The Visual Computer manuscript No.
(will be inserted by the editor)

Sylvain Prat · Patrick Gioia · Yves Bertrand · Daniel Meneveaux

Connectivity Compression in Arbitrary Dimension

Abstract This paper presents a general lossless connectivity compression scheme for manifolds in any dimension with arbitrary cells, orientable or not, with or without borders. Relying on a generic topological model called *generalized maps*, our method performs a region-growing traversal of its primitive elements while describing connectivity relations with symbols. The set of produced symbols is compressed using standard data compression techniques. These algorithms have been successfully applied to various models (surface, tetrahedral and hexahedral meshes), showing the efficiency and genericity of the proposed scheme.

Keywords: compression, connectivity, topological structure, generalized maps

1 Introduction

High-range laser scanners provide highly complex models, demanding more and more space on storage devices. Moreover for remote walkthrough applications (such as navigation in virtual urban environments), dataflow associated with geometry is practically limited by low network bandwidth. Both of these reasons have lead to a research area dedicated to compression algorithms for geometrical models.

Sylvain Prat
France Telecom R&D
E-mail: sylvain.prat@rd.francetelecom.com

Patrick Gioia
France Telecom R&D
E-mail: patrick.gioia@rd.francetelecom.com

Yves Bertrand
Laboratoire SIC
Poitiers, France
E-mail: yves.bertrand@univ-poitiers.fr

Daniel Meneveaux
Laboratoire SIC
Poitiers, France
E-mail: daniel@sic.univ-poitiers.fr

Among geometry compression schemes, connectivity-based approaches preserve neighborhood relationships between vertices, edges, faces, etc. Connectivity information is required in various contexts such as geometrical modeling operations, efficient data structures construction for visualization algorithms (e.g. visibility graphs) or mesh decimation for levels of detail management. According to these methods, geometry is predicted using the previously processed neighborhood and incrementally compressed.

Numerous methods have been proposed to compress the connectivity associated with models for triangle meshes, surface meshes, tetrahedral or hexahedral meshes [1,8,10,3,6]. However, to our knowledge, none of these compression methods can be directly generalized to any dimension and arbitrary cells. In practice, data structures and algorithms are very specific of each class of subdivision. For example, general polyhedral models used in geology applications require some heavy preprocessing before being compressed with existing methods. Such preprocessing would imply to remesh the entire scene, dramatically increasing the number of cells and changing the original connectivity.

In this paper, we propose a new lossless connectivity compression scheme applicable to any manifold, composed of arbitrary cells, with or without boundary, either orientable or not (e.g. Klein bottle or Moëbius strip), with arbitrary cells (arbitrary faces, tetrahedra, hexahedra, polyhedra, etc.). For representing such topological models, we have chosen the *generalized map* data structure since relationships between its primitive elements are homogeneous over all dimensions and simplifies algorithms generalization. Our technique is restricted to manifold meshes. However, to deal with few non-manifoldness, one could duplicate non-manifold vertices, edges, faces and the corresponding geometrical attributes to obtain a manifold mesh. This obviously implies losing some topological information.

Our method carries out a compact description of G-maps while performing a region-growing traversal of primitive elements. Connectivity information is ex-

pressed by symbols corresponding to the traversal histories. This set of symbols is then compressed with existing data compression techniques such as dictionary-based coding. Geometrical attributes are independently compressed using well-known existing methods.

The contributions of our paper include a generic method for compressing manifold models:

- with dimension-independent processing;
- composed of arbitrary cells;
- transparently including border information.

This paper is organized as follows. Section 2 presents previous work related to connectivity compression. Definitions are given in Section 3 while the general description of our algorithm is provided in Section 4. Technical details of our method are explained in Section 5. Finally, some results are shown and discussed in Section 6.

2 Related Work

2.1 Terminology

A space *subdivision* (or mesh) is a decomposition of a space into *cells* (vertices, edges, faces, volumes...). A *topological model* represents such subdivisions and contains two types of information: (i) *connectivity*, which describes the incidence/adjacency relations between cells and (ii) *geometry*, which describes the object shape and appearance such as vertex positions, texture coordinates, colors, textures, etc. In the following, the term *dimension* applies to connectivity independently of geometry. For example, the mesh as depicted in Figure 8(a) is two-dimensional since it is a surface although vertex coordinates are embedded in a 3D Euclidian space.

2.2 Mesh Compression

3D mesh compression is a very active research field which have lead to numerous works, whose most relevant ones could be found in [1] for the surface case. However, two different approaches are used to compress mesh representations:

- connectivity-based techniques proceed in two steps: (i) the connectivity of the mesh is (losslessly) compressed using a region-growing traversal of the elements of the mesh (e.g. triangles); (ii) the connectivity information is efficiently used to predict and (lossily) compress the geometry (e.g. vertex locations) from the previously processed ones during the traversal;
- multi-resolution techniques use a progressive approach: a base mesh and progressive refinements are compactly encoded into the stream. Prior to compression, a regular or semi-regular remeshing is often performed in order to obtain very good compression rates.

