# Progressive Lossless Compression Of Arbitrary Simplicial Complexes

Pierre-Marie Gandoin Olivier Devillers*

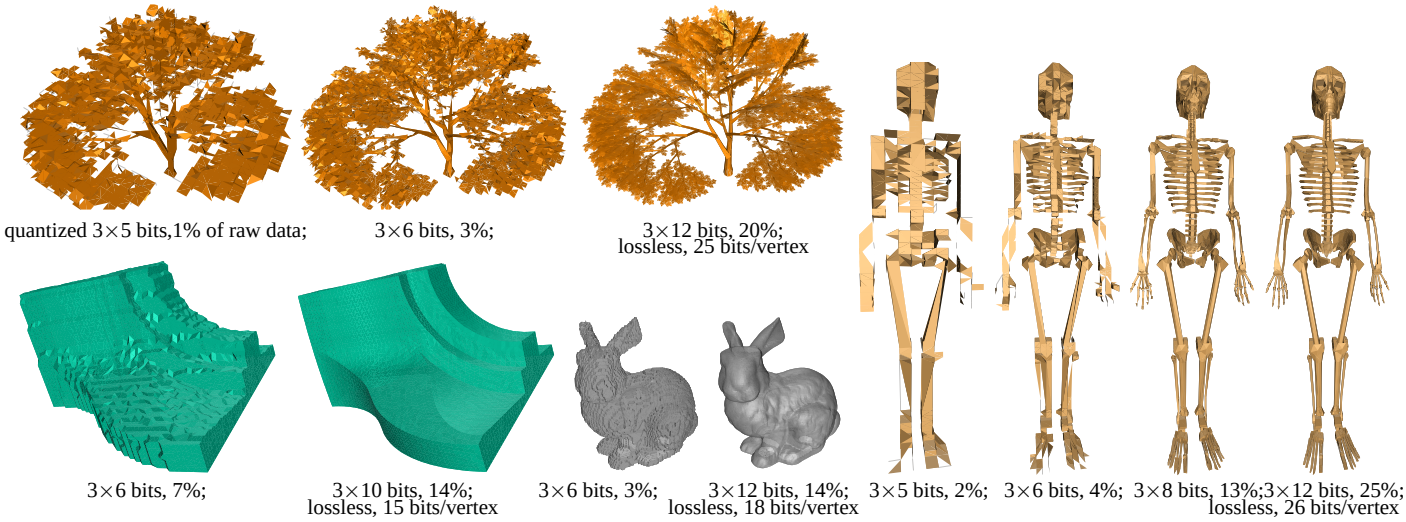


Figure 1: Steps in the progressive decompression of various models.

## Abstract

Efficient algorithms for compressing geometric data have been widely developed in the recent years, but they are mainly designed for closed polyhedral surfaces which are *manifold* or "nearly manifold". We propose here a *progressive* geometry compression scheme which can handle manifold models as well as "triangle soups" and 3D tetrahedral meshes. The method is lossless when the decompression is complete which is extremely important in some domains such as medical or finite element.

While most existing methods enumerate the vertices of the mesh in an order depending on the connectivity, we use a kd-tree technique [Devillers and Gandoin 2000] which does not depend on the connectivity. Then we compute a compatible sequence of meshes which can be encoded using edge expansion [Hoppe et al. 1993] and vertex split [Popović and Hoppe 1997].

The main contributions of this paper are: the idea of using the kd-tree encoding of the geometry to drive the construction of a sequence of meshes, an improved coding of the edge expansion and vertex split since the vertices to split are implicitly defined, a prediction scheme which reduces the code for simplices incident to the split vertex, and a new generalization of the edge expansion operation to tetrahedral meshes.

**Keywords:** Mesh Compression, Non manifold Meshes, Coding, Progressivity, Interactivity

*INRIA Geometrica, BP93, 06902 Sophia-Antipolis, France. Pierre-Marie.Gandoin—Olivier.Devillers@sophia.inria.fr



## 1 INTRODUCTION

Compressing data manipulated by computers has always been, and stays, a crucial necessity since the amount of data grows as fast as the size of computer storage. After text, sound and images, the compression of geometric structures is a new challenge both for storage and for visualization and transmission over the network. For this latter application, we would like to design compression schemes that are progressive, where the information is organized such that a coarse model can be visualized before the transmission is complete.

Most often, a geometric structure consists of a set of points, often referred to as the *geometry* (or *vertex positions*) and the *connectivity* (or *topology*), composed of the adjacency relations between the vertices. The description is sometimes also completed by a list of *attributes* (normals, colors, textures).

### 1.1 Related Work

#### 1.1.1 Connectivity Driven Approach

The main difficulty in the design of a compression scheme is to achieve good compression rates for *both* geometry and connectivity. Regarding the single resolution (non progressive) geometry compression, which dates back to an article by Deering in 1995 using generalized triangle strips [Deering 1995], all the methods *give priority to the connectivity coding*. The common intuitive idea is to describe a spanning tree of vertices and faces by reordering the vertices according to a deterministic strategy to traverse the mesh. This strategy constructs a sequence where each vertex is associated to a label (Rossignac's edge-breaker [1999]), or some other additional information such as the degree of the vertex (Touma and Gotsman's algorithm [1998]), describing the way this vertex is connected to the previous ones in the sequence. Thus, the order of enumeration of the vertices is imposed by the connectivity and the geometric part of the coder tries to get some additional gain, using differential coding or positions prediction: instead of being specified with absolute coordinates, the new vertex position can be expressed relatively to

its predecessors, using a difference vector or some linear predictor. Most of the single-rate compression methods [Deering 1995; Evans et al. 1996; Taubin and Rossignac 1998; Gumhold and Strasser 1998; Touma and Gotsman 1998; Rossignac 1999; Rossignac and Szymczak 1999; King and Rossignac 1999; Isenburg and Snoeyink 1999; Isenburg 2000; Li and Kuo 1998; Bajaj et al. 1999a; Bajaj et al. 1999c; Gumhold et al. 1999; Alliez and Desbrun 2001b] follow this framework and reach costs as low as 2 bits per vertex on average for the most efficient connectivity coders.

