Volume xx (200y), Number z, pp. 1–10

GEncode: Geometry–driven compression for general meshes

Thomas Lewiner[1,2], Marcos Craizer[1], Hélio Lopes[1], Sinésio Pesco[1], Luiz Velho[3] and Esdras Medeiros[3]

[1] PUC–Rio — Departamento de Matemática — Matmídia Project — Rio de Janeiro — Brazil
[2] INRIA — Géométrica Project — Sophia Antipolis — France
[3] IMPA — Visgraf Project — Rio de Janeiro — Brazil

Abstract

Performances of actual mesh compression algorithms vary significantly depending on the type of model it encodes. These methods rely on prior assumptions on the mesh to be efficient, such as regular connectivity, simple topology and similarity between its elements. However, these priors are implicit in usual schemes, harming their suitability for specific models. In particular, connectivity–driven schemes are difficult to generalise to higher dimensions and to handle topological singularities. GEncode is a new single–rate, geometry–driven compression scheme where prior knowledge of the mesh is plugged into the coder in an explicit manner. It encodes meshes of arbitrary dimension without topological restrictions, but can incorporate topological properties, such as manifoldness, to improve the compression ratio. Prior knowledge of the geometry is taken as an input of the algorithm, represented by a function of the local geometry. This suits particularly well for scanned and remeshed models, where exact geometric priors are available. Compression results surfaces and volumes are competitive with existing schemes.

Keywords: Mesh Compression, Geometry–driven techniques, Arbitrary Meshes, Arbitrary Dimension.

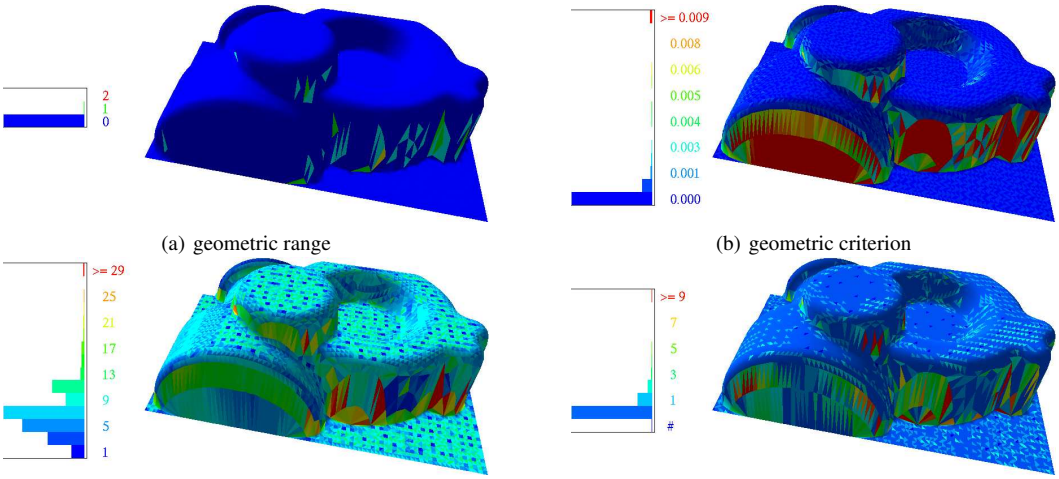
Categories and Subject Descriptors (according to ACM CCS): E.4 [Coding and Information Theory]: Data compaction and compression I.3.5 [Computer Graphics]: Curve, surface, solid, and object representations

1. Introduction

Computer Graphics developments handle each time bigger meshes, using processes of increasing complexity. Compression algorithms followed these developments by improving the compression ratio, enlarging the range of models that can be encoded, simplifying their implementation and increasing execution performances. However, they are still not fully adapted to the wide variety of models and applications of Computer Graphics: scans in artistic and archaeological modelling, isosurfaces for medical and mathematical visualisation, re–meshed models for reverse engineering, finite–element meshes for simulation, high–dimensional meshes for solid representation, meshes with high co–dimension for non–linear optimisation, among others. Actually, the performances of state–of–the–art compression algorithms highly depend on the nature of the model. We will focus here on compression schemes adapted to the priors of specific applications, in particular for the most time–consuming mesh generation algorithms: reconstruction and the re-meshing.

Geometry–driven compression. Meshes are usually described by their geometry (the coordinates of its vertices) and their connectivity (the combinatorial elements that interpolate these vertices, usually triangles or simplices). This composite nature leads to classify compression algorithms between: on one hand connectivity–driven ones, when the connectivity is coded separately and the geometry is partially deduced from it, and on the other side geometry–driven ones, when the geometry is coded separately and the connectivity is coded using the geometry. The efficiency of connectivity–driven algorithms usually relies on a regular connectivity, whereas the newer trend of geometry–driven methods are expected to perform better on geometrical meshes, such as reconstructed or re-meshed models. This work proposes a new geometry–driven method, which encodes arbitrary meshes in arbitrary dimension, and compares nicely to connectivity–driven methods for surfaces (Fig.1) and for volumes (Fig.10).

Related works. The first mesh-compression algorithms were connectivity–driven, in the sense that the geometry encoding depends on the connectivity encoding rules. Among those, the Edgebreaker [TR98, Ros99, LRS*02, LLRV04] performs well on generic meshes, with guaranteed *practi-*



(a) geometric range (b) geometric criterion

(c) list size (d) apex identifier

Figure 1: *GEncode compression: once the geometry is decoded, the decoder attaches triangles to edges of the front by identifying its apex w: A list of candidates is computed from an encoded geometric range (a) and ordered according to a geometric criterion (b), here the distance from w to the edge midpoint. Then w is identified in this ordered list (c) by its position (d).*

cal worst–case close to the theoretical optimum [KR99]. On the other side, Valence Coding [TG98, CD04, KPRW05] has a *theoretical asymptotic compression ratio* close to the optimum [AD01]. It has been widely extended since the original work, and performs very well in practise, especially on meshes with a regular connectivity. Some singularities of the mesh can further be handled by specific algorithms, in particular for the non–manifold case [GBTS99, RC99].

These connectivity–driven approaches can be extended to higher dimensions, but the complexity of the codes increases dramatically. Even for tetrahedral meshes, the extensions of the surface approaches [GGS99, SR00, IA02] are delicate.

