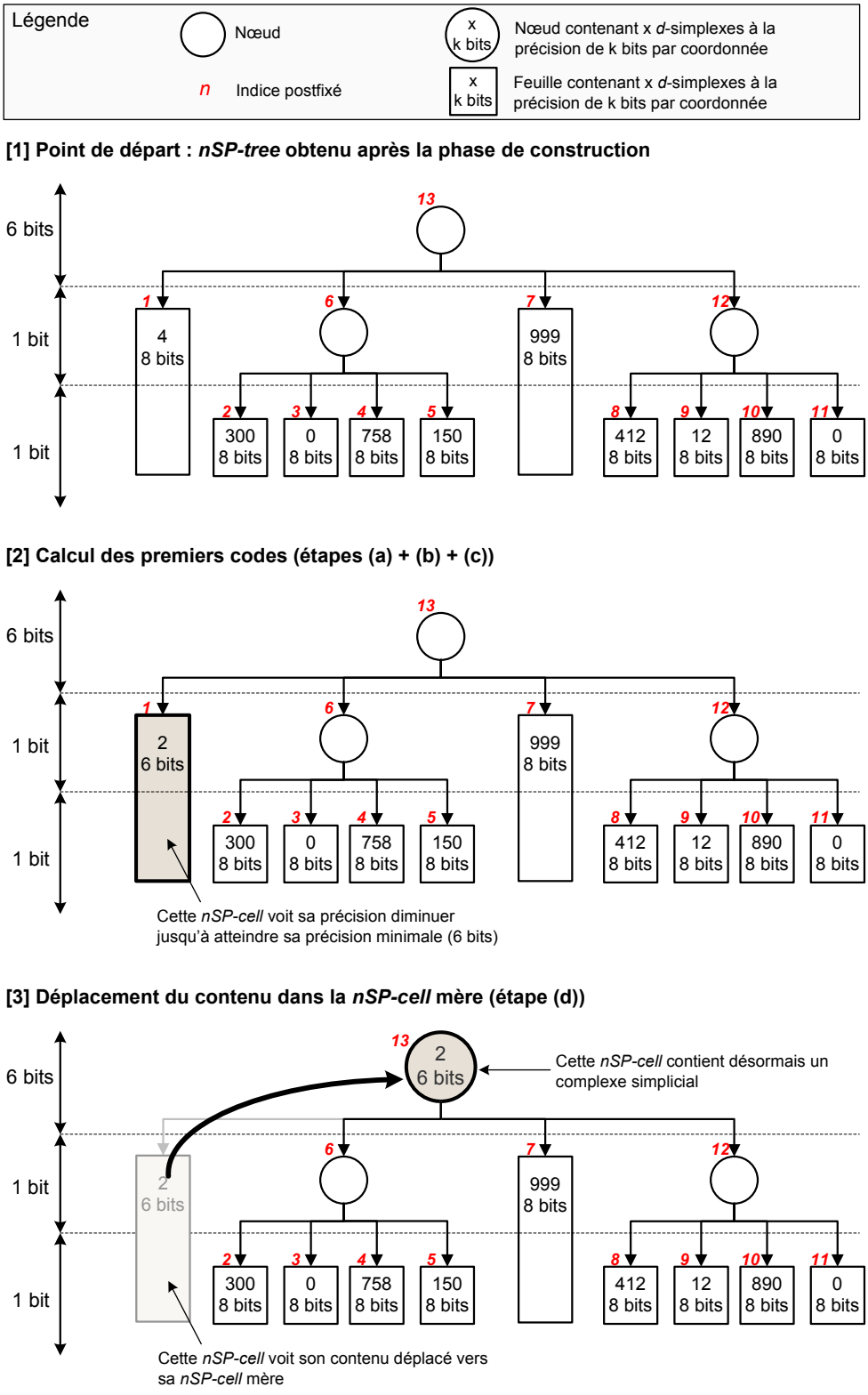


FIGURE 3.9 – Exemple de processus de compression (1/5)

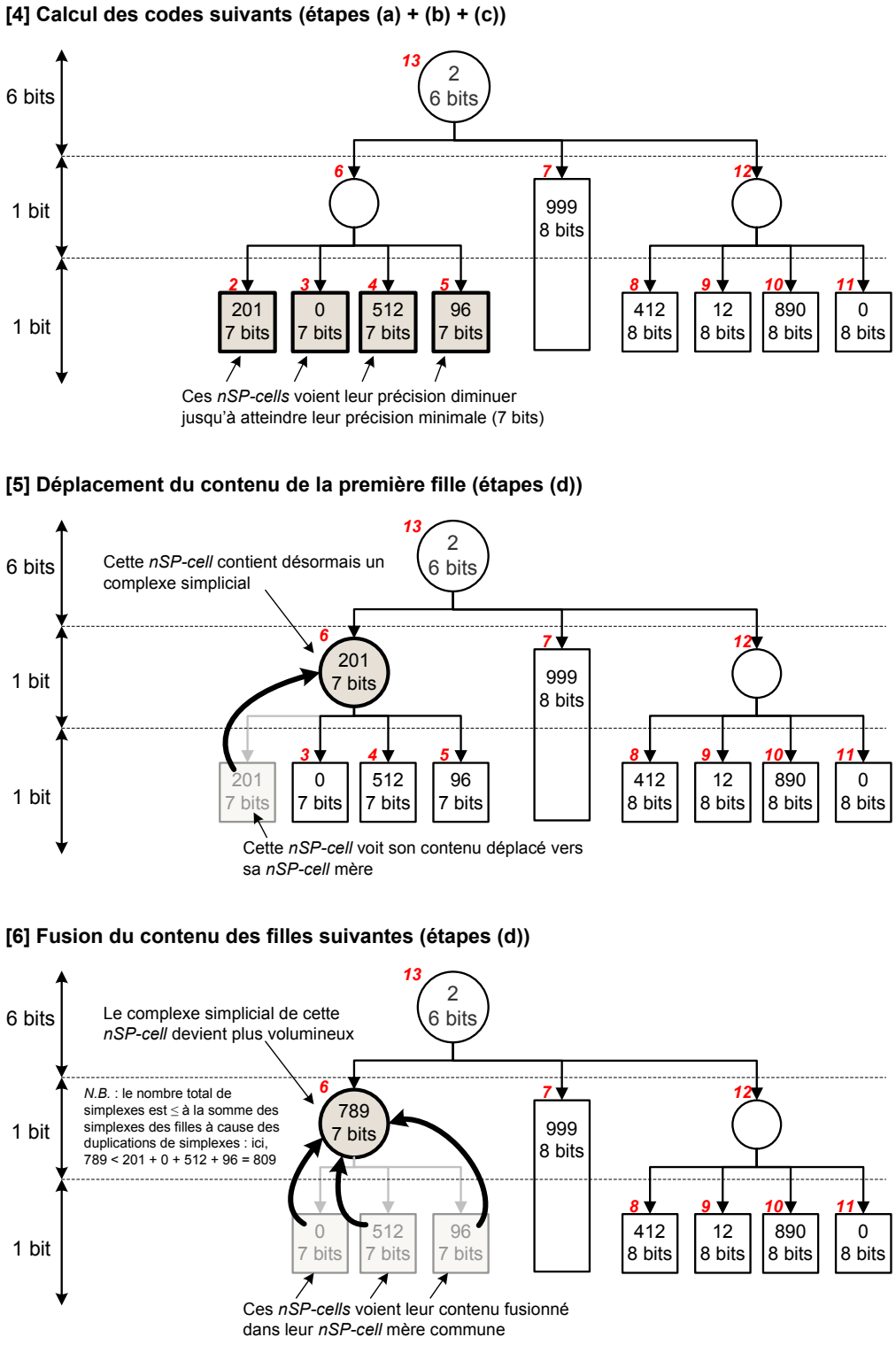


FIGURE 3.9 – Exemple de processus de compression (2/5)

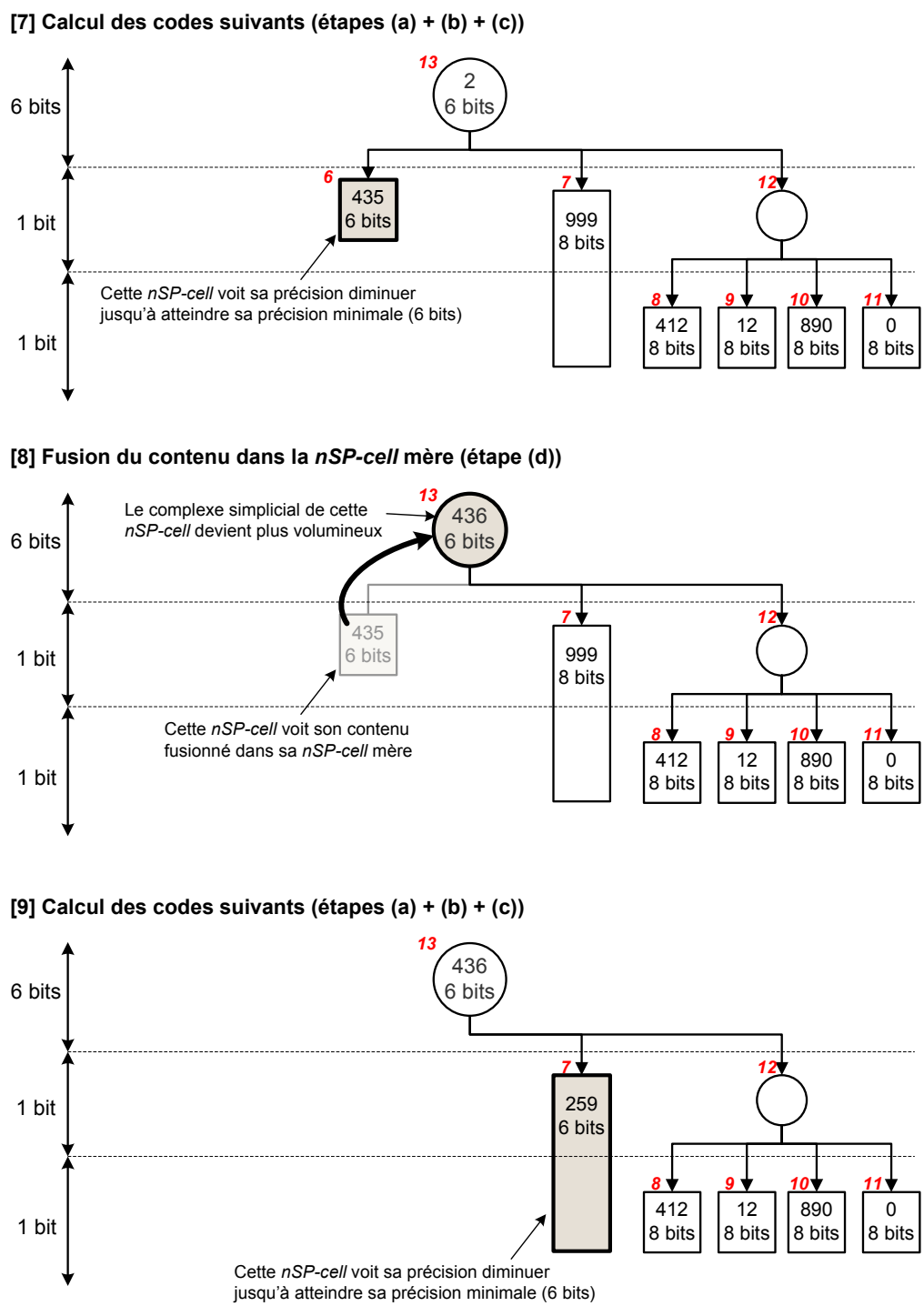


FIGURE 3.9 – Exemple de processus de compression (3/5)

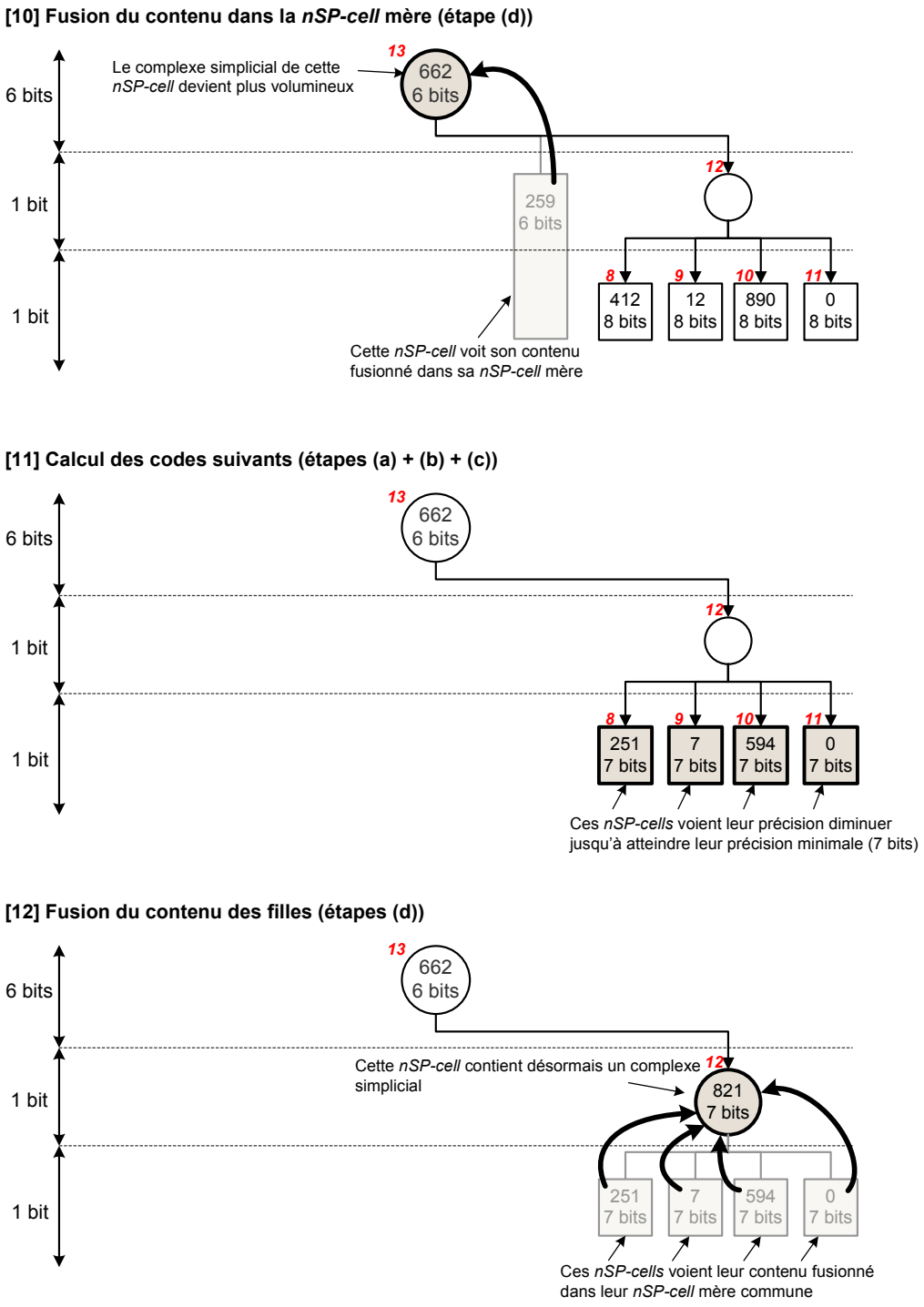
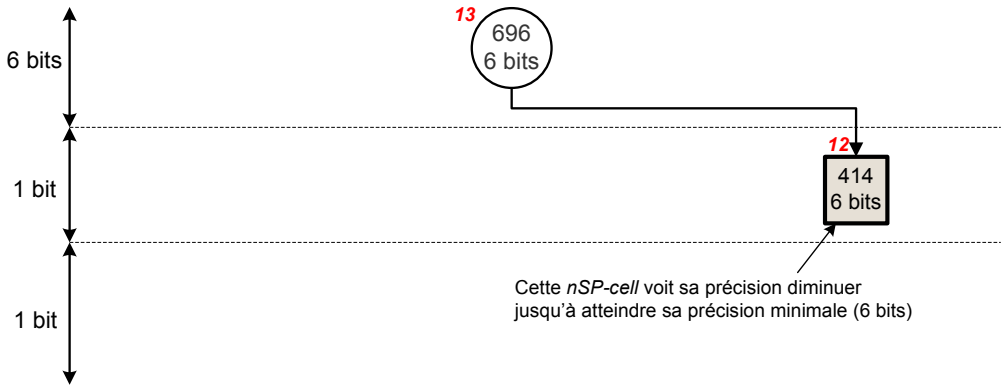
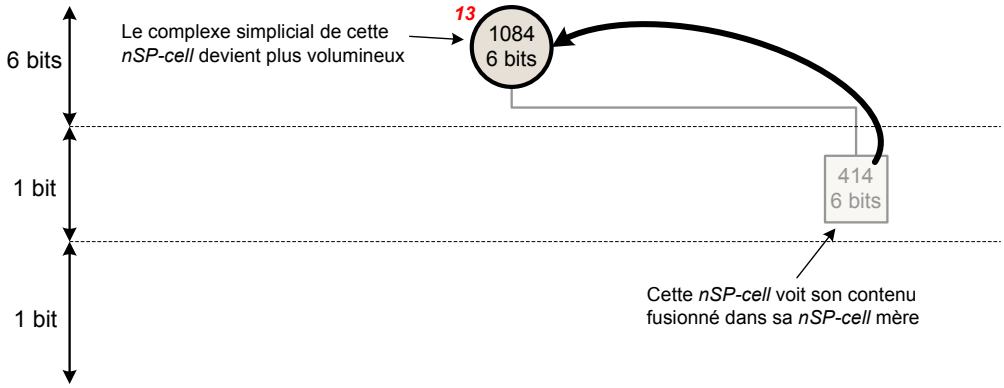


FIGURE 3.9 – Exemple de processus de compression (4/5)

[13] Calcul des codes suivants (étapes (a) + (b) + (c))



[14] Fusion du contenu dans la *nSP-cell* mère (étape (d))



[15] Calcul des codes de la *nSP-cell* racine (étapes (a) + (b) + (c))

