Ecole Doctorale : E.D.I.T.E.

THESE

pour obtenir le titre de
Docteur en Sciences
de l'Université Pierre et Marie Curie (Paris VI)

Mention: Informatique

présentée par

Thomas LEWINER

Compression de Maillages à partir de la Géométrie
(Mesh Compression from Geometry)

Thèse dirigée par **Jean–Daniel BOISSONNAT**
préparée à l'INRIA Sophia Antipolis au sein du Projet Géométrica
et soutenue le *16, Décembre 2005*
devant le jury composé de :

J.–D. Boissonnat	INRIA — Sophia Antipolis	Directeur
J. Rossignac	Georgia Tech — Atlanta	Rapporteur
D. Cohen–Or	Tel Aviv University	Rapporteur
P. Frey	UPMC — Paris	Président
H. Lopes	PUC — Rio de Janeiro	Examinateur
F. Schmitt	ENST — Paris	Examinateur
O. Devillers	INRIA — Sophia Antipolis	Examinateur
F. Lazarus	INPG — Grenoble	Examinateur

Thomas Lewiner

graduated from the Ecole Polytechnique (Paris, France) in Algebra and Computer Science, and in Theoretical Physics. He then specialized at the Ecole Supérieure des Télécommunications (Paris, France) in Signal and Image Processing, and in Project Management, while working for Inventel in wireless telecommunication systems based on BlueTooth technology. He then obtained a Master degree at the Pontifícia Universidade Católica do Rio de Janeiro in computational topology, and actively participated to the Mathematical department's work for Petrobras.

Bibliographic description

Lewiner, Thomas

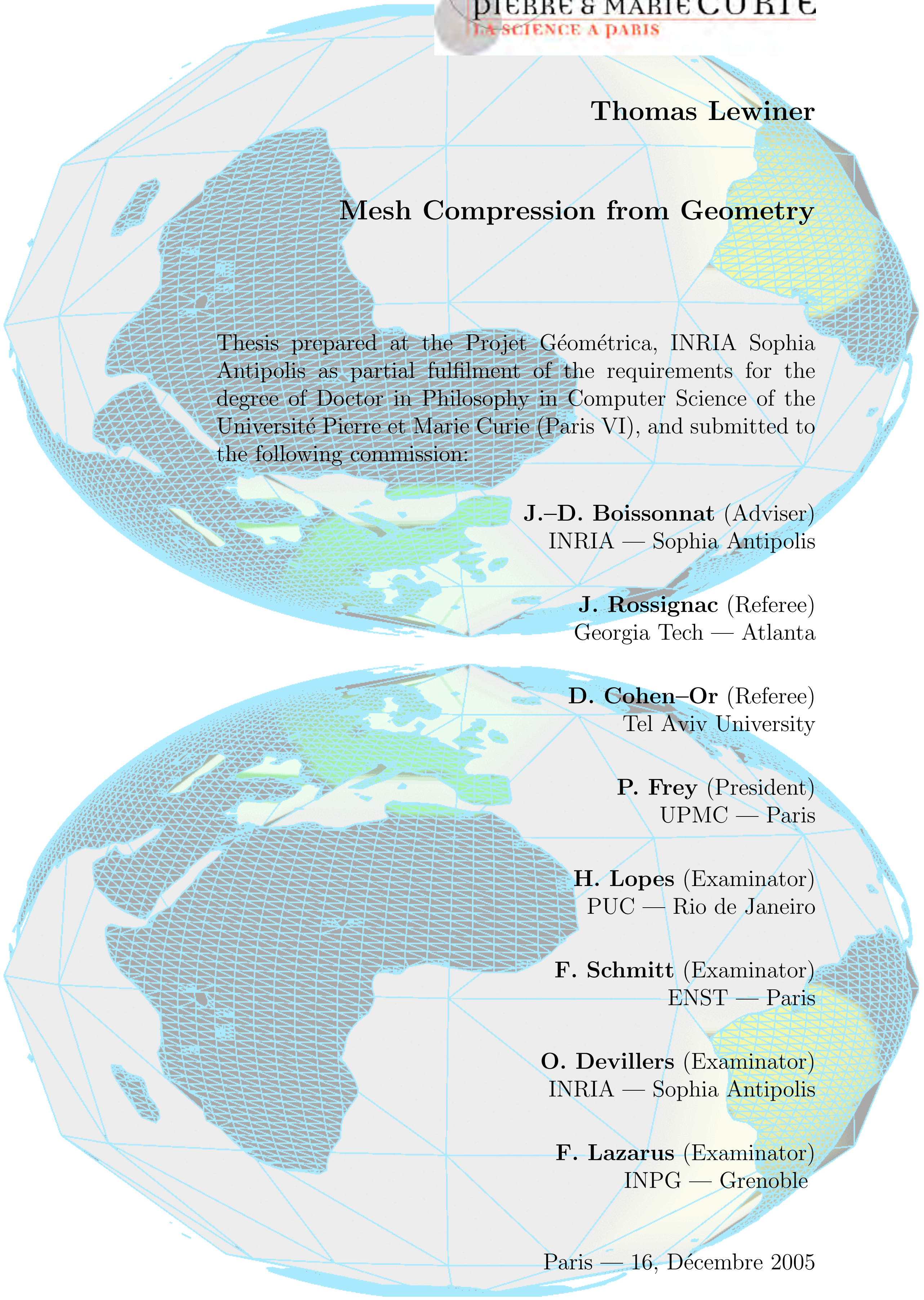
Mesh Compression from Geometry / Thomas Lewiner; adviser: Jean–Daniel Boissonnat.— Paris : Université Pierre et Marie Curie (Paris VI), 2005 [Projet Géométrica, INRIA Sophia Antipolis].—

1 vol. (131 p.) : ill. ; 29,7 cm.—

PhD Thesis.— Université Pierre et Marie Curie (Paris VI) : prepared at the Projet Géométrica, INRIA Sophia Antipolis.—

Bibliography included.—

1. Computer Science – Thesis. 2. Mesh Compression. 3. Geometry–Driven Compression. 4. Level sets. 5. Isosurfaces. 6. Isosurface compression. 7. Computational Geometry. 8. Solid Modelling. 9. Data Compaction. I. Lewiner, Thomas. II. Boissonnat, Jean–Daniel. III. Projet Géométrica, INRIA Sophia Antipolis. IV. E.D.I.T.E., Université Pierre et Marie Curie (Paris VI). V. Mesh Compression from Geometry.



Thomas Lewiner

Mesh Compression from Geometry

Thesis prepared at the Projet Géométrica, INRIA Sophia Antipolis as partial fulfilment of the requirements for the degree of Doctor in Philosophy in Computer Science of the Université Pierre et Marie Curie (Paris VI), and submitted to the following commission:

Paris — 16, Décembre 2005

Acknowledgments

This is my second PhD redaction, and my tribute to the many people who helped me remains. First of all, my father who motivated me for science and always supported me, especially in difficult moments, my mother and my grand mother who made me feel this wonderful familiar environment wherever I was. I am also particularly grateful to my advisor Jean–Daniel for his support and encouragements along these three years. He maintained a very open–minded vision that allowed me to convey many ideas between France and Brazil, and this has been very fruitful for my professional and personal formation. His support has been relayed by the everyday work of Agnès and Creuza, the help of Francis Schmitt in the defence organisation, of Pierre Deransart and Rachid Deriche from the International Relations of the INRIA, the funding of the Fondation de l'École Polytechnique, and the access conceded by the Pontifícia Universidade Católica do Rio de Janeiro. Moreover, I would like to thank particularly Agnès, Marie–Pierre and Dominique for the help in preparing the defence, the referees and the jury members for the exceptional attention they manifested for this work.

This big team that welcomed my projects allowed a wide exchange of point of views, with the practise of Hélio, Pierre and Sinésio, the experience of Jean–Daniel, Olivier, Luiz and Geovan, the lights of Marcos, Carlos and David and the cooperation of my colleagues of both sides: Vinícius, Wilson, Marcos, Rener, Alex, Cynthia, Afonso, Jessica, Christina, Francisco, Fabiano, Marcos and Marie, Luca, David, Marc, Steve, Philippe, Camille, Christophe, Abdelkrim and Laurent. In particular, I would like to thank Vinícius for his great help in the use of his data structure, and Luiz and Hélio for the starting ideas of at least half of this work.

But the scientific help is not sufficient to achieve this work, and the emotive support has been crucial for leading it. This constant energy along these three years came from the presence of my grandmother during the preparation of this work, the souvenir of Fanny and Simone, the eternal support of my parents, the complicity of my sisters, brothers–in–law and of Debora together with the tight friendship of my uncles, ants and cousins. I also felt this warmness with my friends, and I am happy that this includes my professors and colleagues, with Albane, JA, Anne–Laure, Juliana(s), Ana Cristina, Silvana, Tania, Benjamin, Nicolas, David(s), Aurélien, Eric, Mathieu, Sergio(s), Bernardo, . . .

Résumé

Lewiner, Thomas ; Boissonnat, Jean–Daniel. **Compression de Maillages à partir de la Géométrie**. Paris, 2005. 131p. Thèse de Doctorat — Université Pierre et Marie Curie (Paris VI) : preparée au Projet Géométrica, INRIA Sophia Antipolis.

Les images ont envahi la plupart des publications et des communications contemporaines. Cette expansion s'est accélérés avec le développement de méthodes efficaces de compression spécifiques d'images. Aujourd'hui, la génération d'images s'appuie sur des objets multidimensionnels produits à partir de dessins assistés par ordinateurs, de simulations physiques, de représentations de données ou de solutions de problèmes d'optimisation. Cette variété de sources motive la conception de schémas dédiés de compression adaptés à des classes spécifiques de modèles. Ce travail présente deux méthodes de compression pour des modèles géométriques. La première code des ensembles de niveau en dimension quelconque, de manière directe ou progressive, avec des taux de compression au niveau de l'état de l'art pour les petites dimensions. La seconde méthode code des maillages de n'importe quelle dimension ou topologie, même sans être purs ou variété, plongés dans des espaces arbitraires. Les taux de compression pour les surfaces sont comparables aux méthodes usuelles de compression de maillages comme Edgebreaker.

Mots–Clefs

Compression de Maillages. Compression orientée par la Géométrie. Ensembles de Niveau. Isosurfaces. Compression d'Isosurfaces. Géométrie Algorithmique. Modelage Géométrique. Compression de Données.

Abstract

Lewiner, Thomas ; Boissonnat, Jean–Daniel. **Mesh Compression from Geometry**. Paris, 2005. 131p. PhD Thesis — Université Pierre et Marie Curie (Paris VI) : prepared at the Projet Géométrica, INRIA Sophia Antipolis.

Images invaded most of contemporary publications and communications. This expansion has accelerated with the development of efficient schemes dedicated to image compression. Nowadays, the image creation process relies on multidimensional objects generated from computer aided design, physical simulations, data representation or optimisation problem solutions. This variety of sources motivates the design of compression schemes adapted to specific class of models. This work introduces two compression schemes for geometrical models. The first one encodes level sets in any dimension, in a direct or progressive manner, with both state of the art compression ratios for low dimensions. The second one encodes meshes of any dimension and topology, even non–pure or non–manifold models, embedded in arbitrary space. The compression ratios for surfaces are comparable with famous mesh compression methods such as the Edgebreaker.

Keywords

Mesh Compression. Geometry–Driven Compression. Level sets. Isosurfaces. Isosurface compression. Computational Geometry. Solid Modelling. Data Compaction.

Contents

Avant de quitter la vie de ma propre volonté et avec ma lucidité, j'éprouve le besoin de remplir un dernier devoir : adresser de profonds remerciements au Brésil, ce merveilleux pays qui m'a procuré, ainsi qu'à mon travail, un repos si amical et si hospitalier. De jour en jour, j'ai appris à l'aimer davantage et nulle part ailleurs je n'aurais préféré édifier une nouvelle existence, maintenant que le monde de mon langage a disparu pour moi et que ma patrie spirituelle, l'Europe, s'est détruite elle-même.

Stefan Zweig, *Petrópolis, 22 février 1942.*

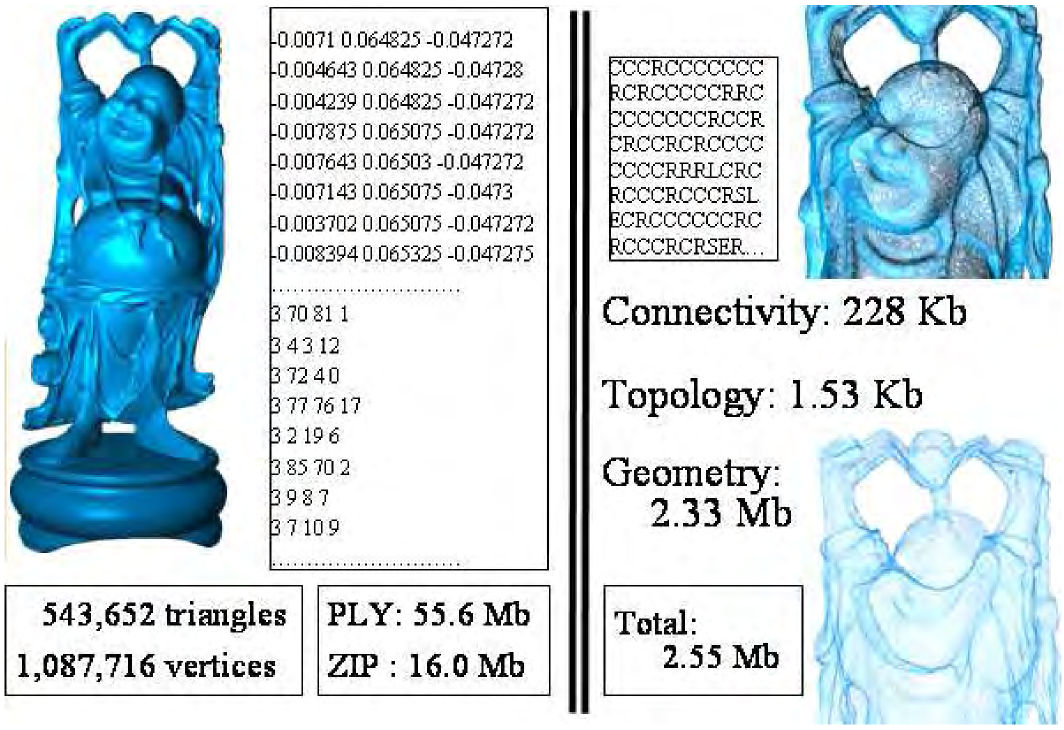
I
Introduction

Images surpassed the simple function of illustrations. In particular, artificial and digital images invaded most of published works, from commercial identification to scientific explanation, together with the specific graphics industry. Technical advances created supports, formats and transmission protocols for these images, and these contributed to this expansion. Among these, high quality formats requiring low resources appeared with the development of generic, and then specific, compression schemes for images. More recently drew on the sustained trend to incorporate the third dimension into images, and this motivates orienting the developments of compression towards higher dimensional images.

There exists a wide variety of images, from photographic material to drawings and artificial pictures. Similarly, higher dimensional models are produced from many sources: The graphics industry designers draw three–dimensional objects by their contouring surface, using geometric primitives. The recent developments of radiology make intense use of three–dimensional images of the human body, and extract isosurfaces to represent organs and tissues. Geographic and geologic models of terrain and underground consist in surfaces in the multi–dimensional of physical measures. Engineering usually generate finite elements solid meshes in similar multi–dimensional spaces to support physical simulations, while reverse engineering, archæological heritage preservation and commercial marketing reconstruct real objects from scanned samplings. In addition, other fields are adding new multi–dimensional modelling, such as data representation and optimisation, particularly for solving financial problems.

Compression methods for three–dimensional models appeared mainly in the mid 1990's with [Deering 1995] and developed quickly since then. This evolution turned out to be a technical necessity, since the size and complexity of the typical models used in practical applications increases rapidly. The most performing practical strategies for surfaces are based on the Edgebreaker of [Rossignac 1999] and the valence coding of [Touma and Gotsman 1998]. These are classified as connectivity–driven mesh compression, since the proximity of triangles guides the sequence of the surface vertices to be encoded. More recently, dual approaches proposed to guide the encoding of the triangle proximity by the geometry, such as done in [Gandoin and Devillers 2002].

Actually, the diversity of images requires this multiplicity of compression programs, since specific algorithms usually perform better than generic one, if they are well adapted, as illustrated on Figure I.1. This work focuses on two new geometry–driven compression methods that compare nicely to the state



```
-0.0071 0.064825 -0.047272
-0.004643 0.064825 -0.04728
-0.004239 0.064825 -0.047272
-0.007875 0.065075 -0.047272
-0.007643 0.06503 -0.047272
-0.007143 0.065075 -0.0473
-0.003702 0.065075 -0.047272
-0.008394 0.065325 -0.047275
...........................
3 70 81 1
3 4 3 12
3 72 4 0
3 77 76 17
3 2 19 6
3 85 70 2
3 9 8 7
3 7 10 9
...........................
```

```
CCCRCCCCCCCC
RCRCCCCCRRC
CCCCCCCRCCR
CRCCRCRCCCC
CCCCRRRLCRC
RCCCRCCCRSL
ECRCCCCCCRC
RCCCRCRSER...
```

Figure I.1: Specific compression methods are more efficient than generic ones.

of the art for surfaces, and further extend to any dimension. Although this area of mesh compression is recent, it has been very productive with great improvements, as testifies the amount of bibliographic references of this work. Many new ideas appeared in these works during this last decade, and could be incorporated to these new proposals in a close future.

I.1 Main Results

Edgebreaker improvement

The first contribution of this work is a small improvement to Edgebreaker compression scheme of [Rossignac 1999], introduced in Chapter IV **Connectivity–Driven Compression**, illustrated on Figure I.2. It provides a simple and efficient way to compress meshes with boundary, as the one of Figure I.3, formulated as an extension of the compression of meshes with handles described in [Lopes *et al.* 2002]. Although the original algorithm already encoded boundaries, the proposed extension lowers the theoretical code entropy and the practical compression ratio. Moreover, it provides a direct way to read the topology of the surface on the compressed format. This part originally appeared in [Lewiner *et al.* 2004].

This contribution is introduced here mainly to state what is intended by connectivity–driven strategies, and the Edgebreaker example has been chosen since it is particularly simple and efficient. Moreover, it emphasizes how the combinatorial study of meshes introduced in the basic concepts of Chapter III **Meshes and Geometry** gives elegant and productive presentations

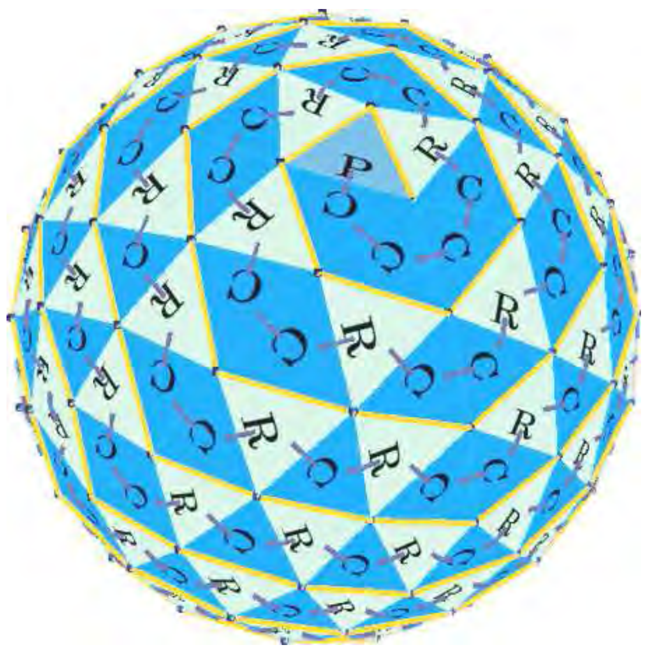


Figure I.2: Compression codes used by the Edgebreaker on a sphere model.