As an intermediate towards geometry–driven approaches, some connectivity–driven schemes use the previously coded geometry to predict the connectivity [KG01, LAD02, CR04, KPRW05]. Each of these algorithms uses a different prior on the geometric regularity of the mesh.

On the contrary, geometry–driven approaches introduced by [GD02] handle gracefully complex connectivity. Still, the compression ratios of the geometry are not yet optimal, since these schemes are quite new to the community. However, for the case of isosurfaces, specific compression schemes [Tau02, LDS03, LVLM04] outperform any connectivity–driven approach.

Contributions. This work proposes a new geometry–driven scheme called GEncode, which works for meshes of arbitrary topology and dimension embedded in spaces of arbitrary dimension. To our knowledge, GEncode is the first compression method that works at that level of generality and still compares nicely to state–of–the–art compression methods for triangulated surfaces and volumes. As opposed to [GD02], GEncode is single rate, but copes with general meshes, and shows better compression ratios: For surfaces, the resulting compression ratios are competitive with the Edgebreaker with the parallelogram prediction, and for volumes it is highly competitive with Grow&Fold [SR00] and with streaming compression [ILGS].

Aside from its generality, GEncode treats the priors of the mesh as an input, and can therefore easily adapt to specific classes of meshes. These priors include on one side global topological properties such as manifoldness, the presence of boundary and eventually the degree of the facets, and on the other side local geometrical properties represented by a scalar function of the vertices of a facet. For example, a common prior for usual Computer Graphics models assumes that the mesh is a triangulated manifold without boundary, and that the triangles maximise their circumradius or their aspect ratio. In particular, if the mesh can be reconstructed from its vertices with a geometric prior, the GEncode connectivity encoding with that prior leads to a *zero entropy* code.

This work is an extended version of [LCL*05] and part of Thomas Lewiner's Ph.D. [Lew05]. It describes GEncode at its high level of generality, investigates different geometric priors and separates the geometric range definition from the geometric prior in order to reduce the constraints on the geometric function defining the prior. Moreover, this version includes tests on tetrahedral meshes, where GEncode turned out to be particularly competitive.

Overview. This work is organised as follow. Sec. 2 recalls the basic notions of meshes, expressed in arbitrary dimension. Then Sec. 3 introduces the two methods we considered for compressing the geometry, and how we synthesised them. The main part of GEncode is introduced at Sec. 4, followed in Sec. 5 by a discussion on priors that can be

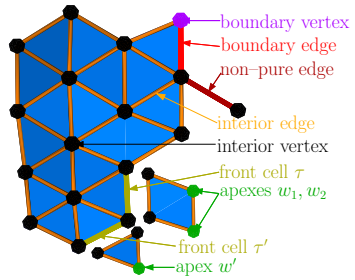


Figure 2: *Cell complex elements topology and cell attachment operation.*

plugged into the algorithm to compress efficiently usual models. Finally, Sec. 6 provides some results and comparisons with state–of–the–art methods on common models, and compares them with the Edgebreaker for surfaces, and with Grow&Fold and streaming methods for volumes.

2. General Meshes

This section introduces the basic definitions that are used in this work, especially the notion of convex cell complexes [Hat02]. This notion is introduced formally, but corresponds to the usual meshes used in Computer Graphics, and the reader can think of this notion as a generalisation of triangulated surfaces. They can be constructed in an incremental manner by the single operation of *cell attachment*. This construction is usually referred as *advancing front*, and entails most of the mesh decoding algorithms.

Convex cells. A *convex cell* σ in $\mathbb{R}^p$ is a non-empty compact subset of $\mathbb{R}^n$ which is the solution set of a finite number of equations $f_i(\mathbf{x}) = 0$ and inequalities $g_i(\mathbf{x}) \geqslant 0$, where f_i and g_i are affine functions of the form $(x_1, x_2, ..., x_p) \mapsto \lambda_0 + \lambda_1 x_1 + \lambda_2 x_2 + ... + \lambda_p x_p$.

A cell σ^d has *dimension* d if it contains $d+1$ affine independent points but no more. A *subcell* τ of σ is a cell obtained by changing some of the inequalities $g_i(x) \geqslant 0$ to equalities. We will say that σ is *incident* to τ. The collection of all the subcells of σ^d of dimension $d-1$ is denoted $\partial\sigma$.

Points, line segments, triangles, quadrangles, tetrahedræ, cubes are examples of convex cells. Among these convex cells are the *simplices*, which generalise the notion of line segment, triangle and tetrahedron: A d–simplex is the convex hull of $(d+1)$ affine independent points in the space.

Convex cell complex. A *convex cell complex* K is a coherent collection of distinct convex cells, where coherence means that the collection contains the subcells of each cell and the intersection of any two cells. A convex cell complex K is *pure* of dimension n if every cell in K is of dimension n or is a subcell of a cell of dimension n belonging to K. A *facet* of a pure n–complex K is a cell of K of dimension n.

The *vertices* of a cell are its subcells of dimension 0. The *geometry* of a complex usually refers to the coordinates of its vertices, while its *connectivity* refers to the incidence of higher dimensional cells on these vertices. Observe that a cell is uniquely determined by its vertices.

Cell attachment. An n–cell σ can be *attached* to a complex K by identifying a collection of its subcells $\{\tau_1, \cdots, \tau_k\}$ with some of the cells of K, preserving its nature of cell complex.

If one of the cells τ of K is of dimension $n-1$, this cell attachment can be considered as the attachment of σ onto τ, and we will write $\sigma = \tau \star \{w_1, \ldots, w_m\}$, where the vertices $\{w_1, \ldots, w_m\}$ are those of σ not subcells of τ (Fig.2). These vertices are called the *apexes* of the cell attachment. If σ is a simplex, there is only one apex ($m = 1$).

Manifolds. Among these convex cell complexes, the class of combinatorial manifolds is the most widely used. A combinatorial n–manifold $\mathcal{M}$ is a pure complex of dimension n where for each vertex v, the union of each open simplex containing v is homeomorphic to the open n–ball $\mathbb{B}^n$ or the intersection of $\mathbb{B}^n$ with a closed half–space. This implies that each $(n-1)$–cell is a subcell of either one or two n–cells. The set of $(n-1)$–cells subcells of only one n–cell is called the *boundary* of $\mathcal{M}$ (Fig.2).