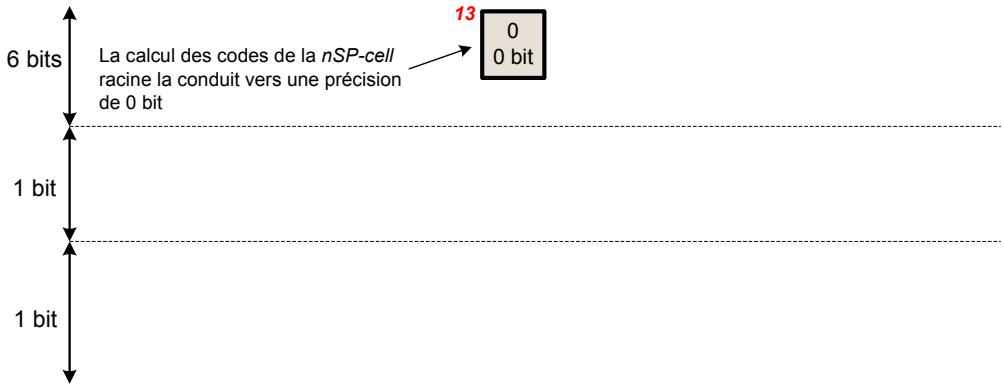
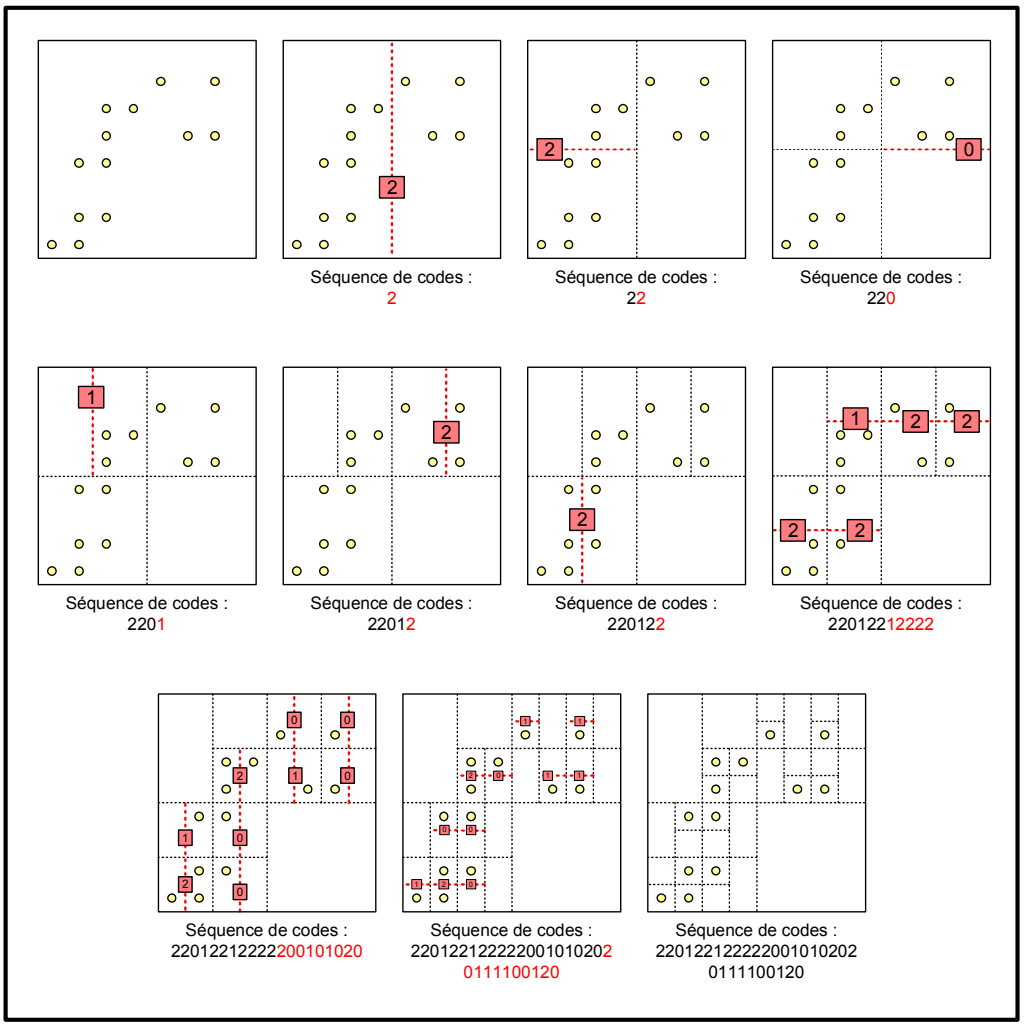


FIGURE 3.9 – Exemple de processus de compression (5/5)



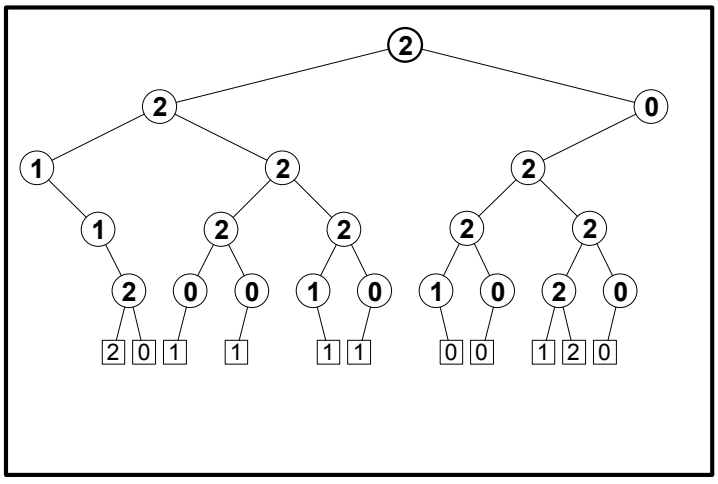


FIGURE 3.10 – Exemple de calcul des codes de géométrie

Concernant la connectivité, nous conservons les deux opérations : expansion d'arête et subdivision de sommet. A chaque fois qu'un code géométrique est de valeur 2 (les deux demi-cellules sont non-vides), une de ces deux opérations est utilisée pour encoder les modifications de connectivité qui en découle : disparition d'une arête ou d'un triangle, dégénération d'un triangle en une arête, etc. Leur encodage est similaire à celui de [GD02], hormis le fait que nous y ajoutons un code décrivant l'orientation de chaque triangle nouvellement créé. Dans le fichier, à la suite du code de géométrie (2 en l'occurrence), un premier symbole est inséré pour indiquer l'opérateur choisi, puis, selon l'opérateur, une suite de symboles est encodée.

Expansion/contraction d'arête L'expansion d'arête — dont l'opération inverse est la contraction d'arête —, définie par Hoppe *et al.* [HDD+93] et présentée à la figure 3.11, est employée lorsque les deux *kd-cellules* à fusionner c_1 et c_2 sont incidentes, et que le voisinage de l'arête reliant c_1 et c_2 est *manifold* et *orientable*. Lors de la contraction d'arête, les deux triangles adjacents à cette arête disparaissent par dégénérescence — ils dégénèrent en deux arêtes a_1 et a_2, déjà existantes. Deux symboles sont encodés, qui correspondent aux indices de ces deux arêtes dans l'étoile formée par le voisinage du sommet central (v_2 et v_6 sur la figure 3.11). S'y ajoutent deux autres symboles binaires, indiquant l'orientation des deux triangles.

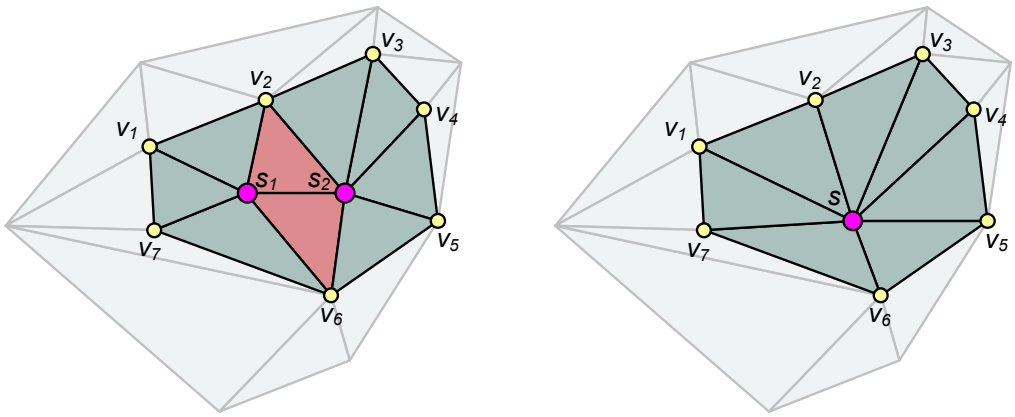


FIGURE 3.11 – Exemple de contraction d'arête

Subdivision/fusion de sommet La subdivision de sommet — dont l'opération inverse est la fusion de sommet —, définie par Popović et Hoppe [PH97] et présentée à la figure 3.12, est employée dans les cas où l'expansion d'arête est impossible : cellules non incidentes, ou voisinage de topologie complexe. Il s'agit d'un opérateur plus général, mais plus coûteux. Il permet d'encoder toutes les modifications qui ont lieu au sein d'un complexe simplicial lors d'une

fusion de deux sommets. Le principal général est de coder une à une les évolutions des simplexes (arêtes, triangles, etc.) entourant les sommets fusionnés. Lors d'une telle fusion de deux sommets s_1 et s_2, tout simplexe T incident à s_2 est remplacé par $T_b = (T \setminus s_2) \cup s_1$. Si T_b existe déjà, il n'est pas conservé. La séquence de codes associée aux modifications doit permettre au décodeur de raffiner le complexe simplicial lors de l'opération de subdivision de sommet. Pour cela, chaque simplexe T incident se voit attribuer un code indiquant son devenir dans le nouveau voisinage :

- code 1 : T est incident à s_1 uniquement
- code 2 : T est incident à s_2 uniquement
- code 3 : T est incident à s_1 et s_2 (un nouveau simplexe est créé)
- code 4 : T est incident à s_1 et s_2 (un nouveau simplexe est créé), et il y a création supplémentaire d'un simplexe de dimension $dim(T) + 1$ incident à s_1 et s_2.

Ces codes sont illustrés par la figure 3.13.

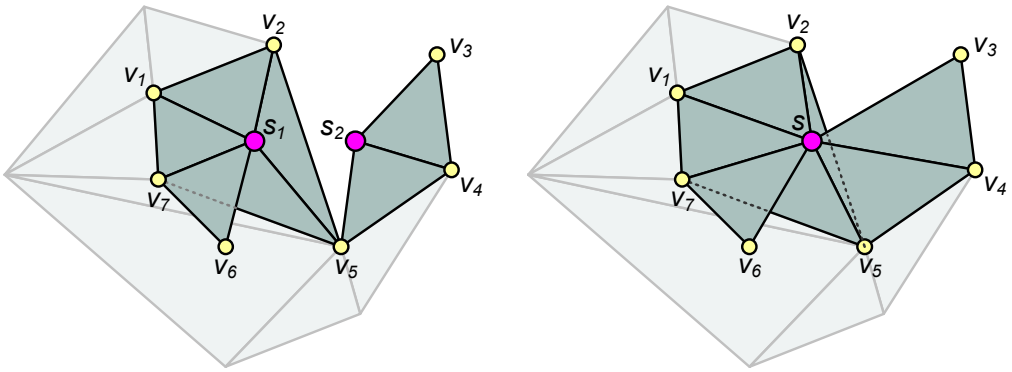


FIGURE 3.12 – Exemple de fusion de sommets

De plus, nous nous basons sur les règles suivantes pour réduire le coût de stockage :

- Si un simplexe T a un code $c \in \{1;2\}$, tous les simplexes adjacents à T de dimension $dim(T) + 1$ ont le code c.
- Si un simplexe T a le code 3, aucun des simplexes adjacents à T de dimension $dim(T) + 1$ n'a le code 4.

Les schémas de prédiction développés dans [GD02] ne sont pas utilisés, au profit de la vitesse de décodage et de l'efficacité de la visualisation. Cependant, rien ne s'oppose à leur inclusion dans le cas d'une application donnant la priorité à la compression ou dans un contexte où les maillages sont visualisés sur des stations de travail très puissantes.

(c) Ecriture des codes dans le fichier SCT : Les codes obtenus sont écrits dans un fichier temporaire semi-comprimé (SCT) à l'aide d'un codeur arithmétique

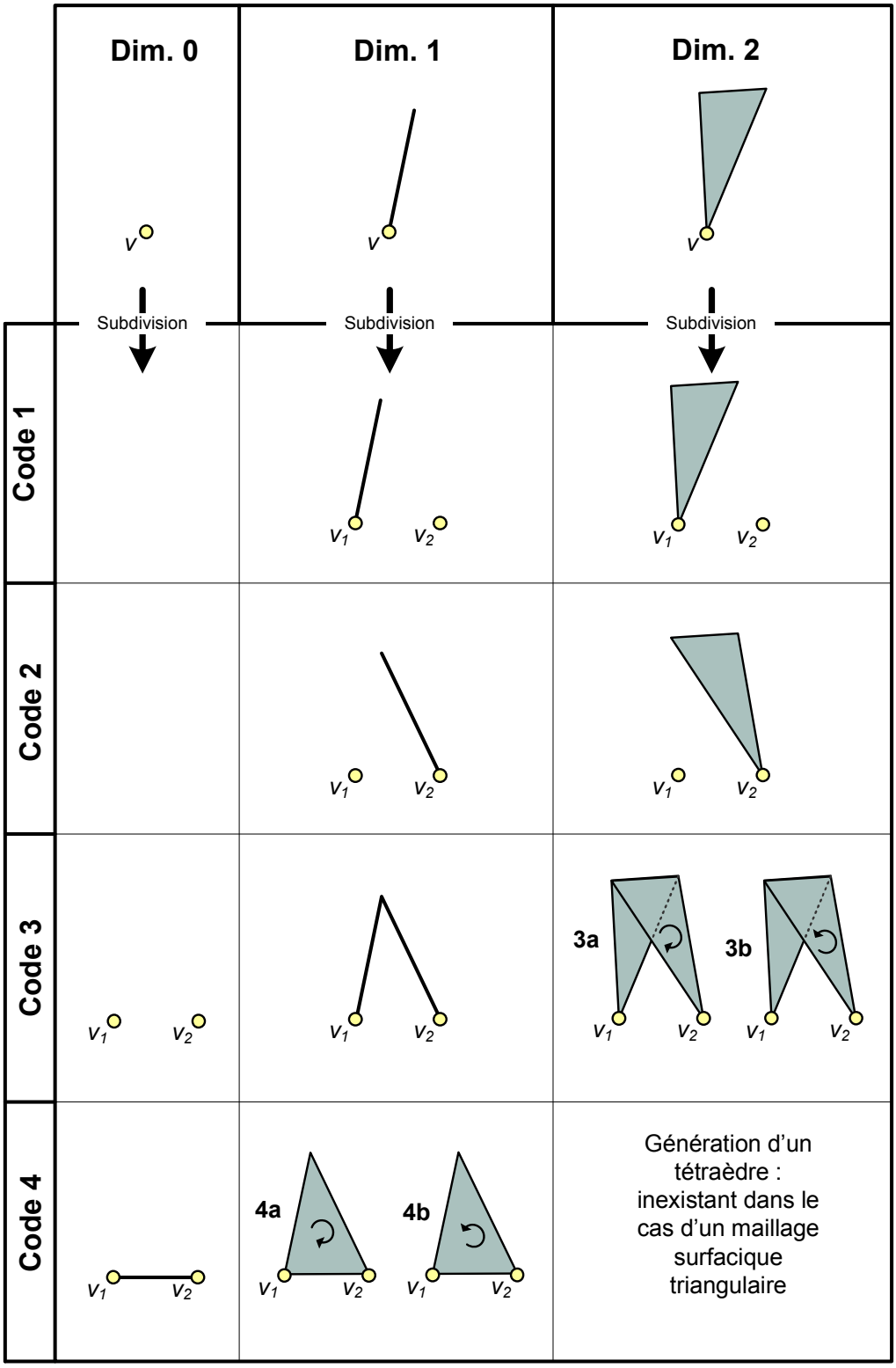


FIGURE 3.13 – Codes employés pour l'encodage de la fusion de sommets

[Amio4]. A ce stade, nous ne pouvons pas écrire dans le fichier final (FEC), les données statistiques n'étant pas encore disponibles.

(d) Fusion dans la *nSP-cellule* mère : Reste désormais un *kd-arbre* et un complexe simplicial dont les sommets ont la précision minimale de *c*. Afin de continuer la traversée du *nSP-arbre*, le contenu de *c* doit être déplacé dans son parent. Si *c* est le premier enfant, son contenu est simplement transféré vers le parent. Sinon, le *kd-arbre* et le complexe simplicial de *c* sont combinés avec le contenu actuel du parent, ce qui implique de détecter et de fusionner les simplexes dupliqués (voir section 4.1.4).

3.5.4 Ecriture finale

Une fois toutes les *nSP-cellules* traitées, nous disposons des données statistiques sur le flux complet et pouvons ainsi appliquer un codage entropique efficace. Les tables de probabilité sont tout d'abord écrites dans le fichier comprimé final (FEC), puis les séquences de codes des *nSP-cellules* du fichier SCT sont écrites à l'aide d'un codeur arithmétique. La fréquence des codes géométriques variant selon le niveau de détail, les probabilités sont calculées indépendamment pour chaque niveau. A la fin du fichier, une description de la structure du *nSP-arbre* qui indique la position de chaque *nSP-cellule* dans le fichier est ajoutée. Cette table permet à l'algorithme de décompression de reconstruire une version initiale du *nSP-arbre* dans laquelle les *nSP-cellules* ne stockent que la position de leur contenu dans le fichier FEC, ce qui fournit un accès direct aux séquences de codes de chaque *nSP-cellule*.

3.6 Décompression et visualisation

Le déroulement du processus de compression conduit à la création d'un fichier (d'extension .chv), qui contient toutes les données nécessaires à la **décompression** et à la **visualisation** du maillage. Ces deux dernières phases sont fortement interconnectées, puisque les données doivent être chargées sélectivement et décodées en fonction du contexte de visualisation. Dans cette section, chaque étape du processus de rendu est détaillée.

3.6.1 Initialisation

La première étape consiste à lire l'en-tête pour en extraire les informations générales. Pour mémoire, il s'agit du système de coordonnées (origine et résolution de la grille), de la dimension *d* du maillage, du nombre *p* de bits

par coordonnée, du nombre n de subdivisions par axe pour le partitionnement de l'espace, ainsi que de la précision p_r de la *nSP-cellule* racine. Puis la table située à la fin du fichier est lue et utilisée pour initialiser le *nSP-arbre* en mémoire vive. A ce stade, chaque *nSP-cellule* contient seulement la position de son contenu dans le fichier. Pour achever le processus d'initialisation, le *kd-arbre* de la *nSP-cellule* racine est créé dans sa forme la plus simple, *i.e.* une cellule racine, et le complexe simplicial associé est initialisé avec un unique sommet.