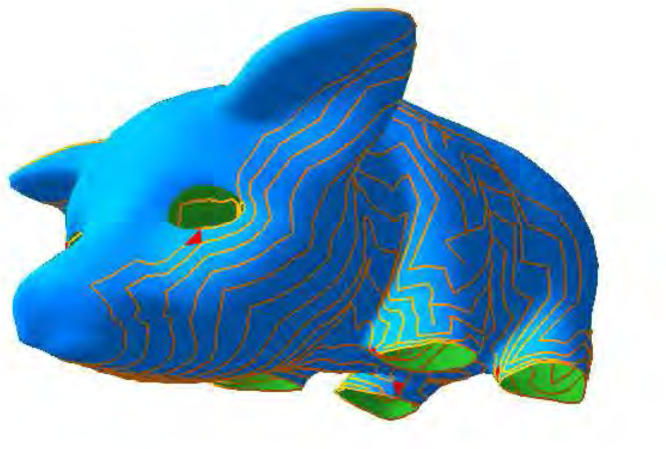


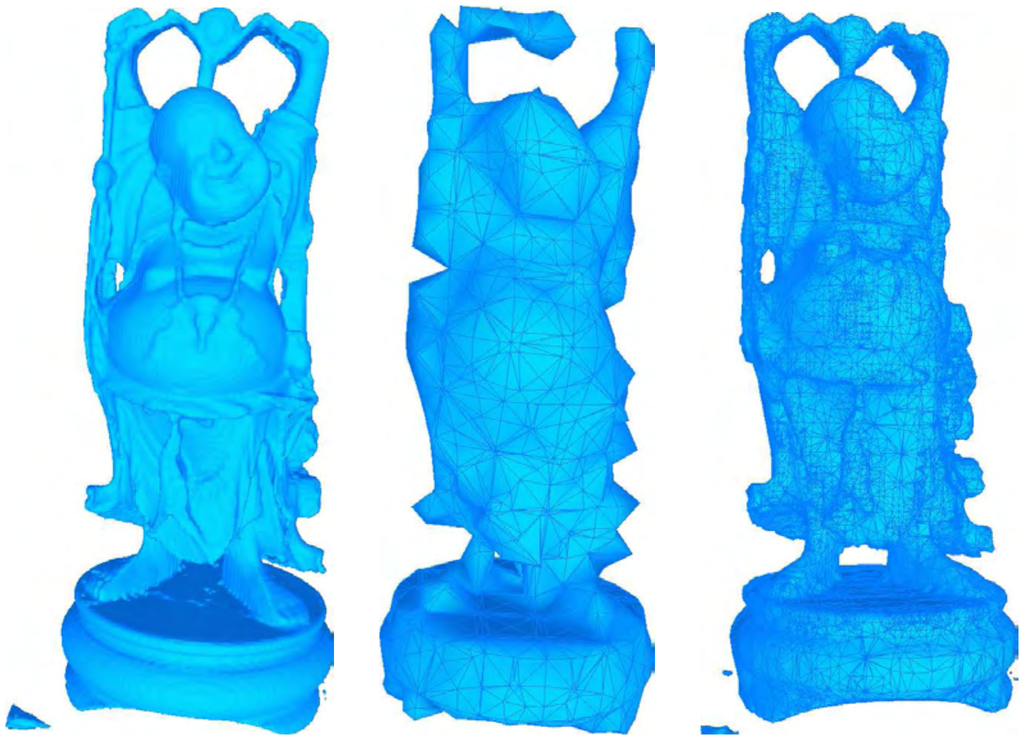
Figure I.3: Edgebreaker compression on a more complex model.

of such algorithms. Actually, most of mesh decompression processes hang on combinatorial and topological operations that have been extensively studied, and which thus provide powerful tools to design mesh compression algorithms.

This analysis provides another improvement on the Wrap&Zip decompression of [Rossignac and Szymczak 1999] for the Edgebreaker. The improvement reduces the execution time of the Zip part. The maintains the linear complexity and the single pass of the compression process, as described in Algorithm 5: compress, calling Algorithm 4: traverse and Algorithm 6: check handle. The decompression is also linear, but requires more than one pass, as described in Algorithm 7: decompress, calling Algorithm 8: wrap, Algorithm 9: fast zip and Algorithm 10: read geometry.

Level sets direct and progressive compression

Although geometry–driven compression compares nicely to connectivity–driven one, the experience acquired in connectivity–driven compression gives for many types of meshes a significant advantage. However, for level sets, geometry–driven compression outperforms any connectivity–driven approaches, as already stated in [Taubin 2002]. Whereas for surfaces the average compression ratios turn around 8 to 10 bits per vertex, level set compression reaches 1 or 2 bits per vertex for the same quality of the decompressed model.



(a) original **(b) without control** **(c) topology control**

Figure I.4: Isosurface multiresolution representation is more correct when preserving its topology.

Chapter V **Level Set Compression** introduces a significant improvement for level set compression. It provides a general method for direct and progressive compression of these level sets, using intensively the stellar theory introduced in Chapter III **Meshes and Geometry** and the powerful structure of [Mello *et al.* 2003] deduced from it. This structure provides a very high degree of adaptability since it is a specification of binary space partitions, together with simple and striking control on the induced multiresolution representation of level sets, as shown on Figure I.4.

For isosurfaces, this method achieves state of the art rate/distortion curves and compression ratio, in particular in comparison to octree–based methods, included in our method as shown on Figure I.5. Moreover, this proposal is described for any dimension in the same manner, using only stellar operations on edges. Its current implementation is still memory consuming. The specification of the algorithm for surfaces appeared first in [Lewiner *et al.* 2004*4], and for curves in [Lewiner *et al.* 2004*2, Lewiner *et al.* 2005*3].

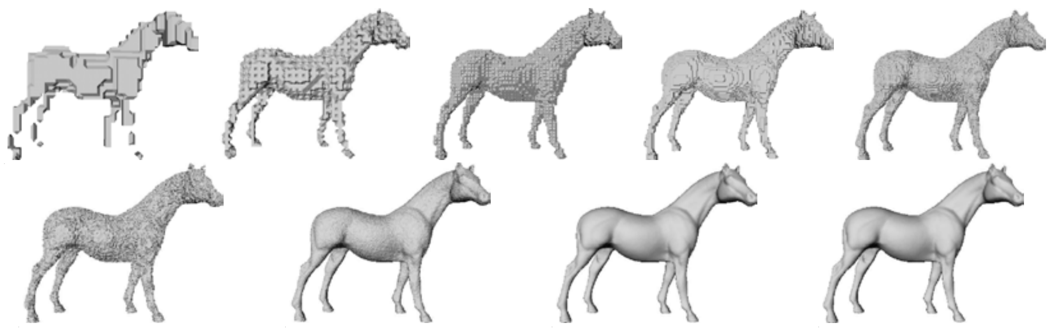


Figure I.5: Decompression of the Horse model as a 257^3 regular sampled data. The first images correspond to the binary multi–triangulation refinement, followed by the geometric refinements.

Generic geometry–driven direct compression

Motivated by these results, confirming the idea that geometry–driven compression systems can be more efficient than connectivity–driven ones, we developed a general geometry–driven direct compression scheme called GEncode. This method, as described in Algorithm 15: gencode and Algorithm 16: gdecode, handles any simplicial complex or polytope of any dimension embedded in any ambient space, as the examples of Figure I.6. For surfaces, GEncode compares nicely to connectivity–driven programs, although the geometrical compression part can still be greatly improved. These results, presented in Chapter VI **GEncode**, already appeared in [Lewiner *et al.* 2005*1].

This general scheme relies on a geometrical criterion to encode the connectivity of the mesh. This criterion is an arbitrary real valued function that should characterise the geometry of a cell of the encoded mesh. The more the criterion fits to the mesh, the better the compression ratio and the algorithm execution time. As an illustration, we derived a geometric criterion from the circumradius of a cell. This criterion fits particularly well to meshes generated by Delaunay–based reconstruction or remeshing. This completed our primary objective: encode the connectivity of a reconstructible mesh at almost zero cost.

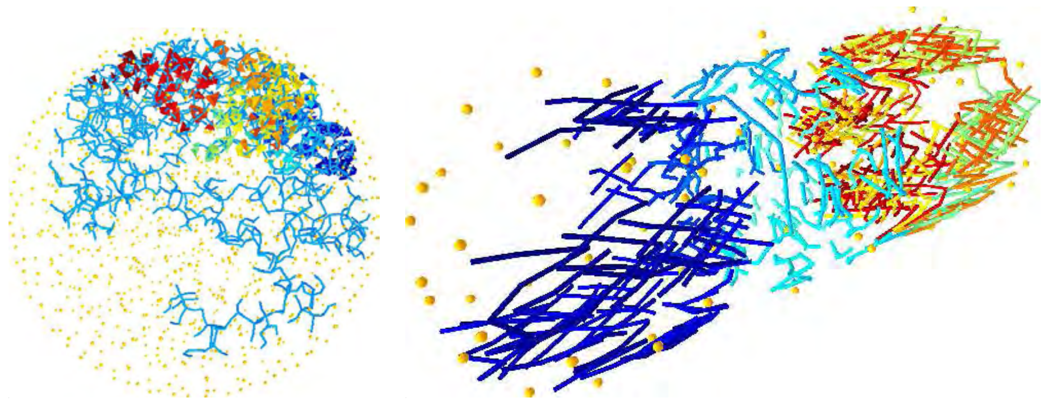


Figure I.6: Encoding traversal of a tetrahedral mesh of a solid sphere in $\mathbb{R}^3$ (left) and of a Cartesian product of a 2–sphere with a circle in $\mathbb{R}^4$ (right).

Furthermore, we present a practical method to derive geometric criterion for a given class of meshes. This method further provides a general description of the relation between the geometry of an object and its discretisation through a mesh model. This relation actually represents the sampling model used by the application that generated the mesh, which is usually implicit in the algorithm design although it represents a fundamental step in its validation. We hope that this method would clarify this intrinsic structure of meshes, which is the proper of efficient information coding.

I.2 Outline

This thesis is organised as follows. Chapter II **Encoding and Compression** gives some general notions of information theory and coding methods. In particular, we tried to emphasize the different parameters a coding system needs to tune to work correctly. This task turns out to be delicate and technical, but remains fundamental to obtain compression ratios comparable with the state of the art. Moreover, the problems involved with this tuning usually put at stake the deep understanding of the proposed encoding, as a opposed to the theoretical entropy eventually achieved by the classical arithmetic coder.

Then, Chapter III **Meshes and Geometry** introduce the main object we will try to encode efficiently: geometrical meshes. These meshes are usually considered as a discretisation of a geometrical object, possibly of high dimension in high dimensional space. However, these meshes have a specific combinatorial structure, partly characterised by the topology of the original object, partly corresponding to the discretisation process. This combinatorial structure can be decomposed by a complete set of construction and edition operators. These operators are described, together with some of the geometry discretisation processes.

A general presentation of classical mesh compression methods is then introduced in Chapter IV **Connectivity–Driven Compression**. These are described in terms of the combinatory of the mesh structure, which differentiates directly the two main proposals for surfaces, the Edgebreaker and the valence coding, as dual and primal strategies. The example of the Edgebreaker is then detailed as a very simple, although very efficient example of mesh encoding. We emphasize the asymmetry between encoding and decoding process, together with some difficulties encountered to encode topological singularities for these connectivity–driven schemes. We introduce some practical solutions for these problems and a simple enhancement of the decompression procedure. The chapter ends with a rough comparison of dual and primal connectivity–driven compression methods.

On a different side, Chapter V **Level Set Compression** proposes a geometry–driven approach to mesh compression, when these meshes are presented as the zero level set of a sampled real valued function. The compression uses specific support space triangulations to represent the sampled function, and this representation provides a high degree of adaptability and regularity. This allows first to generate a controlled multiresolution representation of the level set, and then to encode this multiresolution in a progressive manner. Both the direct and progressive methods are very competitive to state of the art method for contour curve and isosurface compression. Moreover, the compression ratios are much lower than for connectivity–driven approaches, mainly because of the real control of the sampling model such level sets provides.

This control can be extended for generic meshes through a geometric characterisation of the discretisation process. This is the base of the GEncode, introduced in Chapter VI **GEncode**. The proposed scheme is extensively described, and a simple implementation for very general meshes is presented. The required properties of the geometric characterisation are introduced, with the practical example of the Delaunay–based meshes. A general method for producing such criterion is then described. The final results are very motivating, since they compare nicely with the Edgebreaker for usual meshes.

II
Encoding and Compression

This work aims at defining new methods for compressing geometrical objects. We would like first to briefly introduce what we mean by compression, in particular the relation of the abstract tool of information theory [Nyquist 1928, Hartley 1928, Shannon 1948], the asymptotic entropy of codes [Shannon 1948, Huffman 1952, Tutte 1998] and the practical performance of coding algorithms [Huffman 1952, Rissanen 1976, Lempel and Ziv 1977, Moffat *et al.* 1995]. We will then focus on the arithmetic coder [Rissanen 1976, Moffat *et al.* 1995] since we will mainly use it in practise. This coder can be enhanced by taking in account deterministic or statistic information of the object to encode, which translates technically by a shift from Shannon's entropy [Shannon 1948] to Kolmogorov complexity [Li and Vitanyi 1997]. Finally, we will describe how this coding takes part in a compression scheme. General references on data compression can be found in [Salomon 2000].

II.1 Information Representation

(a) Coding

Source and codes. Coding refers to a simple translation process that converts symbols from one set, called the *source* to another, this last one being called the set of *codes*. The conversion can then be applied in a reverse way, in order to recover the original sequence of symbols, called *message*. The purpose is to represent any message of the source into a more convenient way, typically a way adapted to a specific transmission channel. This coding can intend to reduce the size of the message [Salomon 2000], for example for compression applications, or on the contrary increase its redundancy to be able to detect transmission errors [Hamming 1950].

Enumeration. A simple example coder would rely on enumerating all the possible messages, indexing them from 1 to n during the enumeration. The coder would then simply assign one code for each message. In practise, the number of possibilities is huge and difficult to enumerate, and it is hard to recover the original message from its index without enumerating again all the possible messages. However, this can work for specific cases [Castelli and Devillers 2004]. These enumerative coders give a reference for comparing performance of coders. However, in practical cases, we would like the coding to be more efficient for the most frequent messages, even if the performance is altered for less frequent ones. This reference will thus not be our main target.

Coder performance. Two different encodings of the same source will in general generate two coded messages of different sizes. If we intend to reduce the size of the message, we will prefer the coder that generated the smallest message. On a specific example, this can be directly measured. Moreover, for the enumerative coder, the performance is simply the logarithm of the number of elements, since a number n can be represented by $\log(n)$ digits. However, this performance is hard to measure it for all the possible messages of a given application. [Nyquist 1928], [Hartley 1928] and [Shannon 1948] introduced a general tool to measure the asymptotic, theoretic performance of a code, called the *entropy*.

(b) Information Theory

Entropy. The entropy is defined in general for a random message, which entails message generators as symbol sources or encoders, or in particular to a specific message (an observation) when the probabilities of its symbols are defined. If a random message $\mathbf{m}$ of the code is composed of n symbols $\mathbf{s}_1 \dots \mathbf{s}_n$, with probability $p_1 \dots p_n$ respectively, then its entropy $\mathfrak{h}(\mathbf{m})$ is defined by $\mathfrak{h}(\mathbf{m}) = \sum_i -p_i \log(p_i)$. As referred in [Shannon 1948], this definition is natural for telecommunication systems, but it is only one possible measure that respects the following criteria:

1. $\mathfrak{h}()$ should be continuous in the p_i.
2. If all p_i are equal, $p_i = \frac{1}{n}$, then $\mathfrak{h}()$ should increase with n, since there are more possible messages.
3. If the random message $\mathbf{m}$ be broken down into two successive messages $\mathbf{m}_1$ and $\mathbf{m}_2$, then $\mathfrak{h}(\mathbf{m})$ should be the weighted sum of $\mathfrak{h}(\mathbf{m}_1)$ and $\mathfrak{h}(\mathbf{m}_2)$.

Huffman coder. [Huffman 1952] introduced a simple and efficient coder that writes each symbol of the source with a code of variable size. For example, consider that a digital image is represented by a sequence of colours $\mathbf{s}_{black}, \mathbf{s}_{red}, \mathbf{s}_{darkblue}, \mathbf{s}_{lightblue}, \mathbf{s}_{white}$. A simple coder will assign a symbol to each colour, and encode the image as the sequence of colours. This kind of coder will be called next an *order 0 coder*.

If the image is a photo of a seascape, as the one of Figure II.1, the probability to have blue colours in the message will be higher than for red colours. Huffman proposed a simple way to encode with less bits the more frequent colours, here blue ones, and with more bits the less frequent symbols. Consider that each of the colour probabilities is a power of 2: $p_{black} = 2^{-3}, p_{red} = 2^{-4}, p_{darkblue} = 2^{-1}, p_{lightblue} = 2^{-2}, p_{white} = 2^{-4}$.

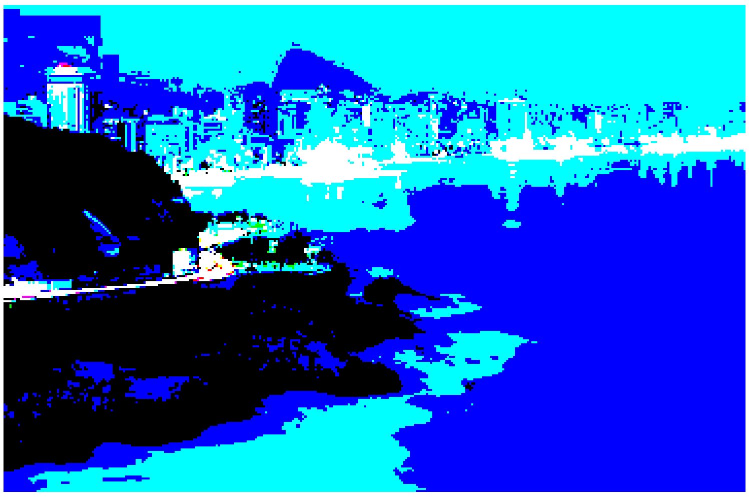


Figure II.1: Huffman coding relies on the frequency of symbols of a message, here the colours inside an image.

These probabilities can be represented by a binary tree, such as each symbol of probability 2^{-b} is a leaf of depth b in the binary tree. Then each symbol is encoded by the left (0) and right (1) choices to get from the root of the tree to that symbol. The decoding is then performed by following the left and right codes until reaching a leaf, and the symbol of that leaf is a new element of the decoded message. In that context, the probability of each left and right operation is $\frac{1}{2}$, which maximises the entropy ($\mathfrak{h}\left(\mathbf{m}\right) = 1$), i.e., the theoretical performance.

Entropy coder. The original Huffman code also worked out for general probabilities, but without maximising the entropy. It uses a greedy algorithm to choose how to round off the probabilities towards powers of 2 [Huffman 1952]. However, Shannon proved that it is asymptotically possible to find a coder of maximum entropy [Shannon 1948], and that no other coder can asymptotically work better in general. This is the main theoretical justification for the definition of $\mathfrak{h}\left(\right)$. [Huffman 1952] introduced a simpler proof of that theorem, by grouping sequence of symbols until their probability become small enough to be well approximated by a power of 2.

(c) Levels of Information

In practise, although the entropy of a given coder can be computed, the theoretical entropy of a source is very hard to seize. The symbols of the source are generally not independent, since they represent global information. In the case of dependent symbols, the entropy would be better computed through the Kolmogorov complexity [Li and Vitanyi 1997]. For example, by increasing the contrast of an image, as human we believe that we loose some of its details, but from the information theory point of view, we added a (mostly) random value to the colours, therefore increasing the information of the image.

An explanation for that phenomenon is that the representation of an image as a sequence of colours is not significant to us. This sequence could be shuffled in a deterministic way, it would not change the coding, but we would not recognise anymore the information of the image. In order to design and evaluate an efficient coding system, we need to represent the **exact** amount of information that is needed for our application, through an **independent** set of codes. If we achieve such a coding, then its entropy can be maximised through a universal coder, such as the Huffman coder or the *arithmetic coder*.

II.2 Arithmetic Coding

The *arithmetic coding* [Rissanen 1976, Moffat *et al.* 1995] encodes the symbols source with code sizes very close to their probabilities. In particular, it achieves an average size of codes that can be a fraction of bits. Moreover, it can encode simultaneously different sources, and provide a flexible way of adapting probabilities of the source symbols. This adaptation can be monitored through rules depending on a *context*, or automatically looking at the previous encoded symbols by varying the *order* of the coder. This section details these points, and provides some sample code inspired from [Arithmetic coding source]. In particular, we would like to detail why the arithmetic coder is usually presented as a universal solution, and how much parameters are hidden behind this universal behaviour. This coder will be used all along this work, and we will detail for each algorithm the hidden parameters which have a significant impact on the final compression ratio.

(a) Arithmetic Coder

Instead of assigning a code for each of the source symbol, an arithmetic coder represent the whole message by a single binary number $\mathbf{m} \in [0, 1[$, with a large number of digits. Where Huffman decoder read a sequence of left/right codes to rebuild the source message, the arithmetic coder will read a sequence of interval shrinking, until finding a small enough interval containing $\mathbf{m}$. At each step, the interval is not shrunk by splitting it in two as in Huffman coding, but each possible symbol of the source is assigned a part of the interval proportional to its probability, and the interval is shrunk to the part of the source symbol. Therefore, a splitting in two corresponds in general to several shrinking, and thus a single bit can encode many symbols, as the next table details on the beginning of the example of Figure II.1, using the above probabilities.