3. Independent Encoding of the Geometry

GEncode is a pure geometry–driven scheme, and the coordinates of the vertices of the mesh are thus encoded separately before the connectivity compression. We considered two geometry coding techniques, described in [GD02] and in [BWK02], and propose a synthesis of them. This synthesis has similar compression ratios as both [GD02] and [BWK02], but emphasises their strong points and could be the basis for further improvements on this part of the coding. In our experiments, the proposed synthesis is generally more efficient on small models (below 3000 vertices) or on the three–dimensional meshes we tested (Fig.10), whereas [GD02] gets the best results for larger models.

Space partition encoding. The coordinates of all the vertices are encoded globally as a space partition tree. This kind of techniques works for vertices with an arbitrary number of coordinates, allowing encoding meshes of arbitrary co–dimension. In particular in [GD02] and [BWK02], the space is divided with a particular binary space partition where each separator is perpendicular to the axis, as an octree for dimension 3: the axis alternates from one level to the next one (X,Y,Z,X,Y... in $\mathbb{R}^3$), and each part is subdivided in two equal sub-parts, as on Figs. 3, 4 and 5. The subdivision is performed until each part contains only one vertex. We will now compare and synthesise these techniques.

Lower nodes efficient. In [GD02], each node of the space partition is encoded by the number of vertices $\#v_l$ its left

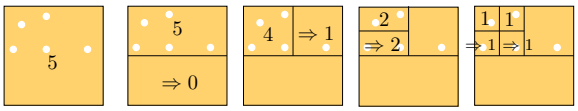


Figure 3: *The geometry encoder of [GD02] codes first* 5 *as a standard* 32 *bits number, then* 5 *on* $\lceil \log_2(6) \rceil$ *bits, and then* 4 *on* $\lceil \log_2(6) \rceil$ *bits. The right vertex relative position is then coded. Then* 2 *is coded on* $\lceil \log_2(5) \rceil$ *bits, and both* 1 *on* $\lceil \log_2(3) \rceil$ *bits. The position of the last vertices is coded.*

children contains. The number of vertices of the other node $\#v_r$ is simply deduced by difference from the number of the vertices of the parent's node $\#v_f$ which is known from the recursion. This technique wastes many bits at the beginning of the encoding: the number of nodes must be encoded on $\lceil log_2(\#v_f + 1) \rceil$ bits, since the number of node of the left child can be $\{0, \ldots, \#v_f\}$ (except the first node which is on 32 bits by convention), as for the example of Fig.3. At the end, when there is only one vertex per node, it is encoded with 1 bit per level, which is optimal.

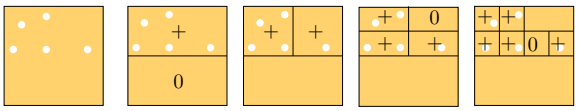


Figure 4: *The geometry encoder of [BWK02] codes the following sequence:* $+0$, $++$, $++$, $0+$, $++$, $++$, $0+$. *Then follow* $0+$ *and* $+0$ *to reach the desired number of bits.*

Higher nodes efficient. In [BWK02], each node is encoded by one out of 3 symbols: $++$ if both children contain at least one vertex, $+0$ if only the left child contains a vertex, and $0+$ if only the right one contains a vertex, as for the example of Fig.4. Note that at least one child must contain a vertex, since the parent did. The encoding stops at a predefined level. This method spends more bits at the end of the encoding, since the decoder does not know when there is only one vertex in a node. Therefore, the encoder sends $log_2(3)$ bit for each level, which is greater than the 1 bit of [GD02] for the last part.

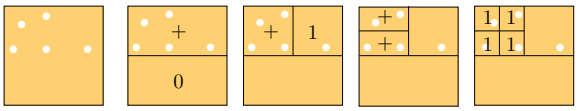


Figure 5: *Synthesis: the encoder codes the following sequence:* $+0$, $+1$ *and the right vertex relative position is then coded, and then* $++$, 11, 11. *The position of the remaining vertices is then coded.*

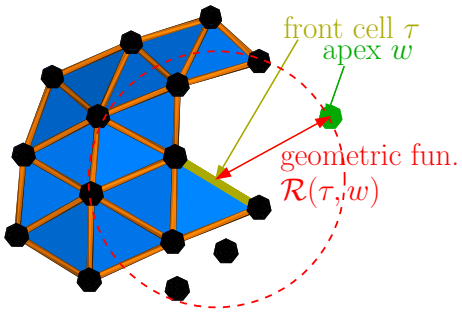
Synthesis proposal. The goal of this synthesis is to take the best part of both. First, it encodes each node by one of 6 symbols: $++$ if both children contains more than one vertex, $+1$ and $1+$ if one child contains more than one vertex, and the other only one, 11 if they both contain only one vertex, and $+0$ and $0+$ if one child contains more than one vertex, and the other child is empty, as for the example of Fig.5. With this encoding, the encoder detects when there is only one vertex in a node, and then uses the technique of [GD02]. The coder further benefits from the different probability of the symbols. Moreover, these probabilities are used differently depending on the level of the node to encode: nodes closer to the root are more frequently of type $++$, whereas these are rare when going closer to the leaves.

4. Connectivity Encoding: Geometric Range and Apex Identifiers

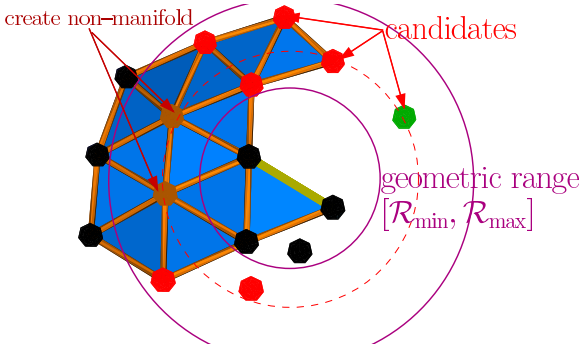
The main contribution of GEncode is its connectivity coding (Algs. 1 and 2). As a generic geometry–driven compression scheme, the geometry is known before the connectivity is decoded, either because it is already available as a point cloud, or because it has been decoded by a method like the one we presented above. The connectivity decoding then works similarly to greedy advancing front reconstruction algorithms such as [BMR*99, MVL04, SFS05], but the best match used by the reconstruction algorithm is continuously corrected by the encoded stream. Therefore, for meshes that can be reconstructed with a greedy strategy, GEncode can achieve a zero entropy message for the connectivity.