3.6.2 Raffinement adaptatif et interactif

A chaque pas de temps du rendu, le maillage est mis à jour pour s'adapter à la fenêtre de visualisation. Les *nSP-cellules* situées dans le cône de vision sont raffinées ou simplifiées de façon à ce que l'erreur maximale de tout sommet garantisse que sa projection satisfasse une précision d'affichage fixée (par exemple, 1 pixel). Les autres *nSP-cellules* sont simplifiées pour minimiser l'occupation mémoire, et ne sont pas envoyées au *pipeline* graphique. En accord avec ces critères, une liste des *nSP-cellules* à afficher associées à leur niveau de détail est maintenue. Une *nSP-cellule* qui atteint sa précision maximale est divisée, et l'on accède directement au contenu de chaque enfant grâce à son indice dans le fichier, indice contenu dans le *nSP-arbre*. A l'inverse, si une *nSP-cellule* doit être simplifiée à un niveau de détail inférieur ou égal à sa précision maximale, ses éventuels enfants sont simplifiés et fusionnés entre eux.

3.6.3 Traitement des frontières multi-résolution

Une fois que la liste des *nSP-cellules* à afficher est établie et que chacune d'elles se trouve au niveau de détail souhaité, deux problèmes principaux — illustrés sur la figure 3.14 — se posent, concernant les frontières entre *nSP-cellules* :

1. les simplexes dupliqués dans différentes *nSP-cellules* peuvent se chevaucher (a_1 et a_2 sur la figure 3.14),
2. des artefacts visuels peuvent être causés par l'affichage de deux *nSP-cellules* adjacentes dont les précisions sont différentes (b sur la figure 3.14).

Seuls les simplexes maximaux frontières — *i.e.* ceux qui se situent dans plusieurs *nSP-cellules* simultanément — peuvent causer ces problèmes ; les autres peuvent être directement affichés. L'algorithme suivant est appliqué pour chaque simplexe maximal frontière s :

1. Soit c la *nSP-cellule* à laquelle s appartient, p_c la précision actuelle de c, et l la liste des *nSP-cellules* à afficher. Soit N le nombre de *kd-cellules* (*i.e.* le nombre de sommets) composant s, c_i (avec i dans $1, \ldots, N$) la *nSP-cellule* dans laquelle se trouve la i-ème *kd-cellule* k_i composant s, et p_i la précision actuelle de c_i (si $c_i \notin l$ alors $p_i = p_c$).

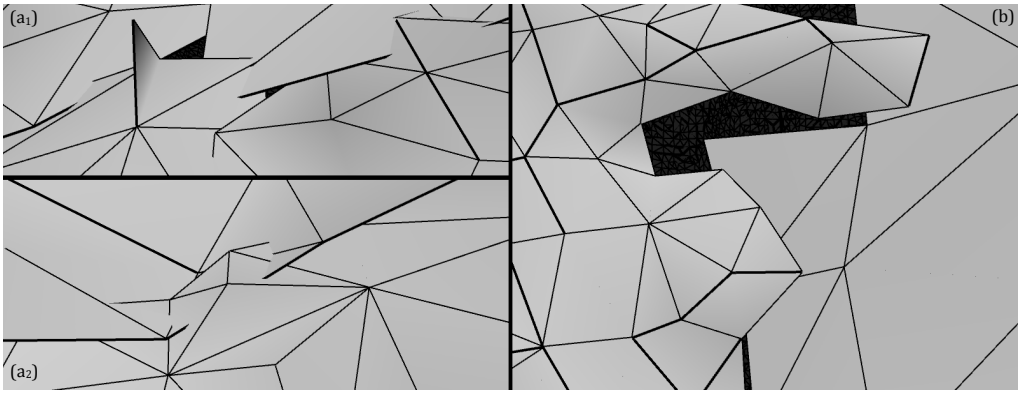


FIGURE 3.14 – Exemples de problèmes liés aux frontières entre les *nSP-cellules* : chevauchements (a_1 et a_2), différence de précision (b)

2. S'il existe un i dans $1, \ldots, N$ tel que $p_i > p_c$ ou ($p_i = p_c$ et indice de $c_i >$ indice de c), s est supprimé et ne sera pas affiché.
3. Sinon, pour chaque *kd-cellule* k_i telle que $c_i \in l$ et $c_i \neq c$, la *kd-cellule* k'_i de c_i contenant k_i est recherchée. Le point représentatif de k'_i est utilisé pour afficher s.

On obtient ainsi une transition lisse et bien formée entre les *nSP-cellules* de différents niveaux de précision. La figure 3.15 montre un exemple de calcul de frontière : la *nSP-cellule* 2 est plus raffinée que la 1. Par conséquent, tous les triangles de la *nSP-cellule* 1 ayant un de leurs sommets dans la *nSP-cellule* 2 sont supprimés, et les sommets de la *nSP-cellule* 2 situés dans la *nSP-cellule* 1 voient leurs coordonnées remplacées par celles du point représentatif de la *kd-cellule* de la *nSP-cellule* 1 dans laquelle ils se trouvent.

3.7 Amélioration du compromis débit-distorsion

Le principal inconvénient de cette méthode est sa tendance à produire de nombreux petits triangles dans les niveaux de précision intermédiaires. Ces triangles, qui ont généralement une taille à l'écran proche de la précision demandée (par exemple, 1 pixel), ralentissent le rendu sans résoudre l'effet de bloc inhérent aux approches de type *kd-arbre* ou *octree*, illustré à la figure 3.16. En réalité, la précision demandée par l'utilisateur concerne généralement la géométrie. Une précision souhaitée de 1 pixel requiert que les sommets soient positionnés sur l'écran à 1 pixel près, mais n'exige pas que les triangles aient une boîte englobante de taille 1 pixel.

Ce problème est résolu en encodant la géométrie en avance sur la connectivité dans le fichier comprimé : à une précision p donnée, le décodeur connaît les codes géométriques (mais pas les codes de connectivité) contenus dans les

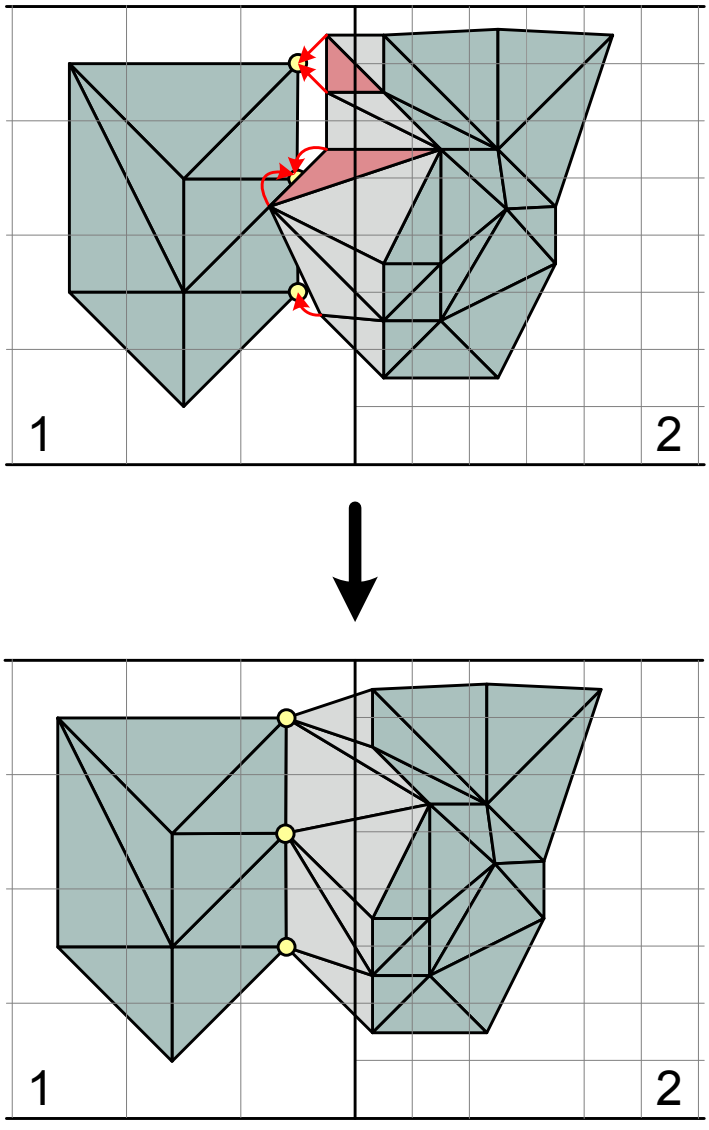


FIGURE 3.15 – Rendu des simplexes maximaux frontières entre deux *nSP-cellules* de niveaux de détail différents

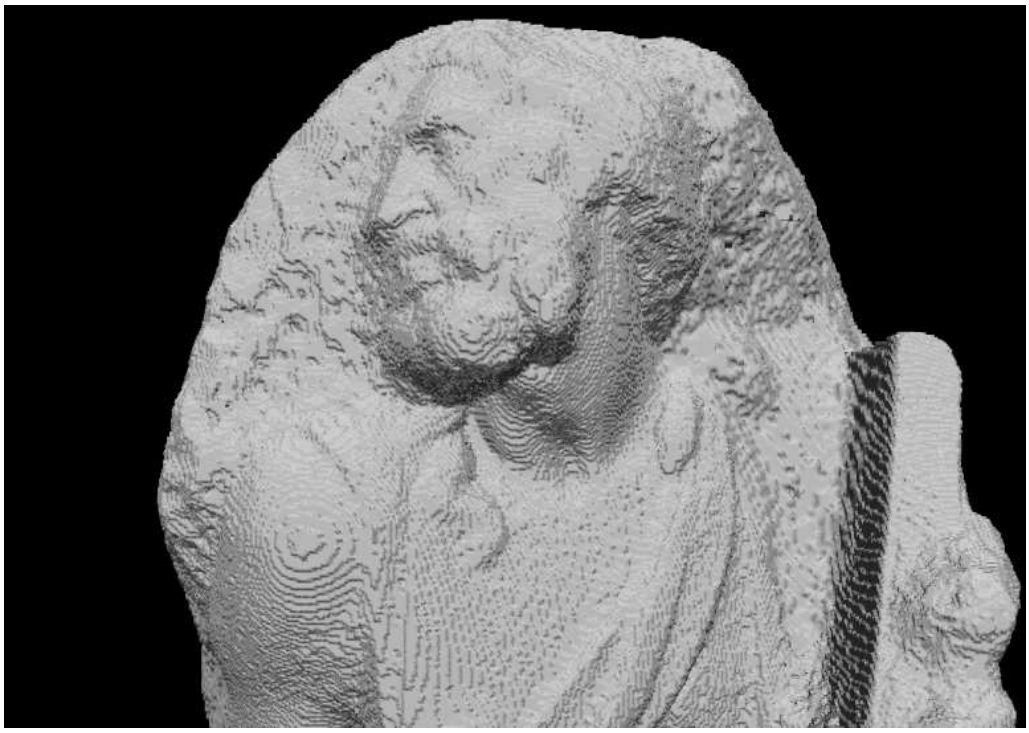


FIGURE 3.16 – Exemple de rendu du Saint Matthieu de Michel-Ange, présentant un effet de bloc dû au découpage en *kd-arbre*

k niveaux suivants du *kd-arbre*. Par conséquent, une *kd-cellule c* sera composée d'un code de connectivité, suivi de plusieurs codes de géométrie qui décrivent les positions des sommets descendants de *c*. Le niveau de ces derniers dépend de *k*, le nombre de niveaux de descendants connus par chaque *kd-cellule*. Pour un maillage plongé dans un espace à *d* dimensions et une avance géométrique de *m* bits, $k = d \times m$. Pour chaque sommet affiché, le nombre et la position géométrique de ses futurs enfants est ainsi connue et leur barycentre peut être utilisé comme position du point représentant. La figure 3.17 illustre ce principe : les dessins (a) et (c) montrent respectivement le maillage avec 2 bits et 3 bits de précision, aussi bien pour la géométrie que pour la connectivité (donc sans encodage anticipé de la géométrie). Pour le schéma (b), la connectivité est celle du dessin (a) (2 bits), mais la géométrie (*i.e.* les sommets) du schéma (c) (3 bits) est connue, ce qui permet de placer les points issus du dessin (a) plus précisément, en utilisant le barycentre des sommets à venir dans le niveau suivant (3 bits). Ainsi, la position géométrique des sommets n'est plus alignée sur une grille régulière, car le sommet associé à une *kd-cellule* n'est plus systématiquement au centre de celle-ci — même s'il demeure bien sûr à l'intérieur de la cellule.

La figure 3.18 montre un exemple de *nSP-arbre* avec une avance géométrique de 1 bit, dans le cas d'un espace 2D ($d = 2$). Lors de la division d'une *nSP-cellule*, certaines *kd-cellules* peuvent être dupliquées dans plusieurs enfants, comme la *kd-cellule* 11 dans cet exemple. Les clones d'une même *kd-cellule*

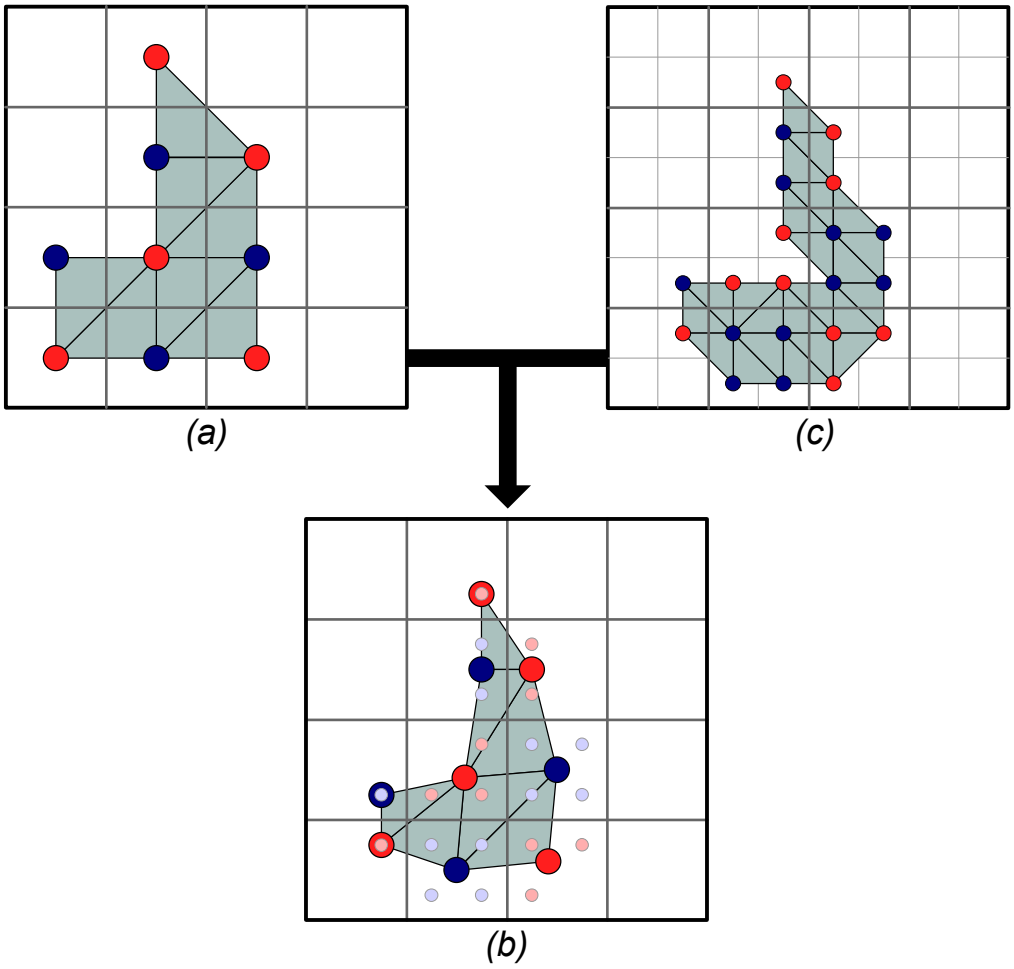


FIGURE 3.17 – Exemple de rendu avec géométrie anticipée : comparaison entre (a) 2 bits de précision pour la géométrie et la connectivité, (b) 2 bits pour la connectivité et 3 bits pour la géométrie ($2+1$ bit d'avance), (c) 3 bits pour les deux

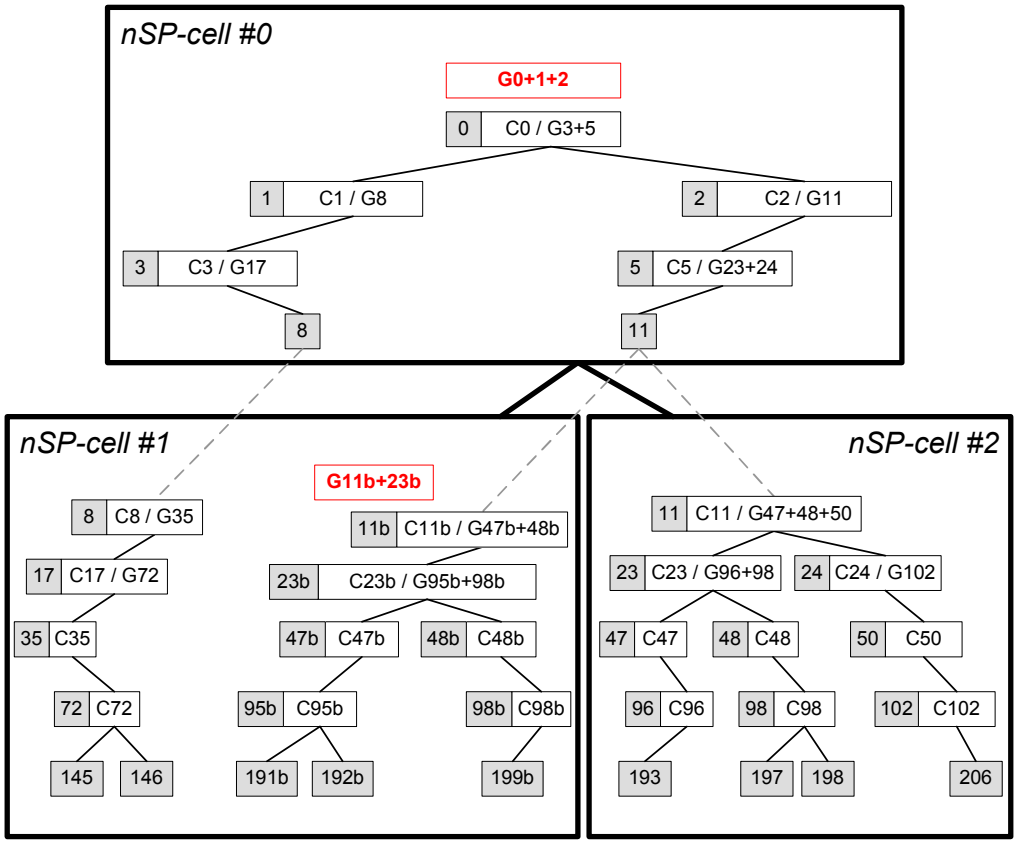
peuvent être amenés à évoluer différemment, puisque leur voisinage est différent selon la *nSP-cellule* à laquelle ils appartiennent. Leurs codes peuvent donc être différents, comme G11 et G11b (la *kd-cellule* 11 est dupliquée lors de la division de la *nSP-cellule* #0). Chaque *kd-cellule* du *kd-arbre* parent (ici, celui de la *nSP-cellule* #0) peut donc stocker les codes géométriques en avance d'une seule *kd-cellule* fille. Par exemple, la *kd-cellule* 2 stocke et utilise le code géométrique G11, mais pas le code G11b. Il faut donc faire un choix parmi les *kd-cellules* dupliquées ; nous avons choisi de stocker dans la *nSP-cellule* parente les codes des *kd-cellules* situées à l'intérieur de leurs *nSP-cellules* respectives (ici, la *kd-cellule* 11 est située à l'intérieur de la *nSP-cellule* #2). Pour les autres *nSP-cellules*, les codes géométriques manquants sont ajoutés au début de la séquence. Dans notre exemple, la *nSP-cellule* #0 stocke G11, G23 et G24, correspondant aux *kd-cellules* 11, 23 et 24 de la *nSP-cellule* #2 ; les codes G11b et G23b manquants sont donc stockés au début de la *nSP-cellule* #1 (en rouge sur le schéma). De même, puisque la *nSP-cellule* racine (#0) n'a pas de parent, quelques codes géométriques sont ajoutés à son début.