In many applications such as CAD modeling or geology, any loss in connectivity is unacceptable. We thus mainly focus on the first approach. The principle of *connectivity-based* methods for surface meshes, or tetrahedral [8,10,3] and hexahedral [6] meshes is as following:

- a region-growing traversal of the mesh elements is performed. Depending on the mesh type, these elements are triangles, faces, tetrahedra or hexahedra;
- during this traversal, symbols describing the connections between the elements are recorded and further compressed using (lossless) standard data compression techniques. They allow the reconstruction of the original mesh connectivity at the decoder side. The symbols could be (i) labels identifying the various incidence cases between the current element and previously visited ones, or (ii) valences and degrees of vertices and faces;
- during the same traversal, each new vertex encountered is encoded: it is first quantized (lossy step), then predicted using previously visited neighborhood and finally compressed using entropy-based compression techniques.

However, to our knowledge, none of these compression methods can be generalized to any dimension and arbitrary cells since structures and algorithms are specific to the class of meshes compressed. We thus have designed a new connectivity-based scheme to solve this problem, which can handle polyhedral meshes.

3 Definitions

A generalized map (or G-map) [7] is a topological model defined for representing manifolds, orientable or not, with or without borders, in arbitrary dimension. It is composed of a single type of primitive elements called *darts* and incidence relations (connectivity) between these elements called *involutions*. In this representation, the cells (vertices, edges, faces, ...) are not explicitly represented but deduced from these relations.

A generalized map of dimension N (or N-G-map) is defined as a $(N+2)$ tuple $G = (B, \alpha_0, \alpha_1, ..., \alpha_N)$ where B is the set of all the darts and α_i are the $N+1$ relations between darts. These relations are permutations which satisfy the following properties:

1. $\forall i, 0 \leq i \leq N$, α_i is a *simple* involution, that is $\alpha_i^2(d) = d$;
2. $\forall i, \forall j, 0 \leq i < i+2 \leq j \leq N$, $\alpha_i.\alpha_j(d)$ is also an involution, called *composed* involution.

This structure is nearly equivalent to that of Brisson [2]: darts correspond to cell-tuples and α_i involutions correspond to `swap(i)` operations. However, additional constraints are present in G-maps to ensure the manifold property.

Figure 1 represents a generalized map corresponding to triangle connected to a quadrilateral. Darts are

represented as black half-split-edges and involutions as colored links on the figure. Table 1 describes connectivity for every dart.

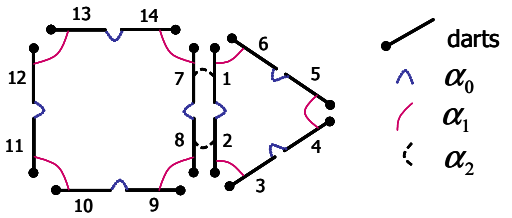


Fig. 1 *2-G-map example representing a triangle connected to a quadrilateral: darts are shown in black and involutions in various colors according to their dimension.*

b	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\alpha_0(b)$	2	1	4	3	6	5	8	7	10	9	12	11	14	13
$\alpha_1(b)$	6	3	2	5	4	1	14	9	8	11	10	13	12	7
$\alpha_2(b)$	7	8	3	4	5	6	1	2	9	10	11	12	13	14

Table 1 *Involutions: the first row represents dart numbers while the other ones indicate incidence relations.*

The first property ensures that darts are paired on every dimension i except if the dart is incident to a i-border. On Figure 1, darts 1 and 7 are linked with α_2 and dart 3 is incident to a 2-border (surface border): $\alpha_2(1) = 7, \alpha_2(7) = 1, \alpha_2(3) = 3$. The second property corresponds to the manifold constraint: it ensures that cells to be stitched together are homeomorphic. This will be explained hereafter.

Orbits denoted as $\langle \alpha_{i_1}, \alpha_{i_2}, ..., \alpha_{i_n} \rangle(d)$ define the set of darts reached by applying any composition of the given set of involutions (called permutation group) to dart d. For example, in 2D, $\langle \alpha_1, \alpha_2 \rangle(1) = \{1, 6, 7, 14\}$ defines the vertex orbit (0-cell) associated to dart 1, i.e. the vertex incident to dart 1. *i-cells* are defined by orbits using all the involutions except α_i. Intuitively, the involution α_i makes it possible to leave a given i-cell and enter into the adjacent one along the considered dart. Traversing along all the involutions composing the orbit provides the set of all darts in the cell. For example, $\langle \alpha_0, \alpha_1 \rangle(1) = \{1, 2, ..., 6\}$ is the face orbit (2-cell) associated to dart 1.

Back to the manifold property, suppose we want to assemble two volumes (3-dimensional cells) along a face (2-dimensional cell) in a volume mesh, both faces from the two volumes to be stitched should have the same number of edges. In terms of generalized maps, we want to sew by α_3 two faces defined by the orbits $\langle \alpha_0, \alpha_1, \alpha_3 \rangle(d_1)$ and $\langle \alpha_0, \alpha_1, \alpha_3 \rangle(d_2)$. Every matching pair of darts from both orbits should then have the same connectivity for α_0 and α_1 according to the second constraint of generalized maps.

Geometrical attributes are often associated to only one dart in the whole orbit. For example darts 1, 2, 4, 10 and 12 on Figure 1 could hold their respective 3D vertex coordinates. To find one attribute attached to an orbit, a traversal of the orbit is performed to find the first dart owning the type of attribute we are looking for. This technique facilitates attributes management during topological or geometrical modifications.