Coding Principle. The algorithm encodes an initial n–cell and then works as an advancing front triangulation, maintaining an ordered queue of $(n-1)$–cells and attaching at each step a cell $\tau \star \{w_1, \ldots, w_m\}$ to τ, the cell of the front with the highest priority, or removing τ from the front, for example when τ is on the boundary of the final mesh. When an n–cell is attached, its $(n-1)$–faces are added to the front. The compression of a connected component ends when the front is empty. The difference with greedy reconstruction algorithms is that vertices w_j are not always the ones that minimise a given *geometric criterion* $\mathcal{G}(\tau, w_j)$, such as the circumradius of $\tau \star w_j$ used by [BMR*99]. Such criteria will be discussed in Sec. 5. To identify the apex w_j, we actually encode its position in a list of *candidates*, generated by a *geometric range* $[\mathcal{R}_{\min}, \mathcal{R}_{\max}]$ and ordered by geometric criterion $\mathcal{G}$.

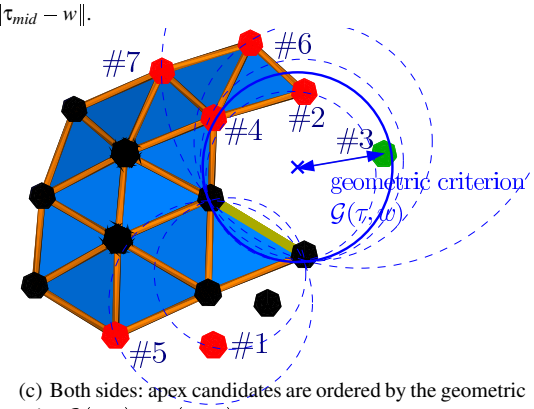
Geometric Range. The list of candidates for w could be the list of all vertices, but that would allow $O(\#_{verts})$ choices, which is too expensive to encode. In order to reduce the size of that list and the time used to compute it, a geometric function $\mathcal{R}(\tau, w)$ is transmitted (Fig.6(a)). However, the quantisation process encodes $\mathcal{R}(\tau, w)$ as one of the predefined ranges: $\mathcal{R}(\tau, w) \in [\mathcal{R}_{\min}, \mathcal{R}_{\max}]$. The list of candidates will then be the vertices v_i such that $\mathcal{R}(\tau, v_i)$ belongs to $[\mathcal{R}_{\min}, \mathcal{R}_{\max}]$ (Fig.6(b)).



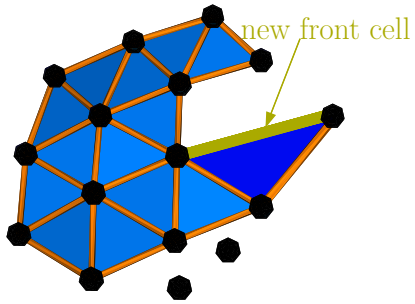
(a) Coder side: geometric function $\mathcal{R}(\tau, w) = \|\tau_{mid} - w\|$.



(b) Both sides: geometric range $\mathcal{R}_{\min} \leqslant \mathcal{R}(\tau, w) < \mathcal{R}_{\max}$.



(c) Both sides: apex candidates are ordered by the geometric prior $\mathcal{G}(\tau, v_i) = \rho(\tau \star v_i)$.



(d) Decoder side: the decoded apex identifier #3 defines the cell attachment $\tau \star w$.

Figure 6: *GEncode principle: the geometric function (a) is encoded. Its quantisation restricts the apex candidates, which are then filtered by the topological priors (b). The candidates are ordered (c) and the encoder transmits the candidate number in the list (here #3), which allows the decoder to perform the cell attachment (d) and continue on the next front cell.*

Definition of the geometric function $\mathcal{R}$ must be effective for identifying the candidate vertices. In particular, for simple geometric priors $\mathcal{G}$ on the mesh, we choose $\mathcal{R} = \mathcal{G}$. Moreover, the binary space partition of the geometric encoding is used to accelerate the localisation of the candidates.

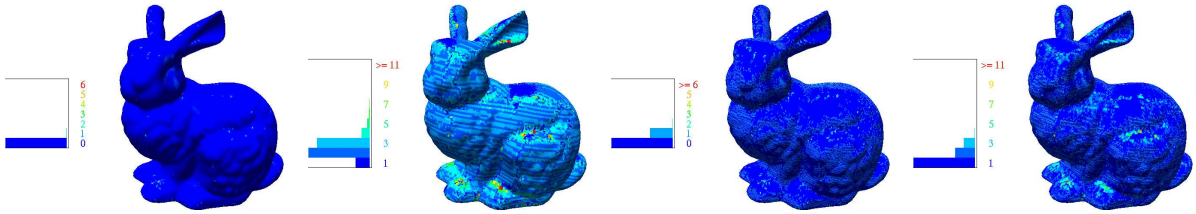
Apex Identifiers. Once the list of apex candidates $\{v_i\}$ is generated, it is ordered by the *geometric prior* $\mathcal{G}(\tau, v_i)$. For example on Fig.6(c), the best candidate according to $\mathcal{G}$ gets identifier 1, the second best match receives identifier 2. The original apex gets identifier 3, and this number is transmitted to the decoder, which can then perform the right cell attachment (Fig.6(d)). This ordering gives a higher probability to the lower identifiers and thus reduces the entropy.

Non–simplicial meshes. Not simplicial meshes require a third number to be encoded: the number m of apexes for the cell attachment $\tau \star \{w_1, \ldots, w_m\}$ (line 8 of Alg. 1). Moreover, there is one range per apex w_j, but only the smallest and the biggest ranges of the w_j are actually encoded. In that case, we know that the second and following apexes are on the affine hyperplane defined by $\tau \star w_1$. This reduces the candidate list, which improves the entropy of the apex identifiers.

Quantisation trade–off. This scheme actually encodes the apex by two means: the geometric range and the apex identifier. The quantisation of the geometric function is thus a trade-off: On the one hand, if the quantisation is rough, there will be many candidates, which requires more time to generate the list and an expensive encoding of the position of w_j in the list of the candidates, as shown on the histograms of Fig.7. On the other hand, if the quantisation is too refined, the quantised geometric range will be expensive to encode.