Historically, progressive compression methods find their origin in mesh simplification. The general idea is to create a decimation sequence from the original mesh using some canonical operator (vertex or face removal, edge collapse, vertex unification) yielding a very coarse version of the mesh. Furthermore, this decimation sequence is driven by a criterion optimizing the choice of the decimated elements in order to maximize the rate/distortion ratio. Thus if the coarse model is transmitted, followed by a sequence of refinements describing finer models, the client, by truncating the bit stream at any point, is guaranteed to obtain the best approximation of the original object. Unfortunately, such a hierarchical organization often leads to a significant overhead cost in bit size. This is why simplification algorithms were not initially used as *compression* methods (an overview of these algorithms can be found in the very complete survey of Garland and Heckbert [1997]). The first methods for progressive geometric compression are extensions of single-rate methods [Taubin et al. 1998; Bajaj et al. 1999b]. By grouping the refinement operations in batches, some techniques yield results relatively close to those of single resolution methods. For instance, the algorithm proposed by Pajarola and Rossignac [2000a; 2000b] uses vertex bit-marking instead of the costly explicit coding of the vertex indices. The two most efficient methods to our knowledge are based on the vertex removal operator, followed by a canonical retriangulation allowing the decoder to identify the patches resulting from the deletions. The first one is due to Cohen-Or, Levin and Remez [1999], and uses triangle coloring to code the patches. This technique results in connectivity costs of around 6 bits per vertex on usual models, but the strip retriangulation produces intermediate meshes whose visual quality is unsatisfactory. To avoid this problem, Alliez and Desbrun [2001a] propose to preserve the regularity of the mesh during the simplification by maximizing the number of degree 6 vertices. To this aim, the algorithm alternates the main decimation phase with a regularization phase where degree 3 vertices are removed. Besides the better quality of the transitional meshes, this method compresses the connectivity of usual meshes (nearly manifold) down to 3.7 bits per vertex on average. As in single-rate methods, the geometry coding follows from the connectivity coding and is generally based on a linear local prediction. Some of these methods can handle non manifold meshes of small genus, either by coding explicitly the changes in topology or by stitching manifold patches [Guéziec et al. 1999], which generally induces important overcosts. It is also important to note that much lower bit-rates can be reached when the algorithm begins with a complete remeshing of the model to generate regularity and uniformity [Khodakovsky et al. 2000; Karni and Gotsman 2000], which is not admissible in many practical applications where data loss is prohibited.

### 1.1.2 Geometry Driven Approach

Schmalstieg and Schaufler [1997], following Rossignac and Borrel [1993] have tackled the problem from a completely different point of view. They group vertices in clusters on a geometric basis and merge them to construct different levels of details. However, the main goal in their approach is to obtain continuity between coarse to fine approximations of the object, and the achieved compression ratios are not competitive with the current state of the art. In a previous paper [Devillers and Gandoin 2000], we have adopted a similar approach: observing that the geometry is, bitwise, the most expensive part of a mesh, and that in many cases, the connectivity can be automatically reconstructed from the vertex positions, we designed an efficient, purely geometric coder, which constitutes the starting point of our present work. The algorithm, valid in any dimension, is based on a kd-tree decomposition by cell subdivision. Given $n$ 2D points with integer coordinates on $b$ bits, the starting cell is a rectangular bounding box of size $2^b$ by $2^b$. The algorithm starts by encoding the *total number of points* $n$ on an arbitrary fixed number of bits (32 for example). Then it starts the main loop, which consists in subdividing the current cell in two halves along the horizontal axis and then coding the *number of points* contained in one of them (the left one for example) on an optimal number of bits: if the parent cell contains $p$ points, the number of points in the half-cell, which can take the $p+1$ values $0, 1, \ldots, p$, will be coded on $\log_2(p+1)$ bits using arithmetic coding [Witten et al. 1987]. The number of points contained in the second half-cell does not have to be explicitly encoded since it can be deduced from the total number and the number transmitted for the first half-cell, after which each one of the two resulting cells is subdivided along the vertical axis according to the same rule. The process, depicted in Figure 2, iterates until there is no non-empty cell greater than 1 by 1. As shown on Figure 2 (in yellow), the corresponding coding sequence consists only of the numbers of points. The positions of these points are hidden in the order of the output. As the algorithm progresses, the cell size decreases and the transmitted data lead to a more accurate localization. The worst case for the algorithm has been proven to correspond to an uniform distribution of points in the bounding box. In the latter case, the gain is equal to $n\,(\log_2 n - 2.402)$ bits. In practice, the method takes advantage of structured distributions containing variations of local density, which yields generally much better performances (see Section 2.5). The method works in any dimension and in the sequel we will use it for points in dimension 3.

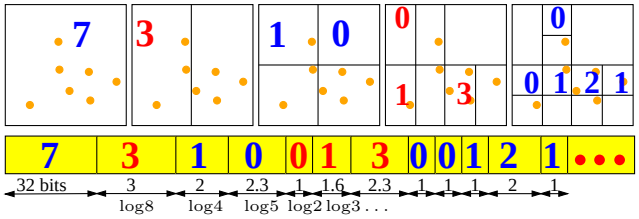


Figure 2: The geometry coder on a two-dimensional example.

Regarding the compression of the connectivity, we propose two alternatives. The first one is to reconstruct the connectivity from the geometry, which is reasonable in special cases such as terrain models or densely sampled objects. The second possibility is an edge-based connectivity coder, but the proposed technique handles only edges and not higher dimensional faces. It is rather expensive and spend more than 12 bits per vertex for a triangular mesh, these bad performances restrict this technique to very specific applications for sparse meshes with few edges.

## 1.2 Overview

In this article, we present a new algorithm for connectivity coding, compatible with the kd-tree geometry coder described above. The general idea is to first run the geometry coder splitting the cells without taking the connectivity into account, then, when the full precision is reached, the connectivity of the model can be added and we can run the splitting process backwards, merging the cells and deducing connectivity between the cells of coarser models (Section 2.1). Connectivity changes between successive models can be encoded by symbols inserted between the numbers of the code of Figure 2.

The changes of connectivity can be described by two well-known decimation operators originally used in a surface simplification context: edge expansion and vertex split. The edge expansion generates short codes in locally manifold cases, while the more expensive (but more general) vertex split operator allows us to treat general models and even unconnected 3D objects like "triangle soups". Compared to classical use of these two operators, we get a cheap description for two reasons: first, in our case, the vertex to be split is implicitly defined and does not need to be referenced explicitly; the second reason is the use of prediction techniques which improve the compression of the connectivity by about $50\%$ (Section 2.4). We get bit-rates of $8$ bits per vertex for the connectivity in the non manifold cases and bit-rates as low as $3$ bits per vertex for nearly manifold meshes usually handled by the geometric compression community. In the manifold case, we are competitive with the most efficient published algorithms [Pajarola and Rossignac 2000b; Cohen-Or et al. 1999; Alliez and Desbrun 2001a], while we reach a continuity in the bit-rate with respect to the manifold/non manifold axis using a unified encoder (Section 2.5).

Furthermore, we show how the method can be extended to volumetric meshes. To this aim, we define a new operator for edge expansion and propose an efficient encoding for it. This approach improves the best progressive method reported for tetrahedral compression [Pajarola et al. 1999] by $25\%$ (Section 3). We finally discuss the possibility to adapt the method to polygonal meshes and conclude in Section 4.