La figure 3.19 illustre les avantages de cette technique en termes de rendu.

Afin de quantifier l'amélioration apportée par cette modification de la distribution des informations au sein du code, nous avons mesuré son apport en termes de compromis débit-distorsion. Nous avons ainsi construit la courbe de débit-distorsion, en utilisant la distance L^2 — aussi appelée distance *RMS* pour *Root Mean Square*, cf. section 1.5 à propos des mesures d'erreur —, calculée à l'aide de l'outil METRO [CRS98]. Celle-ci fait apparaître une amélioration significative dans les niveaux de détail intermédiaires : pour un débit identique, la réduction de l'erreur *RMS* peut atteindre près de -50%.

3.8 Conclusion

Toutes les idées présentées ici ont été implémentées et testées avec soin afin d'en déterminer la pertinence. En particulier, les performances des algorithmes de compression et visualisation sont primordiales pour que la navigation au sein des maillages soient effectivement interactive. Une attention particulière a été portée à l'optimisation, thème développé dans le chapitre suivant et étayé par des résultats quantitatifs.



i | C*i* / G*j*+*k* — *Kd-cell* n°*i* contenant les codes de connectivité de la *kd-cell i* et les codes de géométrie des *kd-cells j* et *k*

G*m*+*n*+... — Au début d'une *nSP-cell* sont stockés les codes géométriques qui n'étaient pas dans la *nSP-cell* parente et qui ne se trouvent pas dans le *kd-tree* suivant.

Encodage du fichier :

#0:**G**0+1+2**C**0**G**3+5**C**1**G**8**C**2**G**11...	#1:**G**11b+23b**C**8**G**35**C**11b**G**47b+48b...	#2:**C**11**G**47+48+50...

FIGURE 3.18 – Exemple de *kd-arbre* avec la géométrie en avance sur la connectivité : cas 2D, 1 bit d'avance (*i.e.* 2 niveaux de *kd-arbre*)

FIGURE 3.19 – Géométrie en avance sur la connectivité : rendu normal (en haut) et rendu avec une avance de 2 bits sur la position géométrique des sommets (en bas)

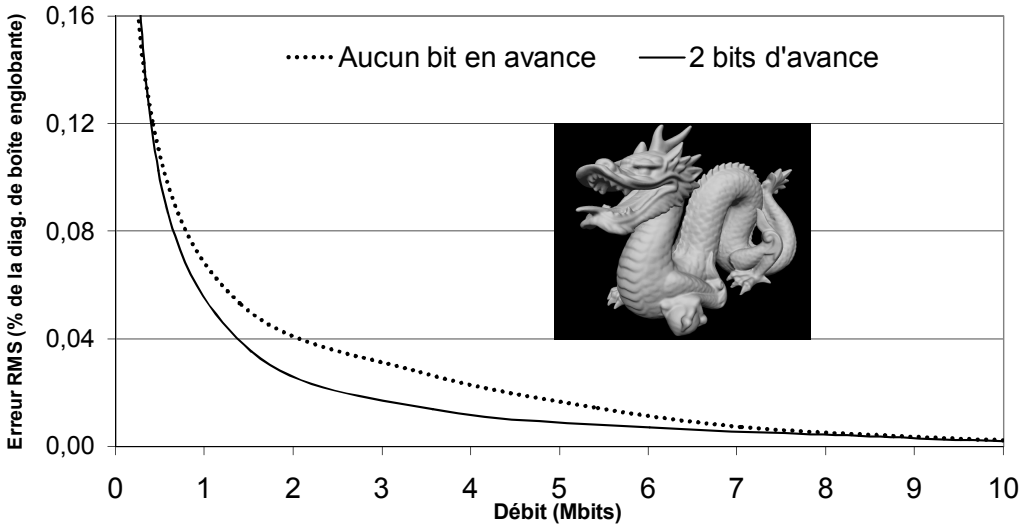


FIGURE 3.20 – Comparaison des courbes de débit-distorsion avec et sans redistribution des informations

Chapitre 4

CHuMI Viewer — Implémentation et résultats

« Si les faits ne correspondent pas à la théorie, changez les faits. »
Albert Einstein.

Parmi les mots-clés caractérisant notre travail se trouve le mot *interactif*. Qui dit interactif dit réactif, rapide, fluide… Ce qui nous interdit de nous contenter de présenter les principes de notre méthode sans évoquer ses performances.

Notre première implémentation était codée de façon « naïve ». La réalisation était certes soignée et documentée, mais la programmation des algorithmes faisaient la part belle à la candeur, sans prêter réellement attention à la complexité ou à l'efficacité. Bien entendu, les résultats étaient désastreux et *interactif* n'était qu'un vain mot.

Cependant, nous avions désormais une vue globale sur la méthode, et une idée précise des points à améliorer : goulets d'étranglement, gestion de la mémoire, performance des accès disque. Partant de là, nous avons réalisé une seconde conception et implémentation de la méthode, *from scratch*, en gardant à l'esprit les notions de performance et de réactivité. Nos efforts furent récompensés lorsque nous avons constaté des améliorations significatives : gains d'un facteur proche de 10 sur les temps de calcul et proche de 20 sur l'occupation mémoire.

Ainsi, le soin apporté à l'implémentation constitue une part très importante de ce travail. Ce chapitre présente les principaux aspects de notre implémentation et les optimisations les plus importantes développées pour atteindre les performances présentées en seconde partie.

4.1 Implémentation et optimisation

Lors de la conception de l'implémentation de notre méthode, nous nous sommes appliqués à soigner trois aspects : les temps de calcul, les accès disque et l'occupation mémoire. Cela nous a conduit à nous intéresser de plus près au *hardware* : fonctionnement des cartes graphiques, exploitation des processeurs multi-cœurs, performance des disques durs, ou encore utilisation efficace des allocations mémoire. Cette section présente les enseignements que nous en avons tirés.

4.1.1 Gestion de la mémoire

Afin de réduire l'occupation mémoire, chaque *nSP-cellule* chargée stocke uniquement la liste des feuilles de son *kd-arbre*, plutôt que l'arbre complet avec ses nœuds internes. Ces derniers peuvent aisément être reconstruits par fusion récursive des feuilles. De plus, les raffinements et simplifications successifs impliquent un grand nombre de créations et de destructions d'objets tels que *kd-cellules*, simplexes et listes. Une gestion naïve de la mémoire nécessiterait de nombreuses allocations et désallocations, causant fragmentation et ralentissements. Par conséquent, des *pools* de mémoire sont systématiquement employés pour pré-allouer des milliers d'objets en une seule opération, et une préférence est donnée à la réutilisation des objets plutôt qu'à leur destruction et création.

Les implémentations habituelles de complexes simpliciaux construisent une structure où chaque d-simplexe ($d > 1$) est composé de $d+1$ $(d-1)$-simplexes. Cette structure est coûteuse à maintenir, et puisque nous n'avons pas réellement besoin de cette relation parent-enfant, nous avons choisi d'utiliser une structure plus simple : une liste de simplexes maximaux, chacun d'eux contenant des références vers les *kd-cellules* correspondant à ses sommets incidents. Réciproquement, chaque *kd-cellule* contient une liste de ses simplexes maximaux incidents.

Le *temps de recherche* des disques durs est le temps que met la tête de lecture pour se déplacer jusqu'au cylindre contenant l'information demandée. Une distance sur disque importante entre deux données accédées successivement conduit à un temps de recherche important, et réduit d'autant la vitesse de lecture des données. De plus, les disques durs actuels embarquent très souvent une mémoire vive sur leur contrôleur, mémoire qui est remplie automatiquement par les blocs qui suivent le bloc actuellement demandé. Ce mécanisme n'est utile que si les accès sont séquentiels, *i.e.* les données sont accédées dans l'ordre dans lequel elles sont stockées sur le disque. C'est pourquoi nous avons organisé les accès à la mémoire externe de la façon la plus séquentielle possible. Comme présenté dans la section 3.4.3, des techniques de pagination et de mémoire tampon sont utilisées durant la compression pour écrire le fichier RWI.

Lors de la visualisation, dès qu'une *nSP-cellule* doit être chargée, sa séquence de codes complète est lue dans le fichier FEC et stockée dans un tampon pour anticiper les accès à venir.

4.1.2 Parallélisation multi-cœurs

La structure du *nSP-arbre* est intrinsèquement propice à la parallélisation. L'algorithme de compression utilise plusieurs *threads* afin de bénéficier des processeurs multi-cœurs actuels. Tandis que les entrées-sorties dans les fichiers demeurent mono-*thread* pour éviter de coûteux accès concurrents au disque dur, les opérateurs de fusion et de subdivision de *kd-cellule* ainsi que de fusion et division de *nSP-cellule* peuvent être exécutés en parallèle sur différentes cellules.

Pour favoriser les accès séquentiels au disque et éviter de trop fréquents blocages des autres *threads* (les entrées-sorties sur disque étant exclusives), le traitement d'une *nSP-cellule* débute par le chargement de la liste des simplexes maximaux contenus depuis les fichiers RWI et RWP dans un tampon. Puis, les étapes suivantes (cf. section 3.5.3) ne nécessitant que des opérations CPU ou RAM, elles peuvent être exécutées dans un *thread* parallèle.

En pratique, un gain global de performance entre 1, 5 et 2 sur un CPU dual-core et entre 2 et 3 sur un CPU quad-core est observé sur les algorithmes de compression. L'étape de décompression et visualisation bénéficie également de cette parallélisation, mais dans ce cas, l'amélioration des performances est plus difficile à estimer.

4.1.3 Compression rapide

La phase de compression, même si elle est *offline* et qu'elle n'est exécutée qu'une seule fois pour chaque maillage, mérite d'être optimisée. Certaines méthodes demandent de solliciter un réseau de plusieurs stations de travail pendant des dizaines d'heures pour comprimer des maillages de plusieurs centaines de millions de triangles. Nous avons souhaité éviter ce cas de figure et permettre l'encodage sur une seule machine en quelques heures, même pour les plus gros maillages. Dans ce but, l'implémentation a été particulièrement soignée.

Un des calculs les plus coûteux de l'algorithme est la construction de la séquence codante de la fusion de sommet. Lors de la compression, il s'agit de cartographier le voisinage des deux *kd-cellules* concernées afin de disposer et d'accéder rapidement aux informations dans un ordre précis. Nous commençons donc par remplir une structure de données spécifique, qui doit satisfaire à plusieurs exigences :

- L'insertion des éléments du voisinage (simplexes maximaux notamment) doit pouvoir se faire dans n'importe quel ordre.
- Cette insertion doit être rapide.
- Le calcul de la séquence codante à partir de cette structure doit être direct et efficace.
- De même, l'application des règles de réduction du coût de stockage (cf. paragraphe « Subdivision/fusion de sommet » de la section 3.5.3) doit être facilitée.

La figure 4.1 présente la structure de données utilisée par notre implémentation. Un dictionnaire est construite :

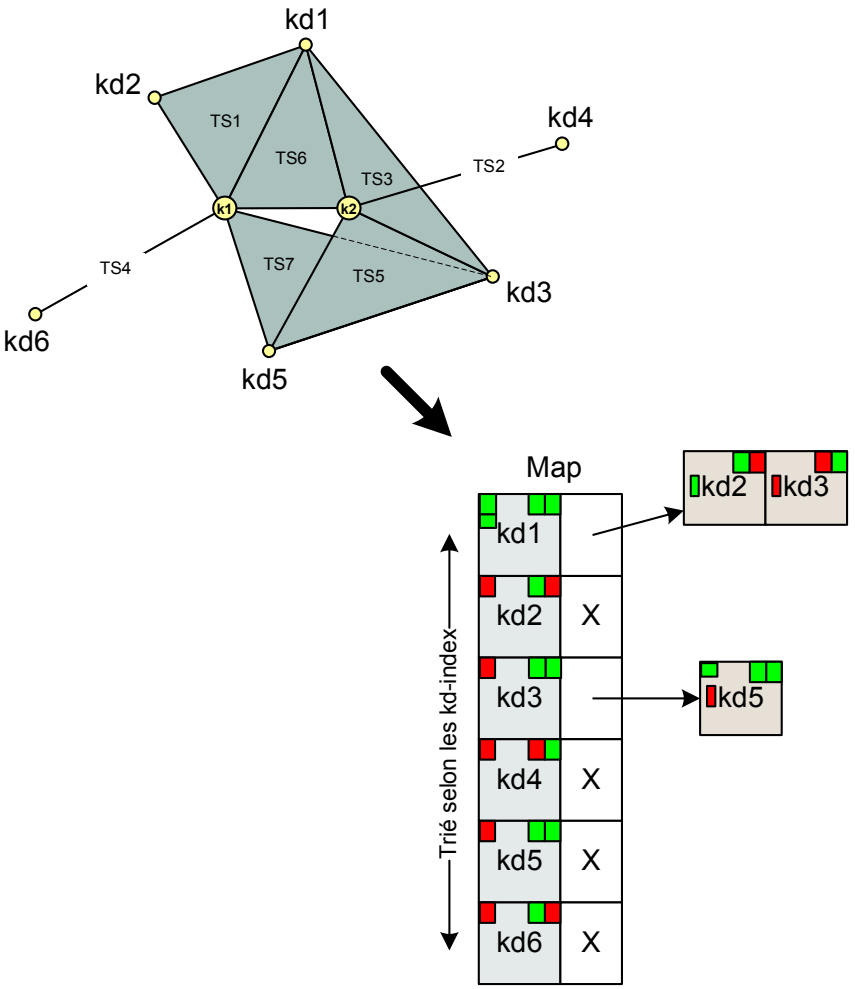
- Les clés correspondent aux *kd-cellules* adjacentes à k1 et k2 (les *kd-cellules* fusionnées).
- S'il existe un triangle entre kd$i$, kd$j$ et k1 ou k2, avec $i < j$, alors une cellule « kd$j$ » est créée dans le tableau associé à la clé « kd$i$ »
- Les clés et les cellules des tableaux associés aux clés contiennent des informations supplémentaires sur les éléments concernés (cf. légende de la figure).

Par exemple, le simplexe maximal TS5, de dimension 2, relie k2, kd3 et kd5. Une cellule « kd5 » est donc créée dans le tableau associé à la clé « kd3 ».

Il est ensuite aisé, à partir de cette structure, d'en déduire la séquence de codes (cf. figure 3.13) :

- Le premier code, de dimension 0, est **3**. Il indique qu'il y a une arête entre k1 et k2. Son calcul n'utilise pas la structure. Il est déterminé directement, en même temps que le remplissage du dictionnaire.
- Les codes de dimension 1 sont déduits à partir de la liste des clés du dictionnaire :
  - **4a** (dans la clé « kd1 », les cases a et t sont vertes)
  - **1** (dans la clé « kd2 », la case a est rouge, la b est verte et la c est rouge)
  - **3** (dans la clé « kd3 », la case a est rouge, les cases b et c sont vertes)
  - **2** (dans la clé « kd4 », la case a est rouge, la b est rouge et la c est verte)
  - **3**
  - **1**
- Les codes de dimension 2 sont déduits à partir des tableaux associés aux clés, parcourus dans l'ordre :
  - **1, non encodé** (en dimension 1, kd2 a un code 1)
  - **2** (dans la cellule « kd3 » du premier tableau, la case a est rouge, la b est rouge et la c est verte)
  - **3a** (dans la cellule « kd5 » du deuxième tableau, la case a est rouge, les cases b et c sont vertes, et t est vert)

De façon similaire, les autres étapes du processus de compression ont été conçues avec soin, notamment la contraction d'arête et la fusion des *nSP-cellules*. La section 4.2 présente les résultats qui attestent des bonnes performances de l'étape d'encodage.



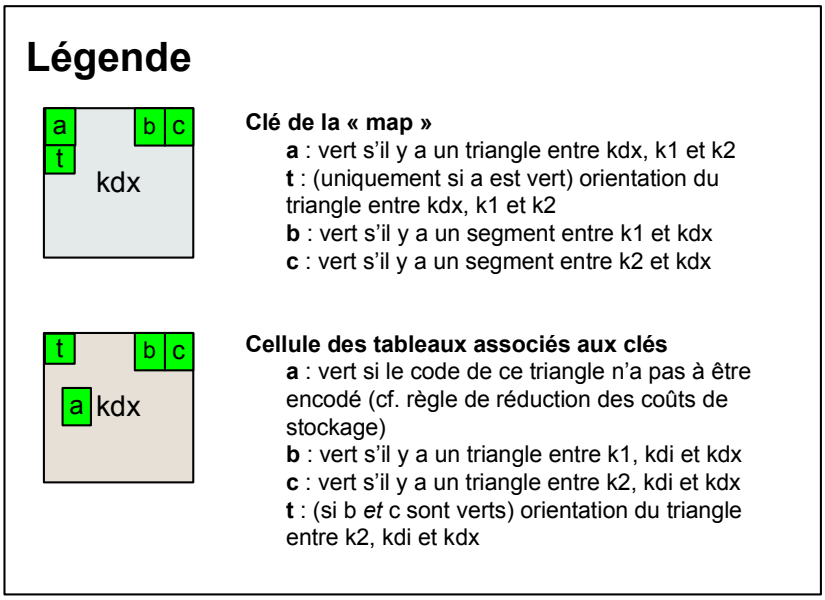


FIGURE 4.1 – Structure de données utilisée pour l'encodage de la fusion de sommets

4.1.4 Rendu adaptatif efficace

En vue d'obtenir de bonnes performances en termes de rendu interactif et fluide, nous avons prêté une attention particulière à l'implémentation de nos algorithmes : conception, programmation, optimisation, en fonction du *hardware*, etc. Nous détaillons ici les principaux choix d'implémentation qui ont conduit à une visualisation *interactive*.

