Geometric Compression For Interactive Transmission

Olivier Devillers Pierre-Marie Gandoin *

INRIA Sophia Antipolis

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 that partially contains the topology of the mesh. In the same time, some simple rules attempt to predict the position of each vertex from the positions of its neighbors that have been previously coded.

In this article, we describe a compression algorithm whose principle is completely different: the coding order of the vertices is used to compress their coordinates, and then the topology of the mesh is reconstructed from the vertices. This algorithm achieves compression ratios that are slightly better than those of the currently available algorithms, and moreover, it allows progressive and interactive transmission of the meshes.

Keywords: geometry, compression, coding, interactivity, mesh, reconstruction, terrain models

1 INTRODUCTION

1.1 Motivations

In the context of image visualization in a network application, a remote server has to transmit data to a client. This data is usually bitmap data and is transferred through some compression algorithm. This method has also been used in the past for computer graphics images, but in that special case, another solution consists in transmitting the scene description and in running the image synthesis program on the client. A 3D geometric scene is made of polygons, and so is typically coded as a sequence of numbers (the vertex coordinates) and tuples of vertex pointers (the edges joining the vertices).

If the problem of bitmap image compression has already been widely studied, the compression of geometric data, lying between computational geometry and standard data compression, is quite a new field of research.

Yet the rapid growth of image synthesis applications makes necessary the manipulation and the exchange of this type of data in a fast and economical manner. In particular, the numerous possibilities given by the World Wide Web in the field of virtual reality could be dramatically restricted without a fast access to the data. This implies — especially for remote access through low bandwidth lines — that the geometric data would be efficiently structured.

*INRIA - 2004, route des Lucioles, B.P. 93 - 06902 Sophia Antipolis Cedex - France, FirstName.LastName@sophia.inria.fr

1.2 Related Work

Among the works dealing with mesh compression, we focus on the following three articles, according to historical and efficiency criteria.

Geometric Compression Through Topological Surgery, by Taubin and Rossignac [16] describes one of the first algorithms that use the transmission order of the mesh vertices to code the topology, and code the vertex positions efficiently by applying prediction schemes. This algorithm — which handles triangle meshes only — decomposes the mesh in triangle strips, and codes the vertices in their order of appearance in the strips, which amounts to code the connectivity of the triangulation. On the other hand, since this order preserves the geometric neighborhood of the vertices, it allows to linearly predict the position of a vertex from the positions of vertices immediately preceeding in the code. So, instead of coding the absolute coordinates of each vertex, the algorithm uses standard entropy coding methods to send only the error resulting from the predictive technique. Compared to the other methods based on the decomposition of the mesh into triangle strips (in particular, those of Deering [8] and Chow [6]), this one seems to give the smallest compression ratios on practical examples [1].

Triangle Mesh Compression, by Touma and Gotsman [17] describes another algorithm, whose general principle is quite close. The first difference is the way to traverse the triangulation. The algorithm maintains a list of vertices forming a polygon which contains the coded triangles. This polygon grows by conquering, for each vertex on its boundary, the outside incident triangles. This gives an order over the vertices of the mesh which allows to reconstruct its topology with few additional information. The second difference with the previous algorithm is the method used to predict a vertex position from its predecessors in the code. Besides a linear prediction technique, the algorithm estimates the crease between the current triangles from the previous creases. This yields better compression ratios in practice. Amongst the other works dealing with single resolution compression of triangle meshes, the method of Gumhold and Strasser [9] stands out as providing fast algorithms intended for real time applications. The compression ratios are close to the results of Touma and Gotsman concerning the connectivity, but the method lacks an efficient scheme for the geometric part.

Progressive Compression Of Arbitrary Triangular Meshes, by Cohen-Or, Levin and Remez [7] proposes a method using multiresolution decomposition to achieve progressive compression of the meshes. An initial simplified mesh is refined by insertion of vertices in patches. The position of each new vertex is predicted from the positions of the vertices of the patch where it is inserted, then corrected by additional information. The algorithm uses an efficient scheme of coloring to allow the decoder to recognize the patches. Concerning progressive geometric coding, this algorithm gives the

[1] In this article, we will measure the efficiency of the compression with the ratio $\frac{compressed\ data\ size}{original\ data\ size}$ (multiplied by 100 to obtain a percentage).

best results published so far, since it yields a code size increase of only 10% compared to the results of Touma and Gotsman. Other progressive methods [15, 11, 2] are slightly less efficient.

These algorithms are designed for triangulated surfaces in the 3-dimensional space. The case of genus greater than 0 is deduced from the null genus case by adding some artificial data. It is also important to note that these algorithms first quantize the vertex coordinates to a number of bits typically lying between 8 and 12.

1.3 Overview

In this article, we tackle the problem of coding geometric structures in a different way. We use the fact that in many cases, the 3D objects are constructed automatically from point samples. Hence the topology of a mesh can often be reconstructed from its vertices.