The main advantage of generalized maps concerns the homogeneous definition of models in arbitrary dimension, thanks to involutions: this is a great advantage for algorithms generalization.

The data structures used to represent generalized maps are depicted in Figure 2. For each dart of the generalized map, the array `alpha` stores $N+1$ pointers to the dart images obtained by each α_i involution. The `marks` attribute allows labeling of darts: every bit can be used to identify a state for the dart during an algorithm. In particular, marks are used during traversals for identifying visited elements. Geometric attributes may be stored in the `attr` list. The G-map structure contains a list of all darts to facilitates the deletion or unmarking of all darts.

```
struct Dart {
  Dart* alpha[N+1]; // the N+1 involutions
  int marks; // marks for traversals
  list<Attributes*> attr; // geometrical attributes
};

struct GMap {
  list<Dart*> darts; // list of darts
};
```

Fig. 2 *C++ style G-Map data structures.*

Assuming 32 bits integers or pointers and a volume mesh (4 involutions), each dart accounts for 24 bytes which leads to 144 bytes per triangle, 576 per tetrahedron and 1152 bytes per hexahedron, and each vertex attribute accounts for 12 bytes. Ignoring the dart list, the structure size for the connectivity and geometry would be 372 MB large for the 2.6 million triangles Buddha model from the Digital Michelangelo project assuming 2 triangles per vertex. The shaft model depicted in Figure 8(d) and the heart model depicted in Figure 8(c) would require respectively 8 MB and 137 MB of storage space. These examples show why compression is required in most situations.

4 Method Overview

Our algorithm follows a strategy similar to previously introduced methods, with a region-growing traversal of the mesh and symbol histories representing the description of incidence relations between visited elements. Histories

are then compressed using a common data compression technique, a dictionary-based coder (a variant of LZW [9]) in this case.

As opposed to other existing methods, the geometry associated with the model is processed independently of connectivity. This allows the processing of any type of embedding, and thus is not limited to per-vertex attributes (e.g. face colors, 3D textures...). Vertices can be compressed using the classical methods mentioned previously, i.e. quantization, prediction using the decoded neighborhood and compression of residuals.

Our algorithm proceeds as follows:

- a region-growing traversal of the mesh is performed in order to describe the connectivity for all darts;
- for a given dart, each involution is described by a symbol depending on the connection configuration;
- the set of symbols stored during the traversal composes the intermediate representation called *histories*;
- histories are compressed;
- for each type of embedding, a specific mesh traversal is performed for processing every attribute using existing techniques.

5 Involution Coder

Once histories have been generated, the available information is sufficient for the decoder to recover the whole connectivity. Histories are compressed to achieve the final binary version of the mesh.

5.1 History Generation

The first step consists of building histories describing the model connectivity. They are produced during a traversal of the generalized map darts by expressing their incidence relations.

5.1.1 Traversal

Let G_N be a N-dimensional generalized map composed of n darts. During the traversal, the visited darts waiting for processing are stored in $N+1$ FIFO queues noted L_i. The L_0 queue contains the darts waiting for the α_0 processing, L_1 contains the darts waiting for the α_1 processing, etc.

During the algorithm initialization, all queues are empty, and one dart is arbitrarily chosen as a traversal starting point and inserted in L_0. The algorithm performs as follows. While queues are not empty, a dart is chosen in the queue with lowest index, i.e. in L_0 first, in L_1 if L_0 is empty, etc. We perform a processing on the chosen dart consisting of encoding the involution into histories. Then, we remove it from L_i and add it to L_{i+1} (if $i < N$) and so on. As soon as a dart has gone through all the queues, it is discarded from the encoding process since its connectivity information has been fully described.

5.1.2 Histories

The processing of each dart consists of generating a symbol describing the connectivity to its incident neighbors. We associate a history noted H_i to each queue L_i, where we store the symbols describing the connectivity information for involution α_i. The information stored in the histories (H_i) allows the reconstruction of the original mesh connectivity along the decoding process.

During the traversal and darts processing, visited darts are marked with a $V(d)$ label. Darts are also assigned a number $N(d)$ according to the traversal order. Number 0 is given to the starting dart. Then, every dart found along an α_i involution is marked, assigned the next available index and added into queue L_0 for further processing. Note that, for a given dart, (α_i) involutions are processed in increasing index order.

Symbols stored in the H_i histories are determined in the following way. Let d' be the image of the current dart d by the involution α_i ($d' = \alpha_i(d)$). Four incidence cases between d and d' can occur (cf. Figure 3):

- if $d = d'$, d is a border dart and thus the symbol `B` (border) is emitted;
- if d' has not been previously visited (i.e. the Boolean value $V(d')$ is *false*), the symbol `N` is emitted, and we add this new dart into queue L_0 after it has been numbered and marked;
- if d' has already been visited, the symbol `W` (waiting) is emitted with an associated index $N(d')$ allowing to identify the corresponding dart;
- if the connectivity information is redundant compared to those of other darts (see below), no symbol is emitted (noted '`-`' in the examples)

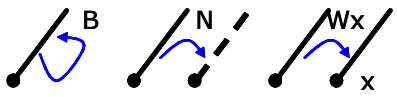


Fig. 3 *Three incidence cases resulting in three distinct symbols: `B`, `N`, `W` correspond to border information, new dart and visited dart respectively.*

The numbering provides indices associated with `W` symbols. The global history $H = \{H_0, H_1, ..., H_N\}$ is incrementally built during the traversal by processing each dart.

