Compression géométrique pour une transmission progressive

Olivier Devillers, Pierre-Marie Gandoin

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Compression géométrique pour une transmission progressive

Olivier Devillers — Pierre-Marie Gandoin

N° 3766

Septembre 1999

THÈME 2

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

Compression géométrique pour une transmission progressive

Olivier Devillers , Pierre-Marie Gandoin

Thème 2 — Génie logiciel
et calcul symbolique
Projet Prisme

Rapport de recherche n° 3766 — Septembre 1999 — 21 pages

Résumé : La compression de structures géométriques est un domaine relativement récent de la compression de données. Depuis 1995, plusieurs articles ont traité le problème du codage optimal de maillages, en utilisant le plus souvent l'approche suivante: les sommets du maillage sont codés dans un ordre établi pour contenir partiellement la topologie du maillage. Parallèlement, un ensemble de règles simples permet de prédire la position du sommet courant à partir des positions de ses voisins qui ont déjà été codés.
Dans cet article, nous décrivons un algorithme de compression dont le principe est complètement différent: l'ordre des sommets est exploité pour comprimer leurs coordonnées, et la topologie est ensuite reconstruite à partir des sommets. Cet algorithme, particulièrement adapté aux modèles de terrains, permet d'atteindre des facteurs de compression légèrement supérieurs à ceux des algorithmes actuellement disponibles pour la compression géométrique, et en outre, il permet le codage et la transmission progressive et interactive des maillages.

Mots-clés : géométrie, compression, codage, triangulation, maillage, reconstruction, modèles de terrain, SIG

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, B.P. 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : 04 92 38 77 77 - International : +33 4 92 38 77 77 — Fax : 04 92 38 77 65 - International : +33 4 92 38 77 65

Geometric compression for progressive transmission

Abstract: The compression of geometric structures is a relatively new field of data compression. Since about 1995, several articles have dealt with the coding of meshes, using for most of them the following approach: the vertices of the mesh are coded in an order such that it contains partially the topology of the mesh. In the same time, some simple rules attempt to predict the position of the current vertex from the positions of its neighbours that have been previously coded.
In this article, we describe a compression algorithm whose principle is completely different: the order of the vertices is used to compress their coordinates, and then the topology of the mesh is reconstructed from the vertices. This algorithm, particularly suited for terrain models, achieves compression factors that are slightly greater than those of the currently available algorithms, and moreover, it allows progressive and interactive transmission of the meshes.

Key-words: geometry, compression, coding, triangulation, mesh, reconstruction, terrain models, GIS

1 Introduction

1.1 Motivations

Dans le contexte de l'imagerie et de la vision pour une application réseau, un serveur doit transmettre des données à un client. Ces données sont habituellement des images bitmaps transférées à travers un algorithme de compression. Cette technique a été utilisée en particulier en infographie, mais dans ce cas précis, une solution plus répandue aujourd'hui consiste à transmettre une description géométrique de la scène et à lancer le programme de synthèse d'image sur le client. Une scène géométrique en 3D est constituée de polygones, et donc est généralement codée comme une séquence de nombres (les coordonnées des sommets) et de n-uplets de pointeurs sur des sommets (les arètes joignant les sommets).
Si le problème de la compression d'images bitmap a été largement étudié dans le passé, la compression géométrique, située entre la géométrie algorithmique et la compression de données, est un domaine de recherche qui commence seulement à émerger.

Cela s'explique par le développement actuel des applications de la synthèse d'image, qui rend nécessaires la manipulation et l'échange de données géométriques de manière rapide et économique. En particulier, les nombreuses possibilités offertes par le World Wide Web dans le domaine de la réalité virtuelle sont conditionnées par un accès rapide aux données. Cela implique — particulièrement pour des connexions grand public à bande passante faible — d'organiser et de comprimer les données géométriques de façon optimale.

1.2 Précédents travaux

Parmi les quelques travaux qui traitent de la compression de maillages (scènes géométriques en 2 ou 3 dimensions composées de polygones), deux articles ont retenu notre attention, pour des raisons historiques et d'efficacité.

Geometric Compression Through Topological Surgery, de Taubin et Rossignac [10] décrit l'un des premiers algorithmes qui exploitent l'ordre de transmission des sommets du maillage pour coder la topologie, et codent les positions des sommets en appliquant des règles de prédiction. Cet algorithme — qui traite uniquement le cas des maillages triangulaires — décompose le maillage en bandes de triangles, et code les sommets dans leur ordre d'apparition dans les bandes, ce qui revient à décrire la

connectivité de la triangulation. D'autre part, comme cet ordre préserve le voisinage géométrique des sommets, il permet de prédire linéairement la position d'un sommet à partir des positions des sommets qui le précède directement dans le code. Ainsi, plutôt que de coder la position absolue de chaque sommet, l'algorithme utilise des méthodes classiques de compression entropique pour envoyer uniquement l'erreur résultant du schéma prédictif.
Comparée aux autres méthodes basées sur la décomposition de maillages triangulaires en bandes (en particulier, celles de Deering [5] et Chow [4]), celle-ci semble donner les meilleurs facteurs de compression sur des exemples pratiques.