# 2 THE CONNECTIVITY CODER

## 2.1 Principle Of The Algorithm

Starting from the principle described in Section 1.1.2, the key idea consists in defining a connectivity between the cells of the kd-tree to approximate the connectivity of the original 3D model. Then the geometric code is enriched: to the number of vertices in the first half of a split cell is appended a code which describes how the connectivity with other cells evolves during the split. There are now two different problems: on the one hand, we have to associate connectivity to coarse levels where the cells contain several points, on the other hand, the way the connectivity evolves has to be coded efficiently.

Let our model be composed of a point set and a set of simplices (edges and triangles). If we consider some intermediate step of the construction of the kd-tree, we embed the connectivity of the model on the set of cells by creating edges and triangles between the cells if they exist in the original model between the points contained by these cells. This sequence of connectivities for the sequence of sets of cells is constructed in the fine to coarse direction, going back through the subdivision process up to the biggest cell (the object's bounding box), using cell merging. After each merge, the information required by the decoder to restore the original connectivity is encoded.

When two cells are merged, they are replaced by a parent cell. Moreover, each cell is identified to its center-point, representative of all vertices contained in the cell. Therefore, merging two cells is equivalent to unifying the two vertices respectively representing them. Accordingly, in the following, we will use tools and vocabulary originating from progressive mesh simplification. Basically, the vertex merging is performed by the two following decimating operators:

• edge collapse, originally defined by Hoppe *et al.* [1993] and widely used in surface simplification (but also for compression purposes by Pajarola and Rossignac [2000a; 2000b]), will be used to merge two adjacent cells under some hypotheses. The two endpoints of the edge are merged, which leads to the deletion of the two adjacent triangles (only one if the edge belongs to a mesh boundary) degenerating in flat triangles (Figure 3).

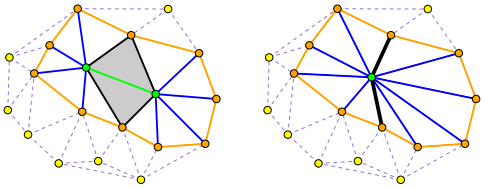

Figure 3: The edge collapse.

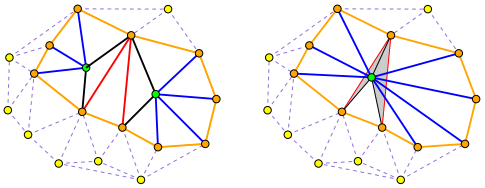

Figure 4: The vertex unification.

• vertex unification, as defined by Popović and Hoppe [1997], is a much more general operation that will allow us to merge any two cells even if they are not adjacent in the current connectivity; the result is non manifold in general (Figure 4).

Each of these operators has a reverse operation: the edge expansion and the vertex split, and their efficient coding will be described in detail in Section 2.2.

In the surface simplification literature discussed previously, the algorithm usually has complete freedom to choose the mesh elements on which the decimating operation is applied. This has two main consequences on the methods. On the one hand, it is possible to optimize the decimation in order to best approximate the original surface as long as possible. Thus to minimize the geometric distortion, a priority queue containing the mesh components is dynamically maintained, and at each decimation, the item minimizing a proximity criterion to the original mesh (or sometimes to the previous version of the mesh) is chosen. On the other hand, in order to let the decoder know which component have been deleted and must be restored, an additional code describing the index of the component among the whole current set must be output.

In our case, on the contrary, the edge to be collapsed or the vertices to be unified are implicitly specified by the subdivision order of the geometric coder. The decoding algorithm uses this implicit definition and the cost of specifying the vertices to apply the operator is avoided. The connectivity coder simply generates a symbol identifying which of the two operators has been used, followed by the parameters detailing the way this operator has modified the connectivity of the current set of cells. The next section shows how these parameters can be efficiently encoded.

## 2.2 Coding Of The Decimation Operators

### 2.2.1 Edge Collapse

The edge collapse operator is very inexpensive in terms of coding, but can be applied only in a quite restrictive context: not only the vertices to merge have to be adjacent, but also the neighborhood of the contracted edge must be manifold and orientable. Under these hypotheses, an edge collapse results in the loss of the two adjacent faces, and the necessary information to code the reverse operation — the edge expansion — consists of the indices of two edges ($VN_2$ and $VN_7$ in Figure 5) among the edges incident to the merged vertex $V$ ($VN_1$ to $VN_{10}$). Actually, since the neighborhood of $V$ is manifold and orientable, the two edges specified unambiguously split the adjacent simplices in two subsets, one of which will be attached to $V_1$, the other to $V_2$.

If the merged vertex has a degree $d$, the description of the two edges to be expanded into triangles costs $\log_2 \binom{d}{2}$. For an average degree equal to $6$, and with arithmetic coding, this leads to a code

size smaller than 4 bits. However, in the context of the cell subdivision, an additional bit is necessary to complete the description. In fact, when a cell is halved, its center-point is split into two vertices whose positions are fixed to the centers of the sub-cells. Consequently, the decoder needs to know which vertex is connected to the red simplices (or equivalently, which one is connected to the blue simplices, see Figure 5).

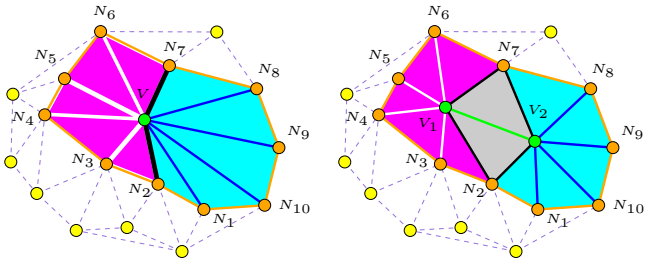


Figure 5: The edge expansion.

In Section 2.4, we will see how simple prediction techniques for the two edges expanded into triangles can result in edge collapse coding using less than 3 bits per vertex on "nearly manifold" and "regular enough" meshes.

### 2.2.2 Vertex Unification

When the vertices to be merged are not adjacent, or when their neighborhood possesses a complex topology, a more general operator is used: the vertex unification, introduced by Popović and Hoppe [1997] in *Progressive Simplicial Complexes*, and whose reverse operation is called generalized vertex split. Using this operator allows us to simplify — but also, in the framework of this article, to efficiently compress — any simplicial complex (i.e. any set containing simplices of dimension 0 (points), 1 (edges), 2 (triangles), 3 (tetrahedra), and so on), which is much more general than the connected manifold case usually handled by previous geometric compression methods.