Figure 4 shows an example for encoding a 1-dimensional mesh (polyline) with two involutions α_0 and α_1. The left part of each line represents: (i) the chosen dart number, (ii) the current involution index, (iii) the resulting symbol; queues contents are on the right.

An arbitrary dart is chosen as a starting point for each connected component. This dart is marked, assigned the number 0 and added into queue L_0. Then, we repeatedly choose the next dart in the queue with lowest index and we encode it. First, dart 0 is chosen in L_0. Since its image by α_0 has not been previously encountered, we store a `N` symbol in the H_0 history. This dart is next added to queue L_1, waiting for the α_1 encoding. The dart found by α_0 is marked, numbered 1 and added to queue L_0. This dart is chosen for the second iteration, but the α_0 connectivity information is redundant. Therefore, no symbol is stored and the dart is moved to L_1 as well. Next, dart 0 is chosen in L_1; the α_1 processing produces a symbol `N` stored in H_1 and discovers a new dart numbered 2, and so on. Later on during the traversal, dart 3 is chosen in L_1 (line 10 in Figure 4). Since its image (dart 5) has been previously visited, the symbol `W5` is written in the H_1 history. At the end of the encoding process, two histories H_0 and H_1 have been generated.

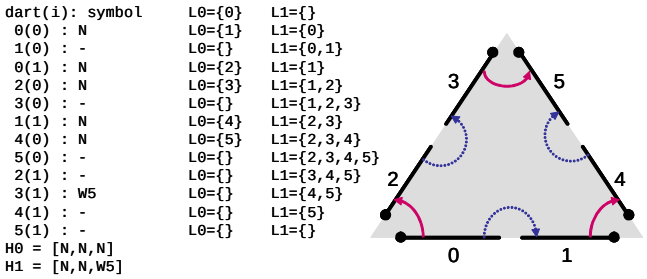


Fig. 4 *A simple 1-dimensional example: the list (left) correspond to the histories produced during the traversal of the polyline (right).*

5.1.3 Avoiding Redundancies

On Figure 4, fake symbols noted '`-`' figure the presence of redundancies in the G-map structure. These redundancies are a consequence of the first property of generalized maps (i.e. $\alpha_i^2(d) = d$) as depicted in Figure 5(a). In dimension higher than one, the second property of the G-map brings another type of redundancy, as depicted in Figure 5(b) and 5(c): border or sewing information needs not be duplicated for a whole orbit. In order to detect these redundancies, another Boolean label $E_i(d)$ is used to indicate whether a symbol must be stored (value *true*) or not (value *false*) into the H_i history for a given dart d. Initially, all $E_i(d)$ labels are set.

Labels are incrementally updated during the encoding and decoding stages of the subdivision. We define the *origin* orbit $O_O = \langle \alpha_0, ..., \alpha_{i-2} \rangle(d)$ and the *destination* orbit $O_D = \langle \alpha_0, ..., \alpha_{i-2} \rangle(d')$ when $d' \neq d$, $i \geq 2$, and $O_O = O_D = \emptyset$ otherwise. For example in Figure 5(b) or 5(c), the darts $\{0, 1\}$ figure the origin orbit and $\{2, 3\}$ the destination orbit while processing dart 0 for α_2

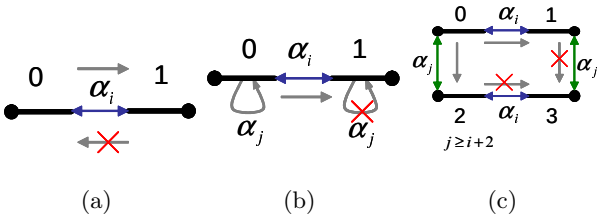


Fig. 5 *Three types of redundancies: stroked red-arrows correspond to the information removed to avoid redundancies.*

(that is $i = 2$). Both orbits are homeomorphic according to the second property of generalized maps. Therefore, describing the connectivity of the destination orbit is not necessary. Labels are updated in the following manner:

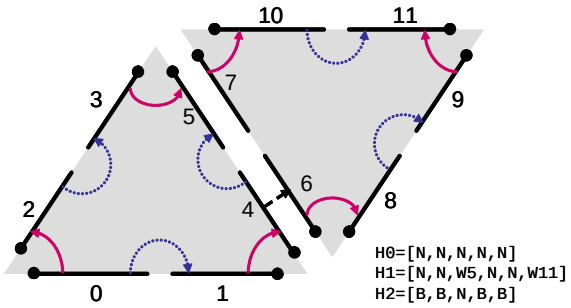
- `N`: the destination orbit has not been previously visited while having the same connectivity as the origin orbit. We thus unset all E_k labels with $k = \{0, ..., i-2, i\}$ for the darts in the destination orbit and the E_i label for the darts in the origin orbit. Destination darts are also marked, numbered in the traversal order and added into queue L_{i-1};
- `B`: since the origin orbit lies on the boundary of the mesh, we unset the E_i labels for the origin darts in order to avoid duplicating the boundary information;
- `W`: the destination orbit has already been traversed; due to the traversal order, all darts in lower dimensions have been visited, so all E_k with $k < i$ should have already been unset. Therefore the E_i labels of the origin and destination orbits are unset.