Triangle Mesh Compression, de Touma et Gotsman [11] décrit un autre algorithme, dont le principe général est assez proche. La première différence est la manière de parcourir la triangulation. L'algorithme maintient une liste de sommets (initialisée avec un sommet arbitraire du maillage) formant un polygone qui contient tous les triangles déjà codés. Le polygone est étendu par l'insertion de la ligne polygonale joignant les sommets adjacents à un sommet donné du polygone et extérieurs à celui-ci. Cela donne un ordre sur les sommets du maillage qui permet de reconstruire sa topologie avec peu d'information supplémentaire. La seconde différence avec l'algorithme précédent est la méthode utilisée pour prédire la position d'un sommet à partir de ses prédécesseurs dans le code. En plus d'un schéma prédictif linéaire, l'algorithme estime le pli entre les triangles courants en fonction des plis précédents, ce qui conduit en pratique à des facteurs de compression supérieurs.

Ces deux algorithmes sont conçus pour des surfaces triangulées dans l'espace tridimensionnel. Le cas de genres non nuls est déduit du genre nul en ajoutant des données artificielles. Il est également à noter que ces algorithmes discrétisent préalablement les coordonnées des sommets sur un nombre de bits généralement compris entre 8 et 12. Les schémas prédictifs sont donc appliqués à ces positions discrétisées.

1.3 Cadre

Dans cet article, nous abordons le problème du codage de structures géométriques de manière différente. Nous utilisons le fait que dans de nombreux cas, les objets tridimensionnels sont construits automatiquement à partir d'échantillons de points. Ainsi, la topologie d'un maillage pourra souvent être reconstruite à partir de ses sommets.
Par conséquent, notre algorithme exploite l'ordre de transmission des sommets pour coder leurs coordonnées uniquement. Il peut donc s'appliquer à n'importe quelle

structure géométrique dès lors que l'on dispose d'un algorithme de reconstruction pour la topologie de l'objet original.

Un exemple de construction automatique de maillage à partir de ses sommets est donné par la triangulation de Delaunay. En 2 dimensions, il s'agit d'une triangulation qui offre certaines propriétés utiles, comme de maximiser le plus petit angle de la triangulation, c'est-à-dire créer des triangles équilibrés. La triangulation de Delaunay a aussi des applications dans le domaine des scènes 3D, en particulier avec les modèles de terrain, dont la topologie est généralement obtenue en triangulant les points sans leurs coordonnées en z.

La transmission efficace de triangulations de Delaunay d'un ensemble de points en 2 dimensions est un problème classique des SIG (systèmes d'information géographiques), traité par Snoeyink et van Kreveld [8], et plus récemment Sohler [9]. Cependant, dans ces travaux, l'objectif est différent du nôtre: l'ordre de transmission sur les points est utilisé pour accélérer la reconstruction de la triangulation (un temps linéaire est obtenu à la place de $n \log(n)$). Accessoirement, un gain est réalisé en codant les coordonnées des points de manière différentielle avec des codes de longueur variable. Cependant, cela ne constitue pas le principal intérêt de ces méthodes, les facteurs de compression obtenus restant relativement faibles.

2 Description de l'algorithme

Dans le but de simplifier la description de l'algorithme, nous commençons par traiter le cas de la dimension 1. Nous verrons ensuite que la généralisation à la dimension n est directe.

Nous décrivons d'abord la partie codage de l'algorithme. Soit S un ensemble de n points situés sur un segment de droite, entre 0 et 2^b (les coordonnées des points sont donc codées sur b bits). L'algorithme commence par coder le nombre total de points sur un nombre de bits arbitrairement fixé (par exemple 32). Puis il entre dans la boucle principale qui consiste à subdiviser le segment courant en deux demi-segments et à coder le nombre de points contenus dans l'un d'entre eux (celui de gauche par exemple) sur un nombre de bits optimal: si le segment courant contient p points, le nombre de points dans le demi-segment sera codé sur $\log_2(p+1)$ bits. Nous verrons dans la section 5 comment il est possible de coder un symbole sur un

nombre de bits non entier.
L'algorithme maintient donc une liste de segments constitués de:

- la longueur du segment,
- la position du segment,
- une liste des points situés sur le segment.

Chaque segment est retiré de la liste, subdivisé en 2 demi-segments insérés en fin de liste s'ils sont non vides, et donne lieu à un code en sortie correspondant au nombre de points contenus dans le demi-segment gauche. L'algorithme termine lorsqu'il n'y a plus de segments subdivisibles dans la liste courante, c'est-à-dire plus de segments de longueur supérieure à 1. Le pseudo-code ci-dessous détaille le fonctionnement de la partie codage.

Algorithme *Codage de points sur un segment de droite*

1. $\mathcal{L} \leftarrow$segment de droite original $\mathcal{S}_0$
2. écrire le nombre de points situés sur $\mathcal{S}_0$ sur 32 bits
3. **tant que** $\mathcal{L}$ non vide
4. **faire**
5. $\quad\mathcal{S} \leftarrow$extraire le premier segment de $\mathcal{L}$
6. $\quad n \leftarrow$nombre de points situés sur $\mathcal{S}$
7. $\quad\mathcal{S}_1 \leftarrow$moitié gauche de $\mathcal{S}$
8. $\quad n_1 \leftarrow$nombre de points situés sur $\mathcal{S}_1$
9. $\quad\mathcal{S}_2 \leftarrow$moitié droite de $\mathcal{S}$
10. $\quad n_2 \leftarrow$nombre de points situés sur $\mathcal{S}_2$
11. $\quad$écrire n_1 sur $\log_2(n+1)$ bits
12. $\quad$**si** $n_1 > 0$
13. $\quad\quad$**alors** ajouter $\mathcal{S}_1$ à la fin de $\mathcal{L}$
14. $\quad$**si** $n_2 > 0$
15. $\quad\quad$**alors** ajouter $\mathcal{S}_2$ à la fin de $\mathcal{L}$

Ainsi, les seuls codes écrits par l'algorithme sont les nombres de points situés sur les segments successifs. Les positions de ces points sont cachées dans l'ordre des codes. En fait, cet ordre contient implicitement une structure d'arbre binaire.