The counterpart of this genericity is that without careful coding, the description of the generalized vertex split can be extremely expensive. Indeed, the principle is to exhaustively detail the evolution of the simplices incident to the vertex $V$ to be split. More precisely, when $V_1$ and $V_2$ are unified into $V$, any simplex $S$ incident to $V_1$ or $V_2$ is replaced by $S' = (S \setminus \{V_1, V_2\}) \cup V$ (in the case where $S'$ is obtained several times, only one occurrence remains). Consequently, the coding sequence associated to the unification must provide the decoder with the information required to reconstruct the original simplex set. In order to do so, each simplex $S'$ incident to $V$ receives a symbol from 1 to 4 determining its evolution after $V$ has been split:

- code 1: $S'$ becomes $S_1$ incident to $V_1$;
- code 2: $S'$ becomes $S_2$ incident to $V_2$;
- code 3: $S'$ becomes $S_1$ incident to $V_1$ and $S_2$ incident to $V_2$;
- code 4: $S'$ becomes $S_1$ incident to $V_1$, $S_2$ incident to $V_2$, and $S$ of dimension $dim(S') + 1$ incident to $V_1$ and $V_2$. (see Figure 6).

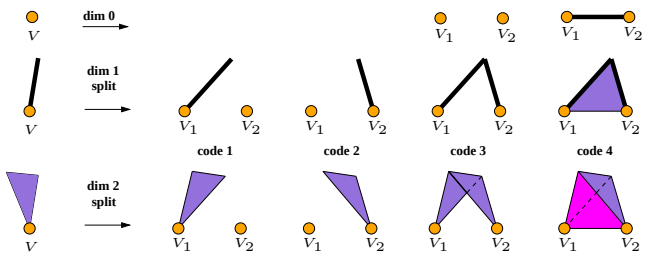


Figure 6: Examples of generalized vertex split.

If naively encoded, this description can lead to prohibitive costs of about 30 bits. Popović and Hoppe propose to optimize the coding by two means. First, they observe that the simplex codes are not independent. Their mutual interaction can be summarized by the following two rules: i) if a simplex $S$ has code $c \in \{1, 2\}$, all the simplices adjacent to $S$ with dimension $dim(S) + 1$ have code $c$; ii) if a simplex $S$ has code 3, none of the simplices adjacent to $S$ with dimension $dim(S) + 1$ have code 4. By coding the simplices adjacent to a vertex to be split by ascending dimensions, and applying these rules and their contrapositives, the cost is reduced by 50% on average. The second optimization suggested by the authors is related to entropy coding. Given a simplex $S$ of dimension $d$ with possible codes $c_1$ to $c_k$, $k \leq 4$ (some codes can be known impossible accordingly to rules i) and ii)), the codes $c_1$ to $c_k$ have not the same probability. Therefore, a statistical array is filled during the decimation process, then the sequence is reversed and the final codes are optimally output: for each simplex, the probability distribution corresponding to its dimension, its potential codes, and its actual code, are sent to the arithmetic encoder. This yields a new gain of about 50% which lowers the final cost down to 8 bits per vertex for usual 3D objects.

## 2.3 Analysis And Features

The decimation algorithm starts from the set of separated vertices with their original connectivity, and performs successive cell/vertex merging according to the fine to coarse sequence of subdivisions generated by the geometric coder, until one vertex representative of the whole point set and centered in the bounding box is obtained. In this section, we give an experimental analysis of the percentage of vertex splits and edge expansions obtained in that context. Since $n - 1$ decimation operations are necessary to completely decimate a set of $n$ points, the global cost of the connectivity coding in bits per vertex is almost:

$$C_{separation} + C_{collapse}\, P_{collapse} + C_{unif}\, P_{unif}$$

where $C_{separation}$ is the size of the header specifying which refinement operator is used, $C_{collapse}$ and $C_{unif}$ are the respective costs of the refinement descriptions, $P_{collapse}$ and $P_{unif}$ are the respective percentages of occurrences of the operators in the coding sequence. The performance of the algorithm thus depends on $P_{collapse}$ and $P_{unif}$; Table 1 shows some statistics for typical models (the last row gives the size of the separation code arithmetically encoded). In usual surface simplification methods where the decimating items are explicitly specified, they are chosen to give priority to edge collapse on vertex unification; here, we do not have this freedom, but we still observe that on nearly manifold models, the edge collapse is quite predominant in the decimation process (up to 96% of the operations).

| models | number of vertices | edge collapse | vertex unifi cation | separation cost |
|---|---|---|---|---|
| triceratops | 2832 | 76.4% | 23.6% | 0.79 bit |
| blob | 8033 | 90.9% | 9.1% | 0.44 bit |
| fandisk | 6475 | 96.0% | 4.0% | 0.24 bit |
| bunny | 35947 | 93.0% | 7.0% | 0.37 bit |
| horse | 19851 | 92.4% | 7.6% | 0.39 bit |
| **average** | 73138 | **92.2%** | **7.8%** | **0.39 bit** |

Table 1: Decimation operators percentages.

The cell subdivision principle that governs the algorithm makes it intrinsically well suited to progressive visualization purposes. As the decoding progresses, cell sizes decrease and the received data allow us to localize the points with more accuracy, and simultaneously, to smooth the object by incorporating the new connectivity information. Therefore it is possible to visualize the set of points at

any stage of the decoding, with smooth transitions between the successive versions using geomorphs. Moreover, the precision over the point coordinates is controlled since it is equal to half the current cell size.

Besides progressivity, the advantage of the method is to provide advanced interactivity to the user. For instance, the geometric coder does not impose an *a priori* quantization of the coordinates: the server can compress the points on as many bits as necessary to preserve the original floating point precision of the model, and the client asks for refinement data until he/she considers the accuracy sufficient for his/her needs. Furthermore, since the cells are structured in a kd-tree, it is possible, during decoding, to select one or more subsets of the scene and to refine them selectively. Hence an interactive navigation through a complex 3D scene is optimized in terms of quantity of data transmitted.

Moreover, the connectivity coder can efficiently handle any simplicial complex in dimension $d$ (see Section 3 for the volumetric meshes extension). In particular, it is important to note that any geometric structure that can be described as a set of edges (i.e. 1-dimensional simplices) is manageable by our algorithm. For instance, this can provide a way to progressively encode polygonal surface meshes (in this case however, a post-process phase is needed to reconstruct the polygons by searching the edge loops). Thus, the edge-based connectivity coder proposed in our previous paper [Devillers and Gandoin 2000] appears as a particular case of our method. Figure 7 gives an example of connectivity code inserted in the geometric code of Figure 2. The portion of code in the figure starts with a geometric vertical split with 3 points on the left, followed by a code of edge expansion and the indices of the two incident edges (1 and 4) expanded in triangles; the three next horizontal geometrical split (001) involves cells with only one point and does not need connectivity code; the last code is a horizontal geometrical split with 2 points above followed by a code of generalized vertex split and the connectivity splitting code for the vertex, the edges and the incident triangles, the simplices are colored on the figure accordingly to their splitting code (due to compatibility rules, only one triangle is really coded).