Lors de la subdivision d'une *nSP-cellule*, les *kd-cellules* et les simplexes maximaux sont transférés, et parfois dupliqués, dans les *nSP-cellules* filles. Une part coûteuse de cette division consiste à déterminer dans quelle(s) *nSP-cellule(s)* fille(s) la *kd-cellule* considérée doit être transférée ou copiée. Plutôt que de calculer cette information pour toutes les *kd-cellules* au moment de la division, nous la calculons dès que la précision de la *kd-cellule* suffit à déterminer la future *nSP-cellule* fille qui la contiendra. Cette information est stockée et peut ainsi être transmise aux *kd-cellules* descendantes, car celles-ci appartiendront à la même *nSP-cellule* fille.

Par ailleurs, cette information permet de prédire l'évolution des simplexes maximaux lors de la prochaine division de la *nSP-cellule* : future(s) *nSP-cellule(s)* contenantes, duplication éventuelle. Afin de lisser la charge de calcul tout au long du raffinement, chaque simplexe maximal est testé à sa création, qui peut avoir lieu suite à :

- Une expansion d'arête ou une subdivision de sommet (cf. paragraphe (b) de la section 3.5.3).
- Une duplication de simplexe lors d'une division de *nSP-cellule*.

A l'inverse, la fusion de deux *nSP-cellules* implique de détruire tous les objets dupliqués, simplexes maximaux et *kd-cellules*. Pour optimiser cette étape et déterminer rapidement les objets concernés, chaque *kd-cellule* et simplexe maximal conserve deux informations :

- Le niveau du *kd-arbre* auquel il/elle a été créé(e).
- S'il/elle a été créé(e) suite à une division de *nSP-cellule*.

Lors de la fusion de *nSP-cellules* dans leur cellule parente, certains simplexes maximaux transférés depuis une fille sont susceptibles d'être composés de *kd-cellules* devant être supprimées (*kd-cellules* dupliquées). Le coût d'une recherche de la *kd-cellule* originale parmi toutes les *kd-cellules* transférées vers la *nSP-cellule* parente est rédhibitoire. C'est pourquoi chaque *kd-cellule* dupliquée stocke un pointeur vers sa *kd-cellule* originale (dont elle est la copie). Pour s'assurer du fonctionnement correct de cette optimisation, les processus de raffinement et de simplification doivent être réversibles : le raffinement suivi de la simplification inverse d'une *nSP-cellule* doit restaurer la liste des *kd-cellules* à l'identique, notamment au niveau des adresses mémoire de celles-ci. Et ce, même si la *nSP-cellule* en question subit des subdivisions et des fusions successives.

Le calcul des changements de connectivité induits par la fusion ou la divi-

sion des *kd-cellules*, ainsi que des séquences de code associées, constitue le cœur de notre algorithme. Ce sont des opérations très coûteuses, sur lesquelles nous nous sommes attardés pour garantir un rendu rapide et réactif.

Tout d'abord, une relation d'ordre efficace sur les *kd-cellules* est nécessaire, ne serait-ce que pour déterminer l'ordre des codes dans la séquence décrivant la subdivision de sommet. Plutôt que de se baser sur la position géométrique, nous utilisons le *kd-index* défini comme suit :

Définition 17. *Le kd-index de la kd-cellule k vaut :*
- *o si k est la racine de l'arbre*
- $2 \times kd\text{-}index(parent(k)) + 1$ *s'il s'agit du fils gauche de* $parent(k)$
- $2 \times kd\text{-}index(parent(k)) + 2$ *s'il s'agit du fils droit de* $parent(k)$

Cette valeur est un entier qui permet la comparaison à l'aide d'une seule opération binaire.

Il est aisé d'en déduire un ordre sur les simplexes :

Définition 18. *Soit* s_1 *et* s_2 *deux simplexes.* $s_1 < s_2$ *:*
- *si* $dim(s_1) < dim(s_2)$
- *sinon, si* $dim(s_1) = dim(s_2)$*, les kd-cellules de* s_1 *et* s_2 *sont triées selon leur kd-index, puis les deux séquences sont comparées suivant l'ordre lexicographique*

De plus, il a été montré précédemment que chaque simplexe maximal stockait le niveau du *kd-arbre* dans lequel il a été créé. Cette valeur est aussi utilisée pour accélérer les opérations de contraction d'arête et de fusion de sommets durant la simplification. Il est ainsi possible de déterminer rapidement quels simplexes maximaux doivent être supprimés lors de la fusion de deux *kd-cellules*. Il ne reste qu'à remplacer la *kd-cellule* disparue par la *kd-cellule* restante dans les simplexes maximaux incidents, puis traiter le cas des triangles dégénérés en arêtes.

Toutes ces optimisations algorithmiques et structurelles, accompagnées d'une conception et d'un emploi efficace du langage C++, permettent d'atteindre nos objectifs en termes de réactivité et de fluidité de la visualisation, comme en attestent les résultats présentés dans la suite.

4.2 Résultats expérimentaux

Afin de démontrer les capacités de notre méthode, cette section présente un ensemble de résultats expérimentaux, tant pour la compression des maillages que pour leur décompression et leur visualisation interactive.

4.2.1 De la comparaison avec les méthodes existantes

Il est difficile de comparer équitablement notre méthode avec les travaux existants, puisqu'à notre connaissance, aucune autre méthode ne combine compression et rendu interactif de maillages polyédriques. Cependant, nous fournissons une comparaison avec les principales méthodes de compression (mono et multi-résolution, *in-core* et *out-of-core*) dans la section 4.2.4, puis avec les méthodes de visualisation de référence dans la section 4.2.5.

Nous tenons à informer le lecteur de la difficulté de comparer notre travail avec les méthodes existantes, en compression mais aussi et surtout pour la visualisation.

Tout d'abord, l'algorithme de compression de Gandoin et Devillers [GD02] sur lequel est basé notre algorithme présente des taux de compression compétitifs au regard de l'étendue des maillages qu'il sait traiter. Cependant, les contraintes de visualisation interactive nous ont amené à ajouter des informations, comme les simplexes dupliqués ou l'orientation des triangles. Nous avons aussi dû abandonner la plupart des schémas de prédiction à cause de leur coût calculatoire élevé. Il s'ensuit une baisse sensible des taux de compression. Il est clair que nous ne nous plaçons pas dans une optique de réduction des taux de compression des meilleures méthodes actuelles, mais plutôt dans l'optique d'introduire un compromis entre compression et visualisation rapide et réactive. Cette compression se doit malgré tout d'être significative et la plus proche possible des taux actuels.

D'autre part, la comparaison avec les méthodes de visualisation existantes présente de nombreuses difficultés. S'il est relativement aisé de comparer les taux de compression de deux algorithmes existants, le simple fait de comparer entre elles les méthodes de visualisation *non compressive* est beaucoup plus difficile, pour plusieurs raisons :

- Comment mesurer et prendre en compte la qualité de la visualisation ?
- Que représente exactement le *nombre d'images par seconde* ou le *nombre de triangles affichés par seconde*, si le maillage visualisé et le parcours de la caméra ne sont pas scrupulusement définis ?
- Comment comparer une méthode qui affiche des triangles avec une méthode qui emploie des approximations basées sur les couleurs ?
- Comment tenir compte précisément de l'évolution de la puissance des stations de travail ? En effet, les résultats de deux articles différents ne sont jamais produits sur les mêmes ordinateurs, et les différents paramètres (architecture, CPU, accès à la mémoire vive, accès au disque dur, puissance de la carte graphique) n'évoluent pas de façon identique.

Dans les articles de visualisation de maillages volumineux actuels, la norme consiste à comparer le nombre de triangles affichés par seconde. Si cette mesure permet généralement d'évaluer l'exploitation de la carte graphique — pour peu

que l'on tienne compte du contexte (année de parution de l'article, machine employée) —, elle est seulement un indicateur parmi d'autres des performances réelles de l'algorithme. Une méthode utilisant moins de triangles qu'une autre pour générer une image identique se verra pénalisée, car l'affichage de très nombreux triangles (de l'ordre du million) ne posent plus de problèmes aux cartes graphiques actuelles, à condition qu'on les stocke dans sa mémoire dédiée et qu'on limite les transferts entre celle-ci et la mémoire vive.

Généralement, l'objectif des résultats présentés est d'évaluer la qualité de l'affichage et la réactivité de l'algorithme lors du déplacement de la caméra par l'utilisateur. En pratique, rien n'est plus efficace pour évaluer la méthode que de visualiser une vidéo ou de manipuler le logiciel, même si la subjectivité est alors inévitable et qu'une comparaison chiffrée est difficile. Il serait peut-être intéressant d'envisager un système d'évaluation par un nombre conséquent d'utilisateurs, en « aveugle » et sur les mêmes maillages.

Dans les sections suivantes, nous donnons un ensemble de mesures, comme autant d'indicateurs des performances réelles de la méthode. En supplément, des vidéos [JGA09a] et le logiciel [JGA09b] sont disponibles en téléchargement sur le Web.

4.2.2 Plateforme et paramètres

Les résultats présentés ont été obtenus grâce à une implémentation C++ de la méthode, utilisant la bibliothèque OpenGL, exécutée sur un PC équipé d'un processeur Intel Q6600 QuadCore 2,4Ghz, 4 Go de RAM, 2 disques durs 10000RPM raccordés en RAID0, et une carte graphique NVIDIA GeForce 8800 GT 512MB. Les modèles testés sont fournis par *The Digital Michelangelo Project*, *Stanford University Computer Graphics Laboratory*, *UNC Chapel Hill* et *Electricité de France (EDF)*.

Concernant les paramètres, la valeur de n (nombre de subdivisions par axe lors de la construction du *nSP-arbre*) doit être suffisamment grande pour :

- permettre la sélection de petites parties du maillage en quelques divisions de *nSP-cellules*,
- limiter le nombre de division/fusion de *nSP-cellules*,

mais suffisamment petite pour :

- éviter l'affichage simultané de nombreuses *nSP-cellules* (ce qui impliquerait de coûteux calculs de frontières),
- afficher un nombre limité de triangles hors champ de la caméra (l'élimination des parties en dehors du cône de vision a la granularité des *nSP-cellules*),
- éviter ainsi de calculer le raffinement de ces parties non visibles

Concernant N_{min} (nombre minimum de triangles contenus dans une *nSP-cellule*

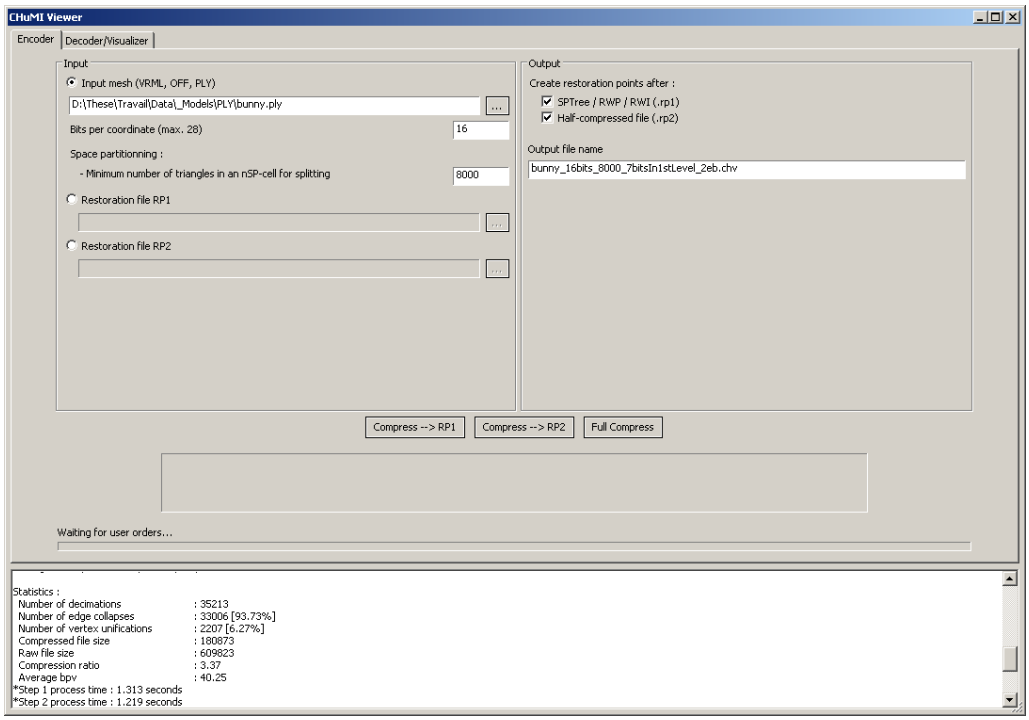


FIGURE 4.2 – Interface de la partie compression de CHuMI Viewer

pour qu'elle soit subdivisée), une petite valeur augmente la duplication de simplexes et dégrade les performances de la compression, tandis qu'une grande valeur induit un *nSP-arbre* de hauteur limitée, au détriment des capacités de multi-résolution et de l'élimination des simplexes hors champ.

Pour satisfaire ces conditions, $n = 4$ et $5000 \leq N_{min} \leq 8000$ apparaissent comme de bons compromis.

Afin de tester et mettre en application les idées présentées dans ce mémoire, le logiciel CHuMI Viewer a été développé en C++. Il est constitué d'environ 26 000 lignes non vides, dont 19 200 de code et 6 800 de commentaires. Les figures 4.2 et 4.3 présentent son interface.

4.2.3 Duplication de triangles

La table 4.1 présente des statistiques sur la duplication des triangles. Le pourcentage donné correspond au ratio entre le nombre de triangles dupliqués (troisième colonne du tableau) et le nombre de triangles originaux (seconde colonne). Sur les maillages testés, le nombre de triangles connait donc une augmentation moyenne de 5,9%, avec un minimum de 4,29% et un maximum de 6,85%. Au regard de la simplicité et de l'efficacité de cette duplication, le surcoût engendré reste tout à fait acceptable.

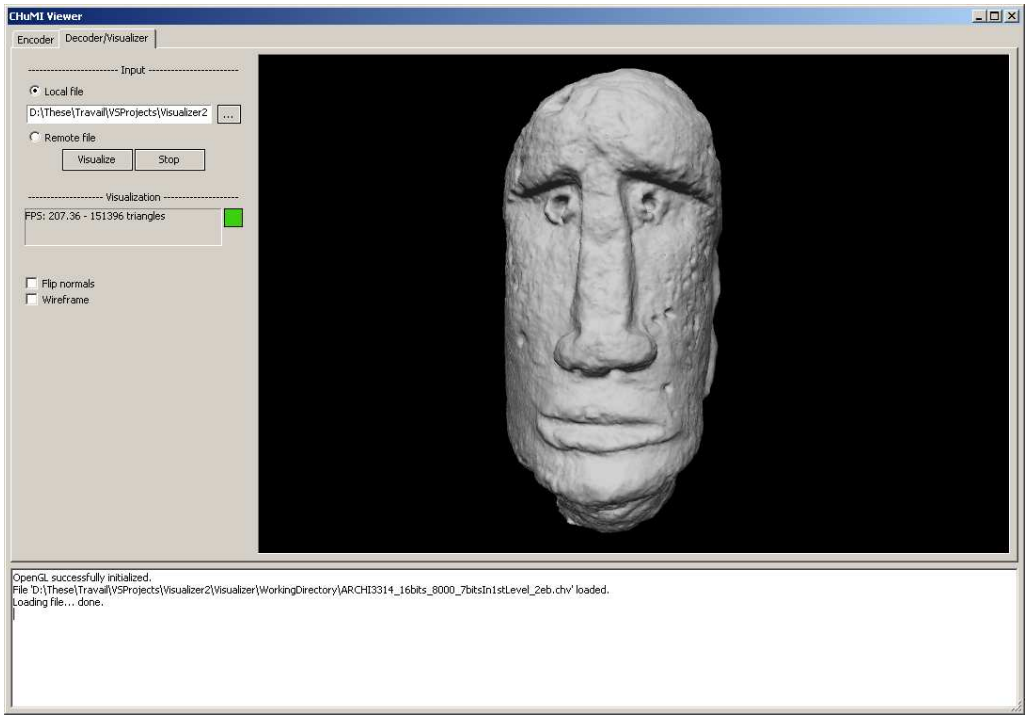


FIGURE 4.3 – Interface de la partie décompression et visualisation de CHuMI Viewer

TABLE 4.1 – Triangles dupliqués lors de la compression avec $p = 16$ bits par coordonnée, $n = 4$, $N_{min} = 8000$ et $p_r = 7$

Modèle	Triangles originaux	Triangles dupliqués	
		Nombre	Augmentation (%)
Dragon	866 508	55981	6,46
David 2mm	8 250 977	564977	6,85
EDF T5	14 371 932	908627	6,32
EDF T8	22 359 212	1284595	5,75
Lucy	27 595 822	1450078	5,25
David 1mm	54 672 488	3591158	6,57
Saint Matthieu	372 422 615	15974825	4,29
Total	500 539 554	23 830 241	4,76

4.2.4 Compression

Nous commençons ici par comparer notre méthode avec les principaux travaux de compression pure.