La partie décodage de l'algorithme répond exactement à sa partie codage. Une liste de segments est maintenue, mais cette fois un segment de droite est constitué de:

- la longueur du segment,
- la position du segment,
- le nombre de points situés sur le segment.

Pour chaque segment de longueur supérieure à 1 dans la liste, l'algorithme lit un nombre dans le flot de données comprimé, correspondant au nombre de points situés sur le demi-segment gauche. Le nombre de points situés sur le demi-segment droit est déduit du nombre de points sur le segment entier et du nombre lu. Ensuite, le segment courant est retiré de la liste, et le ou les demi-segments non vides sont ajoutés en fin de liste. L'algorithme termine lorsqu'il n'y a plus de segment subdivisible dans la liste. L'ensemble de la partie décodage est détaillé ci-dessous.

Algorithme *Décodage de points sur un segment de droite*

1. lire le nombre de points sur le segment de droite original $\mathcal{S}_0$ sur 32 bits
2. $\mathcal{L} \leftarrow \mathcal{S}_0$
3. **tant que** $\mathcal{L}$ contient des segments de longueur supérieure à 1
4. **faire**
5. $\quad \mathcal{S} \leftarrow$extraire le premier segment de $\mathcal{L}$
6. $\quad n \leftarrow$nombre de points situés sur $\mathcal{S}$
7. $\quad$ lire le nombre de points n_1 situés sur le demi-segment gauche de $\mathcal{S}$ sur $\log_2(n+1)$ bits
8. $\quad n_2 \leftarrow n - n_1$
9. $\quad$ **si** $n_1 > 0$
10. $\quad\quad$ **alors** $\mathcal{S}_1 \leftarrow$demi-segment gauche de $\mathcal{S}$
11. $\quad\quad\quad$ ajouter $\mathcal{S}_1$ à la fin de $\mathcal{L}$
12. $\quad$ **si** $n_2 > 0$
13. $\quad\quad$ **alors** $\mathcal{S}_2 \leftarrow$demi-segment droit de $\mathcal{S}$
14. $\quad\quad\quad$ ajouter $\mathcal{S}_2$ à la fin de $\mathcal{L}$

Au fur et à mesure que l'algorithme progresse, les données lues permettent de localiser les points avec plus de précision. Il est donc possible de visualiser l'ensemble des points aux étapes intermédiaires du décodage, avec une précision sur leurs coordonnées égale à la longueur courante des segments. Pour chaque segment $\mathcal{S}_i$, il suffit de générer n_i points (uniformément distribués par exemple) entre ses extrémités.

Pour généraliser cet algorithme à n'importe quelle dimension, définissons une cellule comme l'objet géométrique contenant les points à coder. En dimension 1, 2 et 3, les cellules sont respectivement le segment de droite, le carré et le cube. La seule partie de l'algorithme qui diffère d'une dimension à l'autre est la subdivision de la cellule. En dimension d, une cellule doit être subdivisée d fois (le long de chacun des d axes). Par conséquent, un ordre de subdivision pour les cellules doit être choisi (nous reviendrons sur la question du choix par la suite) et fixé afin que le codeur et le décodeur puissent communiquer.
La figure 1 représente un exemple bidimensionnel. Les nombres de points transmis par le codeur sont accompagnés du nombre de bits correspondant en dessous, et les nombres de points déductibles (donc non transmis) sont écrits entre parenthèses. La figure 2 montre le code résultant.