Consequently, our algorithm exploits the transmission order of the vertices to code only their coordinates (description, analysis and features in Sections 2, 3 and 4). Some additional gain can be obtained by using prediction and entropy coding (Section 5). In this form, the algorithm can be applied to any geometric structure as long as a reconstruction algorithm is available for the topology of the original model. However, to handle cases where automatic reconstruction is not suitable, we propose also a progressive topological coder (Section 6). Finally, in order to compare our method to previous works, we present experimental results (Section 7).

2 DESCRIPTION OF THE ALGORITHM

In order to simplify the description of the algorithm, we start by handling the one-dimensional case. We will see that the generalization to any dimension is straightforward.

We first describe the coding part of the algorithm. Let S be a set of n points lying on a line segment, between 0 and 2^b (the coordinates of the points are integers coded on b bits). The algorithm begins to code the total number of points on an arbitrary fixed number of bits (32 for example). Then it starts up the main loop which consists in subdividing the current segment in two halves and in coding the number of points contained in one of them (the left half-segment for example) on an optimal number of bits: if the current segment contains p points, the number of points in the half-segment will be coded on $\log_2(p+1)$ bits. We will see in Section 5 how it is possible to code a symbol on a fractional number of bits.

So the algorithm maintains a list of segments composed of:

- the segment coordinates,
- the list of all points lying on the segment.

The pseudo-code below details the functioning of the coding part.

Algorithm *Coding of points on a line segment*

```
1.  L ←original line segment S_0
2.  output the number of points on S_0 on 32 bits
3.  while L not empty
4.     do
5.        S ←pop first segment in L
6.        n ←number of points on S
7.        S_1 ←left half of S
8.        n_1 ←number of points on S_1
9.        S_2 ←right half of S
10.       n_2 ←number of points on S_2
11.       output n_1 on log_2(n+1) bits
12.       if n_1 > 0 and length(S_1) > 1
13.          then add S_1 at the end of L
14.       if n_2 > 0 and length(S_2) > 1
15.          then add S_2 at the end of L
```

The loop stops when there are no more divisible segments in the list, that is to say no segment of length greater than 1. It is to be noted that the only output of the algorithm are the numbers of points lying on the successive segments. The positions of these points are hidden in the order of the output. Indeed, this order contains an implicit binary tree structure.

The decoding part of the algorithm matches exactly its coding part. A list of segments is maintained, but this time the line segment data structure is composed of:

- the segment coordinates,
- the number of points lying on the segment.

The entire decoding part is detailed below.

Algorithm *Decoding of points on a line segment*

```
1.  read the number of points on the original line segment S_0 on
    32 bits
2.  L ←S_0
3.  while L contains segments of length greater than 1
4.     do
5.        S ←pop first segment in L
6.        n ←number of points on S
7.        read the number of points n_1 on the left half-segment of
          S on log_2(n+1) bits
8.        n_2 ←n − n_1
9.        if n_1 > 0
10.          then S_1 ←left half of S
11.               add S_1 at the end of L
12.       if n_2 > 0
13.          then S_2 ←right half of S
14.               add S_2 at the end of L
```

As the algorithm progresses, the data read allow to localize the points with more accuracy. Therefore it is possible to visualize the set of points at intermediate stages of the decoding, by displaying the middle point of each segment S_i of the list. Thus the accuracy over the point coordinates is equal to half the length of the corresponding segment.

To generalize this algorithm to any dimension, let us define a cell as the geometric object containing the points to be coded. In dimensions 1, 2 and 3, the cells are respectively line segments, rectangles, and rectangular parallelepipeds. The only part of the algorithm that differs from a dimension to another is the subdivision of the cell. In dimension d, a cell must be subdivided d times (along each of the d axes). Consequently, an order of subdivision for the cells must be chosen (we will come back to this question in the following) and fixed so that the coder and the decoder can communicate.

Figure 1 represents a two-dimensional example. The numbers of points transmitted by the coder are written with the corresponding number of bits below, and the numbers of points deduced by subtraction are written in parentheses. Figure 2 shows the resulting code.

3 THEORETICAL ANALYSIS

3.1 Compression Ratio

To do a theoretical analysis of the algorithm, we will assume that the n points are the vertices of a grid in a d-dimensional hypercube. Let 2^{b_i} (for $i = 1..d$) be the side lengths of the hypercube (the original cell of the algorithm). In the following, Q will denote the number of bits to code the position of a point: $Q = \sum_{i=1}^{d} b_i$.

