AFIG 2008 / Toulouse
IRIT Presse

Compression out-of-core pour la visualisation interactive de maillages

Clément Jamin, Pierre-Marie Gandoin et Samir Akkouche†

Abstract

Le prétraitement des maillages à des fins de visualisation interactive implique une réorganisation complète des données qui induit souvent une augmentation significative de la taille des fichiers. Cela nuit non seulement au stockage et à la transmission réseau, mais pourrait aussi, dans un futur proche, affecter l'efficacité de la visualisation elle-même, au vu de l'écart croissant entre la vitesse des processeurs et la lenteur relative des mémoires externes. Dans cet article, nous tentons de réconcilier compression sans perte et visualisation, grâce à une structure de donnée qui réduit la taille des objets tout en permettant une navigation interactive. En supplément de cette double capacité, cette méthode est out-of-core et peut ainsi traiter des maillages de plusieurs centaines de millions de triangles. En outre, tout complexe simplicial de dimension n est accepté en entrée, que ce soit une soupe de triangles ou un maillage volumétrique, et la compression distribue les données de façon à optimiser le ratio débit-distorsion. Les performances sont proches de l'état de l'art, aussi bien en terme de taux de compression qu'en terme de vitesse d'affichage, ce qui offre un compromis utile à de nombreuses applications.

Keywords: Compression sans perte, visualisation interactive, maillages volumineux, out-of-core

1. Introduction

La compression et la visualisation de maillages sont deux domaines de l'informatique graphique particulièrement actifs, dont les contraintes et les objectifs sont généralement incompatibles, voire contradictoires. La diminution de la redondance implique souvent une complexification du signal, du fait des mécanismes de prédiction dont l'efficacité est directement reliée à la profondeur de l'analyse. Cette couche logique supplémentaire ralentit l'accès aux données et entre en conflit avec les performances requises pour la visualisation temps réel. En effet, afin de naviguer dynamiquement au sein d'un maillage en fonction des mouvements de caméra, le signal doit être soigneusement préparé et hiérarchisé. Cela introduit généralement de la redondance et s'accompagne même parfois de perte de données, lorsque les polyèdres originaux sont approximés par des primitives géométriques simplifiées.

Dans cet article, nous recherchons un compromis entre compression et visualisation interactive en proposant une méthode qui combine de bonnes performances dans les deux domaines. Nous avons choisi comme point de départ l'algorithme de compression *in-core* sans perte de Gandoin et Devillers [GD02]. A ses taux de compression compétitifs, nous avons ajouté des capacités de traitement *out-of-core* et la gestion de niveaux de détails, afin de pouvoir manipuler des maillages sans limitation de taille et de permettre le raffinement local à la demande, via le chargement des données nécessaires et suffisantes à un rendu précis de tout sous-ensemble du maillage. Pour atteindre ces objectifs, l'idée fondamentale consiste à subdiviser l'objet original dans un arbre de partitionnement de l'espace. Cette subdivision est effectuée en introduisant une structure hiérarchique principale (appelée *nSP-tree*) dans laquelle sont plongées les structures de données originales (un *kd-tree* couplé à un complexe simplicial), d'une façon qui optimise la distribution des données de géométrie et de connectivité, supprimant ainsi l'effet de blocs inhérent aux approches de type *kd-tree*.

Après un état de l'art (section 2) et un rappel de la méthode choisie comme point de départ (section 3), notre contribution est introduite par un exemple (section 4), puis détaillée dans deux sections complémentaires. Tout d'abord, les algorithmes et structures de données relatifs à la compression *out-of-core* sont présentés (section 5), puis la description est complétée du point de vue de la visualisation (section 6). Enfin, des résultats expérimentaux sont présentés, accompagnés d'une comparaison avec l'état de l'art (section 7), avant de conclure (section 8).

† Université de Lyon, Lyon, F-69003, France; LIRIS, CNRS UMR 5205, Villeurbanne, F-69622, France - E-mail : prenom.nom@liris.cnrs.fr - Cette recherche a lieu dans le cadre du projet ARA *Eros3D* et du projet ANR *Triangles*, soutenu par le *Centre National de la Recherche Scientifique* (*CNRS*).

2. Travaux antérieurs

2.1. Compression

La compression de maillages est un domaine situé entre la géométrie algorithmique et la compression standard de données. Elle consiste à coupler efficacement l'encodage de la géométrie (position des sommets) et de la connectivité (relation entre les sommets), et traite principalement les modèles triangulaires *manifold*. On distingue les algorithmes mono-résolution, qui exigent un décodage complet avant de pouvoir visualiser l'objet, des méthodes multi-résolution qui permettent de voir le modèle se raffiner progressivement lors du décodage. Bien qu'étant historiquement les premiers algorithmes de compression géométrique, nous avons choisi de ne pas détailler ici les méthodes de compression mono-résolution. Le lecteur peut se référer aux états de l'art [GGK02, AG03] pour plus d'information. Pour comparaison, puisque les méthodes de compression progressive *out-of-core* sont très rares, la section 7 fera référence à l'algorithme de compression *out-of-core* mono-résolution d'Isenburg et Gumhold [IG03]. Par ailleurs, la compression avec perte, dont le principe consiste en une analyse fréquentielle du maillage, est exclue de cette étude.