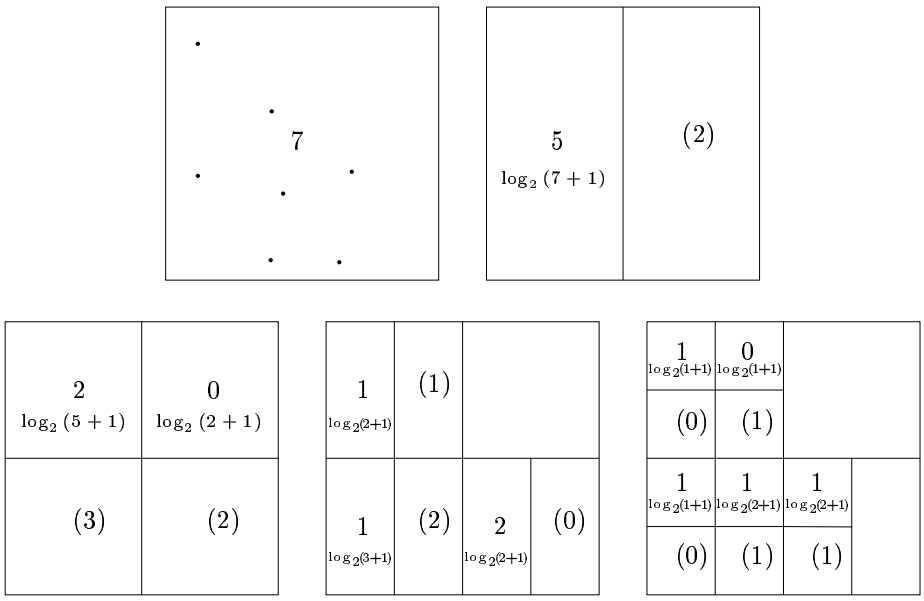

FIG. 1 – *L'algorithme de codage sur un exemple bidimensionnel*

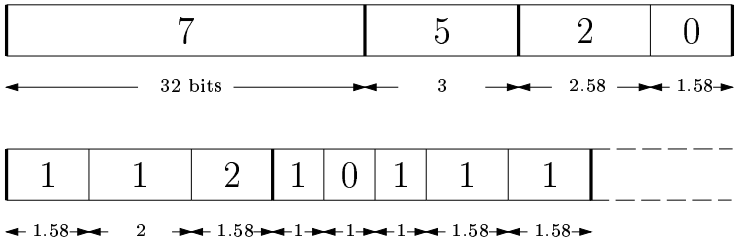

FIG. 2 – *Le code correspondant à l'exemple bidimensionnel*

3 Analyse théorique

3.1 Facteur de compression

Pour réaliser une analyse théorique de l'algorithme, nous allons supposer que les n points sont uniformément distribués dans un hypercube de dimension d. Soit 2^{b_i} (for $i = 1..d$) les longueurs des côtés de l'hypercube (la cellule originale de l'algorithme). Dans la suite, Q désignera le nombre de bits nécessaires pour coder la position d'un point: $Q = \sum_{i=1}^{d} b_i$.
Séparons l'algorithme en deux phases successives:

- séparation des points: les cellules sont subdivisées récursivement jusqu'à ce que chaque cellule de la liste contienne exactement 1 point,
- localisation finale: chaque cellule (contenant un point seulement) est subdivisée jusqu'à ce qu'elle atteigne la taille unitaire.

Calculons le nombre de bits utilisés pour séparer les points. Avec l'hypothèse d'uniformité, la dichotomie d'une cellule contenant c points génère deux cellules contenant chacune $c/2$ points. Par conséquent, pour séparer n points par cette méthode, $\log_2(n)$ subdivisions sont nécessaires. Si l'on décompose l'algorithme en phases définies par la taille des cellules dans la liste courante, le nombre de cellules double et le nombre de points dans chaque cellule est réduit de moitié d'une phase à la suivante. Or, le nombre de bits utilisés lors de la subdivision d'une cellule contenant c points est égal à $log_2(c+1)$. Donc finalement, le nombre total de bits utilisés pour coder la séparation des points est donné par:

$$
\begin{aligned}
& \sum_{i=0}^{\log_2(n)-1} 2^i \log_2(\frac{n}{2^i}+1) \\
= & -\sum_{i=0}^{\log_2(n)-1} i\,2^i + \sum_{i=0}^{\log_2(n)-1} \log_2(n+2^i)\,2^i \\
\leq & -(n\,\log_2(n) - 2\,n + 2) \\
& + \frac{n}{2}\,\log_2(\frac{3}{2}\,n) + \frac{n}{4}\,\log_2(\frac{5}{4}\,n) + \frac{n}{8}\,\log_2(\frac{9}{8}\,n) + \dots \\
\leq & -n\,\log_2(n) + 2\,n \\
& + n\,\log_2(n)\,(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots) \\
& + n\,(\frac{1}{2}\,\log_2(\frac{3}{2}) + \frac{1}{4}\,\log_2(\frac{5}{4}) + \frac{1}{8}\,\log_2(\frac{9}{8}) + \dots)
\end{aligned}
$$

Finalement, les calculs de sommes montrent que le nombre de bits utilisés à l'issue de la phase de séparation des points est inférieur à:

$$N_1 = 2.402\,n$$

Une fois qu'une cellule ne contient plus qu'un point, elle doit être subdivisée jusqu'à ce que le point soit complètement localisé. Comme $\log_2(n)$ subdivisions ont été réalisées au cours de la phase de séparation, il reste à subdiviser chaque cellule $Q - \log_2(n)$ fois. Pendant cette phase, une subdivision coûte 1 bit (le point appartient soit à la première demi-cellule, soit à la seconde). Le nombre de bits utilisés pour coder la localisation finale des points est donc:

$$N_2 = n\,(Q - \log_2(n))$$

Par conséquent, le nombre total de bits utilisés par l'algorithme pour coder les coordonnées des points est:

$$N = n\,(Q - \log_2(n) + 2.402)$$

Si l'on compare N à $n\,Q$ (le nombre de bits nécessaires pour coder les points sans compression), on remarque que le gain est $\log_2(n) - 2.402$ par point, et pour l'ensemble, si l'on néglige la constante additive, il est de $n\,\log_2(n)$, ce qui correspond exactement à l'information d'ordre sur les points ($\log_2(n)$ bits sont nécessaires pour

coder le numéro d'un point parmi n). Autrement dit, l'algorithme fait l'économie du codage de l'information d'ordre sur les points.
Il est à noter que ce gain théorique est une borne inférieure: la distribution uniforme des points constitue le cas le pire pour l'algorithme. En effet, la méthode tire parti de distributions non uniformes, qui génèrent des cellules vides dès les premières subdivisions de la phase de séparation des points. En fait, l'algorithme est d'autant plus efficace que la distribution est structurée, ce qui le rend cohérent avec la théorie de l'information.
Pour cette analyse, nous avons distingué la phase de séparation de la phase de localisation finale. En pratique, pour des distributions arbitraires de points, ces deux phases sont réalisées simultanément.