Example. We will first illustrate the arithmetic coding with our previous example of Figure II.1, reduced to Figure II.2: compressing an image by directly encoding its colours: $\mathbf{s}_{black}, \mathbf{s}_{red}, \mathbf{s}_{darkblue}, \mathbf{s}_{lightblue}, \mathbf{s}_{white}$. In order to simplify the writing, we will consider that the probabilities of the colours are decimals, but this does not make any difference. Each probability is assigned an distinct interval of $[0, 1[$:

symbol	probability	[interval [
$\mathbf{s}_{black}$	0.1	[0 , 0.1 [
$\mathbf{s}_{red}$	0.1	[0.1 , 0.2 [
$\mathbf{s}_{darkblue}$	0.4	[0.2 , 0.6 [
$\mathbf{s}_{lightblue}$	0.3	[0.6 , 0.9 [
$\mathbf{s}_{white}$	0.1	[0.9 , 1 [

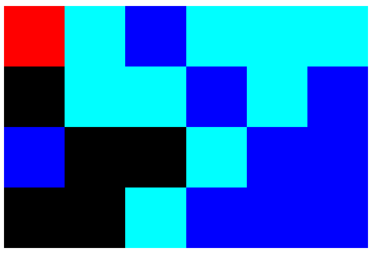


Figure II.2: Reduced image extracted of Figure II.1.

Then, the image to be encoded is the sequence of Figure II.2 :

s_{red}	$s_{lightblue}$	$s_{darkblue}$	$s_{lightblue}$	$s_{lightblue}$	$s_{lightblue}$
s_{black}	$s_{lightblue}$	$s_{lightblue}$	$s_{darkblue}$	$s_{lightblue}$	$s_{darkblue}$
$s_{darkblue}$	s_{black}	s_{black}	$s_{lightblue}$	$s_{darkblue}$	$s_{darkblue}$
s_{black}	s_{black}	$s_{lightblue}$	$s_{darkblue}$	$s_{darkblue}$	$s_{darkblue}$

This sequence will be encoded by progressively shrinking the original interval. The final message is the lower bound of the last interval.

symbol	[proba [	$\lvert I^j \rvert$	interval $I^{j+1} = \inf I^j + \lvert I^j \rvert \cdot proba$
	[, [	1	[0 ,1 [
s_{red}	[0.1 ,0.2 [	0.1	[0.1 ,0.2 [
$s_{lightblue}$	[0.6 ,0.9 [	0.03	[0.16 ,0.19 [
$s_{darkblue}$	[0.2 ,0.6 [	0.012	[0.166 ,0.178 [
$s_{lightblue}$	[0.6 ,0.9 [	0.0036	[0.1732 ,0.1768 [
$s_{lightblue}$	[0.6 ,0.9 [	0.00108	[0.17536 ,0.17644 [
$s_{lightblue}$	[0.6 ,0.9 [	0.000324	[0.176008 ,0.176332 [
s_{black}	[0 ,0.1 [	$3.24\ 10^{-05}$	[0.176008 ,0.1760404 [
$s_{lightblue}$	[0.6 ,0.9 [	$9.72\ 10^{-06}$	[0.17602744 ,0.17603716 [
$s_{lightblue}$	[0.6 ,0.9 [	$2.92\ 10^{-06}$	[0.176033272 ,0.176036188 [
$s_{darkblue}$	[0.2 ,0.6 [	$1.17\ 10^{-06}$	[0.1760338552 ,0.1760350216 [
$s_{lightblue}$	[0.6 ,0.9 [	$3.50\ 10^{-07}$	[0.17603455504 ,0.17603490496 [
$s_{darkblue}$	[0.2 ,0.6 [	$1.40\ 10^{-07}$	[0.176034625024 ,0.176034764992 [
$s_{darkblue}$	[0.2 ,0.6 [	$5.60\ 10^{-08}$	[0.1760346530176 ,0.1760347090048 [
s_{black}	[0 ,0.1 [	$5.60\ 10^{-09}$	[0.1760346530176 ,0.17603465861632 [
s_{black}	[0 ,0.1 [	$5.60\ 10^{-10}$	[0.1760346530176 ,0.176034653577472 [

(b) Algorithms

Decoding algorithm. The decoding procedure is easy to understand once the message $\mathbf{m} \in [0, 1[$ has been completely read. In practise, it is progressively decoded since it is too long to be conveniently represented by a single number in memory, which introduces some extra work referred as renormalisation. First, the probability p_i^j of the source symbols must be known at each step j of the decoding procedure. The initial interval is set to $I^0 = [0, 1[$. Then, at each step j, the interval I^{j-1} is subdivided into parts proportional to the symbol probabilities p_i^j into subintervals sI_i^j as follows :

$$\begin{array}{lcllcll}
sI_1^j & = & [& 0 & , & p_1^j & [\\
sI_2^j & = & [& p_1^j & , & p_1^j + p_2^j & [\\
sI_3^j & = & [& p_1^j + p_2^j & , & p_1^j + p_2^j + p_3^j & [\\
\dots & & & & & & \\
sI_{n-1}^j & = & [& 1 - p_n^j - p_{n-1}^j & , & 1 - p_n^j & [\\
sI_n^j & = & [& 1 - p_n^j & , & 1 & [
\end{array}$$

Then, the message $\mathbf{m}$ belongs to one of the subintervals sI_i^j, corresponding to symbol $\mathbf{s}_i$. This symbol $\mathbf{s}_i$ is added to the decoded message, and the next interval is set to sI_i^j : $I^j = sI_i^j$.

Algorithm 1 aritdecode(*in*,*out*) : decodes stream *in* to *out*

```
1:  $I \leftarrow [0,1[$                                          // initial interval
2:  $in \xrightarrow{+32} \mathbf{m}$                              // reads the first bits of the input
3:  repeat
4:
5:     $(p_i)_{i\in[\![1,n]\!]} \leftarrow$ get_model ()           // retrieves the probabilities
6:     $count \leftarrow p_1$                                      // upper bound of $sI_i^j$
7:     for $i \in [\![2,n]\!]$ do                                  // look the interval containing $\mathbf{m}$
8:        if $\mathbf{m} < count$ then                              // found the subinterval
9:           break                                                 // exits the for loop
10:       end if
11:       $count \leftarrow count + p_i$                            // next i
12:    end for
13:
14:    $\mathbf{s} \leftarrow \mathbf{s}_i$ ; $out \xleftarrow{+\mathbf{s}} \mathbf{s}_i$   // decoded symbol $\mathbf{s}_i$
15:    $I \leftarrow [count - p_{i-1}, count[$                    // updates the current interval
16:
17:    while $\frac{1}{2} \notin I$ or $|I| < \frac{1}{2}$ do      // renormalisation
18:       if $I \subset \left[\frac{1}{2}, 1\right[$ then          // higher half
19:          $I \leftarrow I - \frac{1}{2}$ ; $\mathbf{m} \leftarrow \mathbf{m} - \frac{1}{2}$   // shifts higher half to lower half
20:       else if $I \subset \left[\frac{1}{4}, \frac{3}{4}\right[$ then   // central half
21:          $I \leftarrow I - \frac{1}{4}$ ; $\mathbf{m} \leftarrow \mathbf{m} - \frac{1}{4}$   // shifts central half to lower half
22:       end if
23:       $I \leftarrow 2 \cdot I$                                  // lower half is directly renormalised
24:       $\mathbf{m} \leftarrow 2 \cdot \mathbf{m}$ ; $in \xrightarrow{+1} \mathbf{m}$   // message is shifted by reading in
25:    end while
26:
27: until $\mathbf{s} = stop$                                     // read a stop symbol
```

Renormalisation. Observe that unless the decoder does not stop on its own, as for Huffman coding. The source must have a stop symbol or the decompression must know how to stop the decoder. For large messages, the intervals I^j require more memory to be represented than the usual 2×64 bits offered by computers. Therefore, when an interval I^j is contained in $\left[0, \frac{1}{2}\right[$ or $\left[\frac{1}{2}, 1\right[$, one bit of the message is transmitted, the interval is shifted (scaled by two), and the algorithm goes on. Moreover, the intervals I^j can get arbitrarily small around $\frac{1}{2}$. In order to prevent this, when $I^j \subset \left[\frac{1}{4}, \frac{3}{4}\right[$, two bits of the message are transmitted, the interval is shifted twice (scaled by four). These processes are called *renormalisation*. In parallel, the message does not need to be read entirely, since the only precision needed to decode one symbol is given by the intervals I^j. At each renormalisation, a complement of the message is read to ensure that the precision of the interval matches the precision of the message. This whole process is implemented by Algorithm 1: aritdecode.

Algorithm 2 aritencode(*in*,*out*) : encodes stream *in* to *out*

```
1:  $I \leftarrow [0, 1[$                                                         // initial interval
2:  repeat
3:      $(p_i)_{i \in [\![1,n]\!]} \leftarrow$ get_model ()                      // retrieves the probabilities
4:      $in \xrightarrow{-s} \mathbf{s}_i$                                          // retrieves symbol to encode
5:      $I \leftarrow \left[\sum_{k \in [\![1,i-1]\!]} p_k, \sum_{k \in [\![1,i]\!]} p_k\right[$   // deduce the current interval
6:      while $\frac{1}{2} \notin I$ or $|I| < \frac{1}{2}$ do                      // renormalisation
7:          if $I \subset \left[0, \frac{1}{2}\right[$ then                          // lower half
8:              $out \xleftarrow{+1} 0$                                     // appends a 0 to the coded message
9:          else if $I \subset \left[\frac{1}{2}, 1\right[$ then                     // higher half
10:             $out \xleftarrow{+1} 1$                                     // appends a 1 to the coded message
11:             $I \leftarrow I - \frac{1}{2}$                                   // shifts higher half to lower half
12:         else if $I \subset \left[\frac{1}{4}, \frac{3}{4}\right[$ then          // central half
13:             $out$.repeat_next_bit                      // set out to repeat the next bit to be output
14:             $I \leftarrow I - \frac{1}{4}$                                   // shifts central half to lower half
15:         end if
16:         $I \leftarrow 2 \cdot I$                                                  // scaling
17:     end while
18:
19: until $\mathbf{s} = stop$                                                 // read a stop symbol
```

Encoding algorithm. The encoding procedure is very similar to the decoding one, as details on Algorithm 2: aritencode. Note that in both cases, the finite precision of computer representations forces to use only half of the available bits to represent the probabilities and the intervals, since both have to be multiplied with exact precision. Also, the open intervals are actually represented by closed ones: $[n_i, n_s[= [n_i, n_s - 1[$.

(c) Statistical Modelling

This arithmetic coder provides a powerful engine to encode a universal source. However, it is very sensible to the tuning, which is a very hard task. First, the probability model is very important. Consider a zero–entropy message, i.e. a message with a constant symbol $\mathbf{s}_0$. If the probability model states that $\mathbf{s}_0$ has probability $p_0 \ll 1$, then the encoded stream will have a huge length. Therefore, the arithmetic coder is not close to an entropy coder, unless very well tuned. We will see some generic techniques to improve these aspects.

Adaptive models. A simple solution to the adaptability of the probabilities consists in updating the probability model along the encoding, in a deterministic way. For example, the probability of a symbol can increase each time it is encoded, or the probability of a *stop* symbol can increase at each new symbol encoded, as on the table below. This can be easily implemented through the function get_model() of Algorithm 1: aritdecode and Algorithm 2: aritencode. For the case of the zero–entropy stream, there would be a reasonable amount of encoded stream where p_0 goes closer to 1, and then the stream will be encoded at a better rate. Observe that p_0 cannot reach one, since the probability of each symbol must not vanish, prohibiting an asymptotic zero–length, but respecting the second item of the requirements for the entropy of Section II.1(b) *Information Theory*.

symbol	updated probabilites [						$p_{\mathbf{s}}$ [	$\lvert I^j \rvert$	I^{j+1}
	$\frac{0}{10}$	$\frac{1}{10}$	$\frac{2}{10}$	$\frac{6}{10}$	$\frac{9}{10}$	$\frac{10}{10}$	[, [	1	[0 ,1 [
$\mathbf{s}_{red}$	$\frac{0}{11}$	$\frac{1}{11}$	$\frac{3}{11}$	$\frac{7}{11}$	$\frac{10}{11}$	$\frac{11}{11}$	$[\ \frac{1}{10}, \frac{2}{10}\ [$	0.1	[0.1 ,0.2 [
$\mathbf{s}_{lightblue}$	$\frac{0}{12}$	$\frac{1}{12}$	$\frac{3}{12}$	$\frac{7}{12}$	$\frac{11}{12}$	$\frac{12}{12}$	$[\ \frac{7}{11}, \frac{10}{11}\ [$	0.027272	[0.16363636 ,0.19090909 [
$\mathbf{s}_{darkblue}$	$\frac{0}{13}$	$\frac{1}{13}$	$\frac{3}{13}$	$\frac{8}{13}$	$\frac{12}{13}$	$\frac{13}{13}$	$[\ \frac{3}{12}, \frac{7}{12}\ [$	0.009090	[0.17045454 ,0.17954546 [
$\mathbf{s}_{lightblue}$	$\frac{0}{14}$	$\frac{1}{14}$	$\frac{3}{14}$	$\frac{8}{14}$	$\frac{13}{14}$	$\frac{14}{14}$	$[\ \frac{8}{13}, \frac{12}{13}\ [$	0.002797	[0.17604895 ,0.17884615 [
$\mathbf{s}_{lightblue}$	$\frac{0}{15}$	$\frac{1}{15}$	$\frac{3}{15}$	$\frac{8}{15}$	$\frac{14}{15}$	$\frac{15}{15}$	$[\ \frac{8}{14}, \frac{13}{14}\ [$	0.000999	[0.17764735 ,0.17864635 [
$\mathbf{s}_{lightblue}$	$\frac{0}{16}$	$\frac{1}{16}$	$\frac{3}{16}$	$\frac{8}{16}$	$\frac{15}{16}$	$\frac{16}{16}$	$[\ \frac{8}{15}, \frac{14}{15}\ [$	0.000400	[0.17818015 ,0.17857975 [
$\mathbf{s}_{black}$	$\frac{0}{17}$	$\frac{2}{17}$	$\frac{4}{17}$	$\frac{9}{17}$	$\frac{16}{17}$	$\frac{17}{17}$	$[\ \frac{0}{16}, \frac{1}{16}\ [$	$2.49\ 10^{-5}$	[0.17818015 ,0.17820513 [
$\mathbf{s}_{lightblue}$	$\frac{0}{18}$	$\frac{2}{18}$	$\frac{4}{18}$	$\frac{9}{18}$	$\frac{17}{18}$	$\frac{18}{18}$	$[\ \frac{9}{17}, \frac{16}{17}\ [$	$1.02\ 10^{-5}$	[0.17819337 ,0.17820366 [
$\mathbf{s}_{lightblue}$	$\frac{0}{19}$	$\frac{2}{19}$	$\frac{4}{19}$	$\frac{9}{19}$	$\frac{18}{19}$	$\frac{19}{19}$	$[\ \frac{9}{18}, \frac{17}{18}\ [$	$4.57\ 10^{-6}$	[0.17819851 ,0.17820309 [
$\mathbf{s}_{darkblue}$	$\frac{0}{20}$	$\frac{2}{20}$	$\frac{4}{20}$	$\frac{10}{20}$	$\frac{19}{20}$	$\frac{20}{20}$	$[\ \frac{4}{19}, \frac{9}{19}\ [$	$1.20\ 10^{-6}$	[0.17819948 ,0.17820068 [
$\mathbf{s}_{lightblue}$	$\frac{0}{21}$	$\frac{2}{21}$	$\frac{4}{21}$	$\frac{10}{21}$	$\frac{20}{21}$	$\frac{21}{21}$	$[\ \frac{10}{20}, \frac{19}{20}\ [$	$5.41\ 10^{-7}$	[0.17820008 ,0.17820062 [
$\mathbf{s}_{darkblue}$	$\frac{0}{22}$	$\frac{2}{22}$	$\frac{4}{22}$	$\frac{11}{22}$	$\frac{21}{22}$	$\frac{22}{22}$	$[\ \frac{4}{21}, \frac{10}{21}\ [$	$1.54\ 10^{-7}$	[0.17820018 ,0.17820034 [
$\mathbf{s}_{darkblue}$	$\frac{0}{23}$	$\frac{2}{23}$	$\frac{4}{23}$	$\frac{12}{23}$	$\frac{22}{23}$	$\frac{23}{23}$	$[\ \frac{4}{22}, \frac{11}{22}\ [$	$4.92\ 10^{-8}$	[0.17820021 ,0.17820026 [
$\mathbf{s}_{black}$	$\frac{0}{24}$	$\frac{3}{24}$	$\frac{5}{24}$	$\frac{13}{24}$	$\frac{23}{24}$	$\frac{24}{24}$	$[\ \frac{0}{23}, \frac{2}{23}\ [$	$4.27\ 10^{-9}$	[0.17820021 ,0.17820022 [
$\mathbf{s}_{black}$	$\frac{0}{25}$	$\frac{4}{25}$	$\frac{6}{25}$	$\frac{14}{25}$	$\frac{24}{25}$	$\frac{25}{25}$	$[\ \frac{0}{24}, \frac{3}{24}\ [$	$0.53\ 10^{-9}$	[0.17820021 ,0.17820021 [

Order. This probability model can be enhanced by considering groups of symbols instead of only one. The number of symbols considered jointly is called the *order* of the coder. This is particularly useful for text, where syllables play an important role. An order 0 coder means that the probability model is updated continuously, whereas an order k model will use a different probability model for each combination of the k symbols preceding the encoded one.

Contexts. With this point of view, the arithmetic coder begins with one probability model, and updates it continuously along the encoding process. However, we can actually consider various probability models simultaneously, depending on the *context* of the symbol to encode. For example when coding a text, it is more probable to find a vowel after a consonant. Therefore, the probability of a vowel after another vowel could be reduced to improve the probability model.

Limits. Context modelling and order–based coding allows reducing the interdependence of the symbols (putting the entropy closer to the Kolmogorov complexity [Li and Vitanyi 1997]). This process is the main part of describing the object to encode, but since it is a difficult one, these features can lead to significant improvements of the results. However, the number of contexts and the order must be limited, since for each context the coder builds a probability model through the regular updates, and an exponential number for each order added. This probability needs a reasonable amount of encoded stream to get closer to the real probability model. The encoded stream must be longer than this amount of time for each context.

Prediction. Another way to reduce the covariance relies on *prediction mechanisms*, i.e. deductions that can be equally obtained from the encoder and the decoder. Since we encode the lower part of the interval containing the message, a message ended by a sequence of 0 is cheaper to encode than a message ended with a 1, as on the example of Section II.2(a) *Arithmetic Coder*. Therefore, if the prediction always asserts the results, the message will be a sequence of 0s, with some isolated 1s. This is actually encoded by an arithmetic coder as a run–length encoded stream, since the *stop* characters induce a very tiny last interval. If the stop can be predicted too, then the arithmetic coding spares the last sequence of 0s. In this rough point of view, the better case of arithmetic coding is, for a generic problem, the logarithm of the number of symbols.

II.3 Compression

Coding is only a part of a *compression scheme*. Actually, a compression scheme is composed of various steps of conversions, from the original data to a symbolic representation, from this representation to specifications of sources, from these sources to the encoded message, from this encoded message to a transmission protocol, which entails a re–coding for error detection, and the symmetric parts from the receiver.

This whole process can be designed part by part, or all together. For example, some nice compression scheme already contains error detection using the redundancy of the original data that is left after the encoding. Some lossy or progressive compression schemes perform the encoding directly from the representation and incorporate the specification of sources.

These features optimise compression for specific applications. However, a generic application usually requires a separate design of the parts of a compression scheme. In this context, arithmetic coding turns out to be a very flexible tool to work on the final *compression ratio*, i.e. the ratio of the final size and the original size of the data. Depending on the application, this compression ratio must be reduced to optimize specific characteristics of these applications, leading to different trade–offs. We will now detail three such generic trade–offs.

(a) Compaction

Compaction refers to compact data structures, also called succinct. These structures aim at reducing the size of the memory used during the execution of an application, while maintaining a small execution overhead. This trade–off between memory used and execution time must also allow a random access to the compact data structure. For example for mesh data structures, this trade–off can be simply a elegant data representation with no specific encoding such as [Rossignac *et al.* 2001, Lage *et al.* 2005*1, Lage *et al.* 2005]. It can also involve simple encoding scheme that are fast to interpret as [Houston *et al.* 2005], or involve a precise mixture of very efficient encoding with a higher–level data structure, such as [Castelli *et al.* 2005*1, Castelli *et al.* 2005*2].

(b) Direct Compression

The most used meaning of compression refers to file compression or to compression of exchanged information. Most of the common generic algorithms are based on the LZH (ZIP) algorithm of [Lempel and Ziv 1977], aside from specific image and video compression algorithms such as JPEG [Wallace 1991, Christopoulos *et al.* 2000] and MPEG [le Gall 1991, Pereira and Ebrahimi 2002]. In this case, the goal is to optimise the trade–off between compression rate and compression time: the enumeration method is usually too slow, while a simple coding of the data representation can be in general improved with a minor time overhead. The trade–off can also take into account the amount of memory necessary to compress or decompress the stream. In that case, the process is usually performed *out–of–core*, such as [Isenburg and Gumhold 2003].