Figure 6 illustrates the encoding of a 2-dimensional mesh. The E_k unset during each encoding (or decoding) step appear after the symbol. Note the redundancies for the α_2 involution, especially for darts 4, 5, 6 and 7: E_0 and E_2 are unset for all darts of the orbit $\langle \alpha_0, \alpha_2 \rangle(4)$, as depicted in Figure 5(c).

5.2 History Decoding

The decoding process is symmetric to the encoding process: the decoding algorithm performs the same traversal of the mesh and creates darts on-the-fly using the symbols from histories. As in the encoding process, the E_i labels determine whether a symbol should be read in the H_i history. Depending on the symbol, an adapted process is performed among the followings:

- `B`: the E_i labels are unset;
- `N`: the origin orbit is duplicated to create the destination orbit, with the same connectivity on $\alpha_0, .., \alpha_{i-2}$ involutions. The origin and destination orbits are then sewn together by α_i. The darts of the destination orbit are marked as well, numbered and added in queue L_{i-1}. (E_k) labels of the created darts are initially set; then the E_k labels with $k = \{0, .., i-2, i\}$




```
dart(i):symbol {Ei/Ek unset for darts}
0(0):N   {1}          5(1):-   {}          10(0):N   {11}
* E0 only            0(2):B   {1}         11(0):-   {}
1(0):-   {}           * E2 only            8(1) :-   {}
0(1):N   {2}          1(2):-   {}          9(1) :W11 {11}
2(0):N   {3}          2(2):B   {3}         10(1):-   {}
3(0):-   {}           3(2):-   {}          11(1):-   {}
1(1):N   {4}          4(2):N   {5,6,7}     5(2) :-   {}
* E1 only            * E2+Ek, k=0         6(2) :-   {}
4(0):N   {5}          6(0):-   {}          7(2) :-   {}
* E0 only            7(0):-   {}          8(2) :B   {9}
5(0):-   {}           6(1):N   {8}         9(2) :-   {}
2(1):-   {}           8(0):N   {9}         10(2):B   {11}
3(1):W5  {5}          9(0):-   {}          11(2):-   {}
4(1):-   {}           7(1):N   {10}
```

Fig. 6 *A 2-dimensional example: new redundancies appear at the connection between the two triangles.*

are unset for the destination orbit as well as the labels E_i for the origin orbit;
- `W`: the E_i labels of the origin and destination orbits are unset and the orbits are sewn by α_i.

As an example, the decoding of histories in Figure 6 proceeds as follows. Initially, a first dart numbered 0 is created, marked and added to L_0 with all E_k set. This dart is chosen for the first iteration and a symbol `N` is read for the α_0 involution. Therefore, a dart 1 is created and linked to dart 0 by α_0. Except E_0, all its E_k labels are set. Dart 1 is then added to queue L_0. Next, no information is read during the processing of dart 1 picked from L_0 since E_0 is unset for it. This dart is also moved into L_1 for further processing. The traversal goes on the same way and in the same order as in the encoding process. The case of dart 4 illustrates the duplication of an origin orbit, when picked from L_2; a symbol N is read and thus the origin orbit $\{4, 5\}$ is copied to build the destination orbit, so that two new darts 6 and 7 are created and linked by α_0. Then, the two orbits are sewn together by α_2. The E_k labels are unset, since no information is necessary for `6(0)`, `7(0)`, `5(2)`, `6(2)` and `7(2)` cases.

5.3 History Compression

The compression mechanism is performed in two steps: firstly, indices associated with `W` symbols are predicted; secondly symbols and indices are compressed.

5.3.1 Index Prediction

As explained in the above Sections, an index is associated with every `W` symbol so as to identify the corresponding dart. Empirically, we observed that the number of the `W` symbols is not negligible compared to the number of `N` or `B` symbols. `W` symbols represent up to fifty percent of data in histories. Moreover, the indices are hard to compress and thus difficult to predict because their range extends as the mesh grows.

We thus propose a prediction scheme for the indices based on the visited neighborhood. The idea is to perform a traversal of the $\langle \alpha_i, \alpha_{i-1} \rangle$ orbit in order to find the first dart whose α_i involution has not been processed. This dart belongs to queue L_i thanks to the traversal order of the whole mesh. More precisely, starting from the initial dart, α_{i-1} and α_i involutions are alternately traversed until a dart with a E_i label set to one is found. It corresponds to a dart with unprocessed α_i involution.

The prediction concept is equivalent to a traversal of the edges incident to a face for α_1, to a traversal of the face corners incident to a vertex for α_2 and to a traversal of the volume corners incident to an half-edge for α_3, etc. The average cost for this traversal is then the average degree of faces, vertex valence and edge valence respectively.

The index associated with a `W` symbol is then the difference (delta encoding) between the number of the real dart and the number of the predicted dart. On the example given in Figure 7, the prediction traversal is applied for involutions α_1 and α_2. They both result in a zero index, which is the best possible prediction.