3.2 Complexité

L'algorithme (compression et décompression) est linéaire en temps et en espace dans le nombre de points n de l'objet à coder. Cependant, la constante de temps de la décompression est sensiblement inférieure à celle de la compression. Nous donnons ici ces constantes sans le détail des calculs:

- temps
  - compression: $Q\,n$
  - décompression: $(Q - \log_2(n) + 1)\,n$
- espace: $(Q + 4)\,n$

4 Caractéristiques

4.1 Progressivité

La caractéristique la plus intéressante de l'algorithme est la possibilité de coder (et décoder) les scènes géométriques de manière progressive. Nous avons vu dans la section 2 que les seules sorties du codeur étaient les nombres de points contenus dans les cellules successives, et que les tailles et les positions de ces cellules étaient implicitement codées dans l'ordre des sorties. Le choix de cet ordre, c'est-à-dire de la façon de subdiviser l'ensemble de points original, peut être optimisé pour privilégier le codage progressif de la scène. Puisque l'algorithme structure les cellules en kd-tree, deux parcours sont possibles. Le premier est un parcours en profondeur: chaque point est complètement localisé avant de passer au point suivant. Dans le second (parcours en largeur), toutes les cellules de la même taille sont traitées, générant

des cellules deux fois plus petites qui seront traitées ensemble à l'étape suivante de l'algorithme. Ainsi, après le décodage d'une "vague" entière de cellules, il est possible de reconstruire une version intermédiaire de l'ensemble de points telle que la précision est la même sur chaque point. Un moyen naturel de réaliser cette reconstruction est de générer n_i points uniformément ou régulièrement répartis dans la cellule c_i. Bien sûr, si la précision uniforme n'est pas nécessaire, la scène peut être visualisée à n'importe quel stade de la décompression, et même en temps réel. Ainsi, pour les applications réseau (en particulier, le survol de données), il est possible de comprimer un ensemble de points sans quantification préalable (compression sans perte), et d'envoyer les versions successivement raffinées de la scène 3D à l'utilisateur final jusqu'à ce qu'il considère que la précision suffit à ses besoins.

4.2 Interactivité

En fait, l'algorithme permet d'aller plus loin dans l'interactivité avec l'utilisateur. Puisque les cellules sont structurées en kd-tree, il est possible, pendant le décodage, de sélectionner un ou plusieurs sous-ensembles des données et de raffiner seulement ces parties. De cette manière, une navigation interactive à travers une scène 3D peut être optimisée du point de vue de la quantité d'information transmise.

4.3 Choix de la distribution

Pour obtenir les versions intermédiaires de la scène décodée, on doit générer des points dans l'espace d-dimensionnel à partir d'un nombre de points n_i et d'une cellule c_i. Une façon naturelle de procéder est d'injecter n_i points uniformément répartis dans la boîte englobante c_i, mais ce n'est pas la seule. Une analyse préalable de la scène peut indiquer que les points suivent localement une autre loi de probabilité, ou sont fortement structurés. Par exemple, les modèles de terrain sont souvent construits à partir d'un maillage triangulaire régulier 2D. Il suffit donc d'ajouter en en-tête des données comprimées la méthode de reconstruction la plus adaptée à la scène.

4.4 Dimension

Une autre caractéristique importante de l'algorithme est qu'il peut être appliqué directement à des données en dimension quelconque. Au delà de l'espace tridimensionnel, cela peut être utile pour des données de réalité virtuelle. En effet, le format VRML, largement répandu, associe souvent des données additionnelles aux sommets, telles que normales, surfaces, couleurs ou radiosité. Ces données peuvent être traitées

comme des dimensions supplémentaires et donc comprimées de la même manière que les coordonnées. Cependant, il faut se rappeler que l'ordre de grandeur du gain induit par l'algorithme est $n \log_2(n)$ (où n est le nombre de points de la scène), à comparer à nQ, la taille des données non comprimées. Ainsi, pour être efficace du point de vue du facteur de compression, l'algorithme doit s'appliquer à des données telles que le quotient $Q/\log_2(n)$ ne soit pas trop grand.

En outre, dans le cas de dimensions élevées, le choix de l'ordre de subdivision peut avoir des conséquences importantes sur le facteur de compression. Si la priorité est d'obtenir des représentations intermédiaires fidèles à la scène originale, l'ordre idéal est celui du parcours en largeur, qui consiste à subdiviser toutes les cellules d'une vague suivant la première dimension, puis subdiviser les cellules ainsi obtenues suivant la seconde dimension, ... , jusqu'à la dimension d, et recommencer le même processus jusqu'à la localisation complète des points. En revanche, du point de vue de l'efficacité de la compression, le découpage optimal doit créer des cellules vides en priorité, et donc, suivant la distribution des données, il peut être plus rentable de subdiviser les cellules plusieurs fois suivant la même direction.

5 Codage entropique et prédiction

L'algorithme que nous avons décrit jusqu'ici n'est pas une méthode de compression dans le sens classique de la théorie de l'information. Habituellement, une méthode de compression fournit une manière d'extraire l'information canonique des données (canonique signifiant ici non redondante) et de la coder. Ce que nous faisons ici est une réorganisation des données dans le but d'abandonner une partie de l'information qui ne nous intéresse pas (l'ordre sur les points). Par conséquent, il est naturel de penser qu'il reste une part de redondance dans la partie de l'information que l'on conserve (les coordonnées des points).