La compression progressive est basée sur la notion de raffinement. A tout instant du processus de décompression, il est possible d'obtenir une approximation globale du modèle original, ce qui peut être utile pour les maillages volumineux ou pour la transmission réseau. Ce domaine de recherche est très actif depuis ces dix dernières années, et nous avons ici fait le choix du point de vue historique plutôt que de l'exhaustivité. Les premières techniques de visualisation progressive, basées sur la simplification de maillage, n'étaient pas orientées compression et induisaient souvent une augmentation significative de la taille du fichier, due au coût de stockage d'une structure hiérarchique [HDD*93, PH97]. Par la suite, plusieurs méthode mono-résolution ont été étendues à la compression progressive. Par exemple, Taubin *et al.* [TGHL98] ont proposé un encodage progressif basé sur l'algorithme de Taubin et Rossignac [TR98]. Cohen-Or *et al.* [COLR99] ont proposé une méthode de simplification séquentielle par suppression de sommet pour la connectivité, combinée à de la prédiction de position pour la géométrie. Alliez et Desbrun [AD01] ont ensuite introduit un algorithme basé sur la suppression progressive de sommets indépendants, avec une étape de retriangulation sous la contrainte de maintenir les degrés des sommets autour de 6. Contrairement à la majorité des méthodes, Gandoin et Devillers [GD02] ont choisi de donner la priorité à l'encodage de la géométrie. Leur algorithme, détaillé dans la section 3, obtient des taux de compression compétitifs et peut gérer les complexes simpliciaux de toute dimension, des maillages réguliers *manifold* aux soupes de triangles. Peng et Kuo [PK05] se sont basés sur cet article pour améliorer les taux de compression par l'utilisation de schémas prédictifs efficaces (gain d'environ 15%) qui se limitent cependant aux modèles triangulaires. Cai *et al.* [CLW*06] ont introduit la première méthode de compression progressive adaptée aux maillages volumineux, en offrant la possibilité de rendre *out-of-core* la plupart des méthodes existantes basées sur des octrees.

2.2. Visualisation

La visualisation rapide et interactive de maillages volumineux est un domaine de recherche très actif. Ces travaux sélectionnent les informations pertinentes à afficher : les algorithmes sont dits *out-of-core*, dans le sens où seules les informations nécessaires et suffisantes à un instant donné sont chargées en mémoire. Un arbre ou un graphe est souvent construit pour gérer la structure hiérarchique. Rusinkiewicz et Levoy [RL00] ont introduit QSplat, le premier système *out-of-core* de rendu de nuages de points. Ces derniers sont dispersés dans une structure hiérarchique de sphères englobantes, qui permet de manipuler aisément les niveaux de détail et qui est adapté aux tests de visibilité et d'occlusion. Plusieurs millions de points par secondes peuvent ainsi être affichés grâce à un rendu adaptatif. Par la suite, Lindstrom [Lin03] a développé une méthode pour les maillages avec connectivité. Un octree est utilisé pour répartir les triangles dans des clusters et pour construire une hiérarchie multi-résolution. Une mesure d'erreur quadratique est utilisée pour choisir les positions des points représentatifs pour chaque niveau de détail, et le raffinement est guidé par la visibilité et la mesure d'erreur dans l'espace écran. Yoon *et al.* [YSGM04] ont proposé un algorithme similaire avec une empreinte mémoire bornée : une hiérarchie de clusters est construite, chacun contenant un sous-maillage progressif pour lisser la transition entre les niveaux de détail. Cignoni *et al.* [CGG*04] utilisent une hiérarchie basée sur la division récursive d'un tétraèdre, afin de partitionner l'espace et garantir des frontières variables entre les clusters durant le raffinement. La phase de construction initiale est parallélisable, et le GPU est utilisé pour améliorer la vitesse d'affichage. Gobbetti et Marton [GM05] ont introduit les *far voxels*, capables d'afficher aussi bien des maillages réguliers que des soupes de triangles. Le principe est de transformer les sous-parties volumiques du modèle en approximations de leur apparence à une certaine distance, suivant l'angle de vue. Un arbre BSP est construit, et les noeuds sont discrétisés en voxels cubiques contenant ces approximations. A nouveau, le GPU est largement utilisé pour alléger la charge CPU et améliorer les performances.

2.3. Compression et visualisation combinées

Les méthodes de compression progressive sont désormais parvenues à maturité (les taux obtenus sont proches des limites théoriques) et la visualisation interactive de maillages de grande taille est une réalité depuis plusieurs années. Cependant, même si la combinaison de la compression et de

la visualisation est souvent mentionnée en tant que perspective, très peu d'articles traitent de ce problème, et les fichiers créés par les méthodes de visualisation pure sont souvent bien plus volumineux que les fichiers originaux. En fait, la compression favorise une petite taille au détriment de la rapidité d'accès aux données, tandis que la visualisation se concentre sur la vitesse de visualisation : les deux objectifs s'opposent et s'excluent. Namane *et al.* [NBB04] proposent une version comprimée du QSplat en utilisant Huffman et le codage différentiel pour les sphères (position, rayon) et les normales. Le taux de compression est proche de 50% par rapport aux fichiers QSplat originaux, mais cela se limite au rendu de points. Plus récemment, Yoon et Lindstrom [YL07] ont introduit un algorithme de compression de maillages qui prend en charge l'accès direct aux triangles du maillage. Cependant, leur format n'est pas progressif et n'est donc pas adapté au rendu global d'un modèle puisque cela nécessiterait de charger le maillage complet en mémoire.

3. 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, valide en toute dimension, est basé sur la construction *top-down* d'un *kd-tree* par subdivision de cellule. La cellule racine est la boîte englobante de l'ensemble de points, et chaque subdivision de cellule est encodée par $log_2 3$ bits pour décrire l'un des 3 cas suivants : la première demi-cellule est non vide, la seconde demi-cellule est non vide, ou les deux demi-cellules sont non vides. 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 plus grandes que $1x1x1$, de façon à ce que chaque point soit localisé avec la précision maximale. Dès que les points sont ainsi séparés, la connectivité du modèle est intégrée (un sommet est situé dans chaque cellule) et le processus de subdivision est inversé : les cellules sont fusionnées deux à deux depuis les feuilles du *kd-tree* jusqu'à sa racine, et leur connectivité est mise à jour. La connectivité est modifiée entre deux versions successives du modèle : ces changements sont encodés par des symboles insérés entre les codes géométriques. Les sommets sont fusionnés par un des deux opérateurs de décimation suivants :

- La suppression d'arête, définie par Hoppe *et al.* [HDD*93] et largement utilisé en simplification de surface, fusionne deux cellules adjacentes sous certaines hypothèses. Les deux sommets de 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 3a).
- La fusion de sommets, telle que définie par Popović et Hoppe [PH97], est une opération plus générale qui fusionne deux cellules même si elles ne sont pas adjacentes dans la connectivité courante. Le résultat n'est en général pas *manifold* (cf. figure 3b) et la séquence de codes correspondante est approximativement deux fois plus grande que la séquence de codes d'une suppression d'arête.