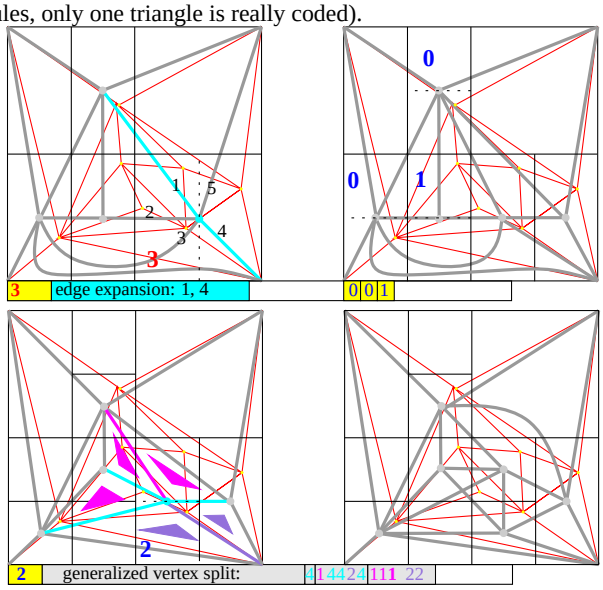


Figure 7: Geometry and connectivity coders.

## 2.4 Prediction

For the encoding of the edge expansion of vertex split, we have consider all cases with the same probability, for example, in Figure 5 the 10 points $N_i$ are considered as possible vertices of the triangles incident to edge $V_1V_2$ with the same probability. If we can compute more realistic probability, then the performance of the arithmetic coding are improved, the most probable $N_i$ get short encoding and the less probable larger one. In the end of this section, we compute such probabilities.

Since the edge collapse operator is by far the most frequent (as shown by Table 1), we focused on the optimization of its description. In the coding sequence, the description of connectivity follows that of geometry, so that when the decoder is about to read the description of an edge expansion, it already knows the geometric position of the vertex $V$ to be split, those of the resulting vertices $V_1$ and $V_2$, and those of the neighbors $N_i$ (see Figure 5). The idea is to exploit this geometric information in order to attempt to guess the two edges having to be expanded into faces to recover the original connectivity (the *cut-edges*). To do so, a score is assigned to each incident edge according to some criterion (defined later), then the scores are normalized and passed to the arithmetic coder; if the prediction scheme is reliable, the actual edges to be expanded obtain a high probability, and thus a short code. Among the numerous criteria we have tested, two stand out by their efficiency and their robustness. The first one is very simple and intuitive: the probability for the edge $VN_i$ to be one of the two cut-edges is defined as a linear function of $|d(N_i, V_1) - d(N_i, V_2)|$ (where $d()$ is the Euclidean distance). For the second predictor, an interpolating plane of $V$, $V_1$, $V_2$ and $N_i$ is first computed, then a Delaunay-like criterion is applied to the projected points: the score of each edge $VN_i$ is inversely proportional to the radius of the circumscribed circle of $\{V_1, V_2, N_i\}$ (see Figure 8). By linearly combining these two criteria, an average gain of up to $40\%$ can be reached for the edge collapse coding sequence. As for the additional bit assigning the two edge subsets (white and blue edges in Figure 5) to the vertices $V_1$ and $V_2$, the gain is even more spectacular since a simple proximity criterion reduces the cost down to $0.2$ bit per operation. The prediction proceeds as follows: if $B_1$ (resp. $B_2$) denotes the barycenter of the neighbors of $V$ in the first (resp. second) subset, the comparison of the expressions $d(V_1, B_1) + d(V_2, B_2)$ and $d(V_1, B_2) + d(V_2, B_1)$ allows us to predict which subset is attached to $V_1$ and which one to $V_2$, with an average reliability of $95\%$ on the models of Table 1.

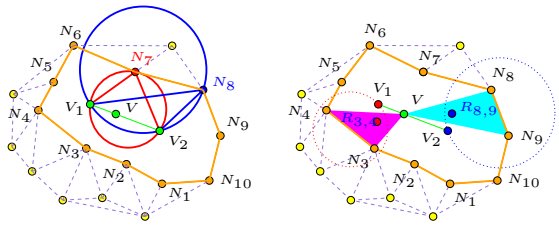


Figure 8: Delaunay (left) and proximity (right) criterions.

For the vertex unification, the same kind of proximity criterion can be used to predict the code of the simplices. In our implementation, we choose to apply prediction schemes only for simplices whose code is 1 or 2, which are much more frequent than codes 3 and 4. We deduce the probabilities for a simplex of barycenter $R$ to be attached to $V_1$ (resp. $V_2$) directly from the distance $d(R, V_1)$ (resp. $d(R, V_2)$) (see Figure 8). The probabilities are then passed to the arithmetic coder and yields a gain of 10 to $20\%$ on the sole vertex unification coding sequence.

It must be noted that, as usually with prediction, the efficiency of all these methods strongly depends on the regularity of the mesh.

## 2.5 Results

Table 2 presents some results of our algorithm compared to those of Pajarola and Rossignac [2000b], Cohen-Or, Levin and Remez [1999] and Alliez and Desbrun [2001a] (a row concerning the Touma and Gotsman [1998] *single-rate* method has been added as reference). For each model and each benchmarked algorithm, the

first line gives the connectivity cost and the second one the geometry cost in bits per vertex. As shown by the last line, our method reaches progressivity with less than $5\%$ overhead compared to the most efficient single resolution algorithms and compares well to other multi-resolution techniques. Generally speaking, the performance comparison between the various published works is made very delicate owing to the disparity of the tested models, as well as the disparity of quantizations on a given model. Since we have not implemented all the methods, Table 2 uses the 3D objects appearing in the different articles. Regarding the quantization, all the models have 12 bits coordinates, except the fandisk whose vertex coordinates are coded on 10 bits.

| models | vertex number | T G 1998 | P R 2000 | C L R 2000 | A D 2001 | our algo |
|---|---|---|---|---|---|---|
| triceratops | 2832 | 2.2<br>20.0 | 7.4<br>21.0 | 5.8<br>20.4 | 5.9<br>25.5 | 6.0<br>19.2 |
| blob | 8033 | 1.7<br>20.0 | 5.9<br>21.0 | 7.6<br>19.7 | 4.3<br>20.6 | 4.1<br>20.1 |
| fandisk | 6475 | 1.1<br>9.0 | 6.8<br>15.0 | ? | 5.0<br>12.3 | 2.9<br>12.1 |
| bunny | 35947 | ? | 7.0<br>16.0 | ? | 4.0<br>15.4 | 3.1<br>14.8 |
| horse | 19851 | 2.3<br>17.0 | ? | 5.7<br>15.4 | 4.6<br>16.2 | 3.9<br>16.4 |
| **average** | **73138** | **2.0**<br>**16.5** | **7.1**<br>**16.9** | **5.8**<br>**17.0** | **4.4**<br>**16.3** | **3.5**<br>**15.7** |
| **total** | | **18.5** | **24.0** | **22.8** | **20.7** | **19.2** |

Table 2: Results on manifold models in bits per vertex for connectivity (number above) and geometry (number below).