5.1 Codage arithmétique

La méthode classique de codage entropique que nous avons choisi d'utiliser ici est le codage arithmétique. Développé dans les années 80 [7, 6], ce principe de compression permet de coder un symbole en fonction de sa probabilité d'occurrence, sur un nombre de bits non nécessairement entier, ce qui constitue un avantage considérable sur les célèbres codes de Huffman. Fondamentalement, le principe de la compression arithmétique est de coder une séquence de symboles par un unique nombre réel ap-

partenant à l'intervalle [0..1[. Pour chaque symbole rencontré, l'intervalle initial [0..1[est raffiné en fonction de la probabilité estimée du symbole, conduisant finalement à un petit intervalle dont n'importe quel nombre code l'ensemble de la séquence. Cette méthode permet de coder chaque symbole s de la séquence sur $\log_2(\frac{1}{P}) + \epsilon$ bits, où P est la probabilité estimée de s, et ϵ une quantité négligeable devant $\log_2(\frac{1}{P})$. Ainsi cette technique peut être très performante si elle est couplée à une modèlisation statistique efficace des données à coder.
Le premier intérêt du codage arithmétique pour notre algorithme est de coder les nombres de points des cellules sur un nombre de bits optimal, y compris quand ce nombre n'est pas entier. En effet, nous avons vu dans la description de l'algorithme que pour une cellule contenant p points, le nombre de points dans la première demi-cellule issue de la subdivision était codé sur $\log_2(p+1)$ bits. En fait, ce n'est possible que grâce au principe du codage arithmétique: sans une méthode adaptée, ce nombre serait codé sur $\lceil \log_2(p+1) \rceil$ bits. Par suite, le gain pour chaque point serait de $\log_2(n) - 3$ au lieu de $\log_2(n) - 2.402$.

5.2 Méthodes de prédiction

En codant le nombre de points dans la première demi-cellule issue de la subdivision sur $\log_2(p+1)$ bits (où p est le nombre de points dans la cellule mère), on suppose que chaque valeur entière comprise entre 0 et p est équiprobable, de probabilité $1/(p+1)$. Pour améliorer les performances de l'algorithme, on peut chercher à estimer plus précisément la probabilité de chacune de ces valeurs. Pour cela, on étudie les densités locales des points au voisinage de la cellule à subdiviser.

La technique de prédiction utilisée repose sur l'hypothèse que les densités locales de points dans la cellule courante sont corrélées avec les densités locales dans son voisinage. Ainsi, l'algorithme analyse le contexte en prenant en compte toute l'information disponible à ce moment précis du codage ou du décodage. Donnons un exemple pour illustrer le principe de la méthode. Supposons que l'on doive subdiviser verticalement la cellule centrale de la figure 3. La figure montre l'état du kd-tree de cellules à ce stade. Un moyen très simple de déterminer la répartition de points la plus probable dans les deux demi-cellules est de calculer le pourcentage de points dans la cellule voisine gauche par rapport au nombre total de points voisins (dans les cellules voisines gauche et droite), puis de supposer que les demi-cellules respectent ce pourcentage. Dans l'exemple, on compte 11 voisins à gauche sur un total de 14 voisins, ce qui conduit à prédire 7 points dans la moitié gauche de la cellule courante et 2 points dans sa moitié droite. De là, une méthode simple consiste à estimer les probabilités des 10 valeurs possibles pour la demi-cellule gauche par une loi gaus-

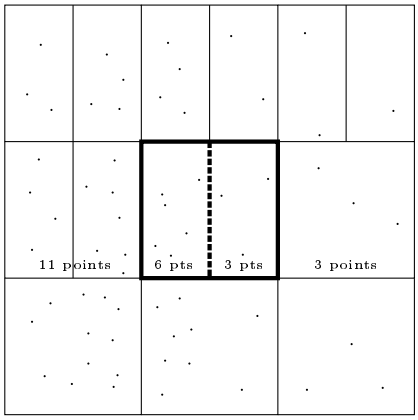

FIG. 3 – *Voisinage d'une cellule bidimensionnelle*

sienne discrète centrée en 7. Ainsi, la valeur effective de la demi-cellule gauche (6 points) aura une probabilité estimée forte, et sera donc codée sur un petit nombre de bits.
Dans cet exemple, la prédiction utilise seulement le voisinage au premier ordre, mais la technique peut être étendue aux ordre supérieurs, en donnant plus de poids aux cellules voisines les plus proches. En fait, l'ordre du contexte analysé peut être optimisé pour atteindre un compromis satisfaisant entre la précision de la prédiction et la complexité de l'algorithme.
Avec notre réglage des paramètres, cette méthode de prédiction apporte un gain supplémentaire d'environ 5% en moyenne, les meilleurs résultats étant obtenus pour les modèles 3D dont les densités locales sont le plus varié. Il est à noter que la simple liste de cellules utilisée dans la section 2 n'est plus suffisante, puisque la prédiction nécessite une structure de données pour un accès rapide au voisinage d'une cellule.

6 Résultats expérimentaux

6.1 Modèles de terrain

La figure 4 donne les résultats de la méthode décrite dans cet article appliquée à quelques modèles de terrain. Les deux premières lignes proviennent d'une base de données de SIG couvrant la région de Vancouver, tandis que la troisième correspond

à un modèle de terrain simple de 3721 points avec des coordonnées sur 10/10/6 bits. Cet exemple permet de visualiser la progression du décodage sur les figures 6 à 10.

	nombre de sommets	données orig.	données comp.	facteur de comp.	facteur théorique
rivers	120998	650364 43	341365 22.6	1.91	1.51
vancouver	908907	4885372 43	2169750 19.1	2.25	1.68
terrain	3721	12092 26	5890 12.7	2.05	1.57

FIG. 4 – *résultats de la compression sur des modèles de terrain*
(dans les colonnes 3 et 4, on donne la taille des données en octets
et le nombre correspondant de bits par sommet)

6.2 Objets 3D standards

Nous présentons dans la figure 5 quelques résultats de notre algorithme de compression comparé à ceux de Taubin et Rossignac [10] et Touma et Gotsman [11]. Nous avons vu en 1.2 que ces algorithmes codaient la connectivité et les coordonnées des sommets d'un maillage. Cependant, les nombres qui apparaissent ici concernent uniquement le codage des coordonnées des sommets, et sont donc comparables aux résultats de notre algorithme. Nous n'avons pas réimplanté les méthodes citées, par conséquent, les colonnes 4 et 5 sont directement extraites de l'article de Touma et Gotsman, et nous avons appliqué notre méthode aux mêmes modèles géométriques. Il est à noter que les coordonnées de ces modèles 3D ont été préalablement quantifiées sur 8 bits pour suivre exactement le même processus que dans les deux articles cités.