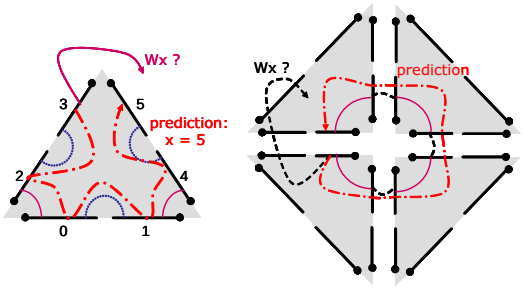


Fig. 7 *Prediction: the best-possible index predictions are shown for α_1 and α_2 involutions.*

5.3.2 Compressed Representation of Symbols

We choose to compress histories with a dictionary-based method such as the LZW [9] algorithm. The principle is to construct a dictionary of words (patterns) from the given symbols. Frequently encountered patterns are given a short code. The choice of this type of compression technique is motivated by the long and repetitive

sequences recorded in practice during the mesh connectivity encoding. On the other hand, since frequencies of `N` and `W0` symbols are similar, the use of an entropy coder would lead to low compression rates.

Before compression, we assign a variable-length byte code to symbols depending on the indices associated to the `W` symbol. In most cases, one byte is sufficient: its two less significant bits designate the symbol and the last six bits give the associated index. `N` and `B` symbols are given the codes `00` and `01`. If the six remaining bits are sufficient to represent the whole index, the `W` symbol is assigned the code `10`. Otherwise, `W` is assigned the code `11`, indicating that additional bytes will be used for the remaining part of the index. For each of these bytes, the most significant bit indicates if one more byte is needed for the index or not. Since indices can be negative, the first bit of the value is used as sign bit and the remaining bits for the absolute value of the indices.

Once this coding is performed, the data is compressed using the Zlib library[1] which is a variant of the LZW method used in Gzip. We thus obtain a binary file.

5.4 Geometry Encoding and Decoding

Attributes associated with the mesh may be vertex coordinates, colors, textures, edge curvatures, etc. The encoder stores values for each type of attribute, read later on by the decoder to provide a shape for the decoded mesh. Encoding and decoding stages should be synchronized for processing geometry: geometrical attributes must be processed only once, in the same order for encoding as for decoding.

In our technique, the whole connectivity is encoded/decoded prior to geometry compression, allowing any kind of attribute to be used, and not only per-vertex ones. We use a distinct traversal for each kind of attribute, starting from the initial dart (dart 0) of the connectivity compression. For example, we perform a vertex traversal to describe their locations, a face traversal for textures, a corner traversal for texture coordinates, etc. Any state-of-the-art predictive scheme (e.g. [1] or [5]) could be used to guess vertex positions from previously encoded ones, and the residuals are compressed as usual.

We use a usual orbit traversal to perform attributes ordering. As for connectivity traversal, we use a new queue L to recall the darts that are waiting to be processed and a label V (visited) avoiding the multiple processing of a single attribute. Initially, dart 0 is added to L and darts are repeatedly chosen in (and removed from) L. The attribute orbit is marked with V and the attribute value is encoded (quantization, prediction and so on). Finally, neighboring darts of the orbit are added to L: it consists of all darts reachable by any α_k not present in the orbit definition. For per-vertex attributes, the neighborhood is given by the α_0 involution since the vertex orbit is defined by $\langle \alpha_1, ..., \alpha_N \rangle(d)$. Since the attribute encoding is not the main topic of our technique, we will not provide results for geometry coding.

[1] http://www.gzip.org/zlib/

6 Results and Discussion

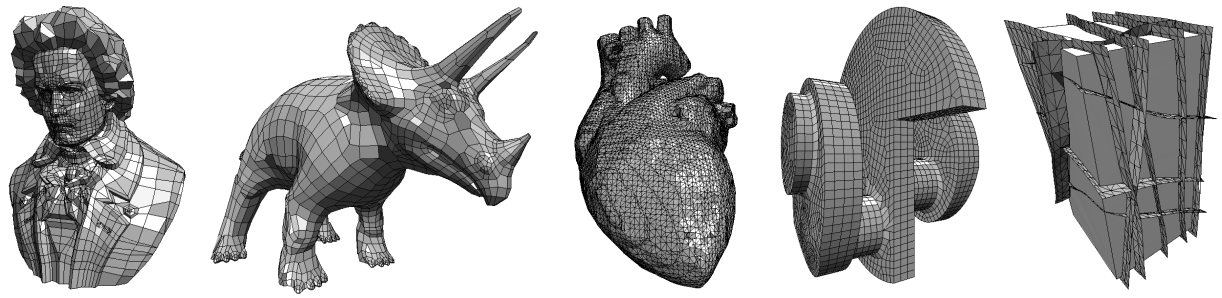
This Section presents some results produced by our method regarding two main aspects: compression rates and index predictions quality. Various models have been used: a common VRML corpus (surface meshes) used to compare several connectivity compression schemes, hexahedral and tetrahedral meshes, regular meshes composed of $N*N$ quads stitched along edges, regular meshes composed of $N*N*N$ cubes stitched along faces, non-orientable meshes (Moëbius band, Klein bottle) and general polyhedral meshes obtained by volumic CSG operations (co-refinement).