(c) Progressive Compression

The compression can also lead to a loss of information when decompressing. This can be useful either when the lost part is not significant, or when it can be recovered by a further step of the compression. In that second sense, lossy compression will generate objects at various levels of detail, i.e. in *multi-resolution*. Each resolution can be compressed separately by the difference from the previous one. A variant of that scheme does not distinguish between levels of details, sending a *coarse level* by direct compression, and refining it by a sequence of local changes. In these contexts, the goal is to optimise the trade–off between compression ratio and the *distortion* of the decompressed model. For the geometrical models, the distortion is usually measured by the geometric distance between the decoded model and the original one.

III
Meshes and Geometry

Geometrical objects are usually represented through meshes. Especially for surfaces in the space, triangulations had the advantage for rendering of representing with a single element (a triangle) many pixels on screen, which reduced the number of elements to store. Although the increasing size of usual meshes reduced this advantage, graphic hardware and algorithms are optimised for these representations and meshes are still predominant over point sets models. Moreover, several parts of the alternative to meshes require local mesh generation, which becomes very costly in higher dimensions. Finally, meshes describe in a unique and explicit manner the support of the geometrical object, either by piecewise interpolation or by local parameterisation such as splines or NURBS.

To a real object correspond several meshes. These meshes represent the same geometry and topology, and thus differ by their *connectivity*. The way these objects are discretised usually depends on the application, varying from visualisation to animation and finite element methods. These variations make it harder to define the geometric quality of a mesh independently of the application, even with a common definition for the connectivity.

This chapter details the combinatorial part, introducing the definition and properties of meshes in Section III.1 *Simplicial Complexes and Polytopes*, and a complete set of operations on the connectivity in Section III.2 *Combinatorial Operators*. Finally, Section III.3 *Geometry and Discretisation* gives some classical examples of interactions of geometry and meshes.

In this work, we do not use a specific data structure for meshes. We will consider the operations described in this chapter as the basic elements of a generic data structure. For further readings, the classical data structures for surfaces are the winged–edge [Baumgart 1972], the split–edge [Eastman 1982], the quad–edge [Guibas and Stolfi 1985], the half–edge [Mäntylä 1988] and the corner–table [Rossignac *et al.* 2001, Lage *et al.* 2005]. For non–manifold 2–meshes, we would mention the radial–edge [Weiler 1985] and [de Floriani and Hui 2003]. Further references on the following definitions can be found in [Munkres 1984, Boissonnat and Yvinec 1998, Hatcher 2002].

III.1 Simplicial Complexes and Polytopes

There are various kind of meshes used in Computer Graphics, Scientific Visualisation, Geometric Modelling and Geometry Processing. However, the graphic hardware is optimised for processing triangles, line segments and points, which are all special cases of *simplices*. We will therefore focus mainly on meshes made of simplices, called *simplicial complex*, and one of its extensions to meshes made of convex elements, which we will refer as *polytopes*. This notion can be further extended to *cell complexes* [Hatcher 2002], but these are only used for high–level modelling and we will not use them in this work.

(a) Simplicial Complexes

Simplex. A simplex is an n–dimensional analogue of a triangle. More precisely, a simplex σ^n of dimension n, or n–simplex for short, is the open convex hull of $n+1$ points $\{v_0, \dots, v_n\}$ in general position in some Euclidean space $\mathbb{R}^d$ of dimension n or higher, i.e., such that no m–plane contains more than $(m+1)$ points. The closed simplex $\bar{\sigma^n}$ is the closed convex hull of $\{v_0, \dots, v_n\}$. The points v_i are called the *vertices* of σ^n. For example, a 0–

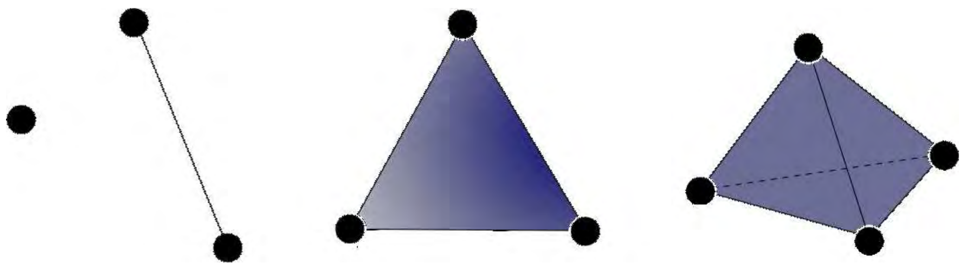
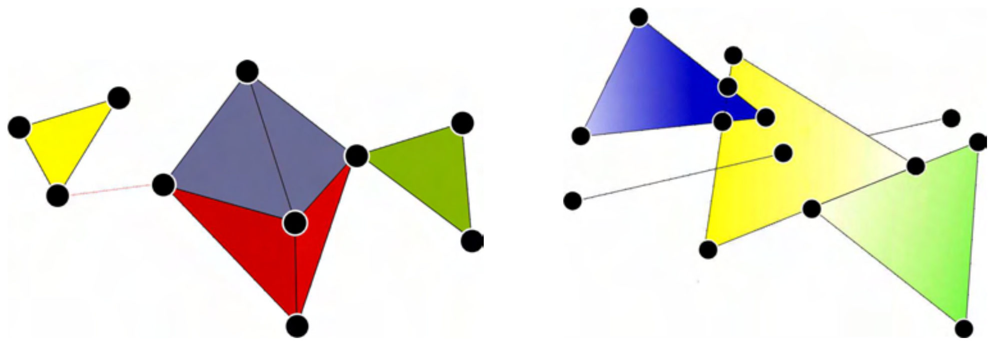


Figure III.1: Simplices from dimension 0 to 3.

simplex is a point, a 1–simplex is a line segment, a 2–simplex is a *triangle*, a 3–simplex is a *tetrahedron*, and a 4–simplex is a *pentachoron*, as shown on Figure III.1.

Incidence. The open convex hull of any $m < n$ vertices of σ^n is also a simplex τ^m, called an m*–face* of σ^n. We will say that σ^n is *incident* to τ^m, and denote $\sigma^n \succ \tau^m$. The 0—faces are called the *vertices*, and the 1–faces are called the *edges*. The *frontier* of a simplex σ, denoted by $\partial\sigma$, is the collection of all of its faces.



(a) Simplicial complex. **(b) Non–complex.**

Figure III.2: Simplicial complex and a set of simplices not being a complex.

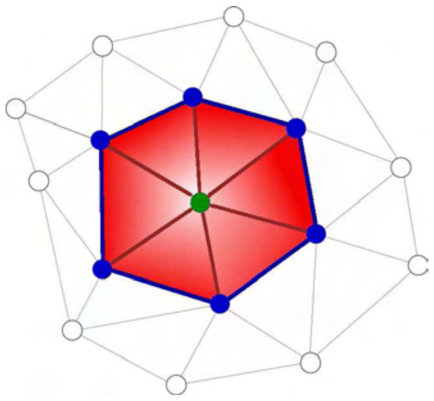
Complex. A *simplicial complex* K of $\mathbb{R}^d$ is a coherent collection of simplices of $\mathbb{R}^d$, where coherent means that K contains all the faces of each simplex ($\forall \sigma \in K, \partial\sigma \subset K$), and contains also the geometrical intersection of the closure of any two simplices ($\forall\,(\sigma_1, \sigma_2) \in K^2, \bar{\sigma_1} \cap \bar{\sigma_2} \subset K$), as illustrated on Figure III.2. Two simplices incident to a common simplex are said to be *adjacent*. The *geometry* of a complex usually refers to the coordinates of its vertices, while its *connectivity* refers to the incidence of higher–dimensional simplices on these vertices.

Skeleton. If a collection K' of simplices of K is a simplicial complex, then it is called a *subcomplex* of K. The subcomplex $K_{(m)}$ of all the p–simplices, $p \leqslant m$, is called the *m–skeleton* of K.

Connected components. A complex is *connected* if it cannot be represented as a union of two non–empty disjoint subcomplexes. A *component* of a complex K is a maximal connected subcomplex of K.

(b) Local Structure

Consider a simplex σ of a complex K. The local neighbourhood of σ is described by its *star* [Alexander 1930].


(a) The star of a vertex is the union of the open star (in red) with the ling (in blue).

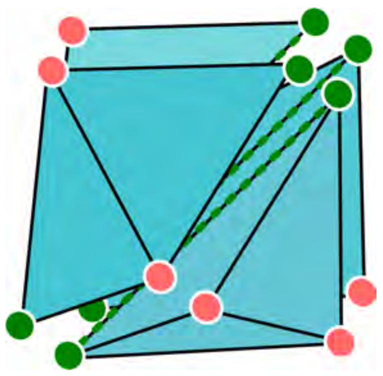

(b) The star of an edge.

Figure III.3: Vertex star in a 2–complex and edge star in a 3–complex.

Link. The *join* $\sigma \star \tau$ of two disjoint simplices σ and τ is the simplex that is the open convex hull of $\sigma \cup \tau$. The *link* of a simplex $\sigma \in K$ is the set of simplices whose join with σ belongs to K: $\operatorname{lk}(\sigma) = \{\tau \in K : \bar{\sigma} \cap \bar{\tau} = \emptyset, \sigma \star \tau \in K\}$.

Star. The *open star* of σ is then the join of σ with its link: $\dot{\operatorname{st}}(\sigma) = \{\sigma \star \tau, \tau \in \operatorname{lk}(\sigma)\}$. Finally, the *star* of σ is the union of all the simplices of the open star together with all their faces: $\operatorname{st}(\sigma) = \dot{\operatorname{st}}(\sigma) \;\cup\; \bigcup_{\rho \in \dot{\operatorname{st}}(\sigma)} \partial\rho$. The *valence* of a vertex is the number of maximal faces in its star.

(c) Pure Simplicial Complexes

Dimension. The *dimension* n of a simplicial complex K is the maximal dimension of its simplices, and we will say that K is an n–complex. A *maximal face* of a simplicial complex of dimension n is an n–simplex of K.

Euler–Poincaré characteristic. Denoting $\#_m(K)$ the number of m–simplices in K, the *Euler–Poincaré characteristic* $\chi(K^n)$ of an n–complex K^n is a topological invariant [Hatcher 2002] defined by $\chi(K^n) = \sum_{m\in\mathbb{N}} (-1)^m \#_m(K^n)$.

Pure complexes. Roughly speaking, a complex is pure if all the visible simplices have the same dimension. More precisely, a simplicial complex K^n of dimension n is *pure* when each p–simplex of K, $p < n$, is face of another simplex of K.

Boundary. The *boundary* ∂K of a pure simplicial complex K^n is the closure of the set, eventually empty, of its $(n-1)$–simplices that are face of only one n–simplex: $\partial K^n = \{\sigma^{n-1} : \#\,\mathrm{lk}(\sigma^{n-1}) = 1\}$. The simplices of the boundary of K and their faces are called *boundary* simplices, and the other simplices are called *interior* simplices.

(d) Simplicial Manifolds

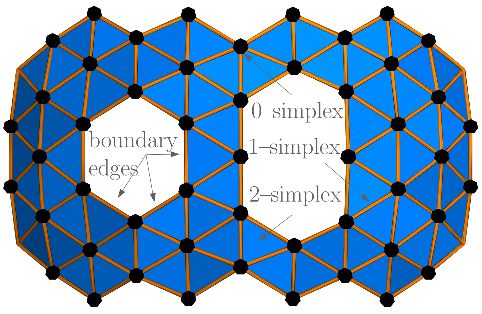


Figure III.4: A surface with two bounding curves

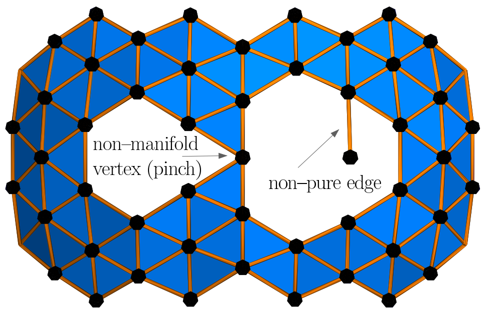


Figure III.5: A non–pure 2–complex with a non–manifold vertex.

Manifolds. A simplicial n–manifold $\mathcal{M}^n$ is a pure simplicial complex of dimension n where the open star of each interior vertex is homeomorphic to an open n–ball $\mathbb{B}^n$ and the open star of each bounding vertex is homeomorphic to the intersection of $\mathbb{B}^n$ with an closed half–space. This implies that each $(n-1)$–simplex of $\mathcal{M}$ is the face of either one or two simplices. In particular, the boundary of an n–manifold is a $(n-1)$–manifold with an empty boundary.

Orientability. An orientation on a simplex is an ordering $(v_0, \ldots, v_n)$ on its vertices. Two orientations are equivalent if they differ by an even permutation. There are therefore two opposite orientations on a simplex. A simplicial manifold $\mathcal{M}^n$ is *orientable* when it is possible to choose a coherent orientation on all its simplices. More precisely, if $\sigma^{n\text{–}1} = (v_1, \ldots, v_n)$ is an oriented interior $(n\text{–}1)$–simplex of $\mathcal{M}^n$, face of $\rho = \sigma^{n\text{–}1}\star v$ and $\rho' = \sigma^{n\text{–}1}\star v'$, then the orientation of ρ and ρ' is coherent the orientation of ρ is equivalent to $(v, v_1, \ldots, v_n)$ and the orientation of ρ' is opposed to $(v', v_1, \ldots, v_n)$. This orientation thus defines the notion of *next* and *previous* vertex inside a triangle of a simplex.

Surfaces. For example, a 2–manifold is a surface, i.e. a simplicial complex made of only vertices, edges and triangles where each edge is in the frontier of either one or two triangles and where the boundary does not pinch. For example, Figure III.4 shows an example of 2–manifold and Figure III.5 illustrates a 2–complex that is neither pure nor a manifold. The topology of surfaces can be easily defined from its orientability and its Euler–Poincaré characteristic, using the Surface classification theorem [Armstrong 1979]: Any oriented connected surface $\mathcal{S}$ is homeomorphic to either the sphere $\mathbb{S}^2$ ($\mathfrak{g}(\mathcal{S}) = 0$) or a connected sum of $\mathfrak{g}(\mathcal{S}) > 0$ tori, in both cases with some finite number $\mathfrak{b}(\mathcal{S})$ of open disks removed. The number $\mathfrak{g}(\mathcal{S})$ is called the *genus* of $\mathcal{S}$, and $\mathfrak{b}(\mathcal{S})$ its *number of boundaries.* The Euler–Poincaré characteristic $\chi(\mathcal{S})$ of $\mathcal{S}$ is equal to $\chi(\mathcal{S}) = \#_2(\mathcal{S}) - \#_1(\mathcal{S}) + \#_0(\mathcal{S}) = 2 - 2 \cdot \mathfrak{g}(\mathcal{S}) - \mathfrak{b}(\mathcal{S})$.

(e) Polytopes

Surfaces in finite element methods are usually represented by a mixture of triangular and quadrangular elements. Although this do not directly fits to the simplicial complexes we just introduced, this structure can be easily extended to that case. For example, one could divide each quadrangle into two coplanar triangles and get a simplicial complex. We define a *polytope* in a similar way.

Along this work, a polytope in $\mathbb{R}^d$ will be a coherent collection of convex open set of $\mathbb{R}^d$, called *cells*, where coherent means again that the collection contains the frontiers and the intersections of its cells. Observe that this implies that each cell is made up with piecewise linear elements, from edges to its maximal faces.

The definition and properties described above are still valid, in particular the notion of boundary, manifold, orientability, and the classification for 2–dimensional manifold polytopes. Moreover, polytopes are useful to define the *dual* of a manifold: The dual of an n–manifold $\mathcal{M}^d$ is the manifold polytope obtained by reversing the incidence relations of its cells, i.e. creating a vertex for each n–cell of $\mathcal{M}^d$, and an m–cell for each $(n-m)$–cell of $\mathcal{M}^d$, spanning the vertices created for each n–cell of its star in $\mathcal{M}^d$.

III.2 Combinatorial Operators

The encoding of meshes describes, in a compact way, how to build the encoded mesh. The decoding operation thus performs a sequence of combinatorial operations on an initial empty or canonically defined mesh together with a reconstruction of its geometry. These combinatorial operations are of two kinds: purely constructive ones and their inverse, namely the Euler and Handle operators, which only increase the number of simplices (or cells) without modifying existing ones; and subdivision ones with their inverse, and Stellar operators is a complete set for this category. The Handle operators are sometimes considered as a special case of Euler operators, since these operations are similar but they alter the topology of the mesh.

We will now describe each of these operators, first because it is a complete set of operators for mesh, and second because each of the three general methods we will introduce in this work is based on one of these operators: the Edgebreaker algorithm [Rossignac 1999] of Chapter IV **Connectivity–Driven Compression** uses the specificities of Handle operators together with Euler operators. The Simplicial Level Set Compression [Lewiner *et al.* 2004*2, Lewiner *et al.* 2004*4] of Chapter V **Level Set Compression** uses intensively Stellar theory, while the geometry–driven compression scheme [Lewiner *et al.* 2005*1] of Chapter VI **GEncode** uses only the basics of Euler and Handle operators. Once again, we do not rely on a specific implementation of these operations, and we will refer to [Velho *et al.* 2003] for a simple, complete and powerful one.

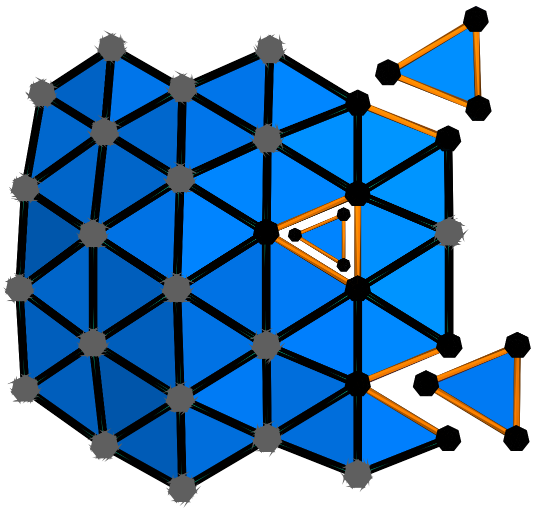


Figure III.6: 3 ways of attaching a simplex to a triangulation. Centre: MSG, Down: MTE, Up: MEV + MTE.

(a) Euler Operators

Generic Euler operators for surfaces. Euler operators were originally defined as operators on surfaces that do not change its manifoldness nor its

Euler characteristic [Mäntylä 1988]. In this restricted definition, there was 5 creation and 5 destruction operators, some illustrated on Figure III.6, namely:

M/K EV	adds/removes an edge and a vertex
M/K TE	adds/removes a triangle and an edge
M/K STV	adds/removes an initial triangle, without edge
M/K SG	closes/opens a bounding curve
M/K EKL	adds/removes an edge, joining two bounding curves

We will now extend these definitions, in order to first distinguish between topology–preserving operators (M/K EV and M/K TE) and Handle operators that will be introduced next, and second to extend these definitions to any dimension. All the definitions of this section also work for polytopes.

Low–level Euler operators. With this distinction, basic Euler operators are reduced to *simplicial collapse* and *simplicial expansion* [Hatcher 2002]. Given a simplicial complex K and σ^m a simplex of K face of only one $(m+1)$–simplex ρ^{m+1}, we say that K collapses to $K' = K \backslash \{\sigma^m, \rho^{m+1}\}$, and that K' expands to K. Observe that since we add or remove pairs of simplices of consecutive dimensions, we do not change the Euler characteristic of K. For $m = 0$, the expansion corresponds to MEV, and the collapse to KEV. For $m = 1$, the expansion corresponds to MTE, and the collapse to KTE.

Euler attachment. The simplicial expansion operator works at low–level, and in particular for pure complexes or manifolds, these operators must be used in group to preserve the purity of the mesh as on the top of Figure III.6. The *simplex attachment* operations compose the minimal group of Euler operators that add a unique maximal face, preserving the purity. For 1–complexes, a simplex attachment of order 1 is a MEV operation. For 2–complexes, a simplex attachment of order 2 is a MTE operation, eventually preceded by a simplex attachment of order 1. In general, a simplex attachment for n–complexes can be described as a simplicial expansion with $m = n-1$, eventually preceded of at most $n-1$ simplex attachments of order $n-1$.

Manifold Euler operators. The simplicial expansion preserves the topology of the mesh (actually this define its simple homotopy type), and combined into simplex attachment operations, it preserves the purity of the mesh. In order to preserve the manifoldness of the mesh, we need to restrict the simplex attachment operations. From the property of manifolds, that each $(n-1)$–simplex is the face of either one or two simplices, we need to restrict these simplex attachment to the boundary of the manifold. Since Euler attachments preserve the topology of the mesh, this is actually sufficient: the manifold Euler operators are attachments involving only bounding simplices.