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). 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 donnés 2 sommets v et w situés dans un niveau intermédiaire du *kd-tree*, il est possible de trouver un descendant v_i de v qui est connecté à un descendant w_j de w. Ainsi, en terme de connectivité, 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 décodé pour tout le maillage. Dans la suite, nous présentons des algorithmes et structures de données basés sur la même approche mais supprimant ces limitations.

4. Vue d'ensemble de la méthode

La figure 1 illustre les principales étapes du raffinement d'un maillage. Les étapes 1 à 6 sont similaires à [GD02] : nous commençons avec une précision nulle et un seul point au centre. Puis nous lisons séquentiellement les codes de géométrie et de connectivité 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 sous-espaces, et le maillage est subdivisé dans ces sous-espaces. On note que les arêtes et les triangles situés sur la frontière sont dupliqués. Les 4 sous-maillages obtenus deviennent indépendants, chacun correspondant à un sous-espace appelé *SP-cell* dont le contenu est directement accessible via un index pointant dans le fichier comprimé. Ainsi, nous pouvons sélectionner une ou plusieurs *SP-cells* et lire leurs codes de raffinement associés dans le fichier comprimé, afin de continuer le processus de raffinement localement, en fonction des mouvements de caméra. Quand l'une d'entre elles atteint sa précision maximale (4 bits dans notre exemple), elle est divisée à nouveau. Ce processus est répété récursivement jusqu'à ce que chaque SP-cell dans le cône de vision ait atteint un niveau de précision déterminé par sa position relative à la caméra. Enfin, on connecte les *SP-cells* ensemble, en gérant

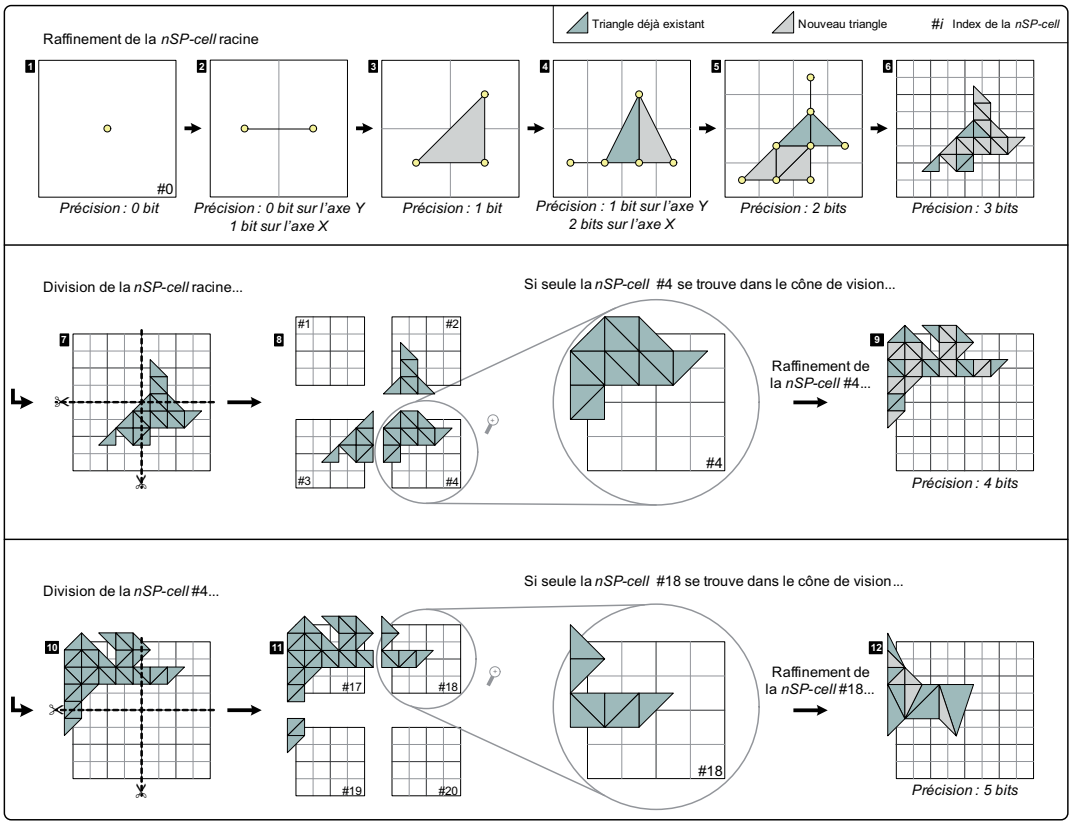

Figure 1: *Vue d'ensemble de la méthode*

les différences de résolution éventuelles pour éviter que les triangles de frontière ne se chevauchent, et le contenu de ces *SP-cells* est envoyé au *pipeline* graphique.

Le processus de compression s'effectue dans le sens inverse. On commence par partitionner l'espace, en créant le *SP-tree* dans lequel on répartit les simplexes. Afin d'éviter les *SP-cells* contenant seulement quelques triangles, une SP-cell doit contenir au moins N_{min} (défini par l'utilisateur) triangles pour être divisée. De plus, les simplexes dont les sommets appartiennent à plusieurs *SP-cells* sont dupliqués dans chacune d'elles. Puis on parcourt le *SP-tree* en ordre postfixe (une cellule est traitée après que tous ses fils ont été traités). Dans chaque *SP-cell*, les *kd-cells* sont progressivement fusionnées comme décrit dans la section 3, et la séquence de codes correspondante est construite, où les codes de géométrie et de connectivité sont entrelacés. La fusion est effectuée jusqu'à ce la précision minimale de la *SP-cell* soit atteinte : dans l'exemple de la figure 1, cela correspond à une fusion des *kd-cells* dans chaque dimension pour une *SP-cell* non racine, et à 3 fusions dans chaque dimension pour la *SP-cell* racine. Quand tous les enfants d'une *SP-cell 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 *SP-cell* racine.