La table 4.2 présente les résultats de l'étape de compression *out-of-core*, en termes de taux de compression. Nous avons choisi $p_r = 7$ bits de façon à ce que le nombre de triangles contenus dans la *nSP-cellule* racine soit suffisamment faible pour permettre à cette version simplifiée du maillage d'être aisément chargée en mémoire. Pour chaque modèle, nous indiquons le nombre de triangles et la taille du codage brut (octets), qui est la version la plus compacte de codage naïf binaire :

Définition 19. *Soit v le nombre de sommets, t le nombre de triangles et p la précision pour chaque coordonnée de sommet (en bits). Alors la taille en bits de l'encodage naïf binaire le plus compact est :*

- $3.p.v$ *pour la géométrie*
- $3.t.log_2\ v$ *pour la connectivité.*

Soit au total : $3.p.v + 3.t.log_2\ v$ *bits.*

Le taux de compression est ensuite présenté. Il s'agit du ratio entre la taille brute et la taille du fichier FEC. Enfin, à titre indicatif, se trouve le nombre total de *nSP-cellules*. La taille des fichiers est divisée par un facteur compris entre 3 et 5 pour les maillages triangulaires de petite taille (inférieure au million de triangles), et par un facteur compris entre 6,5 et 12 pour les fichiers de plus grande taille. Nous constatons que le taux de compression est d'autant plus élevé que le nombre de triangles est grand. Cela s'explique par le fait que pour une précision donnée, le coût d'encodage *par sommet* de la géométrie est d'autant plus faible que le nombre de sommets est élevé. En effet, plus le nombre de sommets est grand, plus la proportion de code géométrique de valeur **2** (les deux demi-cellules sont non-vides) est élevée. Or, un code **2** est « rentable » puisqu'il crée un nouveau sommet tout en raffinant les coordonnées du sommet d'origine, alors qu'un code **0** ou **1** se contente de raffiner les coordonnées d'un sommet existant. La compression de la géométrie est donc d'autant plus efficace que l'occupation de l'espace par les sommets est dense.

La table 4.3 fournit quant à elle les temps de compression, l'en-tête des colonnes se référant aux étapes détaillées dans la section 3.5. Pour mémoire :

- 3.4 = Construction out-of-core
- 3.5.3 = Traitement de chaque cellule du *nSP-arbre*
- 3.5.4 = Ecriture finale

Les deux premières étapes, à savoir la construction du *nSP-arbre* (étape 3.4) et le calcul des séquences de codes (étape 3.5.3), dominent largement le processus en terme de coût. La cohérence spatiale du fichier en entrée a un impact sur la durée de la phase 3.5.3, ce qui explique qu'elle soit plus longue sur le maillage *T8* que sur *Lucy*, alors que *Lucy* contient plus de triangles. Le calcul des

TABLE 4.2 – Résultats de compression avec $p = 16$ bits par coordonnée, $n = 4$, $N_{min} = 8000$ et $p_r = 7$

Modèle	Fichier en entrée (tailles en Mo)			Fichier comprimé	
	Triangles	Ply	Brut	Ratio	*n*SP-C.
Bali*	5 435 004*	-	33	2,78	10 753
Wallis*	6 763 447*	-	41	3,02	13 505
Bunny	69 451	3	0,6	3,37	65
Armadillo	345 932	7	3,3	3,95	1 217
Dragon	866 508	34	8,7	4,64	1 281
David 2mm	8 250 977	173	93	8,05	25 601
UNC Powerplant	12 388 092	503	174	7,85	38 209
EDF T5	14 371 932	388	166	7,36	27 521
EDF T8	22 359 212	693	263	6,82	31 105
Lucy	27 595 822	533	328	7,35	114 113
David 1mm	54 672 488	1 182	671	8,81	181 569
Saint Matthieu	372 422 615	7 838	4 686	11,57	353 473

(*) Ces modèles sont des nuages de points non structurés (la colonne *Triangles* indique le nombre de points)

codes (3.5.3) est linéaire dans le nombre de sommets et bénéficie de notre implémentation parallélisée. Enfin, la table détaille l'utilisation de la mémoire (dont l'espace disque temporairement occupé), attestant des capacités *out-of-core* de la méthode. La taille occupée en mémoire vive augmente très légèrement avec la taille des maillages, car certaines données gardées en mémoire centrale, comme le *nSP-arbre*, sont de taille plus importante. Cependant, elle ne dépasse pas les 320 Mo, ce qui laisse envisager le traitement de maillages bien plus volumineux encore, afin d'exploiter les 4 Go de RAM qui sont désormais la norme sur les ordinateurs courants. La taille utilisée temporairement sur le disque dur — principalement par les fichiers RWI, RWP, SCT — est linéaire dans le nombre de sommets. Ces fichiers sont effacés à la fin du processus de compression. Afin de montrer le bon comportement de la méthode sur les nuages de points — seules les codes de géométrie sont utilisés —, deux exemples de modèles non structurés ont été ajoutés à notre jeu d'objets 3D : *Bali* et *Wallis*.

La table 4.4 montre les résultats de notre algorithme en termes de *bits par sommet*, comparés à ceux de :

- [TG98] : méthode de Touma et Gotsman, référence en compression *in-core* mono-résolution,
- [GD02] : algorithme *in-core* multi-résolution de Gandoin et Devillers,
- [IG03] : méthode *out-of-core* mono-résolution de Isenburg et Gumhold,
- [YL07] : algorithme *out-of-core* mono-résolution de Yoon et Lindstrom, permettant l'accès direct au maillage.

Les trois premières sont des méthodes de compression pure, tandis que la dernière fournit un compromis entre compression et accès aléatoire, très utile pour les applications qui nécessitent le parcours du maillage, mais peu adaptée à

TABLE 4.3 – Performances de compression avec $p = 16$ bits par coordonnée, $n = 4$, $N_{min} = 8000$ et $p_r = 7$

Modèle	Triangles	Temps de compression (mm:ss)				Mem (Mo)	
		3.4	3.5.3	3.5.4	Total	RAM	DD
Bali*	5 435 004*	01:43	00:32	00:06	02:22	173	121
Wallis*	6 763 447*	02:08	00:40	00:07	02:55	174	147
Bunny	69 451	00:01	00:02	00:00	00:03	108	1
Armadillo	345 932	00:02	00:03	00:01	00:06	121	6
Dragon	866 508	00:07	00:10	00:01	00:18	151	15
David 2mm	8 250 977	00:50	00:54	00:09	01:53	290	136
UNC Powerplant	12 388 092	04:26	01:52	00:09	06:27	291	290
EDF T5	14 371 932	01:28	01:50	00:16	03:34	290	254
EDF T8	22 359 212	02:47	03:07	00:26	06:20	289	404
Lucy	27 595 822	03:10	02:48	00:33	06:31	299	522
David 1mm	54 672 488	06:01	04:58	00:57	11:56	304	1035
Saint Matthieu	372 422 615	40:50	33:13	06:26	1:20:29	318	6510

(*) Ces modèles sont des nuages de points non structurés (la colonne *Triangles* indique le nombre de points)

la visualisation interactive, car aucun niveau de détail n'est disponible. Deux méthodes de référence en terme de visualisation, [CGG+04] et [GM05], ont été ajoutées à titre de comparaison. Malheureusement, la comparaison avec [CLW+06] (seule méthode de compression *out-of-core* et progressive à notre connaissance) est impossible puisque les auteurs indiquent seulement les taux obtenus pour encoder la connectivité.

TABLE 4.4 – Comparaison des taux de compression en *bits par sommet*

Modèles	Sommets	p	TG98	IG03	GD02	YL07	CGG+04	GM05	CHuMI
Bunny	35 947	12	-	-	17,8	-	-	-	27,0
Horse	19 851	12	19,3	-	20,3	-	-	-	29,3
Dino	14 050	12	19,8	-	-	31,8	-	-	28,7
Igea	67 173	12	17,2	-	-	25,0	-	-	25,9
David 2mm	4 128 028	16	-	14,0	-	-	306,2	-	22,3
Powerplant	10 890 300	16	-	14,1	-	-	-	-	16,3
Lucy	13 797 912	16	-	16,5	-	-	-	-	25,9
David 1mm	27 405 599	16	-	13,1	-	-	282,3	-	22,2
St. Matthieu	166 933 776	16	-	10,7	-	22,9	282,1	508,0	19,4

Par rapport aux méthodes qui ne permettent pas de visualisation interactive — résultats contenus dans les trois premières colonnes —, un surcoût compris entre 15% et 80%, avec une moyenne de 51%, est observé, principalement dû à la redondance introduite par le partitionnement de l'espace (5—7% des triangles originaux sont dupliqués, cf. section 4.2.3), au codage de l'orientation des triangles, et au fait que certaines prédictions complexes n'ont pas été intégrées pour garantir un rendu rapide.

La comparaison la plus intéressante est celle faite avec la méthode de com-

pression de Yoon et Lindstrom [YL07]. Pour rappel, il s'agit d'un algorithme de compression *out-of-core* de maillages triangulaires qui permet un accès aléatoire optimisé au maillage, directement à partir du fichier comprimé. Même si cet objectif diffère du nôtre, notre point commun est de rechercher un compromis entre compression et accès aux données. La comparaison est plus équitable, et les résultats obtenus sont proches, ce qui confirme que le surcoût induit par notre contribution est acceptable.

Enfin, les résultats obtenus par les deux méthodes de visualisation non compressives [CGG+04] et [GM05] sont indiqués, non pas à titre de réelle comparaison — ces méthodes n'ont pas la prétention de comprimer les données —, mais pour montrer concrètement le gain apporté par notre visualiseur en termes de taille de fichier : comparée à [CGG+04], la taille des fichiers est divisée en moyenne par 13, et comparée à [GM05], par 26.

Dans l'ensemble, si notre méthode reste moins performante que les travaux à vocation purement compressive, les résultats obtenus sont très honorables, si l'on adopte le point de vue de la recherche de compromis. Reste à mesurer les performances de la décompression et de la visualisation pour tirer de réelles conclusions.

4.2.5 Décompression et visualisation

Nous présentons ici les performances de notre méthode en termes de décompression et visualisation. Comme énoncé dans la section 3.6, ces deux tâches sont indissociables car la décompression, effectuée *à la volée*, est guidée par les mouvements de la caméra. Tous les résultats ont été produits avec une erreur maximale à l'écran de 1 pixel.

Pour une première appréciation de la méthode, les figures 4.4, 4.5, 4.6, 4.7 et 4.8 présentent respectivement quelques exemples de rendus à différents niveaux de détail des modèles suivants : Dragon, EDF T8, Lucy, David 1mm, Saint Matthieu. La table 4.5 présente les temps observés pour obtenir une vue donnée par raffinement ou simplification de la vue précédente (ou depuis le début pour la visualisation de la vue (a)). Ces temps peuvent paraître élevés, mais il doit être noté que lors de la navigation, la transition d'une vue à l'autre est progressive et les mouvements de caméra de l'utilisateur sont généralement suffisamment lents pour permettre un raffinement fluide. C'est pourquoi la meilleure façon d'apprécier le comportement de l'algorithme est de manipuler le visualiseur [JGA09b] ou de regarder les vidéos capturées [JGA09a].

La figure 4.9 présente d'autres indicateurs de la vitesse de décodage et de rendu des données. Il s'agit du nombre d'images et de triangles affichés par seconde pendant le déroulement de la vidéo du Saint Matthieu [JGA09a]. Notre décodeur est capable d'afficher jusqu'à 200 millions de triangles par seconde

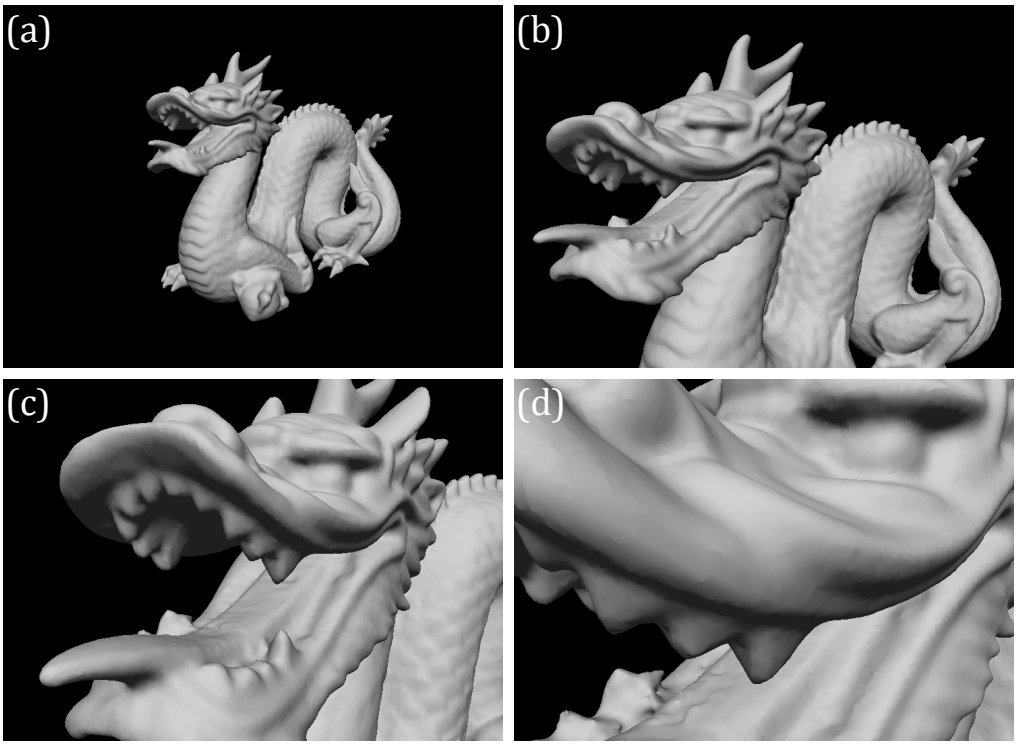


FIGURE 4.4 – Exemple de visualisation du modèle *Dragon*

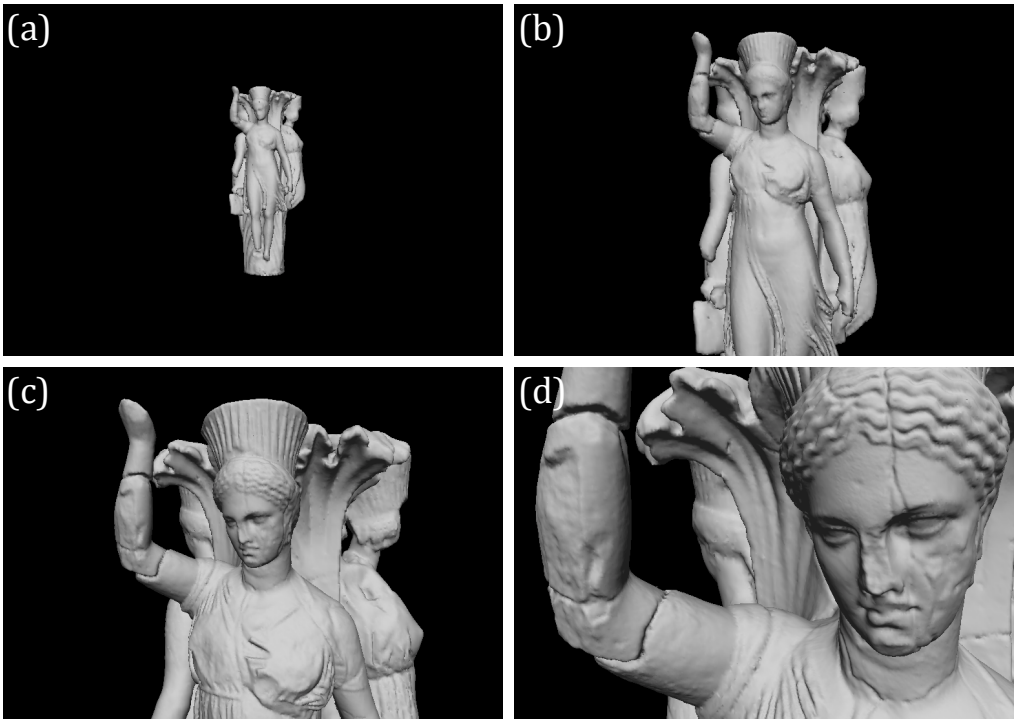


FIGURE 4.5 – Exemple de visualisation du modèle *T8* fourni par EDF

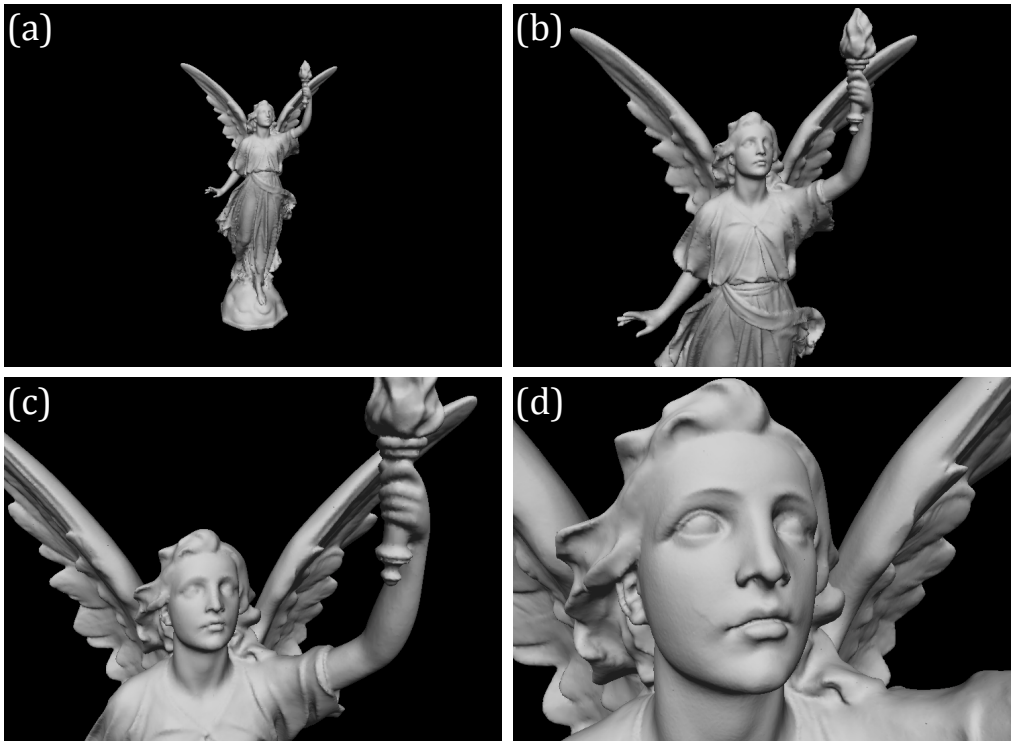


FIGURE 4.6 – Exemple de visualisation du modèle *Lucy*

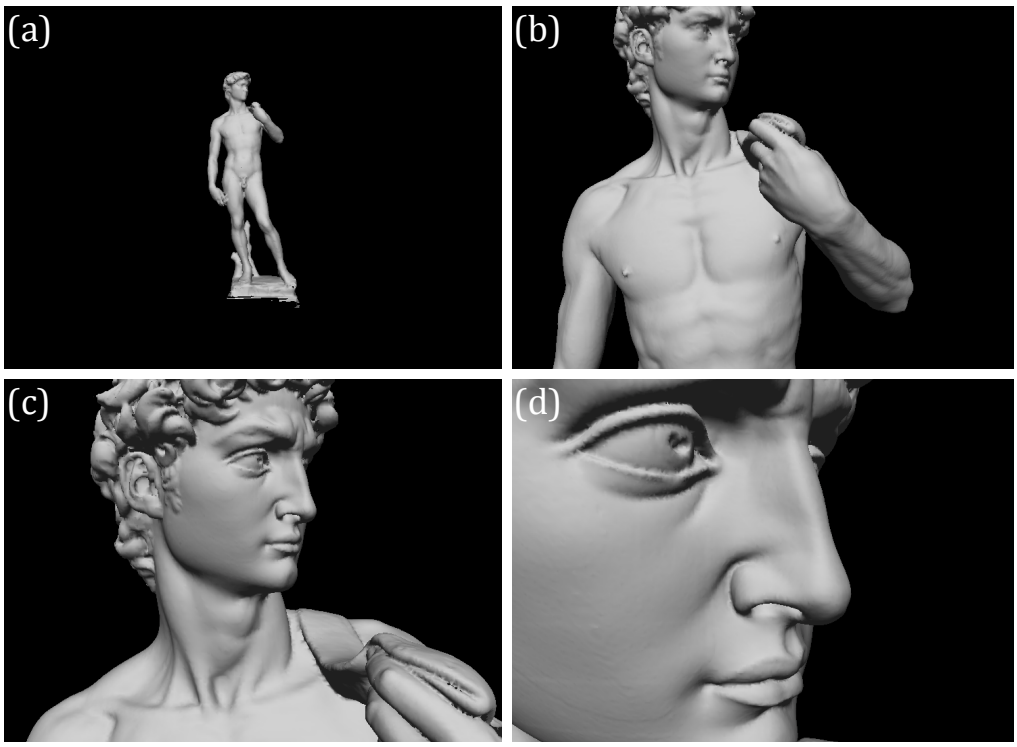


FIGURE 4.7 – Exemple de visualisation du David de Michel-Ange (précision du scanner laser : 1mm)