(b) Handle Operators

Generic attachment. We distinguished inside the generic Euler operators for surfaces between those who preserve and those who change the topology of the simplicial complex. The Euler attachment resulted in adding a maximal face using only low–level Euler operators, and thus preserving the topology. The generalization of this notion, the *generic attachment*, also adds a simplex and its faces, with the only restriction of preserving the properties of a simplicial complex. More precisely, an m–simplex σ^m can be attached to an n–complex K^n by identifying some of the faces of σ^m with some of the simplices of K. In order to preserve the simplicial complex property, we impose that if a face τ of σ^m is identified with a simplex τ' of K^n, then the faces of τ are identified with the faces of τ' in a one–to–one correspondence.
Such operation can alter the topology of the complex, and its manifoldness. In particular, observe that the first step of a mesh construction, i.e. creating the first simplex, is a generic attachment onto an empty complex.

Manifold Handle operators. In order to preserve the manifoldness, we must first restrict the attachment to a maximal face, and identify part of its frontier with the boundary of the manifold, as we did previously. However, this is not enough, since a generic attachment can create a pinch on the manifold. A sequence of Euler attachments can then fatten this pinch in order and thus recover the manifoldness. Therefore, we will define a Handle operator on $\mathcal{M}^n$ as a generic attachment involving only the boundary of $\mathcal{M}^n$ of an n–simplex followed by at most $n-1$ Euler attachments of n–simplices. This operation is also referred as *Handle attachment* [Lopes 1996, Lopes and Tavares 1997].

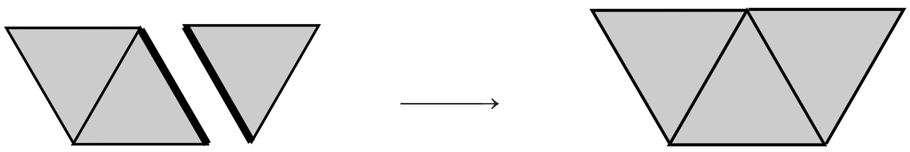


Figure III.7: $\chi + 0$: Euler attachment.

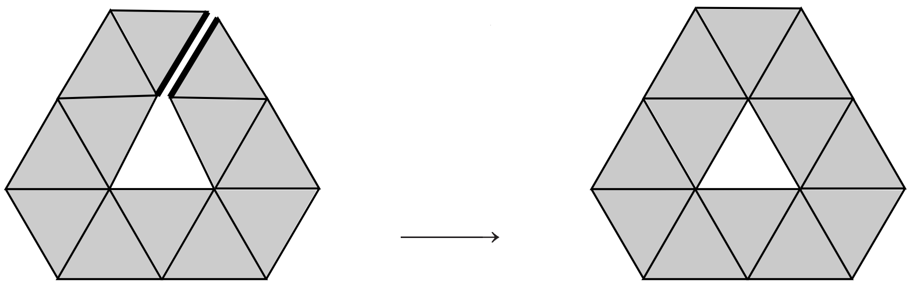


Figure III.8: $\mathfrak{b} + 1$, $\chi - 1$: lower 1–handle operator.

Handle operators for surfaces. These manifold Handle operators can be exhaustively described, and especially for curves and orientable manifold, they

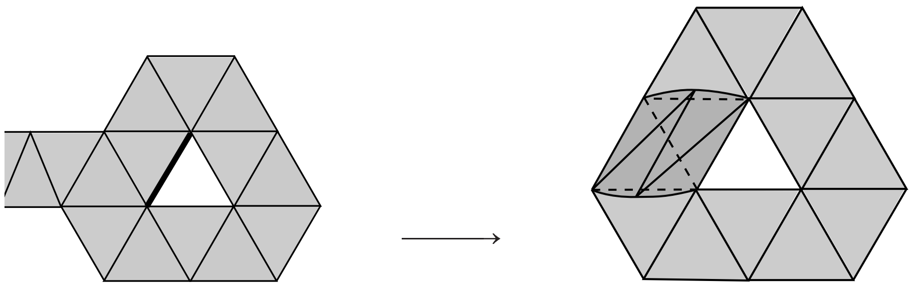


Figure III.9: $\mathfrak{g}+1, \mathfrak{b}-1$, $\chi-1$: upper 1–handle operator.

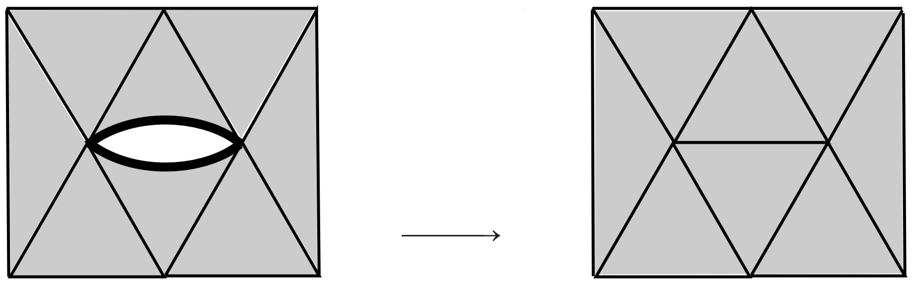


Figure III.10: $\mathfrak{b}-1$, $\chi+1$: 2–handle operator.

characterise exactly the topological change they induce. For 1–complexes, there are four constructive operators: creating a first vertex (generic attachment), adding an edge and a vertex (MEV), adding an edge between two vertices of a boundary (Handle operator) and adding an edge between an interior vertex and another vertex (Non–manifold generic attachment). For surfaces, there are seven surface constructive operations, with their inverse, destructive operators [Lopes 1996, Lopes and Tavares 1997]:

$\chi+1$	creates a new connected component with initial triangle, with its three edges and vertices (Handle operator: 0–handle)
$\chi+0$	completes two consecutive bounding edges with a triangle (MTE, Figure III.6)
$\chi+0$	glues a triangle on a bounding edge (Euler attachment, Figure III.7)
$\chi-1$	glues two bounding edges of distinct connected components (Handle operator: 1–handle)
$\mathfrak{b}+1$	glues two non–consecutive edges of the same bounding curve with two triangles, splitting this curve into two bounding curves (Handle operator: lower 1–handle, Figure III.8)
$\mathfrak{g}+1, \mathfrak{b}-1$	glues two edges of different bounding curves with two triangles, creating a genus (Handle operator: upper 1–handle, Figure III.9)
$\mathfrak{b}-1$	closes a three–edges bounding curves with a triangle (Handle operator: 2–handle, Figure III.10)

(c) Stellar Operators

Stellar theory. Stellar theory studies the combinatorial equivalences between simplicial complexes. It was developed by [Newman 1926] and [Alexander 1930], and more recently consolidated by [Pachner 1991] and [Lickorish 1999]. This theory states another class of operators for editing the combinatorial structure of a manifold with a minimal set of operators while preserving its topology. As opposed to Euler and Handle operators, *stellar operators* intend to modify, as opposed to constructing meshes. The stellar operators were first stated in [Alexander 1930] as *stellar moves*, and then reformulated in [Pachner 1991] as *bistellar moves*. Both cases can be decomposed on edges with the *vertex weld* and *edge split* operations.

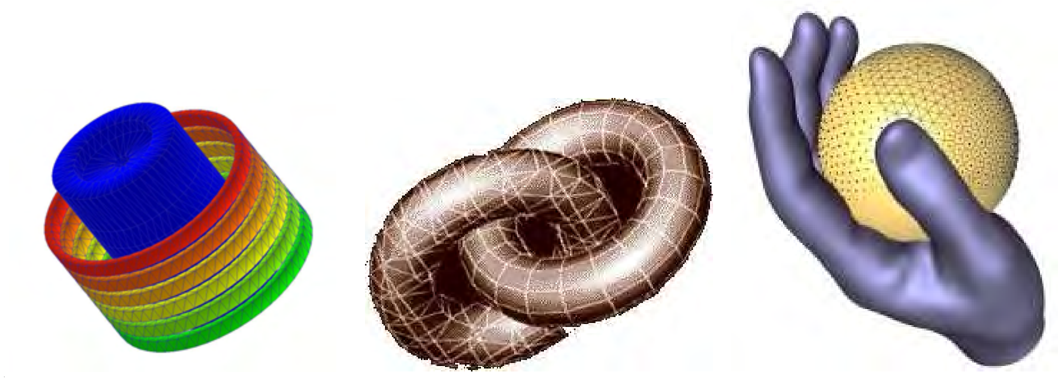
Stellar moves. A stellar move of order m on a simplicial complex K consists of replacing the star of an m–simplex σ^m by the join of a vertex with the link of σ^m: $K \rightarrow K \backslash \mathrm{st}\,(\sigma) \cup v \star \mathrm{lk}\,(\sigma)$. These moves are a powerful operation, as stated in Alexander's theorem [Newman 1926, Glaser 1970, Theorem II.17]: Any two simplicial complexes are piecewise–linear homeomorphic if and only if they are related by a finite sequence of stellar moves.

Bistellar moves. Bistellar moves are somehow more concise than stellar moves, and can be defined as follows. Consider an m–simplex σ^m of an simplicial complex K^n of dimension n, and an $(n-m)$–simplex τ^{n-m} not included in K^n, such that $\mathrm{lk}\,(\sigma^m) = \partial\tau^{n-m}$. Then the complex $K^n \setminus \sigma^m \star \partial\tau^{n-m} \cup \partial\sigma^m \star \tau^{n-m}$ is said to be obtained from K^n by a bistellar move of order m. Note that the inverse of a bistellar move of order m is a bistellar move of order $(n-m)$, and that, for manifolds, a bistellar move does not change the connectivity of the boundary. [Pachner 1991] proved a similar theorem as Alexander's one: two simplicial manifolds with an empty boundary are equivalent if and only if they are related by a finite sequence of bistellar moves.

Weld and split. Actually, Stellar theory states that stellar moves on edges are sufficient to map any two equivalent combinatorial manifolds [Alexander 1930]. This can be proved by decomposing the bistellar moves by stellar moves on edges. The stellar move on edge is called an *edge split*, and the inverse operation, which removes the inserted vertex, is called a *vertex weld*. Therefore, stellar operations can be used as primitives for multiresolution operators. In particular, an edge flip [Hoppe *et al.* 1993] can be decomposed as an edge split followed by a vertex weld. Moreover, the classical edge collapse operator [Hoppe 1996] can be decomposed into a sequence of edge flips, followed by one vertex weld, as shown in [Velho 2001].

III.3 Geometry and Discretisation

The above definition of meshes contains three order of information about a geometrical object: its inner properties, which do not change with large scale deformations and are referred as its topology; its combinatorial representation described by the incidences of its simplices; and its shape, represented by the coordinates of the vertices of the mesh, and interpolated onto its higher order simplices. The relations between the geometry and the topology are very



(a) A finite element volumetric mesh. **(b) A paramrtric and an implicit torus.** **(c) A remeshed hand with a subdivided sphere.**

Figure III.11: Geometrical models generated with different methods.

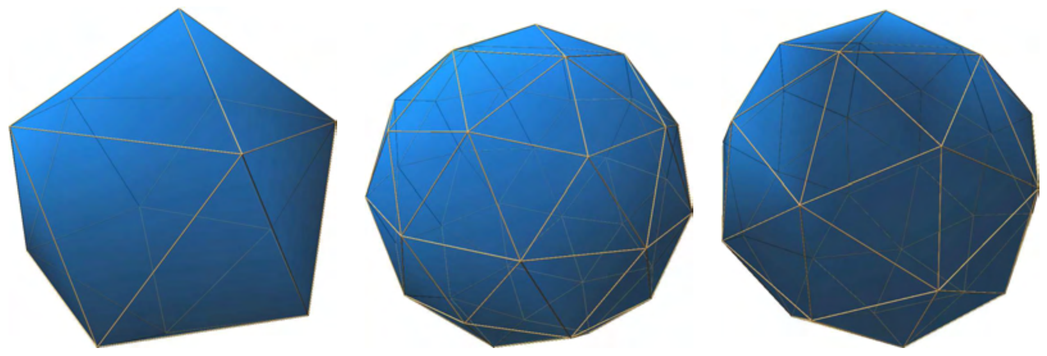
clear, although the connection of geometry and connectivity is still loose. Many different discretisations are commonly used, as illustrated on Figure III.11. There is no standard mesh representation even for simple geometrical models such as a sphere in $\mathbb{R}^3$ [Katanforoush and Shahshahani 2003].

On the way from meshes to geometry, the geometry of an object is usually interpolated or approximated by a mesh using an iterative process of subdivision [Catmull and Clark 1978, Doo and Sabin 1978] and simplification [Garland and Heckbert 1997]. On the reverse way, there are several approaches to define a mesh from a geometrical object, the most famous one being the *Delaunay triangulation* [Boissonnat and Yvinec 1998]. This process is either referred as discretisation if the whole geometrical object is known, or as reconstruction in the other case. Finally, the geometry of an object can be extracted from a mesh, usually by intersection of a standard mesh with a geometrical function, such as for constructive solid geometry models [Rossignac 1986].

In this section, we will give a quick overview on these three elements, detailing only the last one, as we will use it in Chapter V **Level Set Compression**. However, we will not try to define what would be a reliable geometry representation, which restrictions the geometry could impose to the mesh or other definitions of mesh quality. This task is far beyond the scope of this work, although the wide variety of answers is the main motivation for designing the specific compression methods that will be introduced in this work.

(a) Subdivision and Smoothing

Subdivision. The subdivision process was originally defined to state equivalences in piecewise linear topology [Munkres 1984, Hatcher 2002]. In that context, the basic combinatorial operation was the *barycentric subdivision* [Munkres 1984], that inserts the barycentre of each simplex, and thus corresponds to a sequence of stellar moves. In order to refine also the geometry of the surface, the old and new vertices can be shift [Zorin and Schröder 2000]. When iterating the subdivision process, we can obtain a continuous geometrical object from an initial discrete mesh. For example, each of the following subdivision operators generates a different smooth limit.



(a) Original icosahedron. (b) One step of [Loop 1987]. (c) One step of [Kobbelt 2000].

Figure III.12: Example of surface subdivision methods.

Surface subdivision. For surfaces, [Catmull and Clark 1978] first detailed vertex shifts in order to obtain an almost smooth surface after an infinite number of iterations. A dual scheme was introduced at the same time in [Doo and Sabin 1978], which behaves well with noises. The barycentric subdivision creates many simplices with high degrees. The valence can be regularised by inserting only the barycentre's of the edges, as done in [Loop 1987]. Then, the number of new simplices created at each step can be reduced by performing an intermediate dual step [Kobbelt 2000].
Each of these improvements enhanced the smoothness of the result, until the order 5 in [Velho and Zorin 2001], using stellar operations. Actually, we know from the previous section that all these subdivisions can be expressed with stellar operations and vertex shifts, and this provides a unified framework for all the subdivision operations [Velho 2003] which extends in any dimension.

Smoothing. The reverse of the subdivision operation is called simplification [Hoppe *et al.* 1993, Garland and Heckbert 1997, Vieira *et al.* 2004]. Although it could be defined using reverse stellar operators for each of the subdivision above, the geometrical information is difficult to recover: the subdivision filters the mesh. A vertex shift similar to the subdivision one can also be performed without changing the connectivity. The smoothing is then performed with a constant complexity, as a mimic of image filtering [Taubin 1995].

These filters are very useful for practical cases, especially to enhance meshes resulting from noisy acquisition [Fleishman *et al.* 2003] or lossy transmission [Desbrun *et al.* 1999, Taubin 1999], as in Chapter V **Level Set Compression**.

(b) Delaunay Triangulation

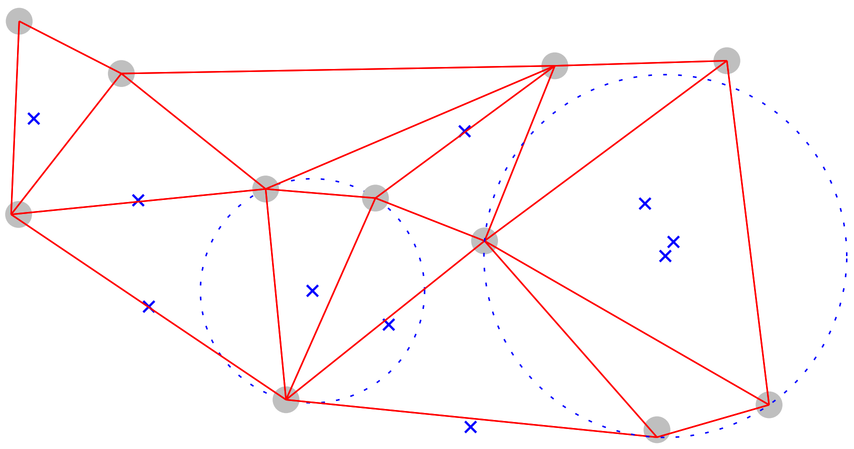


Figure III.13: A Delaunay triangulation, with some of the empty circumcircles attached to each triangle drawn.

Voronoi diagrams. Given a finite set of points $(\mathbf{x}_i)_{i\in[\![0,k]\!]}$ and a distance function $d\,(,)$ on an ambient space $\mathbb{X}$, the most natural way to discretise $\mathbb{X}$ is to draw a curved polytope on it, called the *Voronoi diagram* or the *Dirichlet tessellation* of $(\mathbf{x}_i)_{i\in[\![0,k]\!]}$, where each maximal cell σ_i contains all the points of $\mathbb{X}$ closer to $\mathbf{x}_i$ than to any other $\mathbf{x}_j$: $\sigma_i = \{\mathbf{y} \in \mathbb{X} : \forall j \neq i, d\,(\mathbf{y}, \mathbf{x}_i) < d\,(\mathbf{y}, \mathbf{x}_j)\}$. For the Euclidean distance $d\,(\mathbf{x}, \mathbf{y}) = \|\mathbf{y} - \mathbf{x}\|$, this polytope is straight and convex. The planar version of this polytope goes back to Descartes and Dirichlet, but the general definition was introduced in [Voronoi 1908].

Delaunay tessellation. The Voronoi diagram has many nice properties, but it does not present the original points $\mathbf{x}_i$ as its vertices, which is the usual paradigm for the geometry interpolation. This can be achieved by considering the dual of the Voronoi diagram, in the sense of Section III.1(e) *Polytopes*, where the vertices corresponding to each maximal cell σ_i is shift to the original point $\mathbf{x}_i$. For the Euclidean distance and for the original points in general positions, this dual is a simplicial complex, called the *Delaunay tessellation* of $(\mathbf{x}_i)_{i\in[\![0,k]\!]}$ [Delone 1934]. It can be efficiently computed with randomised algorithms [Boissonnat and Yvinec 1998]. From this point of view, it is the most natural definition of a mesh from the geometry of a set of points, which will be useful for reconstruction applications.

Properties. The Delaunay triangulation K^d of $(\mathbf{x}_i)_{i\in[\![0,k]\!]}$ in $\mathbb{R}^d$ reflects the definition of Voronoi diagrams by the fact that the open circumsphere $\mathbb{S}^{d-1}$

of any maximal simplex is always empty. Moreover, the union of all compact simplices in the triangulation is the convex hull of the original points. For the Euclidean distance, Delaunay triangulations maximise the minimum internal angle of its simplices [Boissonnat and Yvinec 1998]. These properties are useful for packing problems and for differential approximations, since the asymptotic accuracy of derivative estimators is usually related to the angle of the elements of the mesh. Moreover, it relates directly to sampling conditions on the geometry of smooth and non–smooth objects [Boissonnat and Oudot 2005].

Delaunay–based reconstruction. The above properties make Delaunay triangulations a very significant choice for reconstruction algorithms, from slice interpolation [Boissonnat and Geiger 1993, Lopes *et al.* 2000, Nonato *et al.* 2005] to spatial restriction [Chaine and Bouakaz 2000, Boissonnat and Cazals 2002] and greedy traversal [Bernardini *et al.* 1999, Cohen–Steiner and Da 2002] inside the Delaunay triangulation and further more complex methods [Amenta and Kullori 2002]. These methods deduce a mesh from a set of points in a deterministic manner, which can be used as a basis for decompressing a mesh knowing its vertices.

Ball pivoting algorithm. In Chapter VI **GEncode**, we will prefer greedy approaches in order to control the decompression on the fly. Particularly, the Ball Pivoting algorithm of [Bernardini *et al.* 1999] will serve as the basic example. This algorithm creates an initial triangle, a priori the triangle with the smallest possible circumradius for the given point set, and then recursively attach the triangle with the smallest possible circumradius to the boundary of the mesh being reconstructed. The original algorithm put a threshold on the circumradius of the inserted triangle, in order to allow reconstructing open surfaces. It can be directly implemented with the Handle operators of Section III.2(b) *Handle Operators*, as described in [Medeiros *et al.* 2003, Medeiros *et al.* 2004]. The reconstructed surface is then a subcomplex of the volumetric Delaunay triangulation.