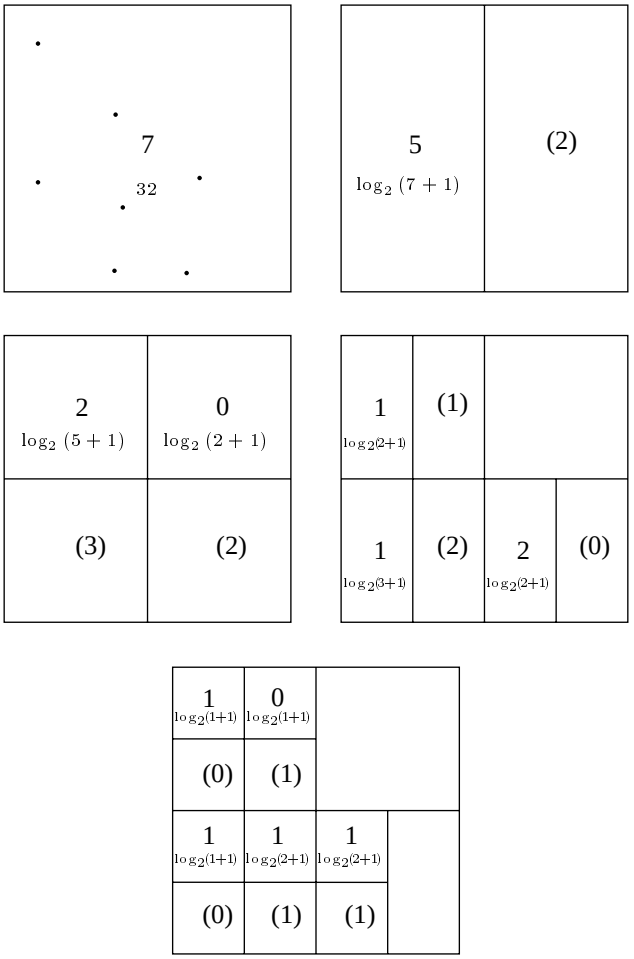


Figure 1: The coding algorithm on a two-dimensional example

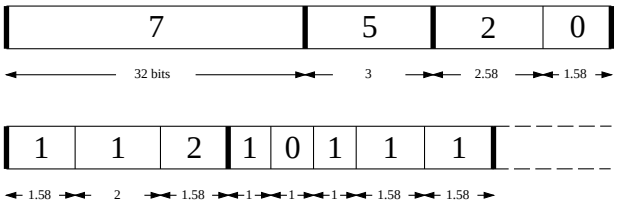


Figure 2: The output code of the two-dimensional example

Let us split up the algorithm in two successive phases:

- separation of the points: the cells are recursively subdivided until each cell of the list contains exactly 1 point,
- final localization: each cell (containing only one point) is subdivided until it reaches the unit size.

Let us calculate the number of bits used to separate the points. With the regularity hypothesis, the dichotomy of a cell containing c points generates two cells containing $c/2$ points each. Therefore, to separate the n points using this technique, $\lceil \log_2 n \rceil$ subdivisions are necessary. If we decompose the algorithm in phases defined by the size of the cells in the current list, the number of cells doubles and the number of points in each cell is reduced by half from a phase to the next one. Now, the number of bits used for the subdivision of a cell containing c points is equal to $log_2(c+1)$. So finally, the total number of bits used to code the separation of the points is given by:

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

Finally, the calculations of the sums show that the number of bits used at the end of the separation of the points is less than:

$$N_1 = 2.402\,n$$

Once a cell contains only one point, it must be subdivided until the point is completely localized. Since $\log_2 n$ subdivisions have been performed during the separation phase, it remains to subdivide each cell $Q - \log_2 n$ times. During this phase, a subdivision costs 1 bit (the point belongs either to the first half-cell or to the second one). Thus the number of bits used to code the final localization of the points is:

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

Consequently, the total number of bits used by the algorithm to code the point coordinates is:

$$N = N_1 + N_2 = n\,(Q - \log_2 n + 2.402)$$

If we compare N to $n\,Q$ (the number of bits used to code the points without compression), we notice that the gain is $\log_2 n - 2.402$ per point, and for the set of points, if we neglect the additive constant, it is $n\,\log_2 n$, which corresponds exactly to the order information over the points ($\log_2 n$ bits are necessary to code the index of a point among n). In other words, the algorithm saves the encoding of the order information over the points.

It is important to observe that this theoretical gain is a lower bound: the regular distribution is the worst-case for the algorithm. In fact, the most non uniform is the distribution, the most efficient is the algorithm. The intuitive idea is that if a cell c is split in two unbalanced parts, then more points share the bit corresponding to c, so the global coding of this bit is cheaper. More formally, let $n_1, n_2, \ldots, n_{2^j}$ be the numbers of points in the cells after j subdivision phases, then the cost of the next phase is $\sum_{i=1}^{2^j} \log(n_i + 1)$, which is maximal when all the n_i's are $n/2^j$, due to the convexity of the $\log$ function.

3.2 Complexity

The algorithm (compression and decompression) is linear in time and space with respect to Qn. However, the time and space constants of the decompression are significantly smaller than the compression ones. Indeed, during the compression, lists of points are stored and must be traversed for each cell subdivision.