Topological Priors. GEncode compresses pure meshes of arbitrary dimension, embedded in any co–dimensional space, orientable or not, with any kind of topology. If the mesh contains multiple connected components, the algorithms are applied separately on each of them. Further information on the mesh will then improve the compression ratio: We already saw that if the mesh is simplicial, the encoding of the face degree (parameter m of line 8 of Alg. 1) can be avoided. Even if this is not known to the coder, the sequences of m will be constant, and thus have entropy zero.

Computer Graphics meshes are usually manifolds. In that case, GEncode is optimised in two ways: firstly, by removing



(a) range: roughly quantised (low entropy). (b) number of candidates (high entropy). (c) range: precisely quantised (high entropy). (d) number of candidates (low entropy).

Figure 7: *Quantisation trade–off: rough quantisation of $\mathcal{R}$ leads to lower entropy for the range, but more candidates, thus higher entropy for the apex identifier.*

from the list the candidates that would create a non–manifold object, i.e. the vertices that are not on the boundary of the reconstructed mesh (Fig.6(b)). Secondly, an $(n-1)$–cell will be processed at most once. Therefore, the $m=-1$ code of line 16 of Alg. 1 only serves as a marker for boundary cells, reducing its range to $0/-1$ and thus improving the entropy of that code. If the complex is a simplicial manifold without boundary, the encoding of m can be omitted for both the number of apexes and the boundary marker (lines 8 and 16 of Alg. 1).

Algorithm 1 gencode($\mathcal{R}$,$\mathcal{G}$): encodes one component of a pure n–complex.

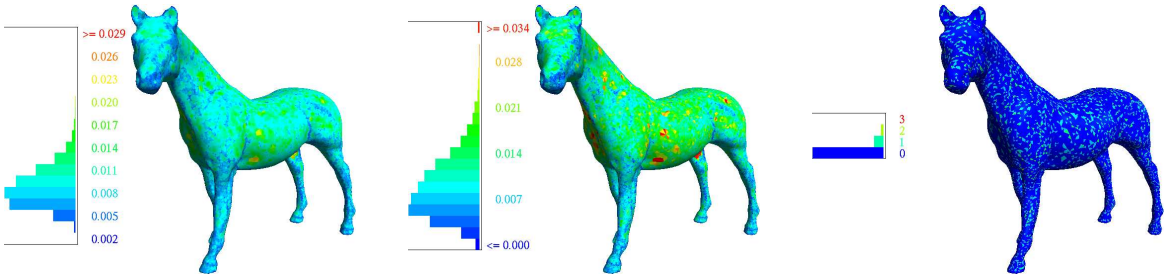
```
// Encodes the first n–cell, and adds its facets to the queue
 1: σ ← encode_first(initial_cell())
 2: q ← ∅ ; q.push(∂σ)
// Front propagation
 3: while τ ← q.top() do
 4:    if τ.mark then continue end
   // Encodes all uncoded the cells incident to τ
 5:    for all σ ≻ τ do
 6:       if σ.mark then continue end
      // Encodes the degree of the facet (non–simplicial case)
 7:       {w_1,...,w_m} ← apexes(σ,τ)
 8:       encode(m)
      // Computes and encodes the geometrical function
 9:       [R_min,R_max] ← quantise_range(τ,{w_1 ... w_m})
10:       encode(R_min,R_max)
      // Apex candidates in that range, best matches first
11:       {v_1,...,v_k} ← candidates(τ,[R_min,R_max])
12:       sort({v_1,...,v_k},G)
      // Encodes the position of each apex in the list
13:       for j ∈ [[1,m]] do encode(i : w_j = v_i) end
14:       q.push(∂σ) ; σ.mark ← true
15:    end for
   // End of incident n–cells (non–closed manifold case)
16:    encode(−1) ; τ.mark ← true
17: end while
```

Non–pure complexes. The above algorithms can be extended to cope with more general topology. If the n–complex K is not pure, as the one of Fig.2, we first encode it as if it were a pure complex. The uncoded cells will then be the non–pure elements of K. They form a complex K' of dimension lower than n. We then encode K' as above and continue recursively. This involves at most $n-1$ calls to Alg. 1 or Alg. 2, which maintains the linear complexity of these algorithms (considering that the localisation of the candidates is constant).

Guarantees. Given a convex cell complex K, the GEncode compression encodes a sequence of cell attachments, starting from an empty cell, and ending at K. The decompression reconstructs K by this sequence, and identifies each cell at-

Algorithm 2 gdecode($\mathcal{R}$,$\mathcal{G}$): decodes one component of a pure n–complex.

```
// Decodes the first n–cell, and adds its facets to the queue
 1: σ ← decode_first()
 2: q ← ∅ ; q.push(∂σ)
// Front propagation
 3: while τ ← q.top() do
 4:    if τ.mark then continue end
   // Decodes the degree of the facet or the code for next τ
 5:    while m ← decode() && m ≠ −1 do
      // Decodes the geometrical range
 6:       [R_min,R_max] ← decode()
      // Apex candidates in that range, best matches first
 7:       {v_1,...,v_k} ← candidates(τ,[R_min,R_max])
 8:       sort({v_1,...,v_k},G)
      // Decodes each apex by its position in the list
 9:       for j ∈ [[1,m]] do i ← decode() ; w_j ← v_i end
      // Attach the new cell
10:       σ ← attach(τ,τ ⋆ {w_1,...,w_m})
11:       q.push(∂σ) ; σ.mark ← true
12:    end while
13:    τ.mark ← true
14: end while
```



(a) circumradius (b) edge length (c) quantised range

Figure 8: *Normalisation of the geometric range significantly reduces the entropy.*

tachment uniquely by the front cell, the number of apexes and their identifiers. The definition of a convex cell complex implies that this is enough to define the geometric realisation of the attached cell $|\sigma|$. However, there can be more than one combinatorial description of $|\sigma|$ if two p–cells of the boundary of σ are aligned (contained in a p–affine plane). If this degenerated case does not occur in K, GEncode is then guaranteed to reconstruct K. If it occurs, the model can be either perturbed, or Algs. 1 and 2 can be modified to encode the combinatorial structure of degenerated cells.