5. Compression out-of-core

Dans cette section, nous détaillons les étapes successives du processus de compression, puis certains aspects de l'implémentation qui permettent d'atteindre de bonnes performances.

5.1. Partitionnement de l'espace

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. Bien sûr, cette étape de quantification n'entre pas en contradiction avec la caractéristique sans perte de la méthode puisque *p* peut être choisi aussi grand que nécessaire pour respecter la précision initiale de l'objet. Les points sont écrits dans un fichier brut (RWP) avec un encodage binaire

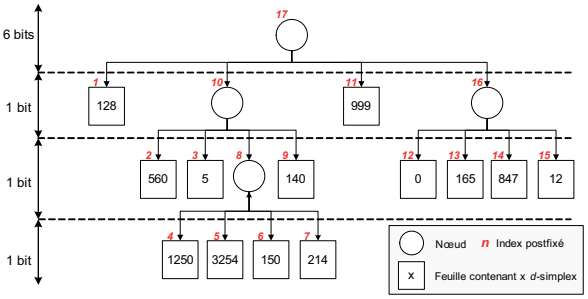

Figure 2: *Exemple 2D d'un nSP-tree avec $p = 9$ bits, $p_r = 6$ bits, $n = 2$ et $N_{min} = 1000$*

standard utilisant des codes de taille constante pour garantir un accès direct dans le fichier.

Définitions : Un *nSP-tree* est un arbre de partitionnement de l'espace avec n subdivisions par axe. Chaque noeud interne délimite un sous-espace cubique appelé *nSP-cell*. Puisque nos maillages sont plongés en 3*D*, une *nSP-cell* possède n^3 enfants, et dans notre cas, les *nSP-cells* sont régulièrement subdivisées, *i.e.* les n^3 enfants sont de même taille.

Chaque *nSP-cell* c contient des sommets dont la précision peut varier 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.

Un top-simplexe est un simplexe (sommet, arête, triangle ou tétraèdre) qui ne possède aucun parent, *i.e.* qui n'entre dans la composition d'aucun autre simplexe. Dans un maillage triangulaire, les top-simplexes sont principalement des triangles, mais dans un complexe simplicial ils peuvent être de dimension quelconque.

Principe : Un *nSP-tree* est construit de telle sorte que chaque feuille contienne l'ensemble des top-simplexes situés dans son sous-espace correspondant. La hauteur du *nSP-tree* dépend de la précision p (nombre de bits par coordonnée) : l'utilisateur peut choisir la précision maximale p_r de la racine, puis la précision p_l gagnée par chaque niveau suivant est calculé à partir de n ($p_l = log_2 n$). Par exemple, pour $p = 12bits$, $p_r = 6$ bits et $n = 4$, la hauteur du *nSP-tree* est 4. Une autre condition est appliquée : une *nSP-cell* ne peut être divisée que si le nombre de top-simplexes qu'elle contient est au moins égal au seuil N_{min}. Par conséquent, la précision minimale des feuilles n'a pas de valeur fixée, tandis que leur précision maximale vaut toujours p. Le contenu des feuilles est stocké dans un fichier d'index (RWI) où les $d+1$ index qui composent un d-simplexe sont écrits dans un format binaire brut pour réduire l'empreinte mémoire sur disque tout en permettant une lecture rapide. La figure 2 montre un exemple 2*D* d'un tel *nSP-tree*.

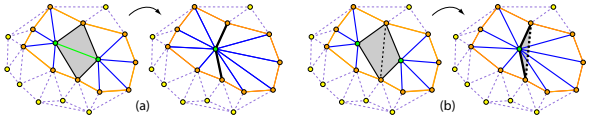

Figure 3: *(a) Suppression d'arête, (b) Fusion de sommets*

Duplication de simplexes : Un simplexe est dit appartenant à une *nSP-cell* c si au moins un de ses sommets incidents se trouve à l'intérieur de c. Ainsi, un simplexe du maillage peut simultanément appartenir à plusieurs *nSP-cells* et par conséquent, certains simplexes sont dupliqués durant la distribution dans le *nSP-tree*, afin d'éviter que le rendu ne présente des trous. Bien que cette technique réduise légèrement les taux de compression, elle a été choisie pour des raisons d'efficacité. Les dessins 7-8 et 10-11 de la figure 1 montrent le partitionnement d'une *nSP-cell*.

Choix de n : Afin d'obtenir un entier pour p_l, n doit être une puissance de 2. Cela garantit en outre que tout descendant d'une *kd-cell* contenue dans une *nSP-cell* c reste à l'intérieur de c. De plus, cela évite que les *kd-cells* ne chevauchent deux *nSP-cells*.

Ecriture de l'en-tête du fichier comprimé : L'en-tête du fichier contient toutes les données générales requises par le décodeur : système de coordonnées (origine et résolution de la grille afin de reconstituer les positions originales des sommets), dimension d du maillage, nombre p de bits par coordonnée, nombre n de subdivisions par axe pour le partitionnement de l'espace, ainsi que les précisions p_r et p_l.

5.2. Traitement de chaque cellule du *nSP-tree*

Un parcours postfixe du *nSP-tree* (nombres rouges dans la figure 2) est effectué et les opérations suivantes sont appliquées à chaque *nSP-cell* c.

(a) Construction du kd-tree et du complexe simplicial : Si c est une feuille : les top-simplexes appartenant à c sont lus dans le fichier RWI et les sommets associés dans le fichier RWP ; puis un *kd-tree* est construit qui sépare les sommets (processus *top-down*), et enfin le complexe simplicial basé sur les feuilles du *kd-tree* est généré.