4 FEATURES

4.1 Progressivity

The most interesting feature of the algorithm is the possibility to apply it for progressive coding (and decoding) of the geometric scene. We saw in Section 2 that the only output of the coder were the numbers of points contained in the successive cells, and that the sizes and the positions of those cells were implicitly coded in the order of the output. The choice of this order, i.e. of the way to subdivide the original set of points, can be optimized in order to prioritize the progressive coding of the scene. Since the algorithm organizes the cells in a kd-tree, two traversals are possible. The first one is a depth-first traversal: each point is completely localized before the next one is handled. In the second one (breadth-first traversal), all the cells of a same size are processed, generating twice as small cells which will be processed together at the next stage of the algorithm. Therefore after the decoding of an entire batch of cells, it is possible to construct an intermediate version of the set of points such that the precision is the same over each point. A typical manner to do this is to display the center point of each cell of the current list. Of course, if there is no need for an uniform precision over the points, the scene can be visualized at any time of the decompression, and even in real-time.

4.2 Interactivity

Thus, for net applications (browsing in particular), it is possible to send successive refined versions of a 3D scene to the final user until he considers that the accuracy is sufficient for his needs. In fact, the method allows to go further in the interactivity with the user. Since the cells are structured in a tree, it is possible, during the decoding, to select one or more subsets of the scene and to refine them and only them. Hence an interactive navigation through a 3D scene, with dilatations and translations, can be optimized from the point of view of the quantity of transmitted information.

4.3 Lossless Compression

Contrary to the previous works on the subject, our method does not introduce any loss over the original data. The algorithm handles integer coordinates, but the conversion from float does not need any quantization (for typical VRML objects, 24 to 32 bits are necessary to store the coordinates lossless, to be compared with the 8 to 12 bits quantization of [16, 17, 7]). Thus the user can choose to bring the decompression to its end and so obtain the real original object.

4.4 Dimension

Another important feature of the algorithm is that it can be applied straightforwardly to data in any dimension. Beyond the three-dimensional space, it can be useful for virtual reality data. Indeed, in the widely spread VRML format, extra data is often associated to the vertices, as normal vectors, surfaces, color, radiosity. This data can be handled as additional dimensions and thus compressed exactly like the coordinates. Again, the algorithm performs better with strongly structured data, thus the compression ratio will be smaller when the additional data is correlated to the spatial data (which is the case in usual models).

Moreover, in high dimensions, the choice of the order of subdivision can have important consequences over the compression ratio. If the priority is to obtain intermediate representations of the scene faithful to the original scene, the ideal order is the one of the breadth-first traversal, which consists in subdividing all the cells of a batch along the first dimension, then subdividing the obtained cells along the second dimension, ..., until the dimension d, then restarting the same process until the complete localization of the points. But from the point of view of the compression efficiency, the optimal cutting must create empty cells as a priority, and thus, according to the data distribution, it can be more economic to subdivide the cells several times along the same direction. It suffices to add in some header of the compressed data which order of subdivision is best suited to the scene.

5 ENTROPY CODING AND PREDICTION

The algorithm we described so far is not a compression method in the classical sense of the information theory. Usually, a compression method gives a manner to extract the canonical information of the data (canonical means here non redundant) and to code it efficiently. Here, what we have done is a reorganization of the data to drop a part of the information which does not interest us (the order over the points). Therefore, it is natural to think that it remains some redundancy in the information part that we kept (the point coordinates).

5.1 Arithmetic Coding

Arithmetic coding is a classical entropy coding method developed in the 1980's [12, 18], that allows to code each symbol s of a sequence on $\log_2 \frac{1}{P} + \epsilon$ bits, where P is the estimated probability of s, and ϵ a small quantity compared to $\log_2 \frac{1}{P}$. Thus this technique can be quite powerful if coupled to an efficient statistic modeling of the data to be coded.

The first utility of the arithmetic coding for our method is to code the numbers of points of the cells on an optimal number of bits, even if this number is not an integer. Indeed, we saw in the description of the algorithm that for a cell containing p points, the number of points in the first half-cell generated by the subdivision was coded on $\log_2(p+1)$ bits. In fact, this is possible thanks to the arithmetic coding principle: without a suitable method, this number would be coded on $\lceil \log_2(p+1) \rceil$ bits. Hence the gain for each point would be bounded by $\log_2 n - 3$ instead of $\log_2 n - 2.402$.

5.2 Prediction Methods

By coding the number of points in the first half-cell generated by the subdivision on $\log_2(p+1)$ bits (where p is the number of points in the parent cell), we assume that each integer value lying between 0 and p is equiprobable, with the probability $1/(p+1)$. To improve the performances of the algorithm, we can try to estimate more precisely the probability of each of those values. To do so, we study the local densities of points in the neighborhood of the cell being subdivided.