5. Geometric Priors

GEncode relies on two complementary geometric functions: the geometric range $[\mathcal{R}_{\min}, \mathcal{R}_{\max}]$ and the geometric prior $\mathcal{G}$ that orders the apexes by their probability to complete the front cell. In a rough sense, $\mathcal{R}$ encodes the highest bits of the apexes, which are usually the same as the highest bits of the front cell τ, and $\mathcal{G}$ predicts the best apex in that range.

We will consider here two uses for the geometric function $\mathcal{R}$: either to encode a specific geometric property of the mesh or to localise the apexes. In the first case, it should be equal to $\mathcal{G}$ (as in [LCL*05]). In the second case, its simplest expression would be the distance to the barycentre of the front cell τ. We will thus focus now on $\mathcal{G}$.

Geometric criterion. The geometric criterion $\mathcal{G}(\tau, w)$ is an arbitrary real valued function that should be minimal with high probability when the cell $\tau \star w$ is a cell of the mesh. This criterion uses the local geometry of $\tau \star w$, and may take into account the decoded mesh, although we will not use this feature here. The more the criterion fits to the coded mesh, the better the compression ratio, since parameter i of line 13 of Alg. 1 will be equal or close to 0 with high probability, reducing the entropy of these codes.

Closest point criteria. The simplest geometric criterion is the distance to the barycentre of τ: $\mathcal{G}_d(\tau, w) = \|w - bary(\tau)\|$. In that case, the geometric function $\mathcal{R}$ can be normalised by the volume (length) of τ quantised on $[0, \infty[$ with an exponential function: $\mathcal{R}_d(\tau, w) = \left\lceil log_2\left(\mathcal{G}_d(\tau, w)/vol(\tau)\right)\right\rceil$. For regularly sampled meshes, the geometric range should therefore be always a low value.

Mesh quality criteria. Geometric Modelling generally aims at generating meshes composed of well–shaped cells. The usual definition for well–shaped triangles are the *aspect ratio* $\mathcal{G}_a(\tau = (v_1, v_2), w) = \frac{\|v_1 - v_2\|^2 + \|w - v_1\|^2 + \|w - v_2\|^2}{area(v_1, v_2, w)}$ and the *Delaunay constraint* $\mathcal{G}_c(\tau, w) = circumradius(\tau \star w)$. The first one usually concerns rendering applications and for local mesh improvement [AMD02], whereas the second one is widely used in reconstruction [BMR*99, ACK01, CSD02] and global remeshing [ACSD*03, ACDI03]. These criteria can also be used for the geometric function $\mathcal{R}$ with an exponential quantisation, noticing that they are bounded from below (by $\frac{16}{\sqrt{5}}$ for the aspect ration and $\frac{1}{2}$ for the circumradius normalised by the edge length (Fig.8)). We can observe that the Delaunay criterion mimics the Ball Pivoting [BMR*99, MVL04] algorithm, where the geometric range encodes the variations of the ball radius. In higher dimension, these criteria can be extended using the Cayley–Menger determinant (especially for computing the circumradius [Blu70]) or replaced by the cell volume for faster computation.

Traversal strategy. The criterion $\mathcal{G}$ actually depends of which front cell is chosen at each step. Therefore, the order in which the complex is traversed has an influence on the compression ratio, as in most advancing front (greedy) algorithms. Normalised criteria as the distance or the circumradius will be *a priori* better quantised if the normalisation is bigger. Since the volume of the front cell is the natural normalisation (Fig.8), we store the front as a priority queue ordered by the volume of the cells.

6. Results

Compression results. GEncode originally intended to compress better meshes that have a nice geometry. The Delaunay criterion is particularly adapted to reconstruction algorithms [BMR*99, ACK01, CSD02] or some techniques of re–meshing [ACSD*03, ACDI03]. The results of Tab. 1 shows this behaviour stands in practise, as the remeshed

	Quantised Range $\mathcal{R}$		Quantisation precision			Geometric Criterion $\mathcal{G}$			
	distance	circumradius	rough	regular	detailed	distance	volume	aspect	circumradius
animal	17%	83%	7%	17%	76%	100%	83%	77%	83%
art	5%	95%	43%	90%	0%	26%	31%	67%	54%
cad	45%	55%	96%	3%	1%	47%	19%	7%	50%
math	67%	33%	55%	21%	24%	54%	5%	22%	35%
medical	0%	100%	100%	0%	0%	61%	0%	0%	39%
sculpture	78%	22%	21%	41%	38%	80%	0%	1%	19%

Table 2: *Percentage of best results obtained by each geometric function, range precision, geometric criterion on the models used for Tab. 1. The best result can be reached by various geometric criteria, leading to a sum over 100%.*

models and the scans sculptures are better compressed than the other models (Fig.9). Although the results presented here are of dimensions two and three, the algorithm has been implemented for any dimension and co–dimension, and the illustrations of this work represent the models *decoded* by our implementation. The only two features that were not implemented in high dimension are the non–pure compression and the manifoldness recognition (which is NP–hard).

Geometric priors. The adaptation power of GEncode allows using different priors on the same model to check which is the most efficient. Tab. 2 compares different methods on the same range of models as Tab. 1, showing the percentage of best results for each quantised range, range precision and geometric criterion. The range precision represents the above–mentioned trade–off between range quantisation and apex identifiers. The best result can be reached by various geometric criteria, particularly when the mesh is very regular or when the quantised range encodes almost completely the apex.

Comparison. GEncode compares nicely to existing compression scheme, although it is able to compress a wider range of models. Tab. 3 details some comparisons with the Edgebreaker algorithm [Ros99, LRS*02, LLRV04] for the meshes illustrating this work. These comparisons were made using the parallelogram prediction for the Edgebreaker, with the same quantisation for the vertices (12 bits per coordinate). Observe that even for surfaces, connectivity–driven compression schemes handle with difficulty non–simple topology, as for the mechanical piece, which has a pinched vertex or the Klein bottle, which is not orientable, whereas these are handled gracefully by the GEncode.