(c) Multi–Triangulations

Order of triangulations. In theory, a geometrical model can be approximated by a sequence of subdivisions on an original, *coarse mesh*. In practise, the size of the mesh must remain reasonable to be handled by a computer, but this sequence of subdivisions still generates a refinement of the resolution of the geometry. We will formalise the fact that a complex K' is obtained by subdivision from K by a hierarchical relation: $K \preccurlyeq K'$.
Actually, these subdivisions can be adapted to refine only a part of a mesh, for example by stellar operations in a specified neighbourhood [Velho 2004] or by inserting points in the Delaunay triangulation [Devillers 2002]. Depending on the neighbourhood chosen, these subdivisions will generate different meshes, and the order $\preccurlyeq$ is thus only a partial order. The hierarchy generated by $\preccurlyeq$ is called a *multi–triangulation* [de Floriani *et al.* 1997, Puppo 1998].

We will look more precisely at one of these multi–triangulations, introduced in [Mello *et al.* 2003] and specified for tetrahedral meshes in [Pascucci 2002].

Binary space partition. On one side, these multi–triangulations actually define generic rules multiresolution data structure of the space they span. On the other side, canonical multiresolution data structures are already widely used, such as the quad–tree [Finkel and Bentley 1974], its generalisation as kd–trees [Friedman *et al.* 1977] and *binary space partitions* [Bentley 1975] or BSP for short. Most of these domain subdivision structures, and in particular the octrees and kd–trees, can be implemented as a BSP with no overhead, which puts this structure as a powerful framework. The abstract definition of multi–triangulation can be combined with the power and simplicity of binary space partitions into *binary multi–triangulations* [Mello *et al.* 2003], or BMT for short.

Binary multi–triangulations. A binary space partition (BSP) is a progressive partition of an original space by a sequence of refinements. Each refinement cut a part of the partition in two subparts. Therefore, such a BSP can be represented by a binary tree with one node per part created along the sequence. The binary multi–triangulation (BMT) is then a specific BSP where each part is a triangle and each refinement is an edge split, as defined in Section III.2(c) *Stellar Operators*. This BMT adapts more nicely than the classical octree or kd–tree for image decomposition, since it provides k times more intermediate levels without extra space used and since it does not require extra operation to remain polytope.

Local refinement and local simplification. The BMT can be seen as a binary tree or as a sequence of simplicial complexes, and the original space corresponds to the coarse mesh. We will thus call the *level* of a maximal face its depth in the binary tree. The BMT is *locally refined* ($\succcurlyeq$) or *locally simplified* ($\preccurlyeq$) by walking up and down the binary tree, creating a hierarchy of triangulations at different *resolutions*. Each refinement operation splits an edge by inserting a vertex w and creating some simplices σ_i. For each of these σ_i, vertex w will be called the *lowest vertex*, and the facet of σ_i not containing w will be called its *higher facet*. Therefore, the lowest vertex characterises the level of a maximal face. Figure III.14 shows a BMT whose root is a

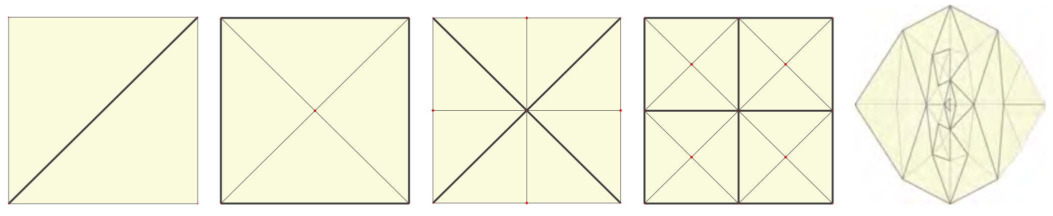
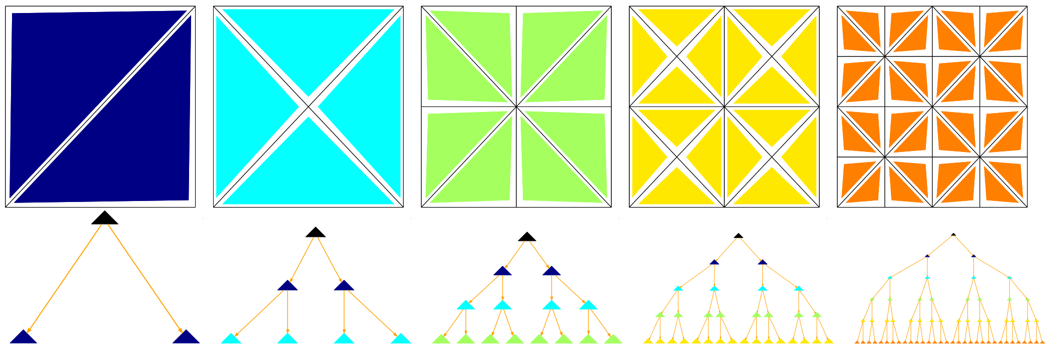

(a) K^1 **(b)** K^2 **(c)** K^3 **(d)** K^4 **(e) Irregular geometry.**

Figure III.14: BMT example: the subdivision edges are marked in bold.

the two triangles of the square (K^1) and whose leaves uniformly covers a grid (K^5). The bold edges are the subdivision edges, and the bold vertices are the simplification vertices. The first triangulation on this figure is the minimum triangulation of a square K^1. The second picture shows the K^2 triangulation that is obtained by the subdivision of the diagonal edge of K^1. The third example of triangulation is the result of four subdivisions applied to the bounding edges of K^2, thus it corresponds to K^3. The fourth triangulation example is K^4 that is obtained from K^3 by subdividing all of its four diagonal edges. The last triangulation on figure Figure III.14, shows a BMT with an irregular geometry. Although we will mainly use regularly spaced vertices, the BMT structure can handle more general configurations. The pictures on Figure III.15 illustrate, the corresponding binary tree representations for the triangulations K^1, K^2, K^3, K^4 and K^5 of the BMT example shown in Figure III.14.



(f) K^1 **(g)** K^2 **(h)** K^3 **(i)** K^4 **(j)** K^5

Figure III.15: The BMT example of Figure III.14 can be represented as a binary tree.

Regular binary multi–triangulation. The adaptability of the BMT relies on local refinements and simplifications. It can be enhanced by maintaining dependencies between adjacent simplices, in particular, by forcing gradual size transitions along each resolution of the triangulation. This means that adjacent maximal faces of a resolution will differ in level by at most 1. With this restriction, the resulting structure is an example of a restricted hierarchy, as defined in [von Herzen and Barr 1987], and we will called it a *regular binary multi–triangulation*, or RBMT for short. Observe that this regularity criterion is purely combinatorial, which allows efficient implementation especially for high dimensions. For example, all the BMT of Figure III.14 are regular.

Non–local refinement and simplification. In order to maintain this regularity, a refinement must propagate to adjacent simplices: a regular refinement of a RBMT is composed of a sequence of local refinements. The RBMT is then a subset of the underlying multi–triangulation, and the order $\rightsquigarrow$ of a regular multi–triangulation is a restriction of the order $\preccurlyeq$ of the multi–triangulation. The regularity restriction is easy to implement in practise (see Algorithm 3: simplify). Simplifying a maximal face σ removes its lower vertex by a stellar move on it. In order to maintain the regularity of the resolution, we

Algorithm 3 simplify(σ): non–local simplification of σ

```
1: $w \leftarrow \sigma$.lowest_vertex          // used to check the level compatibility
2: for all $\sigma_i$ adjacent to $\sigma$ do          // look for simplices of lower level
3:     if $\sigma_i$.lowest_vertex $\neq w$ then          // $\sigma_i$ must be simplified before $\sigma$
4:         simplify($\sigma_i$)          // simplifies $\sigma_i$
5:     end if
6: end for
7: weld($w$)          // locally simplify $\sigma$
```

must check that the adjacent maximal faces σ_i have at most the same level as σ. This can be checked through the lowest vertex w of σ. First, if w is not a vertex of σ_i, then the level of σ_i is greater than the one of σ, and the regularity will be maintained by the simplification. Second, if w is the lowest vertex of σ_i, then σ and σ_i have the same level, and the removal of w will simplify both. Last, if w is a vertex of σ_i, but not the lowest, then σ_i have a lower level than σ and must be simplified first, and here begins the propagation.

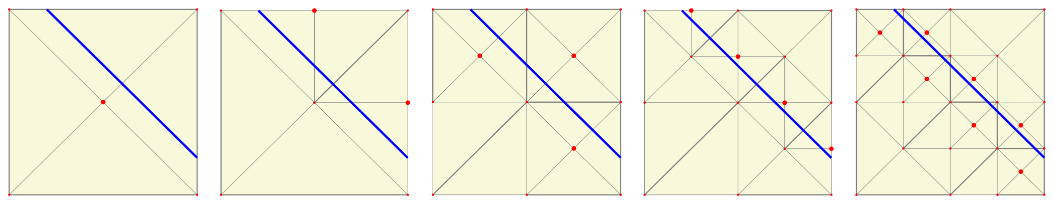


Figure III.16: RBMT adapted to the bold line: at each level, every triangle crossing the bold line is refined, but subdivisions in the upper–right part of the square propagate to the lower–left part.

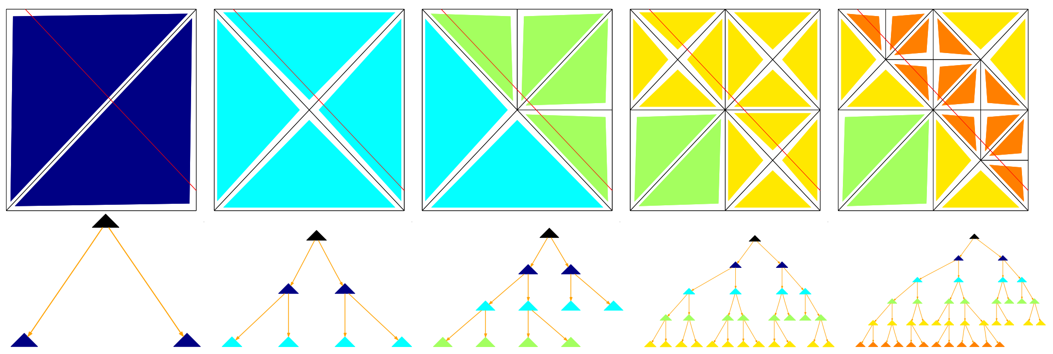


Figure III.17: The BMT example of Figure III.16 can be represented as a binary tree.

For example, Figure III.16 illustrates a sequence of refinements adapting the triangulation to the bold line. In order to preserve gradual transition between resolution levels, local refinements around the bold line propagates inside the triangulation and, in this example, far away from the bold line, as what happened to the bottom left part. The corresponding binary tree is represented on Figure III.17.

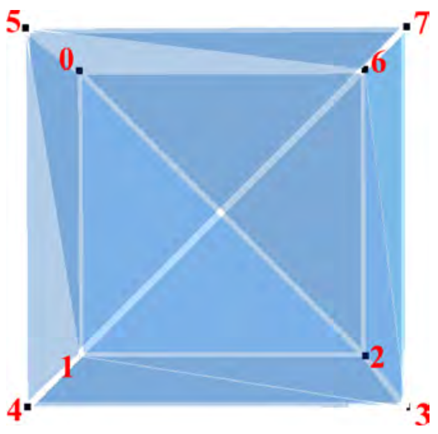
IV Connectivity–Driven Compression

This work aims at compressing geometrical objects represented by meshes. Since there is still no strong relation between the geometry and the connectivity of these meshes for the usual objects considered by graphics applications, dedicated compression schemes consider either that the common information can be deduced from either the connectivity or the geometry. The first option assumes that the star of a simplex has a simple geometry, which can be well approximated by simple methods such as linear interpolation. Then, the geometry can be efficiently encoded by a connectivity traversal of the mesh, leading to *connectivity–driven* compression schemes. The second option predicts the connectivity from the geometry, and will be referred as *geometry–driven* compression schemes. In that case, the connectivity is usually better compressed, but it needs efficient coding of the geometry.

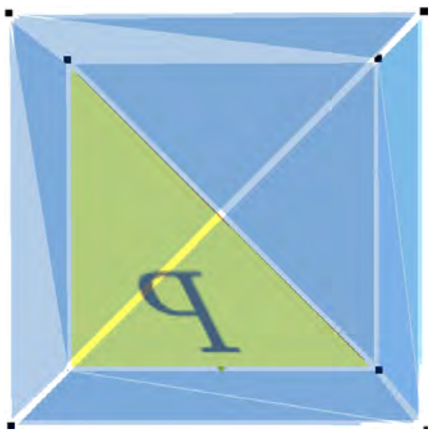
In this chapter, we will focus on the connectivity part of the compression, in order to clarify the position of geometry–driven compression of the next chapters. These connectivity–driven methods improved so much in the last decade that the compression ratio for usual surface connectivity turns around 2/3 bits per vertex. We will give a general framework for handling the critical elements of the connectivity: the topological singularities. These singularities are well understood for surfaces through the Handle operators of Section III.2(b) *Handle Operators*. We will then focus on the Edgebreaker scheme, and introduce two new improvements: the handling of boundary, as a consequence of this framework for singularities, and a small improvement of the decompression algorithm. We will conclude this chapter with compression ratios of the Edgebreaker on usual models with our improvements, and we will detail the specificities of connectivity–driven compression scheme. The goal of this chapter is to state what connectivity–driven compression means, with the detailed example of the Edgebreaker (Figure IV.1 and Figure IV.2), and to show where these algorithms are well suited.

IV.1 Principles

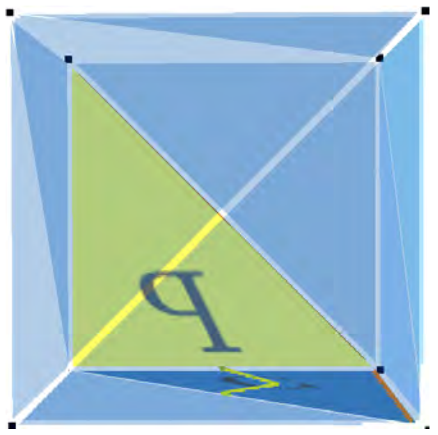
Connectivity–driven compression schemes rely on a traversal of the mesh in order to visit each vertex, and to identify it on further visits. This way, the geometry of the vertex needs to be transmitted only once, and the traversal encodes the connectivity of the mesh. This general framework suits particularly well for manifold polytopes. Most of the existing compression techniques are dedicated to surfaces, and we will focus on these algorithms. Further extensions to non–manifold cases are described in [Guéziec *et al.* 1998],



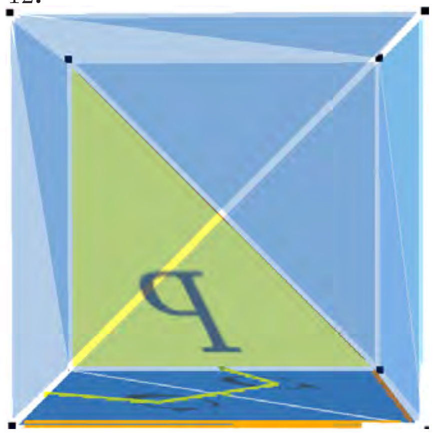
(a) Vertex labels used in the next sequence.



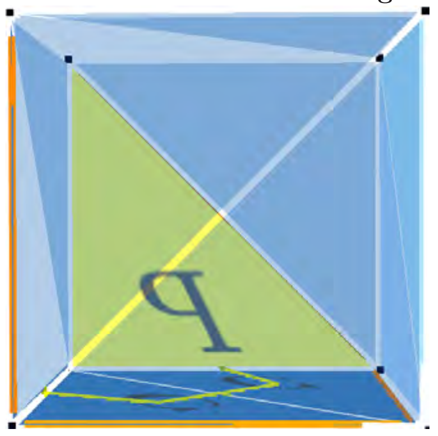
(b) First triangle not encoded: P, vertices 0, 1, 2 are marked. It will be the root of the dual tree. The traversal starts from edge 12.



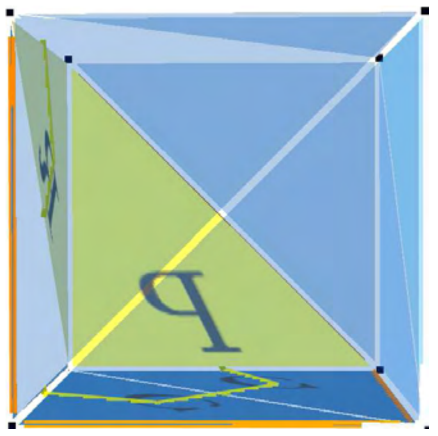
(c) Since vertex 3 is unmarked, 132 is created and 3 is marked: C. This extends the dual tree and the primal remainder. The traversal continues on the right.



(d) Similarly, since vertex 4 is unmarked, 143 is created and 4 is marked: C.

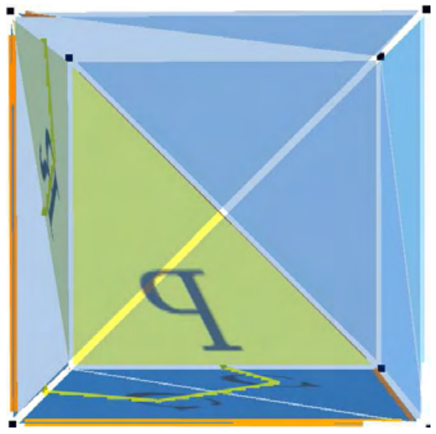


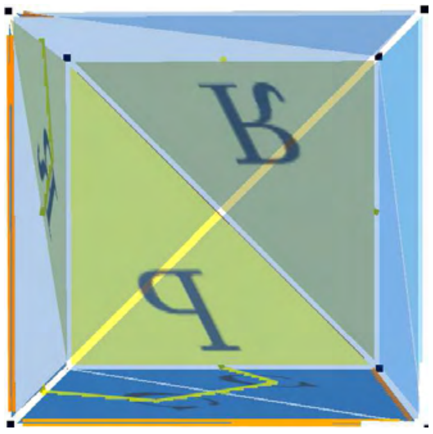
(e) Again, vertex 5 marked: C

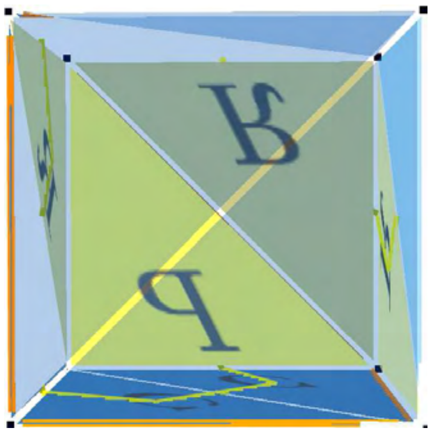


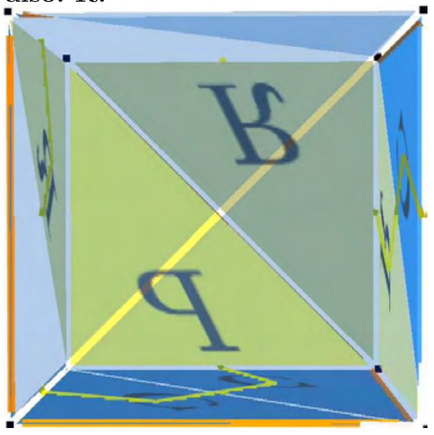
(f) Since vertex 0 is marked and the right triangle is marked already, 105 is attached and the traversal continues on the left: R. This extends only the dual tree.

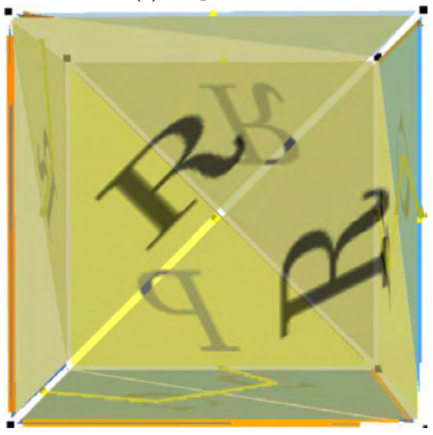
Figure IV.1: Edgebreaker compression of a triangulated cube.


(g) C again: vertex 6 is marked.


(h) Again, vertex 2 is already marked and the right triangle also: R.


(i) Again: R.


(j) C again: vertex 7 is marked.


(k) Again, vertices 4 and then 5 are already marked, with their right triangles also: RR.