(b) Calcul des codes de géométrie et de connectivité : Les feuilles du *kd-tree* sont fusionnées séquentiellement en suivant un processus *bottom-up* analogue à celui de [GD02] (cf. section 3 et figure 3). 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 disparaisse lors de la fusion. Le processus est stoppé lorsque la précision minimale de c est atteinte.

(c) Ecriture des codes dans un fichier temporaire : Les codes obtenus sont écrits dans un fichier temporaire semi-comprimé à l'aide d'un codeur arithmétique. A ce stade, nous ne pouvons pas écrire dans le fichier final, les données statistiques complètes n'étant pas encore disponibles.

(d) Fusion dans la *n*SP-cell mère : Reste désormais un *kd-tree* et un complexe simplicial dont les sommets ont la précision minimale de *c*. Afin de continuer la traversée du *nSP-tree*, 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-tree* 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.

5.3. Ecriture finale

Une fois toutes les *nSP-cells* 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, puis les séquences de codes des *nSP-cells* du fichier temporaire 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-tree* qui indique la position de chaque *nSP-cell* dans le fichier est ajoutée. Cette table permet à l'algorithme de décompression de reconstruire une structure de *nSP-tree* vide qui fournit une accès direct aux séquences de codes de chaque *nSP-cell*.

5.4. Parallélisation multi-core

La structure du *nSP-tree* est intrinsèquement propice à la parallélisation. L'algorithme de compression utilise plusieurs threads afin de bénéficier des processeurs multi-core actuels. Tandis que les entrées/sorties dans les fichiers demeurent mono-thread pour éviter de coûteux accès aléatoires au disque dur, les opérateurs de fusion et de subdivision de *kd-cell* ainsi que de fusions et divisions de *nSP-cell* peuvent être exécutées en parallèle sur différentes cellules. Pour favoriser les accès séquentiels au disque et éviter de trop fréquents blocages des autres threads (les E/S sur disque étant exclusives), le traitement d'une *nSP-cell* débute par le chargement de la liste des triangles contenus depuis les fichiers RWI et RWP dans un tampon. Puis, les étapes suivantes (cf. section 5.2) ne nécessitant que des opérations CPU ou RAM, elles peuvent être exécutées par 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. L'étape de décompression et visualisation bénéficie également de cette parallélisation, mais dans ce cas, l'amélioration des performances est plus difficile à estimer.

6. Décompression et visualisation

Les phases de décompression et de visualisation 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.

6.1. Initialisation

La première phase consiste à lire l'en-tête pour en extraire les informations générales. Puis la table située à la fin du fichier est utilisée pour construire le *nSP-tree*. A ce stade, chaque *nSP-cell* contient seulement sa position dans le fichier. Pour achever le processus d'initialisation, le *kd-tree* de la *nSP-cell* racine est créé dans sa forme la plus simple, *i.e.* une racine, et le complexe simplicial associé est initialisé avec un unique sommet.

6.2. Raffinement adaptatif temps-réel

A chaque pas de temps du rendu, le maillage est mis à jour pour s'adapter à la fenêtre de visualisation. Les *nSP-cells* situées dans le cône de vision sont raffinées ou simplifiées de sorte que l'erreur maximale de tout sommet garantisse que sa projection satisfasse une précision d'affichage fixée. Les autres *nSP-cells* sont simplifiées pour minimiser l'occupation mémoire, et ne sont pas envoyées au *pipeline* graphique. En accord avec ces critères, une liste des *nSP-cells* à afficher associées à leur niveau de détail est maintenue. Une *nSP-cell* qui atteint sa précision maximale est divisée, et l'on accède directement au contenu de chaque enfant grâce à son index dans le fichier, contenu dans le *nSP-tree*. A l'inverse, si une *nSP-cell* doit être simplifiée à un niveau de détail inférieur à sa précision minimale, elle est fusionnée dans son parent.

6.3. Traitement des frontières

Une fois les *nSP-cells* à afficher connues, deux problèmes principaux se posent, tous deux concernant les frontières entre *nSP-cells*. Premièrement, les simplexes dupliqués dans différentes *nSP-cells* peuvent se chevaucher. Et deuxièmement, des artefacts visuels peuvent être causés par l'affichage de deux *nSP-cells* adjacentes dont les précisions sont différentes. Seuls les top-simplexes frontières peuvent causer ces problèmes ; les autres peuvent être directement affichés. Les top-simplexes frontières sont composés d'au moins un sommet situé dans une autre *nSP-cell* à afficher. L'algorithme suivant est appliqué pour chaque top-simplexe frontière s :

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

2. S'il existe un i dans $1,\ldots,N$ tel que $p_i > p_c$ ou ($p_i = p_c$ et index de $c_i >$ index de c), s est supprimé et ne sera pas affiché.
3. Sinon, pour chaque *kd-cell* k_i telle que $c_i \in l$ et $c_i \neq c$, la *kd-cell* 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-cells* de différents niveaux de précision (cf. figure 4).

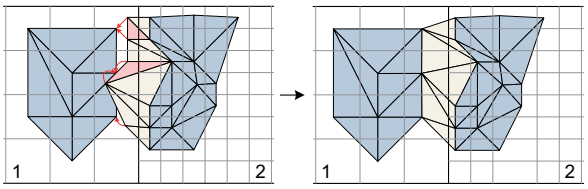

Figure 4: *Rendu des top-simplexes frontières entre deux nSP-cells de niveaux de détail différents.*

6.4. Encodage anticipé de la géométrie

Le principal inconvénient de cet algorithme 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, ralentissent le rendu sans résoudre l'effet de blocs inhérent aux approches de type *kd-tree* ou octree (cf. figure 8, à gauche). Ce problème est résolu en encodant la géométrie en avance sur la connectivité dans le fichier comprimé : le décodeur sera informé des divisions de *kd-cells* m bits avant que les changements de connectivité n'interviennent effectivement. Par conséquent, une *kd-cell* 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 m. Pour un maillage 3D et une avance géométrique de m bits, $3m$ niveaux de descendants sont connus par chaque *kd-cells*. Pour chaque sommet affiché, la position de ses futurs enfants est ainsi connue et leur barycentre peut être utilisé comme position du point représentant (cf. figure 5).