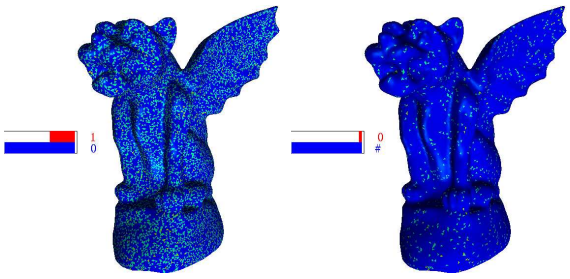
	Geometry	Connectivity
animal	19.343	1.980
art	19.561	1.491
cad	18.566	1.682
math	21.499	1.996
medical	21.220	2.411
scans	18.639	1.372
original	19.334	2.246
re–meshed	18.882	1.269
all	19.089	1.717

Table 1: *GEncode compression ratio, in bits per vertex, using the Delaunay constraint for both the quantised range and the geometric criterion. These results are an average over 200 models, using an order one arithmetic coder.*

GEncode turns out to be very effective for volumes embedded in $\mathbb{R}^3$ (Fig.10), as shown on Tab. 4. We compared with the connectivity–driven compression of [SR00] for the connectivity, and with the streaming compression of [ILGS]. Although this latter method shows some similarity with this work, the compression ratios of GEncode are in average more than 35% superior to streaming compression, at the cost of being slower.

7. Conclusion

This works proposed a new geometry–driven compression scheme that is, to our knowledge, the first compression method that encodes meshes of any dimension and arbitrary topology, while being efficient compared to compression methods for triangulated surfaces. Prior knowledge on the model is used independently by GEncode: as an input for



(a) geometric range (b) apex identifier

Figure 9: *Compression of scanned models: the connectivity is encoded almost at zero rate: # is not transmitted, and thus almost only* 0 *codes are encoded.*

	nv	EB		GEncode	
		geom	conn	geom	conn
terrain	16641	17.09	0.28	13.11	2.71
mechanical	71150	*NA*	*NA*	15.86	0.82
david	24988	25.80	2.71	16.99	1.63
horse	19851	24.86	3.01	18.22	0.85
gargoyle	30059	20.99	2.32	18.38	0.58
bunny	34834	17.05	2.18	18.77	0.97
blech	4102	21.55	2.10	20.73	2.79
fandisk	6475	19.56	2.25	21.52	1.31
klein	4120	*NA*	*NA*	22.11	2.63
sphere	642	27.98	2.27	24.65	0.03
rotor	600	31.67	3.69	24.08	5.18

Table 3: *GEncode compression ratio (in bits per vertex) for triangulated surfaces compared with Edgebreaker [Ros99, LLRV04]. The mechanical model is not manifold, and the Klein bottle is not orientable, which prevented the Edgebreaker to work.*

	nv	G&F	Stream		GEncode	
		conn	geom	conn	geom	conn
sph simul.	12229	76.36	34.00	75.87	22.50	12.76
iron piece	6103	51.93	29.90	25.17	18.17	10.45
molecule	5853	54.00	34.57	23.51	21.77	19.82
triceratops	4344	55.41	34.84	25.82	19.39	18.06
seismic fault	3403	63.50	36.17	35.15	22.57	11.97
dog	3286	55.50	34.31	17.13	20.13	16.86
fandisk	3000	64.85	36.23	18.11	24.26	9.06
turbine	2953	*NA*	33.91	11.80	22.95	14.71
rattle	2514	*NA*	33.84	11.40	22.48	13.12
solid torus	1004	60.52	41.04	15.69	26.68	9.22
sphere1000	986	65.01	41.19	18.76	31.46	10.01
points on $\mathbb{S}^2$	770	43.56	33.14	8.76	21.87	8.34
bended cube	400	57.86	42.52	15.36	23.54	14.59
gear	234	48.62	47.18	15.01	26.18	10.92
finite elem	141	53.22	53.05	19.29	29.77	11.98
sphere100	100	55.84	57.36	19.68	31.49	8.61

Table 4: *GEncode compression ratio (in bits per vertex) for tetrahedral meshes compared with Grow & Fold [SR00] and streaming compression [ILGS].*

the geometric priors and as optimisation of the execution and compression ratio for the topological priors. In particular for scanned and re–meshed models, classical geometric priors such as the aspect ratio or Delaunay constraint improve significantly the compression ratio.

There is a computational price for the gain in compression: the localisation procedure, which consumes the main part of the execution time, is still slow for general geometrical ranges. In particular when comparing with streaming compression methods, the 35% gain required a three or four times more time in our experiments. Moreover, the separate encoding of the geometry still limits the final compression ratio. On one hand, the geometric compression can be improved, and the synthesis proposed in this work is a first attempt in that direction. On the other hand, improvements based on mixed geometry/connectivity encoding are feasible with GEncode.

Acknowledgement

The authors are grateful to the referees, since they motivated, among other things, tests on tetrahedral meshes, where GEncode turned out to be particularly competitive.

References

[ACDI03] ALLIEZ P., COLIN DE VERDIÈRE É., DEVILLERS O., ISENBURG M.: Isotropic surface remeshing. In *Shape Modeling International* (2003), IEEE.

[ACK01] AMENTA N., CHOI S., KOLLURI R.: The Power Crust, unions of balls, and the medial axis transform. *Computational Geometry 19*, 2–3 (2001), 127–153.

[ACSD*03] ALLIEZ P., COHEN-STEINER D., DEVILLERS O., LEVY B., DESBRUN M.: Anisotropic polygonal remeshing. In *Siggraph* (2003), ACM.

[AD01] ALLIEZ P., DESBRUN M.: Valence–driven connectivity encoding of 3D meshes. In *Computer Graphics Forum* (2001), pp. 480–489.

[AMD02] ALLIEZ P., MEYER M., DESBRUN M.: Interactive geometry remeshing. In *Siggraph* (2002), ACM, pp. 347–354.

[Blu70] BLUMENTHAL L. M.: *Theory and applications of distance geometry*. Chelsea, New York, 1970.

[BMR*99] BERNARDINI F., MITTLEMAN J., RUSHMEIER H., SILVA C., TAUBIN G.: The Ball–Pivoting Algorithm for surface reconstruction. *Transactions on Visualization and Computer Graphics 5*, 4 (1999), 349–359.