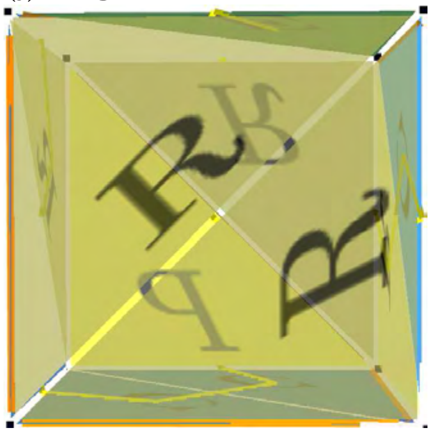
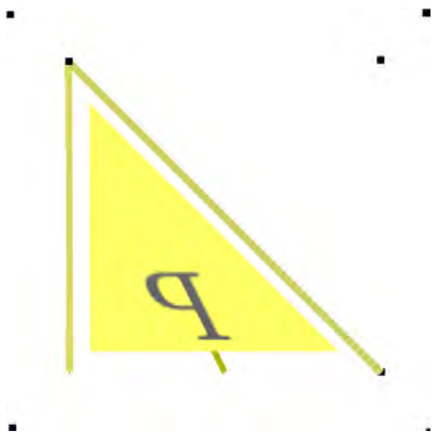
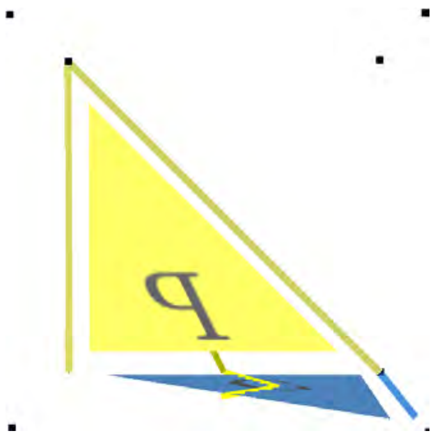
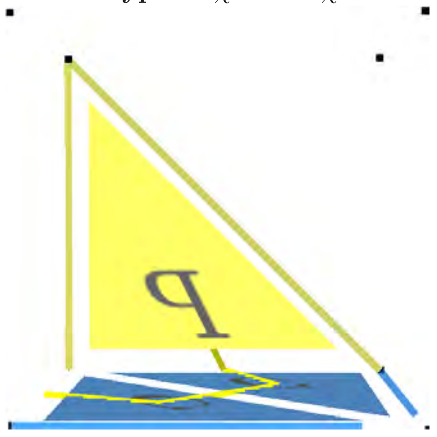
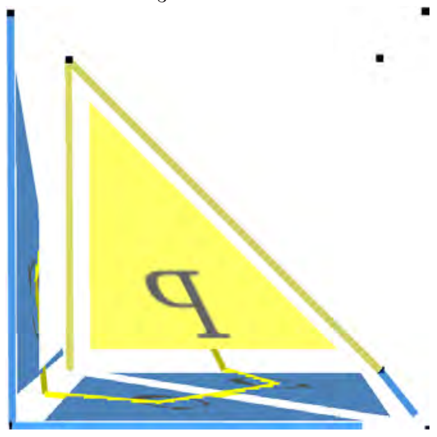

(l) Since vertex 6 is marked, and both the right and left triangles are marked, attach 567: E. This extends the dual tree only.

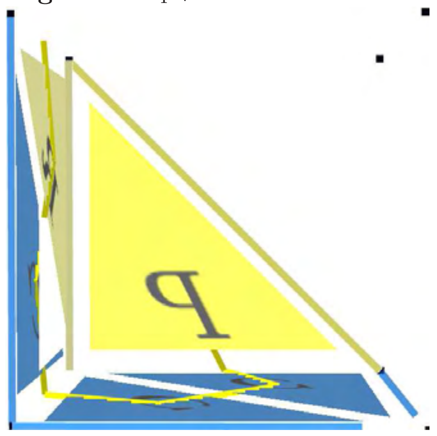
Figure IV.1: **Edgebreaker** compression of a triangulated cube (continued).


(a) Decode P: create the first vertex. This is a Handle operator of type 0: $\chi = 0 \rightarrow \chi = 1$


(b) Decode C: create a new triangle. This is an Euler attachment: $\mathsf{MEV}_3 + \mathsf{MTE}$.


(c) Decode C: create a new triangle: $\mathsf{MEV}_4 + \mathsf{MTE}$.


(d) Decode C: create a new triangle: $\mathsf{MEV}_5 + \mathsf{MTE}$.


(e) Decode R: attach one triangle: **MTE**. The new edge will be identified later by the **Zip** procedure.

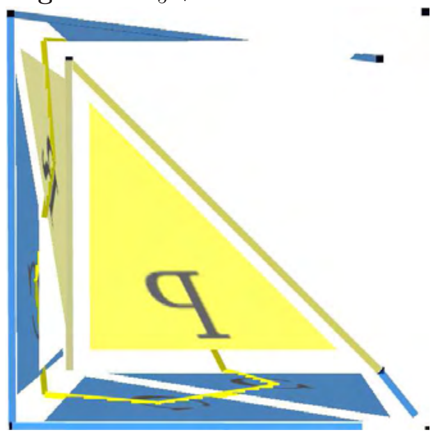
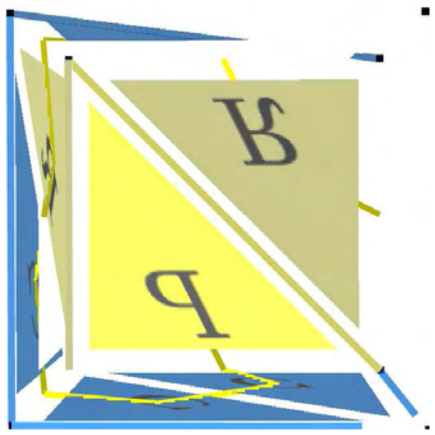
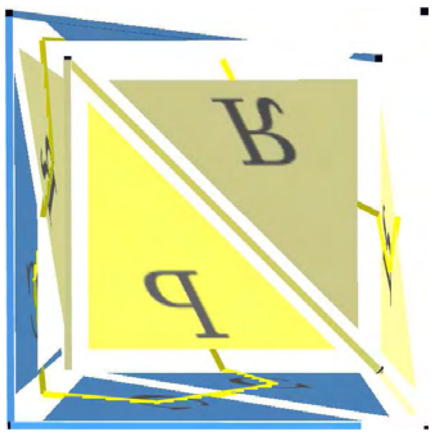
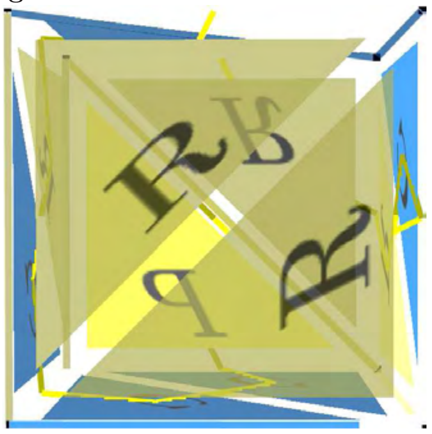
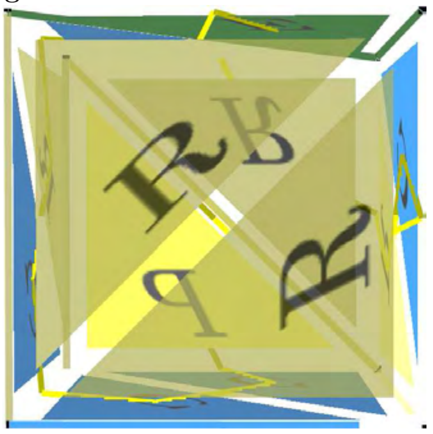

(f) Decode C: create a new triangle: $\mathsf{MEV}_6 + \mathsf{MTE}$.

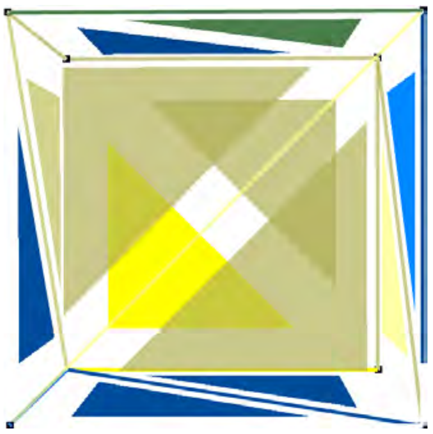
Figure IV.2: **Wrap&Zip** decompression of a triangulated cube.


(g) Decode R: attach one triangle: **MTE**.


(h) Decode R: attach one triangle: **MTE**.


(i) Decode CRR as above.


(j) Decode E: close one triangle. This is a Handle operator of type 2: $\mathfrak{b} = 1 \rightarrow \mathfrak{b} = 0$. The two new edges will be identified by the **Zip** procedure.


(k) The above **Wrap** procedure already decoded the adjacencies of the traversal: this is the dual tree.

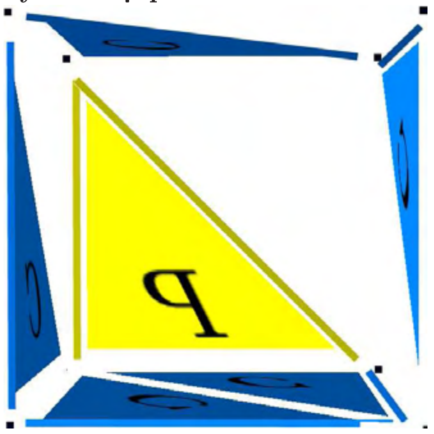

(l) The **Zip** procedure will then identify the edges of the primal remainder, matching edges created by a C with the others.

Figure IV.2: **Wrap&Zip** decompression of a triangulated cube (continued).

while simple extensions of the most common schemes exist for solid models in [Szymczak and Rossignac 2000, Isenburg and Alliez 2002].

(a) Origins

Connectivity–driven compression begun with cache problems in graphic cards: the rough way of transmitting triangle meshes from the main memory to the graphic card is (still) to send the three vertices of the triangle, represented by their three floating–point coordinates. Each triangle is then encoded with 96 bits! [Deering 1995] proposed to represent these triangle meshes by generalised strips in order to share one or two vertices with the last triangle transmitted, reducing by at least a half the memory required previously. This mechanism uses also a small prediction scheme to optimise caching.

Then, these strips were generalised by a topological surgery approach in [Taubin and Rossignac 1998, Taubin *et al.* 1998]. These works introduced the most general framework for connectivity–driven compression, and has been efficiently derived into the Edgebreaker [Rossignac 1999], and with a more flexible way into the valence coding of [Touma and Gotsman 1998, Alliez and Desbrun 2001]. The Edgebreaker has been extended to handle larger categories of surfaces in [Lopes *et al.* 2002, Lewiner *et al.* 2004], while valence coding has been tuned using the geometry in [Alliez and Desbrun 2001], discrete geometry [Kälberer *et al.* 2005]. In addition, the generated traversal of valence coding can be cleaned using [Castelli and Devillers 2004].

With these improvements, the connectivity of usual models can be compressed with less than 3 bits per vertex. Geometry became the most expensive part, which can be reduced using prediction [Touma and Gotsman 1998, Cohen–Or *et al.* 2001] and high–quality quantisation [Sorkine *et al.* 2003, Gumhold and Amjoun 2004]. However, we will not focus here on the compression of the geometry.

(b) Primal or dual remainders

Primal and dual graphs. The main advance of topological surgery [Taubin and Rossignac 1998] was to substitute mesh connectivity compression by graph encoding. A graph can be considered as a simplicial complex of dimension 1. Therefore, the 1–skeleton $K_{(1)}$ of any simplicial complex is a graph, called the *primal graph* of the manifold. Moreover for manifolds, we defined the dual manifold in Section III.1(e) *Polytopes*, and the 1–skeleton of this dual manifold is also a graph, called the *dual graph* of the manifold. For example, Figure IV.3 represents the primal and the dual graph of a triangulated sphere.

Tree encoding. For simplicial surfaces, the dual graph has a very nice property: each node of the graph has three incident links. Encoding the connectivity thus resumes to encoding this dual graph. In order to encode the geometry, this graph must be encoded by traversal, i.e. a spanning forest. Since each connected component can be encoded separately, we will consider

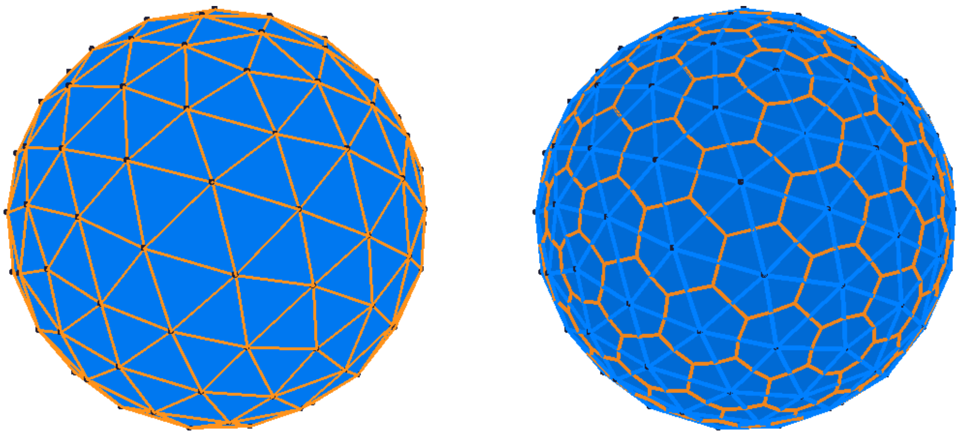


Figure IV.3: (left): the primal graph and (right): the dual graph of a triangulated sphere.

only connected orientable surfaces, and the spanning forest is, in that case, a tree. This tree can be encoded from its root by enumerating for each node how many sons he has. This is the principle of both the valence coding and the Edgebreaker algorithms. The first one encodes the mesh by enumerating the valence of each node of a spanning tree in the primal graph, while the second one encodes a little more than the valence of each node of a spanning tree in the dual graph. In this last case, the valence is either 1, 2 or 3 since the nodes of the dual graph have a constant valence, which simplifies the coding.

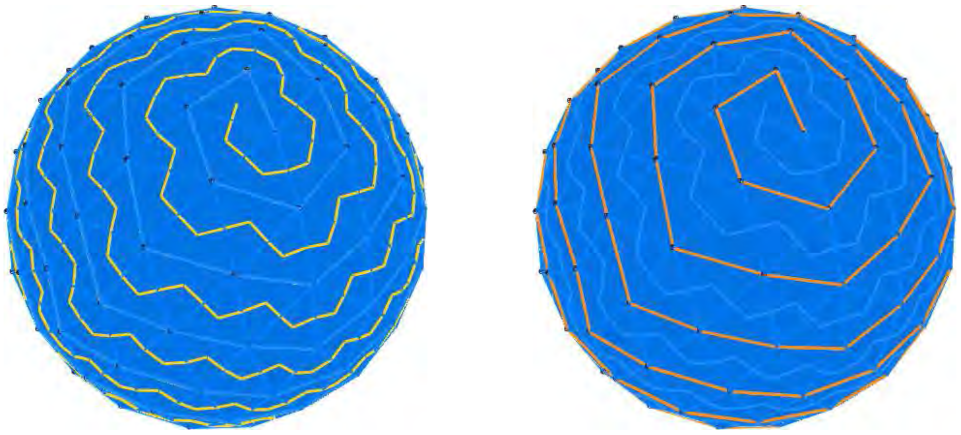


Figure IV.4: (left): a dual spanning tree $\mathcal{S}^{21}$ extracted from the dual graph of Figure IV.3(right). (right): the primal remainder $\mathcal{S}^{01}$ of $\mathcal{S}^{21}$, which is a subgraph of the primal graph of Figure IV.3(left).

Remainders. For clarity of the presentation, we will focus on spanning tree of the dual graph and the primal remainder, which is the focus of the Edgebreaker. What follows can be read identically by considering spanning tree of the primal graph and the dual remainder, which is the point of view of the valence coding. Consider a surface $\mathcal{S}$, with a spanning tree $\mathcal{S}^{21}$ of its dual graph. Observe that the links of $\mathcal{S}^{21}$ correspond to edges of $\mathcal{S}$. Then, consider the primal graph $\mathcal{S}^{1}$ (1–skeleton) of $\mathcal{S}$. Its links also correspond to edges of $\mathcal{S}$. The graph $\mathcal{S}^{01}$ having the same nodes as $\mathcal{S}^{1}$ and the links of $\mathcal{S}^{1}$

not represented in the dual spanning tree $\mathcal{S}^{21}$ is called the *primal remainder* of $\mathcal{S}^{21}$. This remainder is what is left to encode after the traversal of the dual mesh, i.e. $\mathcal{S}^{21}$, has been encoded. For example, the Edgebreaker encodes this primal remainder by specific symbols for the valence 1 and 2 of the dual tree. Moreover, this primal remainder contains all the vertices of the mesh, and will therefore be used to drive the encoding of the geometry.

(c) Topological Singularities

Topology of the remainders. If the remainder is a tree, then it can be easily encoded. The original Edgebreaker works directly in that case. However, this is not always the case, and the topology of the primal remainder actually characterises the topology of the (orientable) surface. For the dual remainder used by the valence coding, there is a detail to assert when the surface has a non–empty boundary. This process relies on a very simple calculus of the Euler characteristic of the remainder. According to Section III.1(c) *Pure Simplicial Complexes*, the Euler characteristic of a surface is given by $\chi(\mathcal{S}) = \#_2 - \#_1 + \#_0$, and according to the surface classification theorem introduces in Section III.1(d) *Simplicial Manifolds*, $\chi(\mathcal{S}) = 2 - 2 \cdot \mathfrak{g}(\mathcal{S}) - \mathfrak{b}(\mathcal{S})$. Since $\mathcal{S}^{21}$ is a tree with exactly one node for each of the $\#_2$ faces, it has $\#_2 - 1$ links. Therefore, the Euler characteristic of the primal remainder $\mathcal{S}^{01}$ is $\chi(\mathcal{S}^{01}) = \chi(\mathcal{S}) - \chi(\mathcal{S}^{21}) = 1 - 2 \cdot \mathfrak{g}(\mathcal{S}) - \mathfrak{b}(\mathcal{S})$. We get the same result for the case of a dual remainder.

Remainder of topological spheres. If the surface $\mathcal{S}$ is a topological sphere, then $\mathfrak{g}(\mathcal{S}) = \mathfrak{b}(\mathcal{S}) = 0$, and the remainders have Euler characteristic 1. From the Jordan curve theorem [Armstrong 1979], the remainders are connected, since they cannot be disconnected by the corresponding spanning tree, which has no closed curve. Then, the remainder is a connected graph with Euler characteristic 1: it is a tree. This primal remainder will be easy to encode, relating topological simplicity to easy compression with connectivity–driven schemes.

Morse edges. For a generic remainder, its Euler characteristic is $1 - 2 \cdot \mathfrak{g}(\mathcal{S}) - \mathfrak{b}(\mathcal{S})$. In the case of a dual spanning tree, the primal remainder is always connected. However, for primal spanning trees on surfaces with a non–empty boundary, the dual remainder can be disconnected. This can be avoided if the primal spanning tree contains all the bounding edges of the surface, except one per boundary components to keep it as a tree. With this restriction, the remainder is a connected graph with exactly $2 \cdot \mathfrak{g}(\mathcal{S}) + \mathfrak{b}(\mathcal{S})$ independent cycles, where a cycle is a sequence of distinct adjacent links whose last one is adjacent to the first one, and where independent means that removing one link of a cycle does not break any other. For each cycle, one edge that would break it will be called a *Morse edge*, since it induces a change in the topology of the surface, and corresponds to a Handle operator introduced in [Lopes and Tavares 1997] and Section III.2(b) *Handle Operators*. Any connectivity–driven compression scheme designed for topological spheres can be extended to any orientable surface by encoding separately these Morse edges. For example, in the case of

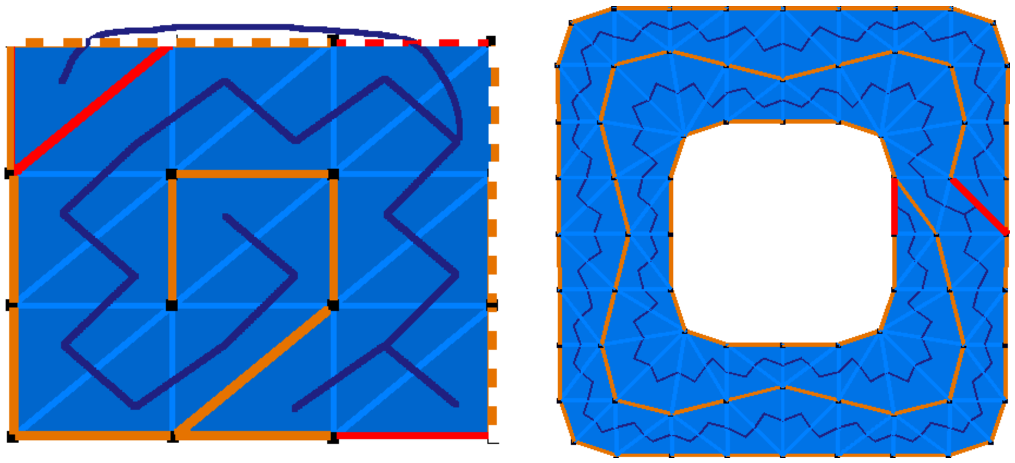


Figure IV.5: (left): a primal remainder on a torus (genus 1): the topmost and bottommost horizontal edges are identified, and so do the leftmost and rightmost ones. (right) a primal remainder on an annulus (two bounding curves).

a sphere, the primal remainder is a tree, as shown on Figure IV.4. For a mesh with genus one or with two boundary curves, the primal remainder is a graph with two cycles, as shown on Figure IV.5.