As shown in Section 2.2.2, the vertex unification operator makes our algorithm applicable to a much wider range of geometric structures than the manifold triangulated surfaces. Table 3 gathers the results of our algorithm on 3D objects modeled with triangle soups (these objects are freely available on the 3DCafe web site: *http://www.3dcafe.com/asp/meshes.asp*). The connectivity and geometry bit-rates are separated as previously, and the vertex coordinates have been quantized on 12 bits for all the models. Contrary to the objects tested in Table 2, the edge collapse occurs barely during the decimation process: it represents less than $5\%$ of the operations. As a result, the average bit-rate of the connectivity goes up to $8$ bits per vertex.

| models | vertex number | results | models | vertex number | results |
|---|---|---|---|---|---|
| aqua05 | 16784 | 8.5<br>16.4 | grass14 | 29224 | 7.5<br>18.8 |
| maple01 | 45499 | 8.2<br>16.9 | skeleton | 6103 | 11.4<br>15.9 |
| m_tree1 | 17782 | 7.9<br>16.0 | **average** | 115392 | **8.2**<br>**17.1** |

Table 3: Results on triangle soups in bits per vertex for connectivity (number above) and geometry (number below).

Figures 9 and 10 show the progressive decompression for a manifold surface (the *triceratops*) and a triangle soup (the *tree*, presented in a global view and a zoomed in region). For the *triceratops*, the size of the compressed data goes from $4\%$ (for the first low precision version) to $25\%$ (for the final lossless version) of the original data in their raw form (3 x 12 bits per vertex for the positions plus 3 x $\log_2 n$ bits per triangle for the connectivity). As for the *tree*, the compressed size spreads from $8\%$ to $45\%$ of the original raw size. We also draw the average positionning error of the vertices as the decompression evolves. Figures 1 and 12 show more examples of progressive decompression.

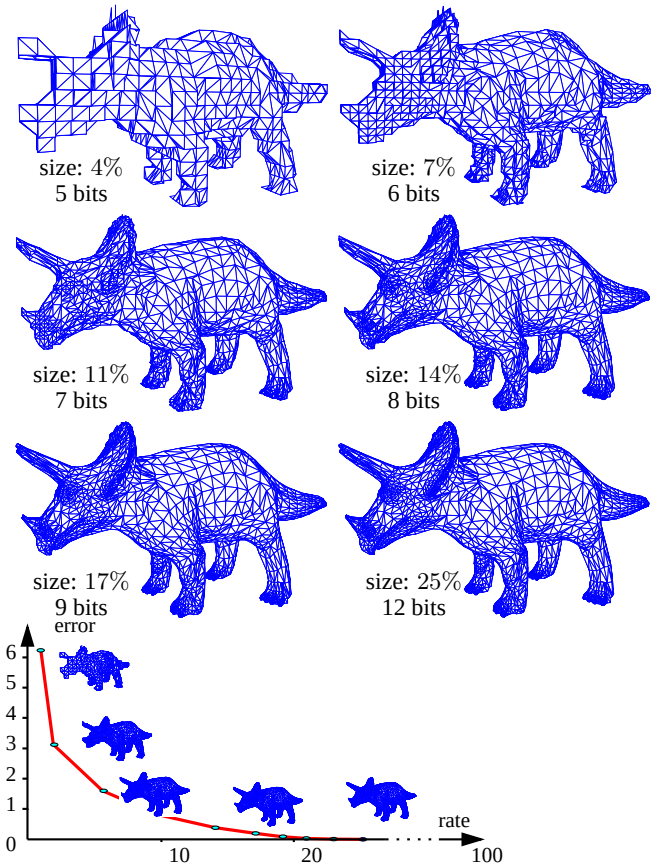


Figure 9: Rate distorsion on the *triceratops* model.

## 3 VOLUMETRIC MESHES EXTENSION

Volumetric meshes have been extensively used in finite element for years, and are also more and more widespread in volume visualization. More specifically, tetrahedral meshes, which offer a direct and flexible way to interpolate numerical values in any point in space, have established themselves as the most natural and powerful tool for volume representation.

Although the need for tetrahedral compression is clear (the connectivity being by far more expensive than for surface meshes), relatively few methods have been proposed up to now. Regarding the progressive coding, the only article tackling the progressive tetrahedral mesh compression is due to Pajarola, Rossignac and Szymczak [1999], and uses an edge collapse operator applied to successive batches of independent edges. Besides the cost of the vertex split description (the *implant*), an additional bit per vertex and per batch is used to identify the vertices to be split. Thus the global cost depends on the number of independent edges collapsed in each batch. To avoid the appearance of non manifold regions during the simplification process, some edge collapses are forbidden. As for the compression of the vertex positions, the authors suggest to exploit the prediction techniques designed for triangular meshes.

### 3.1 Generalization Of The Decimation Operators

The generalization of the vertex unification operator described in Section 2.2.2 is straightforward in any dimension. As a result, the progressive geometry and connectivity coding algorithms we proposed in the framework of triangular structures are easily applica-

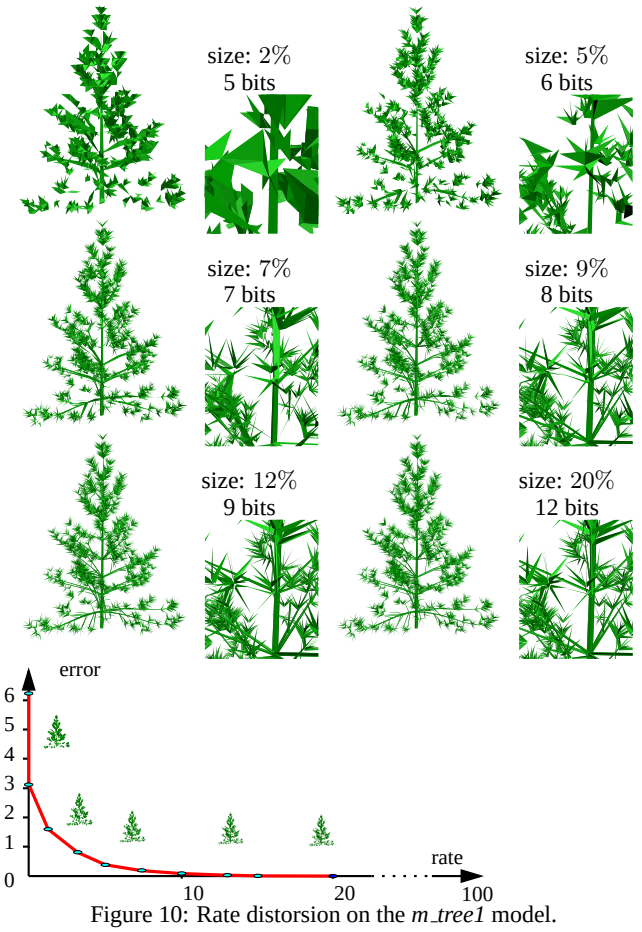

Figure 10: Rate distorsion on the *m_tree1* model.

ble to any simplicial complex in dimension $d$. However, without a low cost operator equivalent to the edge collapse described in Section 2.2.1, the performances regarding the connectivity compression will not be competitive. More precisely, the average cost of a vertex unification for tetrahedral meshes is 120 bits without optimization, and around 40 bits by applying the two rules stated in Section 2.2.2 plus arithmetic coding.