The prediction technique relies on the assumption that the local densities in the current cell are correlated with the local densities in its neighborhood. Thus the algorithm analyses the context taking into account all the available information at this precise time of the coding or decoding. Let us give an example to explain the general principle of the method. Let us assume that we have to subdivide vertically the central cell of the Figure 3. The figure shows the state of the kd-tree of cells at this stage. A very simple manner to determine the most probable repartition of the points in the two half-cells is to calculate the percentage of points in the left neighbor cell with respect to the total number of neighbor points (in the left and right neighbor cells), and then to assume that the half-cells will match this percentage. Here, we count 11 left neighbors over 14 neighbors, which leads to predict 7 points in the left half of the current cell and 2 points in its right half. From that, a basic method consists in estimating the probabilities of the 10 possible values for the left half-cell with a discrete gaussian law centered at 7. Thus

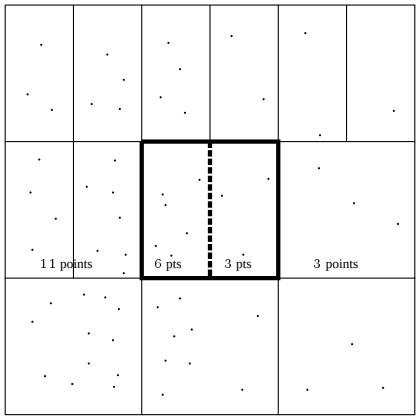


Figure 3: Neighborhood of a two-dimensional cell

the actual value of the left half-cell (6 points) will have a strong estimated probability, and so will be coded on a small number of bits.

In this example, the prediction uses only the first order neighborhood, but the technique can be enlarged to higher orders, by giving more weight to the nearest neighbor cells. In fact, the order of the analysed context can be optimized to achieve a satisfying trade-off between the accuracy of the prediction and the algorithm complexity.

With our setting of the parameters, this prediction method provides an additional gain of about 5% on average, the best results being reached for 3D models whose local densities are the most various. It is to be noted that the simple list of cells used in Section 2 is no longer sufficient, since the prediction needs a suitable data structure for a quick access to the cell neighborhood.

6 TOPOLOGY CODING

We saw in Section 1.2 that the previous methods of geometric compression needed the topology of the object to perform some prediction on the position of the points and so manage compression of this data. The original point in our method is that we obtain similar results (see Section 7.2) without the support of the topological information. That is why this method is particularly well-suited for models whose topology can be automatically reconstructed.

However, for many 3D objects, this automatic reconstruction cannot be done in a lossless way. Moreover, reconstruction algorithms are generally more time-consuming than a single topology decompression, and in some contexts, this overcost can be undesirable (the actual decoding and reconstruction time must be maintained under the transmission time).

Consequently, we propose an algorithm that progressively codes the topology of a geometric structure. This algorithm relies on the same principle of cells subdivision used by the positions coder, so can be applied to the object simultaneously.

6.1 Description Of The Algorithm

Our algorithm codes the minimal connectivity information of the structure in the sense that only the object edges are coded. Consequently, some extra work is necessary to reconstruct the facets of the mesh and the adjacency relations between them when the final application needs these informations. In the following, we will call super-edge an edge between 2 cells. We say that there is a super-edge between 2 cells when there is an edge joining a point of one cell to a point of the other. We call neighbors 2 cells joined by a super-edge.

The algorithm codes these super-edges. Each cell stores a list of its neighbors (at the beginning, the first cell, which is the bounding box of the object, has no neighbor). For each subdivision, we code how the super-edges incident to the original cell are connected to its 2 sub-cells. More precisely, there are 3 possibilities for each super-edge E incident to the original cell C:

- E is incident to the first sub-cell c_0,
- E is incident to the second sub-cell c_1,
- E is incident to c_0 and c_1.

Consequently, for each cell subdivision, each super-edge generates a code on $\log_2(3)$ bits. Besides, one additional bit is necessary to code the possible appearance of a new super-edge between the sub-cells c_0 and c_1. Of course, when a cell subdivision generates an empty cell, there is no topological information to be transmitted. The non empty sub-cell inherits all the super-edges of its parent cell.

Regarding the visualization of intermediate versions of the original model, some post-processing regularization work must be done, since the early stages of the decompression can create edge crossings (see Figure 4). It is only during the final localization, when each cell contains one point, that the topology of the intermediate object has the same properties as the original one.

Figure 4 shows a two-dimensional example (including the original mesh). The code of Figure 5 corresponds to a clockwise description of the super-edges, starting with the bottom-left cell.