7 Conclusion

Nous avons présenté une nouvelle technique de compression géométrique adaptée à n'importe quelle structure dont la topologie est reconstructible à partir de ses sommets. En plus d'atteindre des facteurs supérieurs à ceux des autres algorithmes pour la compression de coordonnées, son principal avantage réside dans la progressi-

	nombre de sommets	données orig.	IBM 1996	G & T 1998	notre algo.
engine	2164	6222 23	4703 17.4	3425 12.7	2492 9.2
shape	2562	7686 24	4578 14.3	2990 9.3	4052 12.7
beethoven	2655	7965 24	4982 15.0	3576 10.8	3201 9.6
triceratops	2832	7788 22	3673 10.4	2937 8.3	2843 8.0
cow	3066	8815 23	4878 12.7	3376 8.8	3419 8.9
dumptruck	11738	32280 22	20351 13.9	11162 7.6	6858 4.7
total	25783	72767 22.6	44311 13.7	28149 8.7	24083 7.4

FIG. 5 – *banc d'essai sur la compression de modèles 3D*

vité du codage. En outre, l'algorithme est valide en dimension quelconque, et simple à implanter. Son originalité (mais aussi sa principale limitation) est d'abandonner la topologie de la structure géométrique, et donc d'être utilisable uniquement en association avec une technique de reconstruction. C'est pourquoi ses applications pratiques sont pour l'instant restreintes aux modèles de terrains. Cependant, des travaux à venir élargiront les domaines d'application en utilisant certaines méthodes de reconstruction de surfaces tridimensionnelles [1, 3, 2], qui donnent une condition nécessaire et suffisante sur l'échantillonnage de l'objet pour garantir une reconstruction valide. Ainsi, en ajoutant éventuellement un petit nombre de points à l'ensemble original, il serait possible de reconstruire la topologie de l'objet par une de ces méthodes.

Une autre perspective importante est d'améliorer la technique de prédiction, dont les résultats actuels ne sont pas complétement satisfaisants. Cela pourrait se faire par une analyse plus précise de la distribution de points au voisinage de la cellule courante, et en appliquant des méthodes d'optimisation pour le réglage des paramètres.

Références

[1] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 39–48, 1998.

[2] Fausto Bernardini and Chandrajit L. Bajaj. Sampling and reconstructing manifolds using alpha–shapes. In *Proc. 9th Canad. Conf. Comput. Geom.*, pages 193–198, 1997.

[3] Jean-Daniel Boissonnat and Bernhard Geiger. Three dimensional reconstruction of complex shapes based on the Delaunay triangulation. In R. S. Acharya and D. B. Goldgof, editors, *Biomedical Image Processing and Biomedical Visualization*, volume 1905, pages 964–975. SPIE, 1993.

[4] M. Chow. Optimized geometry compression for real-time rendering. In *Proc. IEEE Visualization*, pages 347–354, 1997.

[5] M. Deering. Geometry compression. In *Proc. SIGGRAPH*, pages 13–20, 1995.

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

[7] J. Rissanen and G. G. Langdon. Arithmetic coding. *IBM J. Res. Develop.*, 23(2):149–162, 1979.

[8] J. Snoeyink and M. van Kreveld. Linear-time reconstruction of Delaunay triangulations with applications. In *Proc. Annu. European Sympos. Algorithms*, number 1284 in Lecture Notes Comput. Sci., pages 459–471. Springer-Verlag, 1997.

[9] C. Sohler. Fast reconstruction of delaunay triangulations. In *Proc. 11th Canad. Conf. Comput. Geometry*, 1999.

[10] G. Taubin and J. Rossignac. Geometric compression through topological surgery. Research Report RC-20340, IBM Research Division, 1996.

[11] C. Touma and C. Gotsman. Triangle mesh compression. In *Proc. Graphics Interface*, pages 26–34, 1998.