La figure 6 montre un exemple de *nSP-tree* avec une avance géométrique d'1 bit. Lors de la division d'une *nSP-cell*, certaines *kd-cells* peuvent être dupliquées dans plusieurs enfants, comme la *kd-cell* 11 dans cet exemple. Les clones d'une même *kd-cell* peuvent être amenés à évoluer différemment, puisque leur voisinage est différent selon la *nSP-cell* à laquelle ils appartiennent. Leurs codes peuvent donc être différents, comme dans G11 et G11b. Le *kd-tree* parent peut stocker les codes géométriques d'une seule *nSP-cell* fille. Nous avons choisi la fille contenant la *kd-cell* originale. Pour les autres *nSP-cells*, les codes géométriques manquants sont ajoutés au début de la séquence. Dans notre

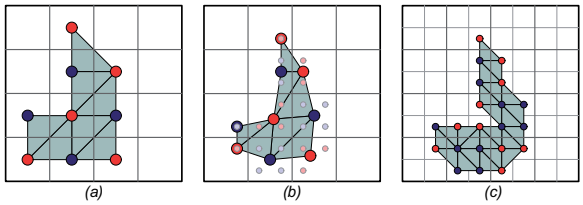

Figure 5: *Exemple de rendu avec géométrie en avance : comparaison entre (a)* 2 *bits de précision pour la géométrie et la connectivité, (b)* 2 *bits pour la connectivité et* 3 *bits pour la géométrie (*$2+1$ *bit d'avance), (c)* 3 *bits pour les deux*

exemple, la *nSP-cell* #0 stocke G11, G23 et G24, correspondant aux *kd-cells* 11, 23 et 24 de la *nSP-cell* #2 ; les codes G11b et G23b manquants sont stockés au début de la *nSP-cell* #1 (dans la boîte rouge). De même, puisque la *nSP-cell* racine (#0) n'a pas de parent, quelques codes géométriques sont ajoutés à son début. Les figures 7 et 8 illustrent les avantages de cette technique en terme de rendu. La courbe de débit-distorsion, construite en utilisant la distance *L2* fournie par l'outil METRO [CRS98], montre une amélioration significative dans les niveaux de détail intermédiaires.

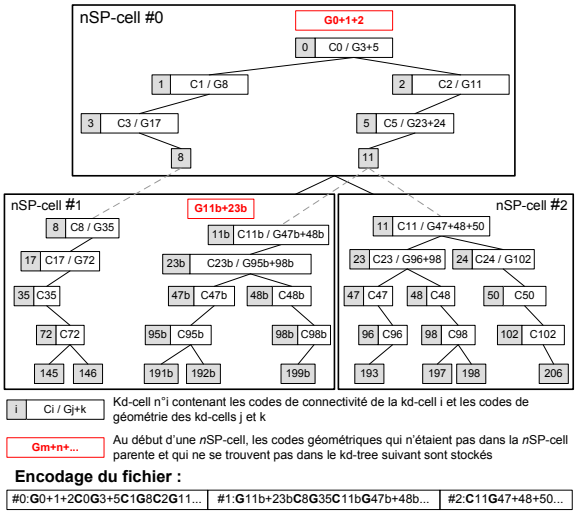

Figure 6: *Exemple de kd-tree avec la géométrie en avance sur la connectivité : cas 2D,* 1 *bit d'avance (i.e.* 2 *niveaux de kd-tree)*

7. Résultats expérimentaux

Il est impossible de comparer équitablement notre méthode avec des travaux existants, puisqu'à notre connaissance, aucune autre méthode ne combine la compression

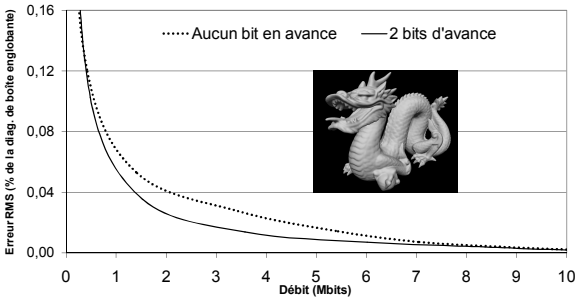

Figure 7: *Comparaison des courbes de débit-distorsion*

Figure 8: *Géométrie en avance sur la connectivité : rendu normal (à gauche) et rendu avec une avance de* 2 *bits sur la position des sommets (à droite)*

et le rendu interactif de maillages polyèdriques. Cependant, nous fournissons une comparaison avec quelques méthodes de compression bien connues (mono et multi-résolution, *in-core* et *out-of-core*) dans la section 7.1, puis avec les méthodes de visualisation de référence dans la section 7.2. Les résultats présentés ont été obtenus grâce à une implémentation C++/OpenGL de la méthode, sur un PC équipé d'un processeur Intel Q6600 QuadCore 2,4Ghz, 4 Go de RAM, 2 disque dur 10000RPM en RAID0, et une carte graphique NVIDIA GeForce 8800 GT 512MB. Les modèles sont fournis par *The Digital Michelangelo Project*, *Stanford University Computer Graphics Laboratory*, *UNC Chapel Hill* et *Electricité de France (EDF)*. Concernant les paramètres, n doit être suffisamment grand pour permettre la sélection de petites parties du maillages en quelques divisions de *nSP-cells*, mais suffisamment petit pour éviter l'affichage simultané de nombreuses *nSP-cells* (ce qui impliquerait de coûteux calculs de frontières). Une petite valeur de N_{min} augmente la duplication de simplexes et dégrade les performances de la compression, tandis qu'une grande valeur induit un petit *nSP-tree*, au détriment des capacités de multi-résolution. $n = 4$ et $5000 \leq N_{min} \leq 8000$ semblent être de bons compromis.