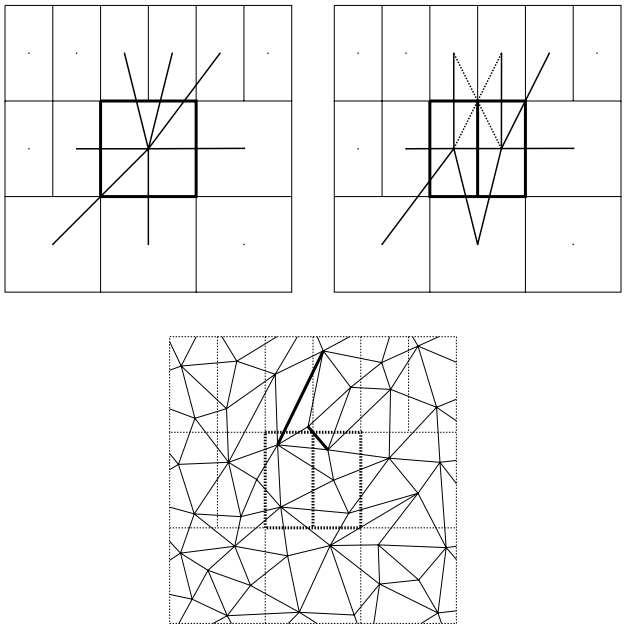


Figure 4: Coding the topology on a two-dimensional example

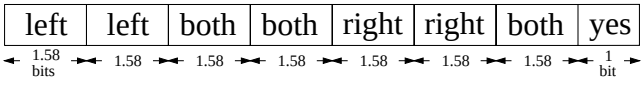


Figure 5: The output code of the two-dimensional example

6.2 Theoretical Analysis

The amount of memory used by this method to code the topology of a geometric structure depends on the average degree of a cell. Indeed, the subdivision of a cell having d neighbors costs exactly $d \log_2(3) + 1$ bits (if it is a "true" subdivision, i.e. if it does not generate an empty cell). Now, it takes exactly $n - 1$ "true" subdivisions to separate the n points of an object, i.e. to code its whole topology. Consequently, if the average degree of a cell is d, the progressive coding of the topology takes less than $d \log_2(3) + 1$ bits per vertex.

The difficulty is to evaluate this average degree, which must not be confused with the average degree of the original geometric object: it is the average number of neighbors of a cell during the separation phase of the algorithm, therefore, it is slightly greater than the average degree of a vertex. Typically, for a triangular mesh, the average degree of a cell is about 7, which corresponds to a cost of 12 bits per vertex for the topology coding.

6.3 Future Work

6.3.1 Prediction

As it is, the compression achieved by this topology coding algorithm is far from the results of [17] or [7]. However, this method should allow us to handle special connectivity cases, like triangular meshes, in a much more efficient way using prediction and entropy coding. For example, the appearance of a new super-edge between the sub-cells is highly probable and can be coded efficiently using entropy coding. Moreover, the connectivity of a parent cell neighbors to the two sub-cells can often be predicted from various informations like cells positions, sizes, or number of points. This work will be described in a forthcoming paper.

6.3.2 Automatic Reconstruction

An example of application where the topology can be reconstructed from the set of points is given by terrain models. In most cases, the connectivity of such models is obtained by computing the Delaunay triangulation of the 2D-projected set of points. In some cases however, slight variations are introduced in the object topology (in order to follow roads or rivers for instance). In such cases, 5 to 10% only of the edges are not Delaunay edges. It suffices to code them, then to compute the constrained Delaunay triangulation to reconstruct the model topology. In 3D, the same technique can be applied to tetrahedral meshes. More generally, the topology coder allows to "help" any reconstruction algorithm by coding a subset of the object edges.

Kim et al. have developed a compression method for terrain models that uses this idea to code only the non Delaunay edges of the model [10]. However, if the algorithm reaches very good results regarding the connectivity, it does not compress the geometric information.

It is worth to be noted that the efficient transmission of 2D Delaunay triangulations is a classical problem in GIS (Geographic Information System). It has been handled by Snoeyink and van Kreveld [13], and more recently, by Sohler [14]. However, in these works, the main purpose is different from ours: the vertex transmission order is used to speed up the triangulation reconstruction (a linear time is obtained instead of $O(n \log n)$). In addition, some compression is achieved by coding the vertex coordinates in a differential way, using variable length codes, but the gain remains relatively small compared to the one generated by pure compression methods.

7 EXPERIMENTAL RESULTS

7.1 Terrain Models

Figure 6 gives results of our method applied to some terrain models. The first two lines come from a GIS database covering the region of Vancouver, whereas the third one corresponds to a simple terrain model composed of 14641 vertices with $12/12/10$ bits coordinates. This example allows to visualize the progression of the decoding on Figures 8 to 11.

	source data	comp. data	comp. ratio	theor. ratio
rivers (120998)	650365 43	341365 22.6	52%	66%
vancouver (908907)	4885376 43	2169750 19.1	44%	60%
terrain (14641)	62225 34	33776 18.5	54%	66%

Figure 6: compression results on terrain models[2]

The last two columns allow to compare the theoretical minimal gain obtained in Section 3.1 to the actual gain achieved by the algorithm.