Tables 2, 4, 3 and Figure 9 present results about prediction quality and compression rates. `D`, `V`, `F`, `T`, `H` and `P` correspond respectively to the number of darts, vertices and faces, tetrahedra, hexahedra and polyhedra. `topo` is the size of compressed connectivity. The number of bits per dart, per vertex, per edge, per tetrahedra or per hexahedra are denoted by `BpD`, `BpV`, `BpE`, `BpT` and `BpH` respectively. When possible, we use a compression ratio to compare our method against existing schemes: [4] for surface meshes, [3] for tetrahedral meshes and [6] for hexahedral meshes. We also provide the proportion of bad prediction (`BP`) w.r.t. the number of `W` symbols, considering a bad prediction as a non zero predicted index. Encoding `time` is given in milliseconds (including geometry) and was obtained with an AMD Athlon XP 2200+ with 1 GB of RAM. It does not include the time used to construct the generalized map structure. We also used the default parameter for the compression rate given by the Zlib library. In all cases, the topological properties (number of borders, genus, orientability, number of cells) of the decoded meshes remain exactly the same as the original one.

Note that no bad prediction have been observed for regular meshes (quads and cubes meshes) and less than five percents of bad predictions otherwise in average. Prediction errors are due to borders and local topological irregularities. This is quite similar to the "splits" cases of the valence-based compression schemes.

As for compression rates, our method is less efficient than the best state-of-the-art techniques: compression factor of 2 to 3 have been observed for surface meshes, a factor of 1 to 3 for tetrahedral meshes and a factor of 2 to 8 for hexahedral meshes. However, regarding the genericity brought to connectivity compression, our method remains efficient even for processing specific meshes for which the existing methods have been especially designed.

The main problem of our method lies in the dictionary-based coding which is only efficient for long



(a) beethoven (b) triceratops (c) heart (d) shaft (e) geo_result

Fig. 8 *Various models used for the results. (a) and (b) are surface meshes, (c) is a tetrahedral mesh, (d) is a hexahedral mesh and (e) is a general polyhedral mesh.*

name	D	V	F	topo	time	BpD	BpV	Ise02 (BpV)	Invo/ Ise02	BP
teapot	9916	1189	1290	742	110	0.599	4.992	1.13	4.42	1.0%
galleon	18932	2373	2384	1377	187	0.582	4.642	2.09	2.22	4.4%
shark	20480	2560	2562	1490	219	0.582	4.656	0.76	6.13	1.4%
beethoven	21308	2655	2812	1903	219	0.714	5.734	2.10	2.73	1.9%
sandal	21716	2636	2953	1626	204	0.599	4.935	2.08	2.37	2.5%
triceratops	22656	2832	2834	1602	250	0.566	4.525	1.19	3.80	1.9%
cupie	24016	2984	3032	1857	234	0.619	4.979	1.64	3.04	3.0%
cow_poly	24660	2904	3263	1565	265	0.508	4.311	1.53	2.82	1.8%
cessna	30600	3745	3927	2953	328	0.772	6.308	2.55	2.47	4.3%
al	31004	3618	4175	2810	281	0.725	6.213	2.42	2.57	3.5%
tommygun	32340	4171	3980	2516	281	0.622	4.826	2.32	2.08	8.7%
cow	34824	2904	5804	1813	391	0.416	4.994	1.78	2.81	1.2%
moebius	96	24	12	73	16	6.083	24.333	3.54	6.87	7.7%
klein	7200	900	900	275	78	0.306	2.444	0.68	3.59	0.7%

Table 2 *Compression results for the surface models.*

name	D	V	T	topo	time	BpD	BpV	BpT	GGS99 (BpV)	Invo/ GGS99	BP
random	301728	2000	12572	11213	3250	0.297	44.852	7.135	15.12	2.97	2.4%
proto	310464	2896	12936	9303	3265	0.240	25.699	5.753	9.55	2.69	1.0%
blunt	4497480	40960	187395	30168	53563	0.054	5.892	1.288	6.00	0.98	0.0%
heart	5950176	50906	247924	168705	86125	0.227	26.512	5.444			0.9%

Table 3 *Compression results for the tetrahedral models.*

name	D	V	H	topo	time	BpD	BpV	BpH	IA03 (BpH)	Invo/ IA03	BP
bump2	57072	1665	1189	1043	609	0.146	5.011	7.018	2.1	3.34	0.08%
fru	209280	5124	4360	2206	2344	0.084	3.444	4.048	0.98	4.13	0.00%
hanger	8208	398	171	311	78	0.303	6.251	14.550	5.3	2.75	0.55%
hutch	392256	8790	8172	2740	4406	0.056	2.494	2.682	0.31	8.65	0.01%
mdg-1b	178080	4510	3710	2160	1922	0.097	3.831	4.658	0.77	6.05	0.02%
shaft	330384	9218	6883	5147	3672	0.125	4.467	5.982	1.7	3.52	0.05%
test	114528	3198	2386	1196	1187	0.084	2.992	4.010	0.87	4.61	0.01%
warped	384000	9261	8000	1282	4218	0.027	1.107	1.282	0.18	7.12	0.00%

Table 4 *Compression results for the hexahedral models.*

name	D	V	E	P	topo	time	BpD	BpV	BpE	BP
geo_result	107324	4746	11210	40	18947	9328	1.412	31.938	13.521	11.4%
geo_cut	41636	2314	4834	30	8318	3625	1.598	28.757	13.766	14.9%