7.1. Compression

La table 1 présente les résultats de l'étape de compression *out-of-core*. 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 : si v est le nombre de sommets, t le nombre de triangles et p la précision pour chaque coordonnée de sommet (en bits), cette taille en bits est $3\,p\,v$ pour la géométrie $+3t\,log_2 v$ pour la connectivité. Le taux de compression est ensuite présenté (ratio entre la taille brute et la taille comprimée) avec le nombre total de *nSP-cells*. Les temps de compression sont aussi fournis, l'en-tête des colonnes se référant aux étapes détaillées dans la section 5. Comme attendu, le calcul des séquences de codes (étape 2*b*) est la partie la plus coûteuse. Cependant, cette dernière, qui est directement proportionnelle au nombre de sommets, bénéficie de notre implementation 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 table 2 montre les résultats de notre algorithme en terme de bits par sommet, comparés à ceux de [TG98] (méthode de référence *in-core* mono-résolution), de l'algorithme original *in-core* multi-résolution [GD02], de la méthode *out-of-core* mono-résolution [IG03] et de l'algorithme *out-of-core* mono-résolution avec accès direct [YL07]. 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 de maillage mais peu adaptée à la visualisation. Les deux méthodes 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 ne donnent des taux que pour la connectivité. Comparé aux méthodes qui ne permettent pas de visualisation interactive (trois premières colonnes), un surcoût moyen compris en 15% et 80% est observé, principalement dû à la redondance introduite par le partitionnement de l'espace (5 à 7% des triangles originaux sont dupliqués), au codage de l'orientation des triangles, et au fait que certaines prédictions complexes n'ont pas été intégrées pour garantir un rendu rapide.

7.2. Décompression et visualisation

La figure 10 présente quelques exemples de niveaux de détail du modèle *St. Matthew* avec 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 ou de regarder les vidéos jointes [JGA08]. Tous les résultats ont été produits avec une erreur maximale à l'écran de 1 pixel.

Pour apprécier la vitesse de décodage des données, la figure 9 présente le nombre de triangles affichés par seconde

Table 1: *Résultats de compression avec p = 16 bits par coordonnée, $n = 4$, $N_{min} = 8000$ et $p_r = 7$*

Modèles	Fichier en entrée		Fichier comprimé		Temps de compression (mm :ss)				Mem (Mo)	
	Triangles	Taille brute	Taux	nSP-C.	1	2	3	Total	Ram	DD
Bunny	69 451	609 823	3,37	65	00 :01	00 :02	00 :00	00 :03	108	1
Armadillo	345 932	3 295 034	3,95	1 217	00 :02	00 :03	00 :01	00 :06	121	6
Dragon	866 508	8 697 708	4,64	1 281	00 :07	00 :10	00 :01	00 :18	151	15
David 2mm	8 250 977	92 767 630	8,05	25 601	00 :50	00 :54	00 :09	01 :53	290	136
UNC Powerplant	12 388 092	173 938 325	7,85	38 209	04 :26	01 :52	00 :09	06 :27	291	290
EDF T5	14 371 932	165 869 111	7,36	27 521	01 :28	01 :50	00 :16	03 :34	290	254
EDF T8	22 359 212	263 381 699	6,82	31 105	02 :47	03 :07	00 :26	06 :20	289	404
Lucy	27 595 822	328 231 060	7,35	114 113	03 :10	02 :48	00 :33	06 :31	299	522
David 1mm	54 672 488	671 000 862	8,81	181 569	06 :01	04 :58	00 :57	11 :56	304	1035
St. Matthew	372 422 615	4 685 766 446	11,57	353 473	40 :50	33 :13	06 :26	80 :29	318	6510

Table 2: *Comparaison des taux de compression en bits par sommet*

Modèles	Sommets	p	[TG98]	[IG03]	[GD02]	[YL07]	[CGG*04]	[GM05]	Notre méthode
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
UNC 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. Matthew	166 933 776	16	-	10,7	-	22,9	282,1	508,0	19,4

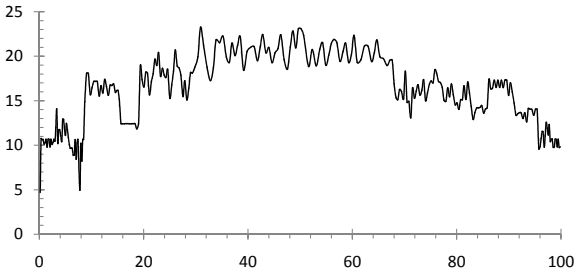

Figure 9: *Millions de triangles par seconde affichés au cours du temps pendant la vidéo du St. Matthew accompagnant l'article [JGA08]*

pendant le déroulement de la vidéo du *St. Matthew*. Notre décodeur est capable d'afficher jusqu'à 23 millions de triangles par seconde (tps), avec une moyenne d'environ 20 millions quand il tourne à plein régime (entre $t = 30$ et $t = 70$ s. dans la figure 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. Nous rappelons qu'aucune de ces méthodes de visualisation ne compresse les données : par exemple, [CGG*04] et [GM05] augmentent le fichier brut original, respectivement, de 10% et 80% en moyenne (cf. table 2). Par ailleurs, les petits trous polyédriques qui peuvent être observés sur les captures et vidéos ne sont pas des artefacts provoqués par la méthode mais proviennent des maillages originaux. Il est utile de remarquer que la transition entre les niveaux de précision k et $k+1$ d'une *nSP-cell*, visibles sur les vidéos, ont lieu relativement rarement (plus précisément, lorsque le zoom sur la *nSP-cell* 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.