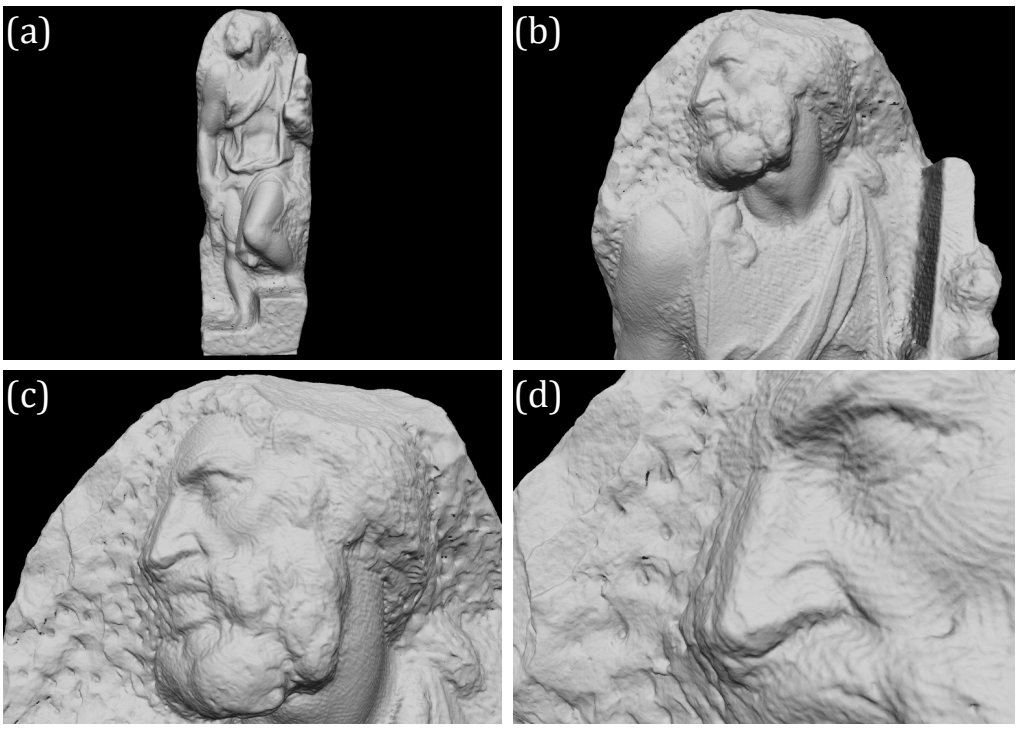


FIGURE 4.8 – Exemple de visualisation du Saint Matthieu de Michel-Ange

(TPS), avec une moyenne d'environ 173 millions quand il tourne à plein régime (entre $t = 30$ et $t = 65$ s. sur la figure 4.9). A titre de comparaison, [Lin03] affiche jusqu'à 3 Mtps, [CGG+04] affiche une moyenne de 70 Mtps, et [GM05] une moyenne de 45 Mtps. En terme de *FPS*, la moyenne se situe autour de 150 images par seconde lorsque le visualiseur est très sollicité (nombre de triangles à l'écran de l'ordre du million), et cette valeur peut atteindre plus de 500 lorsque le nombre de triangles affichés est plus faible (quelques centaines de milliers).

Bien entendu, cette comparaison n'a de sens qu'en tenant compte de l'année de publication de chaque méthode et de la progression des performances du matériel, et plus généralement, des remarques faites dans la section 4.2.1. De plus, [CGG+04] et [GM05] raffinent le modèle sur des critères de distance et de direction de la caméra, alors que *CHuMI* utilise seulement le critère de distance. Nous rappelons également que ces méthodes de visualisation ne compriment pas les données : par exemple, [CGG+04] et [GM05] produisent respectivement des fichiers de taille supérieure aux fichiers bruts originaux de 10% et 80% en moyenne.

Nous pensons qu'un indicateur plus fiable de la vitesse de décompression des modèles est donné par le nombre de triangles que le décodeur est capable de créer ou de supprimer par seconde. La figure 4.10 montre ces données dans le cas de la vidéo du Saint Matthieu [JGA09a]. Lors d'un zoom avant, peuvent être décodés et créés jusqu'à 310 000 triangles en une seconde, tandis qu'un

TABLE 4.5 – Temps de raffinement en secondes, avec, dans l'ordre des colonnes : temps pour afficher (a), temps de (a) à (b), de (b) à (c), de (c) à (d), de (d) à (c), de (c) à (b), et enfin de (b) à (a)

Modèle	Init.	Zoom avant			Zoom arrière		
	(a)	(b)	(c)	(d)	(c)	(b)	(a)
Dragon	0,6	0,5	0,4	0,3	0,2	0,3	0,2
EDF T8	0,4	0,9	0,9	3,8	0,8	0,5	1,2
Lucy	0,4	0,8	0,5	2,6	0,6	0,3	0,2
David 1mm	1,0	0,7	1,0	1,6	0,6	0,5	0,3
Saint Matthieu	0,7	1,2	2,0	1,7	0,8	0,6	0,4

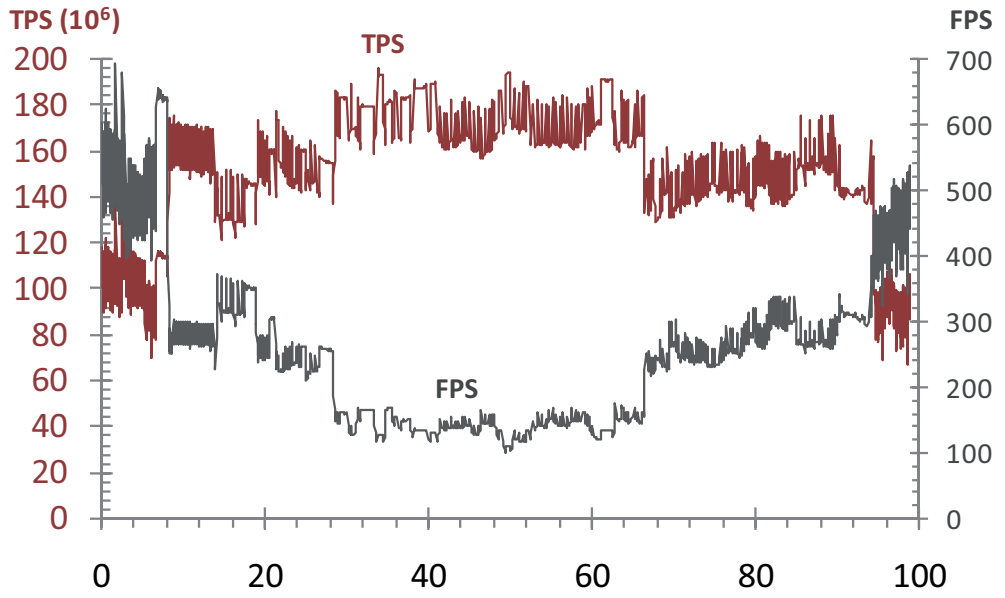


FIGURE 4.9 – Millions de triangles par seconde et images par seconde affichés au cours du temps pendant la vidéo du Saint Matthieu [JGA09a]

zoom arrière peut mener à la destruction d'un maximum de 565 000 triangles par seconde. Malheureusement, nous ne disposons pas de ces données pour les méthodes de visualisation non compressives. On peut cependant s'attendre à des résultats qui leur seraient favorables, car dans leur cas, les données n'ont qu'à être lues, sans travail de décodage. Cependant, l'écart grandissant entre puissance calculatoire et temps d'accès aux données pourrait nous réserver des surprises dans un avenir proche.

Nous tenons à préciser que les petits trous polyédriques qui peuvent être observés sur les captures et vidéos ne sont pas des artefacts provoqués par la méthode. Ils proviennent des maillages originaux et correspondent généralement à des zones inaccessibles au laser du scanner. Par ailleurs, le lecteur attentif notera que les transitions entre les niveaux de précision k et $k+1$ d'une *nSP-cellule*, visibles sur les vidéos, ont lieu relativement rarement. Plus précisé-

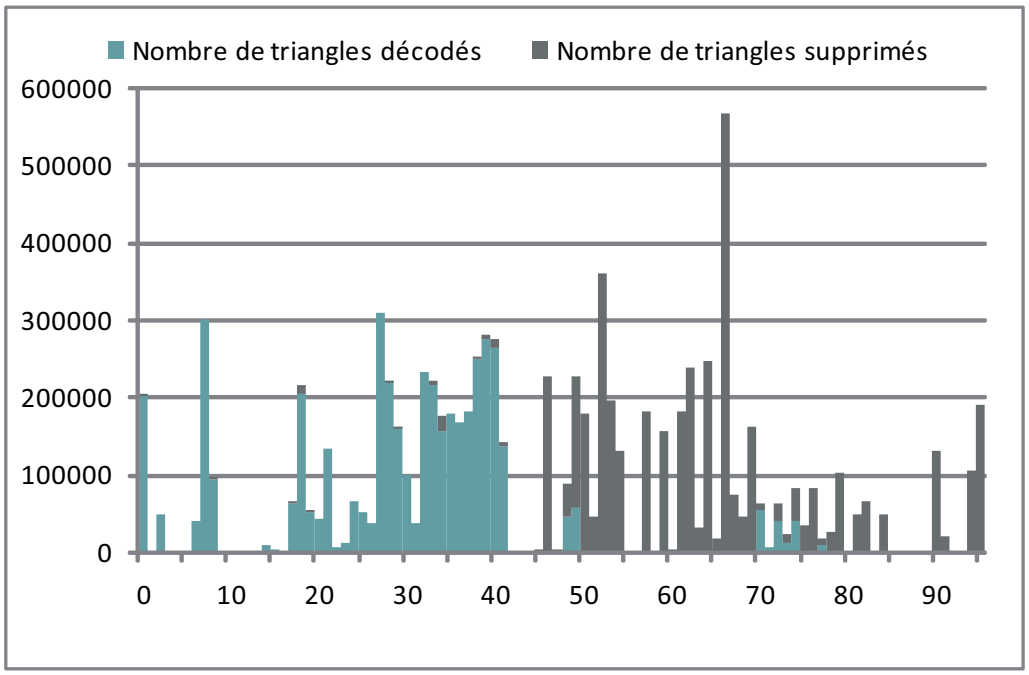


FIGURE 4.10 – Histogramme empilé montrant le nombre de triangles créés et supprimés par seconde au cours du temps pendant la vidéo du Saint Matthieu accompagnant le mémoire [JGA09a]

ment, elles se produisent lorsque le zoom sur la *nSP-cellule* est doublé, puisque chaque bit additionnel sur les coordonnées des points double la précision d'affichage. Par conséquent, si une telle transition ne se produit pas durant un zoom, cela ne signifie pas que le processus de raffinement est bloqué mais que le seuil n'est pas encore atteint.

4.3 *Streaming* à travers un réseau

Le succès des sites web de diffusion de média en *streaming*, tels *YouTube* ou *Deezer*, montre que la généralisation des connexions haut-débit rend la consultation de données *via* Internet de plus en plus fréquente. CHuMI Viewer n'échappe pas à la règle, et nous avons inclus la possibilité de consulter un maillage à distance. La figure 4.11 présente l'architecture générale simplifiée du visualiseur, lorsqu'il est utilisé à travers un réseau.

Le fichier comprimé par notre méthode est structuré en blocs, chacun correspondant à la séquence codante d'une *nSP-cellule*. Le principe général du fonctionnement en *streaming* consiste à recréer progressivement le fichier complet, en téléchargeant les blocs dans un ordre guidé par les besoins de la visualisation. La première étape consiste à télécharger les informations générales, *i.e.* l'en-tête du fichier, qui est de petite taille et donc rapide à télécharger. Par la suite, lorsque le décodeur a besoin d'un bloc, il teste si celui-ci est déjà présent

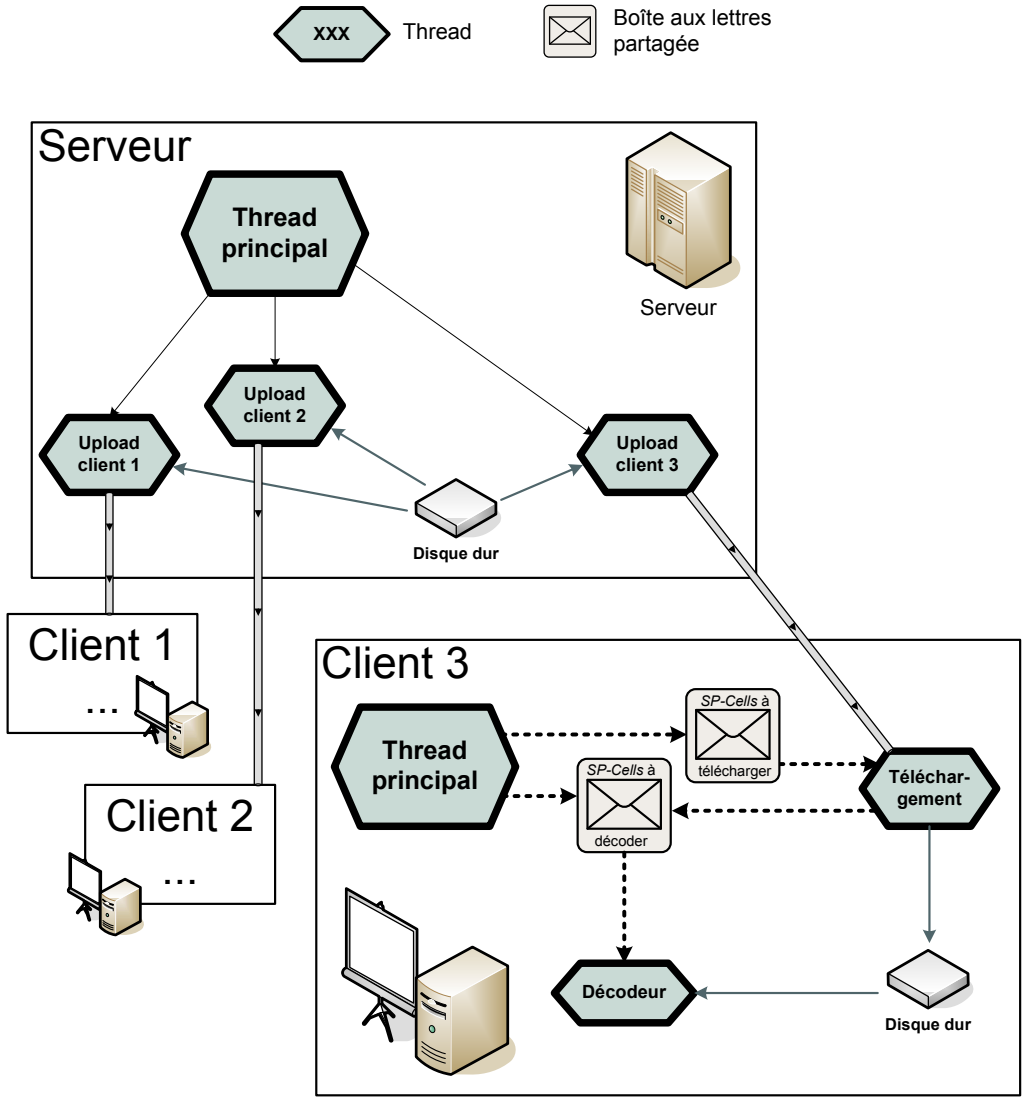


FIGURE 4.11 – Architecture générale simplifiée de CHuMI Viewer utilisé en mode *streaming*

sur le disque dur. Si ce n'est pas le cas, le décodeur télécharge le bloc depuis le serveur, et l'écrit ensuite sur le disque dur. Ainsi, aucune partie du fichier n'est téléchargée deux fois. A la fin de la visualisation, le fichier est plus ou moins complet selon l'exhaustivité de la navigation réalisée.

Bien entendu, ce mode réseau est agrémenté des fonctions de base d'un tel logiciel, telles que le choix du serveur hébergeant les modèles 3D comprimés *via* son URL, ou encore le téléchargement de la liste des fichiers disponibles sur le serveur.

4.4 Conclusion

CHuMI Viewer, disponible publiquement sur le web [JGA09b], a permis d'implémenter, tester, mesurer et optimiser la méthode présentée dans ce mémoire. Concrétisation appréciable de ce travail, ce logiciel a été installé au *C2RMF*, laboratoire associé au musée du Louvre, qui souhaite l'intégrer dans sa base de données *Eros*. Il fournira aux chercheurs de cette équipe un outil précieux pour l'exploration visuelle des œuvres d'art numérisées.