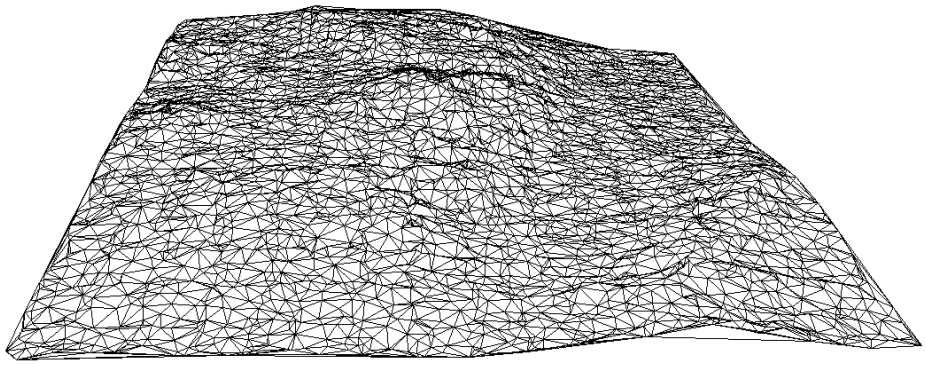

FIG. 6 – *précision* = 6 *bits, facteur de compression* = 5.44

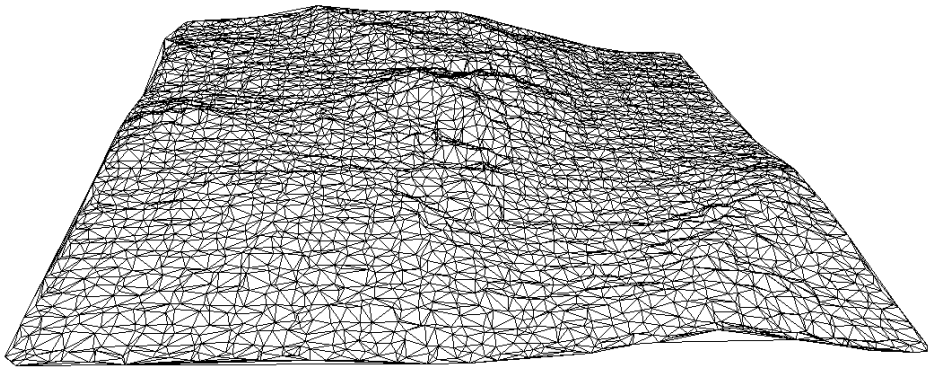

FIG. 7 – *précision* = 7 *bits, facteur de compression* = 3.90

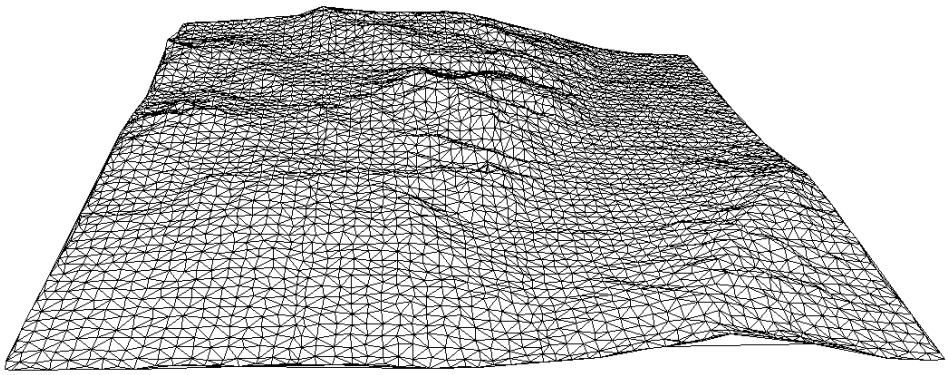

FIG. 8 – *précision* = 8 *bits, facteur de compression* = 3.00

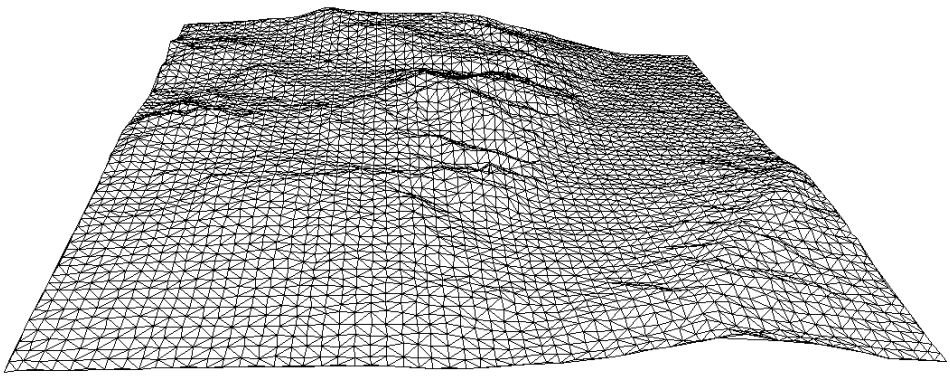

FIG. 9 – *précision* = 9 *bits, facteur de compression* = 2.44

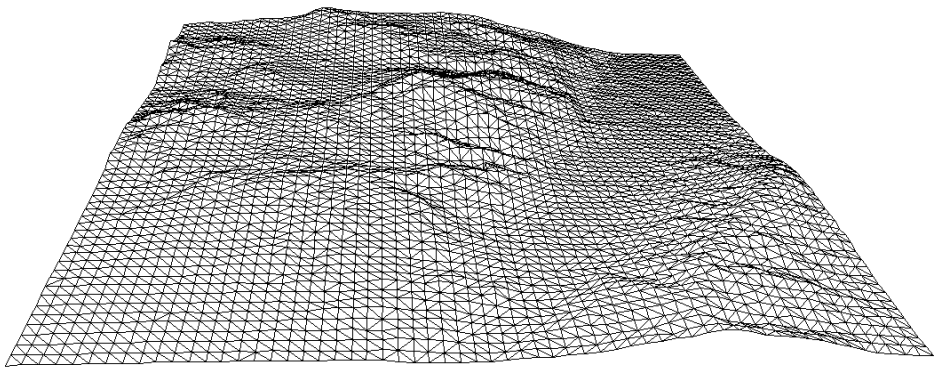

FIG. 10 – *précision* = 10 *bits (compression sans perte), f. c.* = 2.05

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399