8. Conclusion et perspectives

Comme présenté dans la section 7, les structures de données et les algorithmes présentés dans cet article obtiennent de bons résultats comparés à l'état de l'art, aussi bien en tant que méthode *out-of-core* de compression sans perte que comme méthode de visualisation de maillages de tout type et de toute taille, offrant ainsi une combinaison appréciable pour de nombreuses applications. Cependant, nous travaillons sur plusieurs points qui devraient lever les principales limitations actuelles. Notre premier objectif est d'augmenter sensiblement la vitesse d'affichage. Pour cela, trois pistes au moins peuvent être explorées : l'introduction de techniques d'occlusion, l'optimisation de l'algorithme de décompression (notamment par préchargement et utilisation intensive du GPU), et le recours éventuel à des méthodes de simplification pour réduire le nombre de simplexes affi-

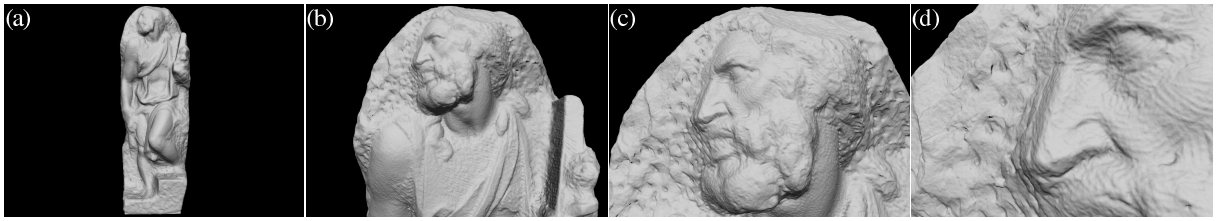

Figure 10: *Temps de raffinement du St. Matthew : 1,4 s pour afficher (a), 2,4 s de (a) à (b), 2,6 s de (b) à (c), 3,0 s de (c) à (d) ; temps de simplification : 1,5 s de (d) à (c), 1,1 s de (c) à (b), 0,6 s de (b) à (a).*

chés dans le pire des cas. Par ailleurs, nous pensons qu'il est possible d'améliorer les taux de compression en raffinant les schémas de prédiction. La difficulté réside dans la conception d'un modèle de probabilité précis dont les coûts calculatoires ne pénalisent pas la visualisation.

References

[AD01] ALLIEZ P., DESBRUN M. : Progressive compression for lossless transmission of triangle meshes. In *SIGGRAPH 2001 Conference Proc.* (2001).

[AG03] ALLIEZ P., GOTSMAN C. : Recent advances in compression of 3d meshes. In *In Proc. of the Sym. on Multiresolution in Geometric Modeling* (2003), Springer.

[CGG*04] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R. : Adaptive tetrapuzzles : Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Transactions on Graphics 23*, 3 (2004), 796–803.

[CLW*06] CAI K., LIU Y., WANG W., SUN H., WU E. : Progressive out-of-core compression based on multi-level adaptive octree. In *ACM international Conference on VRCIA* (2006), ACM Press, New York, pp. 83–89.

[COLR99] COHEN-OR D., LEVIN D., REMEZ O. : Progressive compression of arbitrary triangular meshes. In *IEEE Visualization 99 Conf. Proc.* (1999), pp. 67–72.

[CRS98] CIGNONI P., ROCCHINI C., SCOPIGNO R. : Metro : Measuring error on simplified surfaces. *Comput. Graph. Forum 17*, 2 (1998), 167–174.

[GD02] GANDOIN P.-M., DEVILLERS O. : Progressive lossless compression of arbitrary simplicial complexes. In *ACM SIGGRAPH Conference Proc.* (2002).

[GGK02] GOTSMAN C., GUMHOLD S., KOBBELT L. : Simplification and compression of 3d meshes. In *Tutorials on Multiresolution in Geometric Modelling* (2002), Springer, pp. 319–361.

[GM05] GOBBETTI E., MARTON F. : Far voxels : a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. In *ACM SIGGRAPH* (2005), ACM Press, pp. 878–885.

[HDD*93] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W. : Mesh optimization. In *SIGGRAPH 93 Conference Proc.* (1993).

[IG03] ISENBURG M., GUMHOLD S. : Out-of-core compression for gigantic polygon meshes. In *SIGGRAPH 2003 Conference Proc.* (2003).

[JGA08] JAMIN C., GANDOIN P.-M., AKKOUCHE S. : *http ://liris.cnrs.fr/ cjamin/AFIG2008/*, 2008.

[Lin03] LINDSTROM P. : Out-of-core construction and visualization of multiresolution surfaces. In *Sym. on Interactive 3D Graphics* (2003), ACM Press, pp. 93–102.

[NBB04] NAMANE R., BOUMGHAR F. O., BOUATOUCH K. : Qsplat compression. In *ACM AFRIGRAPH* (2004), ACM Press, New York, pp. 15–24.

[PH97] POPOVIĆ J., HOPPE H. : Progressive simplicial complexes. In *SIGGRAPH 97 Conference Proc.* (1997).

[PK05] PENG J., KUO C.-C. J. : Geometry-guided progressive lossless 3d mesh coding with octree decomposition. In *ACM SIGGRAPH Conference Proc.* (2005).

[RL00] RUSINKIEWICZ S., LEVOY M. : Qsplat : a multiresolution point rendering system for large meshes. In *Conf. on Computer Graphics and Interactive Techniques* (2000), ACM Press, New York, pp. 343–352.

[TG98] TOUMA C., GOTSMAN C. : Triangle mesh compression. In *Graphics Interface 98 Conference Proc.* (1998), pp. 26–34.

[TGHL98] TAUBIN G., GUÉZIEC A., HORN W., LAZARUS F. : Progressive forest split compression. In *SIGGRAPH 98 Conference Proc.* (1998), pp. 123–132.

[TR98] TAUBIN G., ROSSIGNAC J. : Geometric compression through topological surgery. *ACM Transactions on Graphics 17*, 2 (1998).

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

[YSGM04] YOON S.-E., SALOMON B., GAYLE R., MANOCHA D. : Quick-vdr : Interactive view-dependent rendering of massive models. In *Proc. of Visualization* (2004), IEEE Computer Society, pp. 131–138.