Conclusion et perspectives

L'objectif initial du travail présenté ici était de coupler les avantages de la compression sans perte avec les possibilités de visualisation adaptative par niveaux de détail d'objets 3D trop volumineux pour tenir en mémoire centrale. Or, dans l'état de l'art, le prétraitement des maillages à des fins de visualisation implique une réorganisation complète des données qui induit une augmentation souvent considérable de la taille des fichiers et entrave le stockage et la transmission des modèles.

La structure de données hiérarchique conçue et développée à l'occasion de cette thèse réduit significativement la taille des objets tout en offrant à l'utilisateur final les pleines possibilités d'une navigation interactive. En supplément de cette double capacité, la caractéristique *out-of-core* des algorithmes autorisent le traitement de maillages contenant plusieurs centaines de millions de triangles (par exemple, le Saint Matthieu de Michel-Ange).

La compression est rapide et efficace, l'encodeur requiert seulement 1h20 pour comprimer 372 millions de triangles sur un seul PC, là où certaines méthodes comparables mobilisent une grappe de 16 PC durant plusieurs heures. Avec un facteur de compression situé autour de 7, la taille des fichiers générés induit un stockage facilité et un transfert rapide, et autorise l'utilisation en *streaming* grâce à un module client-serveur. Ce gain en espace disque favorise l'exploitation de la puissance de calcul des processeurs, en perpétuelle évolution et désormais multi-cœurs, et réduit les accès mémoire dont la progression en termes de performance est faible et sporadique. L'implémentation parallélisée est tournée vers l'avenir, qui promet la multiplication des unités de calcul, à l'image de l'architecture des cartes graphiques.

Le décodeur, couplé au visualiseur, autorise une navigation fluide et interactive, afin d'accéder rapidement et à la volée à tous les détails des objets les plus précis. Ici aussi, le parallélisme et l'optimisation des algorithmes et des accès mémoire permettent d'atteindre de bonnes performances et d'assurer une visualisation réactive.

Bien entendu, nous avons à l'esprit de nombreuses perspectives pour pour-

suivre ces travaux. En premier lieu, bien que théoriquement capable de traiter les maillages de toutes dimensions, les algorithmes requièrent un travail d'implémentation et d'optimisation supplémentaire pour pouvoir gérer correctement les maillages volumiques. Par exemple, les opérateurs de contraction d'arête et de fusion de sommets doivent être étendus de façon efficace aux tétraèdres. De même, la prise en compte des occlusions entre simplexes devient primordiale dans le cas volumique.

D'autre part, nous pensons qu'il est possible d'améliorer significativement les taux de compression, en développant des techniques de prédiction efficaces et rapides. Avec les gains à venir en termes de puissance de calcul, il deviendra rapidement possible d'effectuer des calculs complexes sans pénaliser la vitesse de décodage.

Notre méthode ne traite actuellement que la géométrie et la connectivité des maillages. Il serait intéressant d'encoder les couleurs et/ou les coordonnées de texture. Ce problème est un sujet de recherche à part entière si l'on souhaite obtenir des résultats compétitifs : compression efficace, anti-aliassage, etc.

Concernant la visualisation, il serait bien entendu appréciable de réduire encore la latence de mise à jour du maillage. Pour cela, trois chemins principaux peuvent être explorés : optimisation *cache-oblivious* — *i.e.* exploitant la mémoire cache du CPU sans que la taille de celle-ci soit connue —, utilisation de structures de données exploitant l'architecture des cartes graphiques, telles que des niveaux de détail gérés à l'aide de sous-ensembles de triangles, et enfin usage intensif du GPU par la programmation de *shaders*.

Bibliographie

[AD01] P. Alliez and M. Desbrun. Valence-driven connectivity encoding for 3d meshes. In *Eurographics 2001 Conference Proc.*, 2001. 30

[AG03] Pierre Alliez and Craig Gotsman. Recent advances in compression of 3d meshes. In *In Proc. of the Sym. on Multiresolution in Geometric Modeling*. Springer, 2003. 28

[Ami04] Said Amir. Introduction to arithmetic coding theory and practice. Technical report, HP Labs report HPL-2004-76, April 2004. 78

[BY95] J-D. Boissonnat and M. Yvinec. *Géométrie algorithmique*. Ediscience international, Paris, 1995. 7

[BY98] J.-D. Boissonnat and M. Yvinec. *Algorithmic geometry*. Cambridge University Press, UK, 1998. traduit de la version française (Ediscience international) par Hervé Brönnimann. 7

[Car76] M. P. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976. 6

[CGG+03a] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. BDAM – batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22(3) :505–514, September 2003. Proc. Eurographics 2003. 45

[CGG+03b] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Planet-sized batched dynamic adaptive meshes (p-bdam). In *VIS '03 : Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 20, Washington, DC, USA, 2003. IEEE Computer Society. 45

[CGG+04] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Adaptive tetrapuzzles : Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Transactions on Graphics*, 23(3) :796–803, 2004. x, 37, 38, 40, 42, 46, 102, 103, 106

[CGG+05] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Batched multi triangulation. In *Proceedings IEEE Visualization*, pages 207–214, Conference held in Minneapolis, MI, USA, October 2005. IEEE Computer Society Press. x, 41, 42, 43

[CH87] G. V. Cormack and R. N. S. Horspool. Data compression using dynamic markov modelling. *Comput. J.*, 30(6) :541–550, 1987. 15

[Cho97] M. M. Chow. Optimized geometry compression for real time rendering. In *IEEE Visualization 97 Conference Proc.*, pages 347–354, 1997. 26, 27

[CLW+06] K. Cai, Y. Liu, W. Wang, H. Sun, and E. Wu. Progressive out-of-core compression based on multi-level adaptive octree. In *ACM international Conference on VRCIA*, pages 83–89. ACM Press, New York, 2006. 34, 102

[CMRS03] Paolo Cignoni, Claudio Montani, Claudio Rocchini, and Roberto Scopigno. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*, 9(4) :525–537, 2003. 32, 33

[COLR99] D. Cohen-Or, D. Levin, and O. Remez. Progressive compression of arbitrary triangular meshes. In *IEEE Visualization 99 Conf. Proc.*, pages 67–72, 1999. 30

[CPCS08] Marco Callieri, Federico Ponchio, Paolo Cignoni, and Roberto Scopigno. Virtual inspector : A flexible visualizer for dense 3d scanned models. *IEEE Comput. Graph. Appl.*, 28(1) :44–54, 2008. 42

[CRS98] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro : Measuring error on simplified surfaces. *Comput. Graph. Forum*, 17(2) :167–174, 1998. 23, 84

[CW84] John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32 :396–402, 1984. 15

[Dee95] M. Deering. Geometry compression. In *SIGGRAPH 95 Conference Proc.*, pages 13–20, 1995. 26, 27

[ESjCoo] Jihad El-Sana and Yi jen Chiang. External memory view-dependent simplification. *Computer Graphics Forum*, 19 :139–150, 2000. 36

[ESV96] F. Evans, S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *IEEE Visualization 96 Conference Proc.*, 1996. 26, 27

[ESV99] J. El-Sana and A. Varshney. Generalized view-dependent simplification. *Computer Graphics Forum*, 18, No. 3 :83–94, 1999. 36

[GD02] P-M. Gandoin and O. Devillers. Progressive lossless compression of arbitrary simplicial complexes. In *ACM SIGGRAPH Conference Proc.*, 2002. x, 2, 30, 49, 52, 53, 54, 55, 56, 60, 68, 75, 76, 96, 101

[GGK02] Craig Gotsman, Stefan Gumhold, and Leif Kobbelt. *Simplification and compression of 3d meshes*, pages 319–361. Springer, 2002. 28

[GM05] E. Gobbetti and F. Marton. Far voxels : a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. In *ACM SIGGRAPH*, pages 878–885. ACM Press, 2005. x, 39, 41, 102, 103, 106

[GMC+06] Enrico Gobbetti, Fabio Marton, Paolo Cignoni, Marco Di Benedetto, and Fabio Ganovelli. C-bdam - compressed batched dynamic adaptive meshes for terrain rendering. *Computer Graphics Forum*, 25(3), sep 2006. To appear in Eurographics 2006 conference proceedings. x, 45, 47

[GS98] S. Gumhold and W. Strasser. Real time compression of triangle mesh connectivity. In *SIGGRAPH 98 Conference Proc.*, pages 133–140, 1998. 27

[HDD+93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *SIGGRAPH 93 Conference Proc.*, 1993. 52, 75

[Hop96] H. Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proc.*, 1996. 29, 43

[HSH09] L. Hu, P. Sander, and H. Hoppe. Parallel view-dependent refinement of progressive meshes. In *ACM Symposium on Interactive 3D Graphics and Games 2009*, 2009. 43, 45

[Huf52] D. A. Huffman. A method for the construction of minimum-redundancy codes. In *Proc. Inst. Electr. Radio Eng.*, September 1952. 11

[IG03] M. Isenburg and S. Gumhold. Out-of-core compression for gigantic polygon meshes. In *SIGGRAPH 2003 Conference Proc.*, 2003. ix, 28, 32, 33, 34, 101

[IL05] Martin Isenburg and Peter Lindstrom. Streaming meshes. In *IEEE Visualization*, pages 231–238, 2005. x, 33, 34

[ILS05] Martin Isenburg, Peter Lindstrom, and Jack Snoeyink. Streaming compression of triangle meshes. In *SGP '05 : Proceedings of the third Eurographics symposium on Geometry processing*, page 111, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association. 34, 47

[isb85] *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std* 754 – 1985. New York, NY, 1985. Reprinted in SIGPLAN Notices, 22(2) :9–25, 1987. 16

[JGA09a] Clément Jamin, Pierre-Marie Gandoin, and Samir Akkouche. *CHuMI videos*, 2009. http://clementjamin.free.fr/CG/. xii, 97, 103, 106, 107, 108

[JGA09b] Clément Jamin, Pierre-Marie Gandoin, and Samir Akkouche. *CHuMI Viewer*, 2009. http://clementjamin.free.fr/CHuMI/. 97, 103, 110

[KBG02] Zachi Karni, Alexander Bogomjakov, and Craig Gotsman. Efficient compression and rendering of multi-resolution meshes. In *VIS '02 : Proceedings of the conference on Visualization '02*, pages 347–354, Washington, DC, USA, 2002. IEEE Computer Society. 30

[KR99a] D. King and J. Rossignac. Guaranteed 3.67v bit encoding of planar triangle graphs. In *Canadian Conference on Computational Geometry Proc.*, 1999. 27

[KR99b] D. King and J. Rossignac. Optimal bit allocation in compressed 3D models. *CGTA : Computational Geometry : Theory and Applications*, 14, 1999. 30

[LH04] Frank Losasso and Hugues Hoppe. Geometry clipmaps : terrain rendering using nested regular grids. In *SIGGRAPH '04 : ACM SIGGRAPH 2004 Papers*, pages 769–776, New York, NY, USA, 2004. ACM. x, 44, 45, 46

[Lin00] Peter Lindstrom. Out-of-core simplification of large polygonal models. In *SIGGRAPH '00 : Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 259–262, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 32

[Lin03] P. Lindstrom. Out-of-core construction and visualization of multiresolution surfaces. In *Sym. on Interactive 3D Graphics*, pages 93–102. ACM Press, 2003. x, 36, 38, 39, 106

[LK98] J. Li and C.-C. Jay Kuo. Progressive coding of 3d graphic models. *IEEE Computer Graphics*, 86, 1998. 30

[LS01] Peter Lindstrom and Cláudio T. Silva. A memory insensitive technique for large model simplification. In *VIS '01 : Proceedings of the conference on Visualization '01*, pages 121–126, Washington, DC, USA, 2001. IEEE Computer Society. 32

[NBB04] R. Namane, F. O. Boumghar, and K. Bouatouch. Qsplat compression. In *ACM AFRIGRAPH*, pages 15–24. ACM Press, New York, 2004. 44

[NYC05] Jose Luis Nunez-Yanez and Vassilios A. Chouliaras. A configurable statistical lossless compression core based on variable order markov modeling and arithmetic coding. *IEEE Transactions on Computers*, 54(11) :1345–1359, 2005. 15

[PH97] J. Popović and H. Hoppe. Progressive simplicial complexes. In *SIGGRAPH 97 Conference Proc.*, 1997. 52, 75

[PK05] J. Peng and C-C. J. Kuo. Geometry-guided progressive lossless 3d mesh coding with octree decomposition. In *ACM SIGGRAPH Conference Proc.*, 2005. 31, 53

[PR00] R. Pajarola and J. Rossignac. Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics*, 6(1) :79–93, January–March 2000. 30

[Pup96] Enrico Puppo. Variable resolution terrain surfaces. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 202–210. Carleton University Press, 1996. 41

[RB93] J. Rossignac and P. Borrel. *Geometric Modeling in Computer Graphics*, chapter Multi-Resolution 3D Approximations for Rendering Complex Scenes, pages 455–465. Springer-Verlag, July 1993. 32

[Ris76] Jorma Rissanen. Generalized kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20(3) :198–203, 1976. 12

[Ris83] J. Rissanen. A universal data compression system. *IEEE Transactions on Information Theory*, 29, 1983. 12

[RJ79] Jorma Rissanen and Glen G. Langdon Jr. Arithmetic coding. *IBM Journal of Research and Development*, 23(2) :149–162, 1979. 12

[RL00] S. Rusinkiewicz and M. Levoy. Qsplat : a multiresolution point rendering system for large meshes. In *Conf. on Computer Graphics and Interactive Techniques*, pages 343–352. ACM Press, New York, 2000. 36

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27 :379–423, 623–656, 1948. 10

[SS82] James A. Storer and Thomas G. Szymanski. Data compression via textual substitution. *J. ACM*, 29(4) :928–951, 1982. 11

[TG98] C. Touma and C. Gotsman. Triangle mesh compression. In *Graphics Interface 98 Conference Proc.*, pages 26–34, 1998. 27, 30, 101

[TGHL98] G. Taubin, A. Guéziec, W. Horn, and F. Lazarus. Progressive forest split compression. In *SIGGRAPH 98 Conference Proc.*, pages 123–132, 1998. 30

[TR98] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2), 1998. 27, 30

[Tut62] W. Tutte. A census of planar triangulation. *Canadian Journal of Mathematics*, 14, 1962. 26

[Wel84] T. A. Welch. A technique for high-performance data compression. *Computer*, 17(6) :8–19, 1984. 11

[WNC87] I.H. Witten, R. Neal, and J.G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6) :520–540, 1987. 12

[XHM99] X. Xiang, M. Held, and J. D. B. Mitchell. fast and effective stripification of polygonal surface models. In *ACM Symposium on Interactive 3D Graphics Proc.*, 1999. 26

[YL07] S. Yoon and P. Lindstrom. Random-accessible compressed triangle meshes. *IEEE Trans. on Visualization and Computer Graphics*, 13(6) :1536–1543, 2007. 46, 101, 103

[YSGM04] Sung-Eui Yoon, Brian Salomon, Russell Gayle, and Dinesh Manocha. Quick-vdr : Interactive view-dependent rendering of massive models. In *Proc. of Visualization*, pages 131–138. IEEE Computer Society, 2004. 37

[ZL77] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3) :337–343, 1977. 11

[ZL78] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5) :530–536, 1978. 11

Publications

Revue Internationale avec comité de lecture

CHuMI Viewer : Compressive Huge Mesh Interactive Viewer. C. Jamin, P.-M. Gandoin, S. Akkouche. Computer & Graphics, 2009.
http://dx.doi.org/10.1016/j.cag.2009.03.029
http://clementjamin.free.fr/CG/

Revue Nationale avec comité de lecture

Compression out-of-core pour la visualisation interactive de maillages volumineux. C. Jamin, P.-M. Gandoin, S. Akkouche. REFIG (Revue Electronique Francophone d'Informatique Graphique), 2009. (à paraître)

Conférence Nationale

Compression out-of-core pour la visualisation interactive de maillages volumineux. C. Jamin, P.-M. Gandoin, S. Akkouche. AFIG, Toulouse, 2008.

Rapport de recherche

A Compact Data Structure For The Navigation Into Arbitrary Meshes. C. Jamin, P.-M. Gandoin, S. Akkouche. Rapport de recherche RR-LIRIS-2008-014, 2008.

Logiciel

CHuMI Viewer, C. Jamin. http://clementjamin.free.fr/CHuMI/

Algorithmes et structures de données compactes pour la visualisation interactive d'objets 3D volumineux

Résumé : Les méthodes de compression progressives sont désormais arrivées à maturité (les taux de compression sont proches des taux théoriques) et la visualisation interactive de maillages volumineux est devenue une réalité depuis quelques années. Cependant, même si l'association de la compression et de la visualisation est souvent mentionnée comme perspective, très peu d'articles traitent réellement ce problème, et les fichiers créés par les algorithmes de visualisation sont souvent beaucoup plus volumineux que les originaux. En réalité, la compression favorise une taille réduite de fichier au détriment de l'accès rapide aux données, alors que les méthodes de visualisation se concentrent sur la rapidité de rendu : les deux objectifs s'opposent et se font concurrence. A partir d'une méthode de compression progressive existante incompatible avec le raffinement sélectif et interactif, et uniquement utilisable sur des maillages de taille modeste, cette thèse tente de réconcilier compression sans perte et visualisation en proposant de nouveaux algorithmes et structures de données qui réduisent la taille des objets tout en proposant une visualisation rapide et interactive. En plus de cette double capacité, la méthode proposée est *out-of-core* et peut traiter des maillages de plusieurs centaines de millions de points. Par ailleurs, elle présente l'avantage de traiter tout complexe simplicial de dimension n, des soupes de triangles aux maillages volumiques.

Mots clés : Compression sans perte, Visualisation interactive, Maillages volumineux, *Out-of-core*

Algorithms and compact data structures for interactive visualization of gigantic 3D objects

Abstract : Progressive compression methods are now mature (obtained rates are close to theoretical bounds) and interactive visualization of huge meshes has been a reality for a few years. However, even if the combination of compression and visualization is often mentioned as a perspective, very few papers deal with this problem, and the files created by visualization algorithms are often much larger than the original ones. In fact, compression favors a low file size to the detriment of a fast data access, whereas visualization methods focus on rendering speed : both goals are opposing and competing. Starting from an existing progressive compression method incompatible with selective and interactive refinements and usable on small-sized meshes only, this thesis tries to reconcile lossless compression and visualization by proposing new algorithms and data structures which radically reduce the size of the objects while supporting a fast interactive navigation. In addition to this double capability, our method works out-of-core and can handle meshes containing several hundreds of millions vertices. Furthermore, it presents the advantage of dealing with any n-dimensional simplicial complex, which includes triangle soups or volumetric meshes.

Keywords : Lossless compression, Interactive visualization, Large meshes, Out-of-core