7.2 Standard 3D Objects

We present in Figure 7 some results of the method described in this paper compared to those of Taubin and Rossignac [16] and Touma and Gotsman [17]. We saw in Section 1.2 that these two methods coded the connectivity and the coordinates of the mesh vertices. However, the figures that appear here concern only the coding of the coordinates, so are comparable to the results of our algorithm. The columns 3 and 4 have been extracted from the article of Touma and Gotsman, and we have applied our method to the same geometric models to obtain the column 5. It is to be noted that the coordinates of these 3D models have been prior quantized on 8 bits to follow exactly the same process as in the first two cited articles. The differences of efficiency come from the structure of the models: "eight" and "shape" have a very regular structure which suits the predictive method of Touma and Gotsman. On the contrary, "dumptruck" and "engine" are more complex models containing variations of local density more suitable for our algorithm. Regarding the method of Cohen-Or et al. [7], we do not have results for the same models, but the authors report an average code size increase of about 10% compared to the results of Touma and Gotsman.

In Figure 12, we show what can be obtained by automatic reconstruction [4] from the set of points, without any connectivity information.

8 CONCLUSION

We have presented a new method of geometric compression whose originality is to perform coordinates compression without the support of the connectivity information. Besides reaching slightly better ratios than the current available algorithms, this technique allows lossless coding and interactive decoding. Moreover, the algorithm is valid for any dimension, and simple to implement. This method is particularly well-suited to any geometric structure whose

[2]The first column gives the number of vertices of the model. The next columns give the size of the data in bytes, and the corresponding number of bits per vertex (bpv).

	source data	IBM 1996	G & T 1998	our algo.
beethoven (2655)	7965 24	4982 15.0	3576 10.8	3201 9.6
blob (8036)	24108 24	10352 10.3	7951 7.9	8937 8.9
cow (3066)	8815 23	4878 12.7	3376 8.8	3419 8.9
dumptruck (11738)	32280 22	20351 13.9	11162 7.6	7218 4.9
eight (766)	2011 21	1146 12.0	683 7.1	1018 10.6
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
triceratops (2832)	7788 22	3673 10.4	2937 8.3	2843 8.0
total (33819)	96875 22.9	54663 12.9	36100 8.5	33180 7.8

Figure 7: compression results on 3D models[2]

topology is reconstructible from its vertices, which is the case of terrain models. However, the progressive topology coding feature permits to handle any geometric structure (not only triangular meshes). Concerning the special case of triangular meshes, a future publication will show how prediction and entropy coding can lead to competitive compression ratios for the connectivity.

On the other hand, future works will consider the use of some 3D surface reconstruction methods [1, 3, 5, 4], which give a necessary and sufficient condition over the object sample to guarantee a valid reconstruction. Thus, by possibly adding a small number of points (or edges, using the topology coder) to the original set, it will be possible to reconstruct the object topology.

Another important perspective is to improve the predictive scheme, whose current results are not completely satisfying. It could be achieved by a more precise analysis of the point distribution in the neighborhood of the current cell, and by applying optimization methods for the parameters setting.

References

[1] Nina Amenta and Marshall Bern. Surface reconstruction by Voronoi filtering. In *ACM SoCG '98 Proc.*, pages 39–48, 1998.

[2] Chandrajit L. Bajaj, Valerio Pascucci, and Guozhong Zhuang. Progressive compression and transmission of arbitrary triangular meshes. In *IEEE Visualization '99 Proc.*, October 1999.

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

[4] Jean-Daniel Boissonnat and Fréderic Cazals. Smooth shape reconstruction. In *ACM SoCG '00 Proc.*, 2000.

[5] Jean-Daniel Boissonnat and Bernhard Geiger. 3-dimensional reconstruction of complex shapes based on the Delaunay triangulation. In *Biomedical Image Processing and Biomedical Visualization*, volume 1905, pages 964–975, 1993.

[6] Mike M. Chow. Optimized geometry compression for real time rendering. In *IEEE Visualization '97 Proc.*, pages 347–354, 1997.

[7] Daniel Cohen-Or, David Levin, and Offir Remez. Progressive compression of arbitrary triangular meshes. In *IEEE Visualization '99 Proc.*, pages 67–72, 1999.

[8] Michael Deering. Geometry compression. In *SIGGRAPH '95 Proc.*, pages 13–20, 1995.

[9] Stefan Gumhold and Wolfgang Strasser. Real time compression of triangle mesh connectivity. In *SIGGRAPH '98 Proc.*, pages 133–140, 1998.

[10] Yang-Soo Kim, Dong-Gyu Park, Ho youl Jung, and Hwan-Gue Cho. An improved tin compression using delaunay triangulation. In *Pacific Graphics '99 Proc.*, October 1999.