We suggest here a decimation operator equivalent to the edge collapse adapted to tetrahedral meshes. Similarly to the case of polyhedral surfaces, we improve the coding of special vertex unifications that do not create topological changes in their neighborhood. More precisely, this 3D edge collapse can be used when the reverse operation fulfills the following conditions:
• the vertex $V$ to be split has a code 4 (i.e. the split is an edge expansion);
• the set of triangles incident to $V$ having a code 4 (i.e. the triangles generating a tetrahedron after the split) forms a manifold surface $M$, and $V$ is not on its boundary;
• the remaining edges (i.e. the edges adjacent to $V$ that do not belong to $M$) have a code $c \in \{1, 2\}$;
• $M$ separates the set of code 1 edges from the set of code 2 edges (see Figure 11).

When the conditions are satisfied, the piece of code that allows to restore the neighborhoods of the vertices $V_1$ and $V_2$ before the unification is composed of: i) the number of code 4 triangles adjacent to $V$; ii) the indices of these triangles among the set of triangles incident to $V$; iii) the indices of the edges lying on the "first" side of $M$. As shown in Section 2.4, the cost can be lowered with the help of prediction techniques combined with arithmetic coding. On average, the final cost of this 3D equivalent of the edge collapse is about 20 bits.

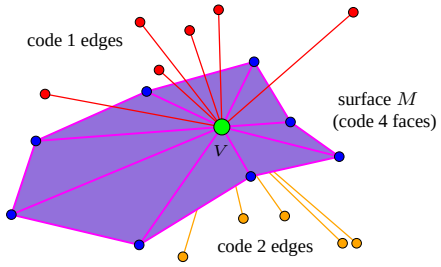

Figure 11: Good case for the 3D edge expansion.

### 3.2 Results

As in the coding of triangular structures, the global cost depends essentially on the occurrence percentages of the two operators. For the Delaunay tetrahedralization of $10,000$ points points uniformly distributed in a sphere, the 3D edge collapse occurs in less than $45\%$ of the vertex merging; we thus reach a bit-rate of 34 bits per vertex to encode the mesh connectivity and 35 bits per vertex for the geometry if the point coordinates are quantized on 16 bits. We obtain a final compression rate of $15\%$ if we compare with a direct storage with 16 bits per coordinate and $4 \times \log_2 n$ bits per tetrahedron.

These results can be compared with those obtained by Pajarola, Rossignac and Szymczak [1999]for progressive tetrahedral compression. For a random Delaunay tetrahedralization containing $10,000$ vertices, their edge collapse operator, whose functionalities and coding are quite different from ours, yields a total cost around 45 bits per vertex for the connectivity (taking into account the base mesh plus the refinement cost). Moreover, the progressivity is limited since the base mesh cannot be arbitrary small (it represents about 1/4 of the total cost). The geometry coding is not addressed in this article, just like in Yang, Mitra and Chiueh work [2000], which tackle the progressive coding of tetrahedral meshes from a rendering point of view.

We have also tested our algorithm on meshes coming from real applications: for a mesh of a Falcon business jet, (courtesy of Dassault-Aviation) with 10188 vertices and 54911 tetrahedra with coordinates on 16 bits, we obtain $41\%$ of edge collapses and a bit-rate of 23 bits per vertex for the geometry and 25 bits per vertex for the connectivity.

## 4 CONCLUSION AND FUTURE WORK

We have presented a new progressive connectivity coding algorithm based on surface simplification techniques optimized for compression purposes. This work is built on the kd-tree geometric coder [Devillers and Gandoin 2000], and as such, allows an efficient joint compression of the positions and the connectivity of the mesh. Besides the good compression rates, which are competitive with the most efficient progressive methods, the algorithm has the advantage of being applicable to any simplicial complex, including in particular non manifold triangulations and triangle soups. Furthermore, the method can be extended to any dimension and yields in the case of tetrahedral meshes a cost reduction of about $25\%$ compared with previous work.

Future work will address the extension of the algorithm to polygonal meshes. Indeed, the method permits the compression of such geometric structures by describing them as a set of edges, and by reconstructing the polygons from this set by loop searching, but it should be more efficient to define an edge collapse operator adapted to polygons with a size greater than 3. Also, we think the compression ratios can be still improved, in particular by combining connectivity and geometry prediction as the refinement process proceeds.