Table 5 *Compression results for the polyhedral models.*

symbol sequences. Further possible improvements could be:

- initializing a dictionary before the encoding process using repetitive patterns of `N`, `WO` and `B`;
- using a statistical model to predict the symbols;
- influencing the traversal order with heuristics;

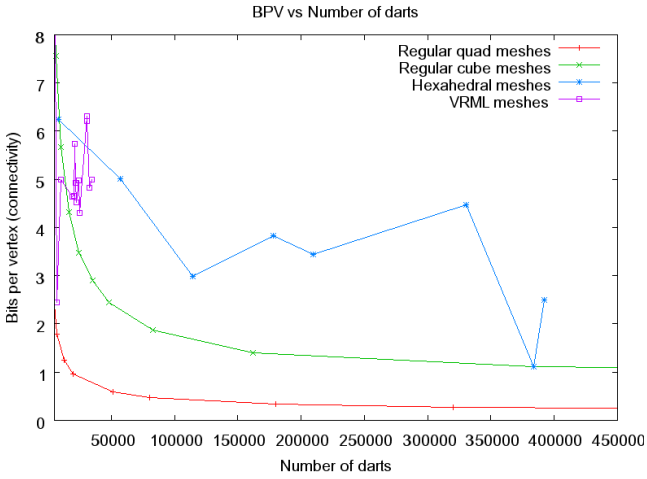


Fig. 9 *Graph of the BPV against the number of darts: regular meshes are easier to compress than the general surface or volume meshes.*

- using arithmetic coding with an adequate transform of the symbols (occurence counts for example).

Another problem is the random distribution of indices in bad prediction cases which implies low compression rates for indices, especially for volume meshes. The reason for this is that the prediction is performed from the current dart whereas it could be performed from any dart of the face for α_3 or of the edge for α_2. Simple tests have shown that predictions could be improved but there is no straightforward way to detect the best candidate from which to perform the prediction. Improving this prediction (or traversal order) would be beneficial.

7 Conclusions

In this paper, we have proposed a new lossless connectivity compression scheme designed for meshes with arbi-

trary topology and cells. This method relies on generalized maps for building histories during a region-growing traversal of the darts. Histories are compressed using a dictionary-based coder to form the final binary representation of the model's connectivity. Geometrical attributes attached to the model are treated independently using common techniques such as quantization, prediction and data compression.

The main advantage of our method concerns its genericity. Thanks to this technique, the compression of irregular volume models becomes possible, this is particularly interesting for geology applications or for the (volumic) representation of urban environments with ground.

In the future, we aim at improving geometry coding using vector quantization. We also plan to use this work for an urban walkthrough application: scenes are streamed over networks while visibility computations are performed during navigation using the scene topology. In this context, our technique allows to reduce both bandwidth and latency needed for geometrical data.

Acknowledgements We would like to thank Alla Sheffer for the bump2, fru and shaft models, Steven Owen for mdg-1, Scott Mitchell for hanger and hutch, the NASA for blunt, Bruno Notrosso for proto (spx), Yongjie Zhang and Chandrajit Bajaj for the heart model [11].

References

1. Alliez, P., Gotsman, C.: Recent advances in compression of 3D meshes. In: Proceedings of the Symposium on Multiresolution in Geometric Modeling. Cambridge (2003)
2. Brisson, E.: Representing geometric structures in d dimensions: topology and order. In: Proceedings of the fifth annual symposium on Computational geometry, pp. 218–227. ACM Press, Saarbrücken, Germany (1989)
3. Gumhold, S., Guthe, S., Strasser, W.: Tetrahedral mesh compression with the cut-border machine. In: Proceedings of the conference on Visualization '99, pp. 51–58. IEEE Computer Society Press, San Francisco, California, United States (1999)
4. Isenburg, M.: Compressing polygon mesh connectivity with degree duality prediction. In: Graphics Interface '02 Conference Proceedings, pp. 161–170. Calgary, Alberta (2002)
5. Isenburg, M., Alliez, P.: Compressing polygon mesh geometry with parallelogram prediction. In: Proceedings of the conference on Visualization '02, pp. 141–146. IEEE Computer Society, Boston, Massachusetts (2002)
6. Isenburg, M., Alliez, P.: Compressing hexahedral volume meshes. In: Graphical Models, vol. 65, pp. 239–257. Academic Press Professional, Inc. (2003)
7. Lienhardt, P.: N-dimensional generalized combinatorial maps and cellular quasi-manifolds. International Journal of Computational Geometry and Applications **4**(3), 275–324 (1994)
8. Szymczak, A., Rossignac, J.: Grow and fold: compression of tetrahedral meshes. In: Proceedings of the fifth ACM symposium on Solid modeling and applications, pp. 54–64. Ann Arbor, Michigan, United States (1999)
9. Welch, T.: A technique for high-performance data compression. IEEE Computer pp. 8–19 (1984)
10. Yang, C.K., Mitra, T., Chiueh, T.c.: On-the-fly rendering of losslessly compressed irregular volume data. In: IEEE Visualization 2000, pp. 101–108 (2000)
11. Zhang, Y., Bajaj, C.: Finite element meshing for cardiac analysis. ICES Technical Report 04-26, University of Texas (2004)