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

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

[13] Jack Snoeyink and Marc Van Kreveld. Linear time reconstruction of delaunay triangulations with applications. In *European Symposium on Algorithms Proc.*, 1997.

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

[15] Gabriel Taubin, André Guéziec, William Horn, and Francis Lazarus. Progressive forest split compression. In *SIGGRAPH '98 Proc.*, pages 123–132, 1998.

[16] Gabriel Taubin and Jack Rossignak. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2), 1998.

[17] Costa Touma and Craig Gotsman. Triangle mesh compression. In *Graphics Interface '98 Proc.*, pages 26–34, 1998.

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

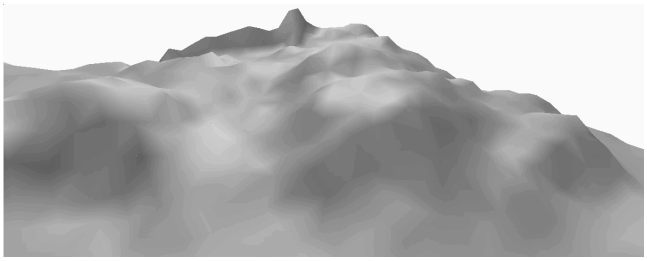


Figure 8: precision = 5 bits, comp. ratio = 3%

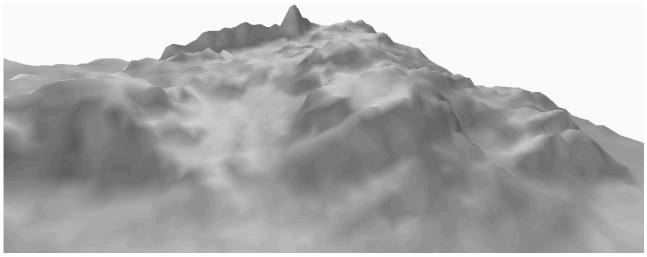


Figure 9: precision = 6 bits, comp. ratio = 8%

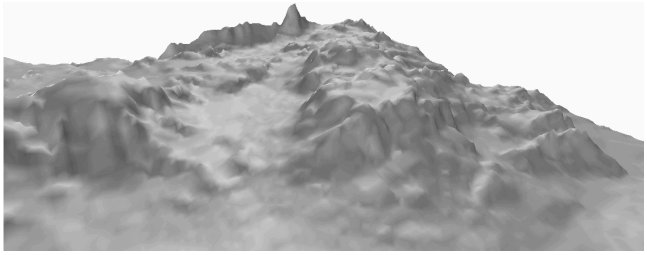


Figure 10: precision = 7 bits, comp. ratio = 16%

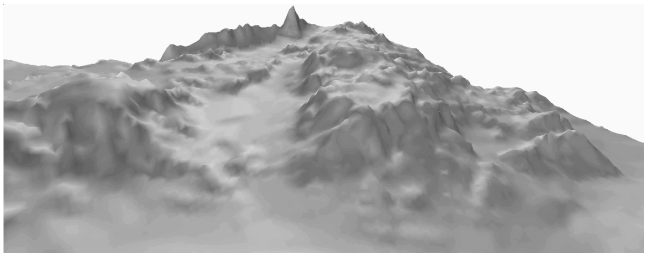
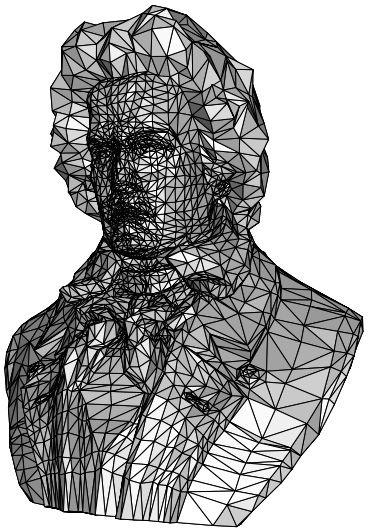
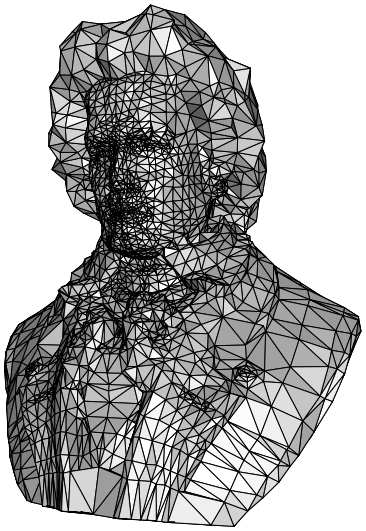


Figure 11: precision = 12 bits (lossless), c. r. = 54%



(a) Original topology



(b) Reconstructed topology

Figure 12: Beethoven (geometry cost: 9.6 bpv)