**Acknowledgments:** Thanks to Pierre Alliez, Alain Dervieux, George Drettakis and Monique Teillaud for their help.

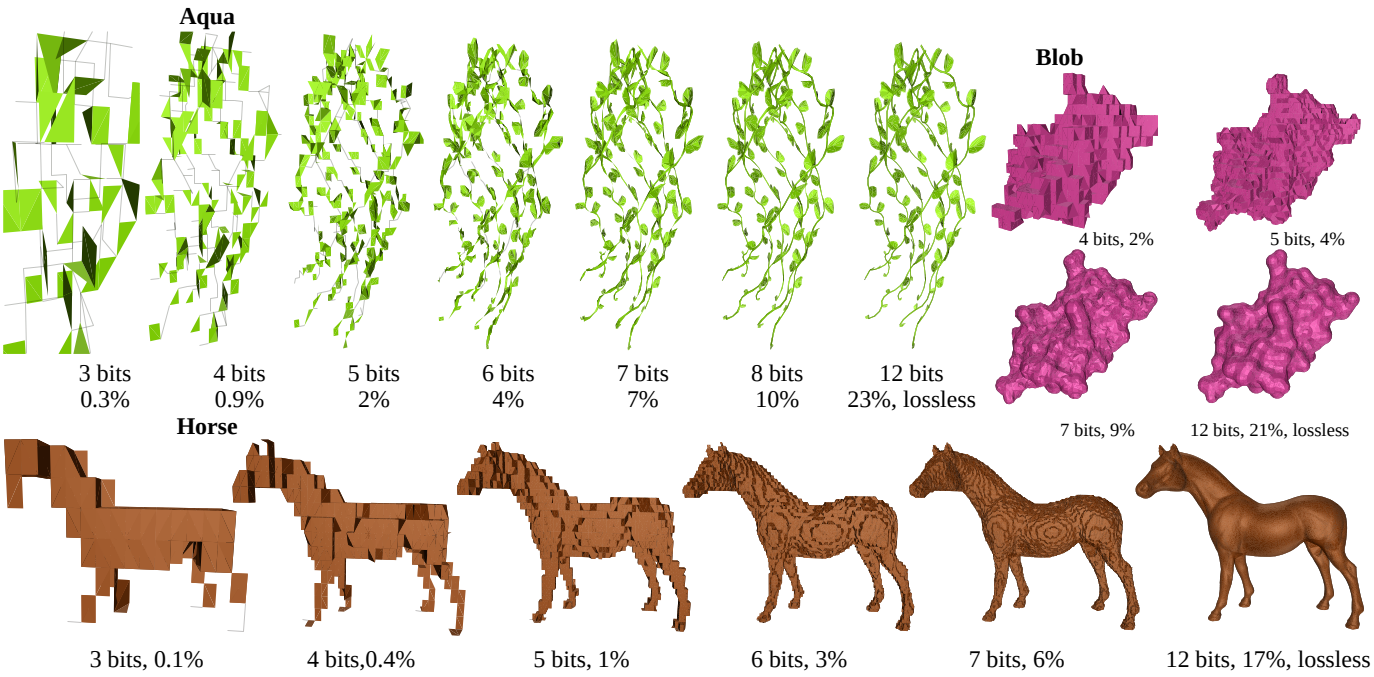


Figure 12: More examples of progressive decompression.

## References


ALLIEZ, P., AND DESBRUN, M. 2001. Progressive compression for lossless transmission of triangle meshes. In *SIGGRAPH 2001 Conference Proc.*, 199–202.

ALLIEZ, P., AND DESBRUN, M. 2001. Valence-driven connectivity encoding for 3d meshes. In *Eurographics 2001 Conference Proc.*, 480–489.

BAJAJ, C., CUTCHIN, S., PASCUCCI, V., AND ZHUANG, G. 1999. Error resilient streaming of compressed vrml. Tech. rep., University of Texas.

BAJAJ, C., PASCUCCI, V., AND ZHUANG, G. 1999. Progressive compression and transmission of arbitrary triangular meshes. In *IEEE Visualization 99 Conference Proc.*, 307–316.

BAJAJ, C., PASCUCCI, V., AND ZHUANG, G. 1999. Single resolution compression of arbitrary triangular meshes with properties. *Computational Geometry : Theory and Applications*, 247–296.

COHEN-OR, D., LEVIN, D., AND REMEZ, O. 1999. Progressive compression of arbitrary triangular meshes. In *IEEE Visualization 99 Conference Proc.*, 67–72.

DEERING, M. 1995. Geometry compression. In *SIGGRAPH 95 Conference Proc.*, 13–20.

DEVILLERS, O., AND GANDOIN, P.-M. 2000. Geometric compression for interactive transmission. In *IEEE Visualization 2000 Conference Proc.*, 319–326.

EVANS, F., SKIENA, S., AND VARSHNEY, A. 1996. Optimizing triangle strips for fast rendering. In *IEEE Visualization 96 Conference Proc.*, 319–326.

GUÉZIEC, A., BOSSEN, F., TAUBIN, G., AND SILVA, C. 1999. Effi cient compression of non-manifold polygonal meshes. *Comput. Geom. Theory Appl. 14*, 137–166.

GUMHOLD, S., AND STRASSER, W. 1998. Real time compression of triangle mesh connectivity. In *SIGGRAPH 98 Conference Proc.*, 133–140.

GUMHOLD, S., GUTHE, S., AND STRASSER, W. 1999. Tetrahedral mesh compression with the cut-border machine. In *IEEE Visualization 99 Conference Proc.*, 91–98.

HECKBERT, P. S., AND GARLAND, M. 1997. Survey of polygonal surface simplifi cation algorithms. Tech. rep., Carnegie Mellon University.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1993. Mesh optimization. In *SIGGRAPH 93 Conference Proc.*, 19–26.

ISENBURG, M., AND SNOEYINK, J. 1999. Mesh collapse compression. In *Symposium on Computational Geometry*, 419–420.

ISENBURG, M. 2000. Triangle fi xer : Edge-based connectivity encoding. In *16th European Workshop on Computational Geometry Proc.*

KARNI, Z., AND GOTSMAN, C. 2000. Spectral compression of mesh geometry. In *SIGGRAPH 2000 Conference Proc.*, 279–286.

KHODAKOVSKY, A., SCHRÖDER, P., AND SWELDENS, W. 2000. Progressive geometry compression. In *SIGGRAPH 2000 Conference Proc.*, 271–278.

KING, D., AND ROSSIGNAC, J. 1999. Guaranteed 3.67v bit encoding of planar triangle graphs. In *Canadian Conference on Computational Geometry Proc.*, 146–149.

LI, J., AND KUO, C.-C. J. 1998. A dual graph approach to 3d triangular mesh compression. In *IEEE International Conference on Image Processing Proc.*

PAJAROLA, R., AND ROSSIGNAC, J. 2000. Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics 6*, 1 (January–March), 79–93.

PAJAROLA, R., AND ROSSIGNAC, J. 2000. Squeeze : Fast and progressive decompression of triangle meshes. *CGI 2000 Proc.*, 173–182.

PAJAROLA, R., ROSSIGNAC, J., AND SZYMCZAK, A. 1999. Implant sprays : Compression of progressive tetrahedral mesh connectivity. In *IEEE Visualization 99 Conference Proc.*, 299–306.

POPOVIĆ, J., AND HOPPE, H. 1997. Progressive simplicial complexes. In *SIGGRAPH 97 Conference Proc.*, 217–224.

ROSSIGNAC, J., AND BORREL, P. 1993. *Geometric Modeling in Computer Graphics*. Springer-Verlag, July, ch. Multi-Resolution 3D Approximations for Rendering Complex Scenes, 455–465.

ROSSIGNAC, J., AND SZYMCZAK, A. 1999. Wrap&zip : Linear decoding of planar triangle graphs. *Computational Geometry : Theory and Applications*, 119–135.

ROSSIGNAC, J. 1999. Edgebreaker : Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 47–61.

SCHMALSTIEG, D., AND SCHAUFLER, G. 1997. Smooth levels of detail. In *IEEE Virtual Reality Annual International Symposium*, 12–19.

TAUBIN, G., AND ROSSIGNAC, J. 1998. Geometric compression through topological surgery. *ACM Transactions on Graphics 17*, 2, 84–115.

TAUBIN, G., GUÉZIEC, A., HORN, W., AND LAZARUS, F. 1998. Progressive forest split compression. In *SIGGRAPH 98 Conference Proc.*, 123–132.

TOUMA, C., AND GOTSMAN, C. 1998. Triangle mesh compression. In *Graphics Interface 98 Conference Proc.*, 26–34.

WITTEN, I., NEAL, R., AND CLEARY, J. 1987. Arithmetic coding for data compression. *Communications of the ACM 30*, 6, 520–540.

YANG, C., MITRA, T., AND CHIUEH, T. 2000. On-the-fly rendering of losslessly compressed irregular volume data. In *11th IEEE Visualization Conference*, 329–336.