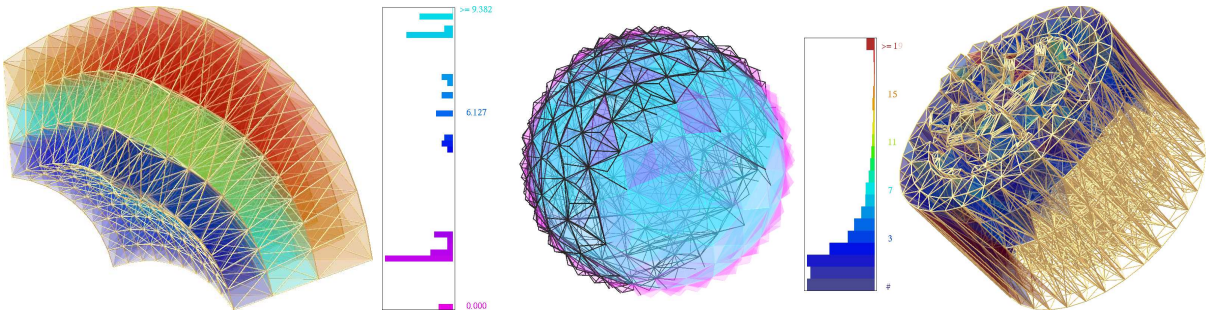
[BWK02] BOTSCH M., WIRATANAYA A., KOBBELT L.: Efficient high quality rendering of point sampled geometry. In *Eurographics Workshop on Rendering* (2002), pp. 53–64.

[CD04] CASTELLI ALEARDI L., DEVILLERS O.: Canonical triangulation of a graph, with a coding application. INRIA preprint, 2004.

[CR04] COORS V., ROSSIGNAC J.: Delphi: geometry-based connectivity prediction in triangle mesh compression. *The Visual Computer 20*, 8–9 (2004), 507–520.

[CSD02] COHEN-STEINER D., DA T. K. F.: A greedy Delaunay–based surface reconstruction algorithm. *The Visual Computer 20*, 1 (2002), 4–16.

[GBTS99] GUÉZIEC A., BOSSEN F., TAUBIN G., SILVA C.: Efficient compression of non–manifold polygonal meshes. *Computational Geometry 14*, (1999), 137–166.



(a) A bended cube, decompressed from cold to hot colours.

(b) A mesh of poor quality, generated from points closed to the sphere $\mathbb{S}^2$, with the distribution of their circumradii.

(c) A turbine model, similar to the rotor surface, with the distribution of the apex identifiers during the decoding.

Figure 10: *Compression on tetrahedral meshes works exactly as for triangulated surfaces.*

[GD02] GANDOIN P.-M., DEVILLERS O.: Progressive lossless compression of arbitrary simplicial complexes. In *Siggraph* (2002), vol. 21, ACM, pp. 372–379.

[GGS99] GUMHOLD S., GUTHE S., STRAŠER W.: Tetrahedral mesh compression with the Cut–Border machine. In *Visualization* (1999), IEEE, pp. 51–58.

[Hat02] HATCHER A.: *Algebraic topology*. Cambridge University Press, 2002.

[IA02] ISENBURG M., ALLIEZ P.: Compressing hexahedral volume meshes. In *Pacific Graphics* (2002), pp. 284–293.

[ILGS] ISENBURG M., LINDSTROM P., GUMHOLD S., SHEWCHUK J.: Streaming compression of tetrahedral volume meshes.

[KG01] KRONROD B., GOTSMAN C.: Efficient coding of nontriangular mesh connectivity. *Graphical Models 63* (2001), 263–275.

[KPRW05] KÄLBERER F., POLTHIER K., REITEBUCH U., WARDETZKY M.: Freelence — coding with free valences. *Computer Graphics Forum 24*, 3 (2005), 469–478.

[KR99] KING D., ROSSIGNAC J.: Guaranteed 3.67v bit encoding of planar triangle graphs. In *Canadian Conference on Computational Geometry* (1999), pp. 146–149.

[LAD02] LEE H., ALLIEZ P., DESBRUN M.: Angle-analyzer: A triangle-quad mesh codec. In *Eurographics* (2002), vol. 21(3).

[LCL*05] LEWINER T., CRAIZER M., LOPES H., PESCO S., VELHO L., MEDEIROS E.: Gencode: geometry–driven compression in arbitrary dimension and co–dimension. In *Sibgrapi* (2005), IEEE, pp. 249–256.

[LDS03] LEE H., DESBRUN M., SCHRÖDER P.: Progressive encoding of complex isosurfaces. *Transactions on Graphics 22*, 3 (2003), 471–476.

[Lew05] LEWINER T.: *Mesh Compression from Geometry*. PhD thesis, Géométrica Project, INRIA–Sophia Antipolis, delivered by Université Paris VI, 2005.

[LLRV04] LEWINER T., LOPES H., ROSSIGNAC J., VIEIRA A. W.: Efficient Edgebreaker for surfaces of arbitrary topology. In *Sibgrapi* (2004), IEEE, pp. 218–225.

[LRS*02] LOPES H., ROSSIGNAC J., SAFONOVA A., SZYMCZAK A., TAVARES G.: Edgebreaker: a simple compression for surfaces with handles. In *Solid Modeling and Applications* (2002), ACM, pp. 289–296.

[LVLM04] LEWINER T., VELHO L., LOPES H., MELLO V.: Simplicial isosurface compression. In *Vision, Modeling and Visualization* (2004), IOS Press, pp. 299–306.

[MVL04] MEDEIROS E., VELHO L., LOPES H.: Restricted bpa: applying ball–pivoting on the plane. In *Sibgrapi* (2004), IEEE, pp. 372–379.

[RC99] ROSSIGNAC J., CARDOZE D.: Matchmaker: manifold BReps for non–manifold r–sets. In *Solid Modeling and Applications* (1999), ACM, pp. 31–41.

[Ros99] ROSSIGNAC J.: Edgebreaker: connectivity compression for triangle meshes. *Transactions on Visualization and Computer Graphics 5*, 1 (1999), 47–61.

[SFS05] SCHEIDEGGER C. E., FLEISHMAN S., SILVA C. T.: Triangulating point–set surfaces with bounded error. In *Symposium on Geometry Processing* (2005), Eurographics, pp. 63–72.

[SR00] SZYMCZAK A., ROSSIGNAC J.: Grow & Fold: compressing the connectivity of tetrahedral meshes. *Computer–Aided Design 32*, 8/9 (2000), 527–538.

[Tau02] TAUBIN G.: Blic: bi–level isosurface compression. In *Visualization* (2002), IEEE, pp. 451–458.

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

[TR98] TAUBIN G., ROSSIGNAC J.: Geometric compression through topological surgery. *Transactions on Graphics 17*, 2 (1998), 84–115.