IV.2 The Edgebreaker example

The Edgebreaker scheme has been enhanced and adapted from the Topological Surgery [Taubin and Rossignac 1998] to yield an efficient but initially restricted algorithm [Rossignac 1999], which encodes the connectivity of any simplicial surface homeomorphic to a sphere with a guaranteed worst case code of 1.83 bits per triangle [King and Rossignac 1999]. The Wrap&Zip algorithm introduced in [Rossignac and Szymczak 1999] enhanced the original Edgebreaker decompression worst–case complexity from $O\left(n^2\right)$ to $O\left(n\right)$, where n is the number of triangles of the mesh. It decompresses the mesh in two passes, a direct and a recursive one. It is possible to decompress it in only one pass using the Spirale Reversi algorithm of [Isenburg and Snoeyink 2000], but it requires to read the encoded backwards, which is not appropriate for the Huffman encoding of [King and Rossignac 1999] or the arithmetic encoding. But the true value of Edgebreaker lies in the efficiency and in the simplicity of its implementations [Rossignac *et al.* 2001], which is very concise. This simple algorithm has been extended to deal with non–simplicial surfaces [Kronrod and Gotsman 2001] and the compression of simplicial surfaces with handles has been enhanced in [Lopes *et al.* 2003] using *handle data*. Because of its simplicity, Edgebreaker is viewed as the emerging standard for 3D compression [Salomon 2000] and may provide an alternative for the current MPEG–4 standard, which is based on the Topological Surgery approach [Taubin and Rossignac 1998].

In this section, we will enhance the Edgebreaker compression for surfaces with a non–empty boundary. [King and Rossignac 1999] encoded these sur-

	operator	$\mathcal{S}^{21}$ val.	$\mathcal{S}^{21}$ pos.	apex	left tri.	right tri.
C	attach	2	$\varnothing$	unmarked	unmarked	unmarked
R	MTE	2	left	marked	unmarked	marked
L	MTE	2	right	marked	marked	unmarked
E	2–handle	1		marked	marked	marked
S	1–handle	3		marked	unmarked	unmarked

Table IV.1: The CLERS codes.

faces by closing each bounding curve with a dummy vertex. This is a very simple but expensive solution: first, it requires encoding each bounding edge with a useless triangle; second, it requires extra code to localise the dummy vertex; and third, it gives bad geometrical predictors on the boundary. The original solution of [Rossignac 1999] however encodes bounding curves a special symbol containing their length, which solves the first item but does not describe explicitly the topology of the surface, and gave bad prediction on the boundary. As we introduced in [Lewiner *et al.* 2004], we use directly the handle data to encode the boundaries, which solves the above mentioned problems and enhances the compression ratio. We will also introduce a small acceleration to the Wrap&Zip procedure in order to avoid the recursion, accelerate the decompression and reduce the amount of memory used.

(a) CLERS encoding

Gate based compression. Edgebreaker encodes the connectivity of the mesh by producing the stream of symbols taken from the set **C**,**L**,**E**,**R**,**S**, called the *clers stream*. It traverses spirally the dual graph of a surface in order to generate a spanning tree. At each step, a decision is made to move from one triangle t to an adjacent triangle t' through an edge e' called the *gate*. The vertex v of t not contained in the previous gate e is called the *apex* of the gate. This decision depends on the previously visited triangles, which are marked together with their incident vertices.

Right–first traversal. The spiral traversal means that the next triangle is chosen to be the one on the right if not marked, where the right triangle means that the link of the new gate e' contains the vertex next to the apex v of the previous gate e (see Section III.1(d) *Simplicial Manifolds* for the definition of next). This gives a direct construction of the dual spanning tree and an order on it.

CLERS codes. The traversal is then encoded by the valences (1, 2 or 3) of the nodes of the dual spanning tree $\mathcal{S}^{21}$, and for the valence 2 case, by the current position ($\varnothing$, left or right) of the primal remainder $\mathcal{S}^{01}$ with respect to the new triangle. The corresponding symbols are stated on Table IV.1. The valence of the nodes of $\mathcal{S}^{21}$ can be easily detected during the traversal, using the rules of Table IV.1 [Rossignac 1999].

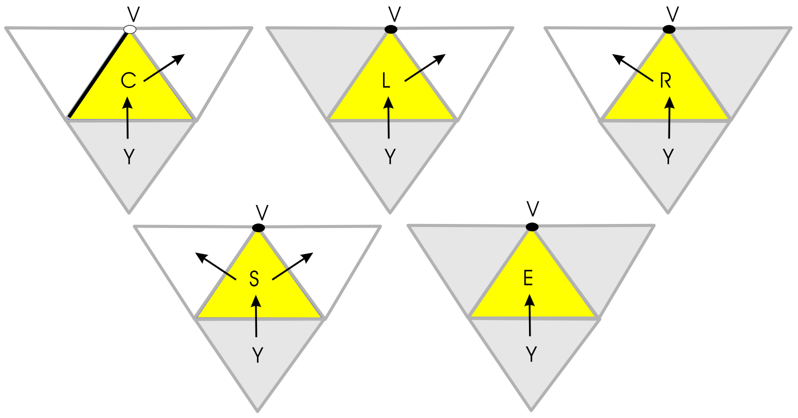
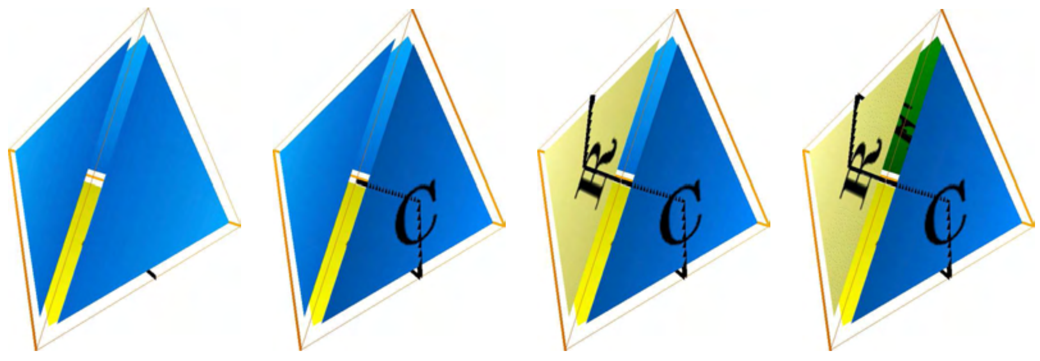


Figure IV.6: The Edgebreaker encoding. A C corresponds to a vertex Creation. With the outward orientation, an L means that the Left triangle has been visited, whereas an R means that the Right triangle has been visited. S stands for Split, and E for End.

Original compression. We will now describe directly the above formal presentation of the Edgebreaker. The algorithm starts by encoding the geometry of a first triangle, that will be the root of $\mathcal{S}^{21}$. In the text, we will call it a **P** triangle. It corresponds to a 0–handle Handle operator. The traversal begins right after with the rules of Table IV.1: if the apex is not marked, a **C** is encoded with the geometry of the apex, and the traversal continues on the right triangle. Otherwise, if the left triangle is marked, an **R** symbol is encoded and the traversal continues on the right triangle. Similarly, if the right triangle is marked, an **L** symbol is encoded and the traversal continues on the left triangle. If none of the triangles are marked (but the apex is), an **S** symbol is encoded. The traversal splits since the spanning tree has a branching here. The first traversed branch begins with the right triangle, and continues on the left one when the first branch ends. Finally, if both adjacent triangles are marked, the branch ends with an **E** symbol. This branching mechanism can be simply implemented with an **S** stack that stores the left triangle of each **S** triangle.

(b) Fast decompression



(a) P (b) C (c) R (d) E

Figure IV.7: Coding of a tetrahedron: PCRE.

Wrap&Zip decompression. The original Wrap&Zip procedure of [Rossignac and Szymczak 1999] decodes the clers stream in two passes. The Wrap simply decodes the dual spanning tree, with the geometry of each vertex at each **C** symbol. It decodes the **S**/**E** branchings and positions correctly the adjacent triangles using the branching order and the distinction between the **C** or **L** symbols and the **R** symbols. Then, the Zip procedure completes this spanning tree to obtain the dual graph. If the surface has the topology of a sphere, then there is enough information to recover the entire dual graph, as we will see next. The procedure is very similar to the enumeration of [Poulalhon and Schaeffer 2003]: it looks for the star of each vertex v, and if its star is not closed, and if the two bounding edges of its star are associated to a **C** on one side, and on another symbol on the other side, then these two edges are identified. A recursive implementation of this procedure is necessary to achieve a linear complexity, using the fact that the closure of a star usually allows closing adjacent stars, except when reaching an **L** or **E** symbol.

Fast Zip. Actually, the Zip procedure is a recursive traversal of the dual spanning tree, and it closes the stars from the leaves to the root. Actually, since the algorithm just built the spanning tree, there is no need to traverse it all to find the leaves. It is sufficient to use a **C** stack during the Wrap that stores each **C** triangle. Popping the **C** stack reads it in the reverse way, and the algorithm closes one star at each **C** symbol, and three for each **P** symbol, instead of trying all triangles. This spares half of the tests. Moreover, stars can be closed at some **R** and **E** symbols during the Wrap. This can be used to keep the size of the **C** stack small, and allows a better usage of the multiway geometry prediction of [Cohen–Or *et al.* 2001].

(c) Topology encoding

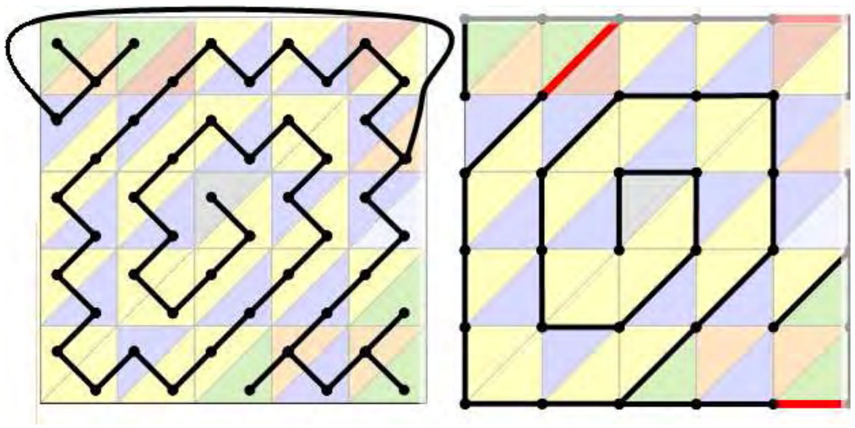
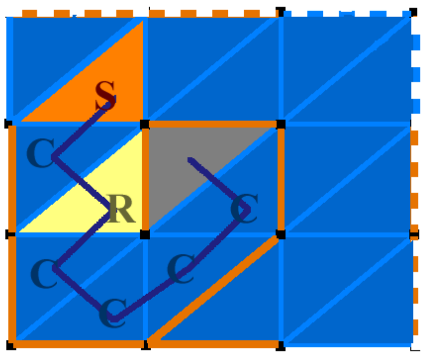
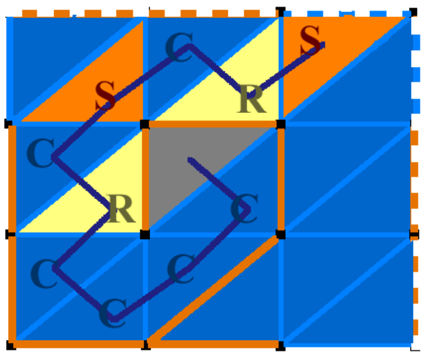


Figure IV.8: Dual tree generated by the Edgebrealer traversal and the primal remainder, with the two Morse edges in red.

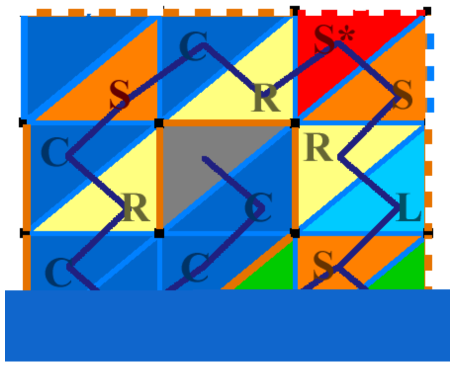
Handle $\mathbf{S}^h$ ***symbols.*** As we said earlier, if the surface $\mathcal{S}$ has genus $\mathfrak{g}(\mathcal{S}) > 0$, the primal remainder $\mathcal{S}^{01}$ is not a tree anymore, as illustrated on Figure IV.8. For a surface with an empty boundary, $\mathcal{S}^{01}$ has $2 \cdot \mathfrak{g}(\mathcal{S})$ cycles. These cycles can be simply detected during the traversal and efficiently encoded using [Lopes *et al.* 2003], while preserving the original Edgebreaker compression scheme. These cycles correspond to a branching, and thus to an **S** symbol. However, the two branchings induced by each genus of the surface loops back, and the left edge of the **S** triangle is visited before its right branch ends. During the execution, this is easily detected when popping the **S** stack containing the triangles left to **S** symbols: if the top of the **S** stack is not marked, the algorithm continues as normally. If the left triangle was marked, the **S** symbol actually corresponds to a handle, and will be marked as a *handle* $\mathbf{S}^h$ symbol. This symbol is encoded as a normal **S**, and special information identifying this $\mathbf{S}^h$ symbol is encoded in the *handle data*. In order to decompress handles directly, the position of the left triangle in the clers stream can be encoded, for example by the number of **S** symbol that preceded the $\mathbf{S}^h$ symbol and by the number of **R**, **L** and **E** symbols that preceded the left triangle, since *handle* $\mathbf{S}^h$ triangles are obviously closed by only these kind of triangles. These numbers can be encoded by differences to spare even more space.



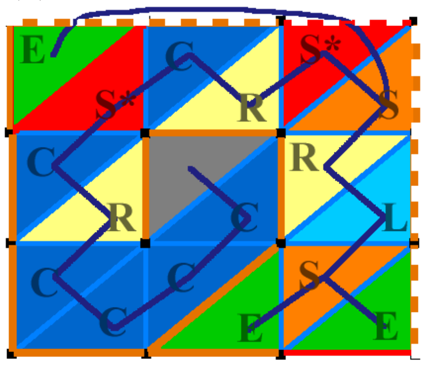
(a) Reaching first S triangle



(b) Reaching second S triangle



(c) The lower–right E triangle closes the handle.



(d) The upper–left E triangle closes the handle.

Figure IV.9: Coding of a torus: the creation of two *handle* S triangles: the first and the second S symbols.

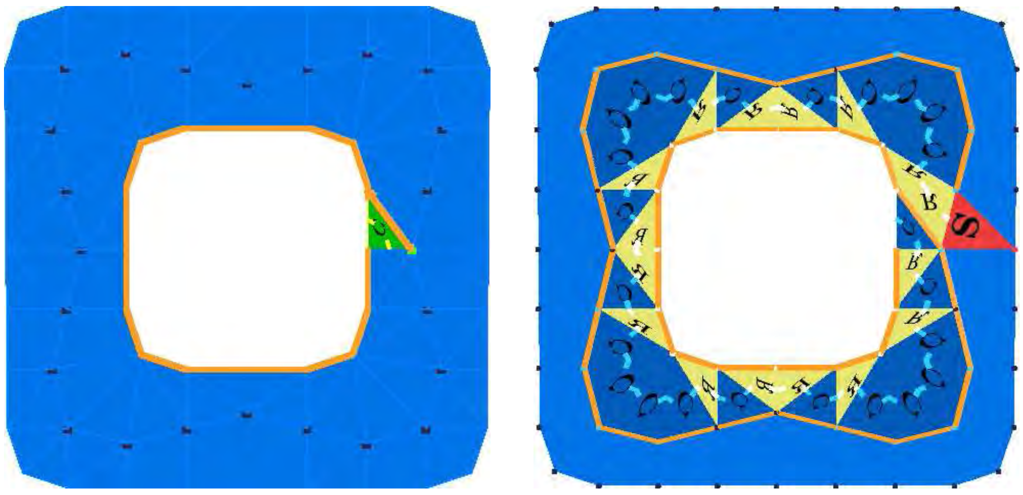
Example. To illustrate the algorithm, consider the triangulated torus of Figure IV.9, where the edges on the opposite sides of the rectangle are identified.

This simplicial complex can be embedded in $\mathbb{R}^3$. The Edgebreaker compression algorithm encodes the connectivity of the mesh though the following clers stream: **CCCCRCSCRSSRLSEEE**, completed with the following handle data: 0—4^-,0—3^+. There are four triangles labelled with an **S** symbol. The left triangles of the two last ones are visited when popping the **S** stack. On the contrary, the two first ones are visited before the being popped out of the **S** stack. These two triangles are detected as handle $\mathbf{S}^h$ symbols. This is encoded in the handle data as follows: the first handle $\mathbf{S}^h$ symbol is also the first **S** symbol, and the first number encoded is therefore 0. There are four possible matches (**R**, **L** and **E** symbols) for its left triangle before the good one, which is encoded by the 4. Since it is an **E** triangle, it can be glued on both sides, and the left side is indicated by the $\epsilon =^-$. The encoding is done the same way for the second handle $\mathbf{S}^h$ symbol.

First bounding curve. This scheme can be extended to boundary compression, since they correspond to the same Handle operators. Using the handle data to encode boundaries is then more coherent, gives a direct reading of the surface topology through this handle data even before decoding the mesh, and allows a specific prediction scheme for boundaries. Consider first a connected surface $\mathcal{S}$ with one bounding curve. Suppose that we close it by adding a face incident to each bounding edge of $\mathcal{S}$, called the *infinite face*. The resulted surface $\mathcal{S}^+$ has no boundary, and can almost be encoded by the previous algorithm. However, the infinite face is not a triangle. In the same way that the **P** triangles are not explicitly encoded, we will not encode this infinite face, and start the compression directly one of its adjacent triangle. As in the original Edgebreaker algorithm, we encode and mark first all its vertices, e.g., all the vertices belonging to the boundary of $\mathcal{S}$. Then, for the first boundary, we only need to know if the surface component has a boundary or not.

Boundary S^b ***symbols.*** Now, consider a connected surface has more than one bounding curve. Then, we distinguish arbitrarily one of them as the first boundary and the encoding uses the technique of the last paragraph. During the traversal, we label each triangle touching a new bounding curve as a boundary $\mathbf{S}^b$ triangle. As for handles, we encode it as a normal **S** symbol in the clers handle, and specify that it is a boundary $\mathbf{S}^b$ symbol in the handle data. To distinguish with handle $\mathbf{S}^h$ symbols, their first number is negative. Also, due to the orientation of the bounding curve, the left triangle is always glued on its left side, and we do not need to specify the last $\epsilon =^+$ or $\epsilon =^-$, and we can avoid counting the **L** symbols to localise it. From the Euler characteristic, we know that there is exactly one boundary $\mathbf{S}^b$ symbol per bounding curve. On Figure IV.10, the only *handle* **S** triangle is the first triangle with a vertex on the internal boundary that we encounter during the traversal. As said before, there are $2 \cdot \mathfrak{g}(\mathcal{S}) + \mathfrak{b}(\mathcal{S}) - 1$ such *handle* **S** triangles for each surface component with genus $\mathfrak{g}(\mathcal{S})$ and $\mathfrak{b}(\mathcal{S})$ bounding curves.

Multiple components. The compression processes successively each surface component. When the component has no boundary, the compression en-



(a) The first triangle is chosen adjacent to a boundary. The vertices of the central infinite face are encoded.

(b) An unmarked boundary is reached: the corresponding S triangle is a *boundary* S triangle.

Figure IV.10: Coding of an annulus: initialisation and creation of *boundary* S triangles.

codes explicitly the vertices of the first triangle (uncoded **P** symbol). Otherwise, it encodes the vertices of the first bounding curve. In practise, we only need to transmit the number of components with boundary of $\mathcal{S}$. Then we transmit first all the components with a non–empty boundary, and then the other ones.

(d) Compression algorithms

The compression scheme then decomposes in handling the multiple components and their first boundaries (Algorithm 5: compress), compress each component by the dual spanning tree traversal (Algorithm 4: traverse). The handles are tested along the traversal with Algorithm 6: check handle. The whole process is linear and performed in one pass only.

(e) Decompression algorithms

The decompression is performed in three passes, controlled by Algorithm 7: decompress. The first pass decodes the dual spanning tree (Algorithm 8: wrap), which is further zipped using the backward sequence of **C** symbols (Algorithm 9: fast zip). The compression described here encodes boundary curves, which improves prediction for the interior. However this means that the size of the boundary is not known to the decoder at the first pass, and the geometry must be decoded in a posterior step (Algorithm 10: read geometry). This pass could be done at the wrap stage if we encode the boundaries when they are closed, or if we encode the geometry of the bounding curves in a separate stream.

Algorithm 4 traverse(t): encode one component starting from triangle t

```
stack $Sstack \leftarrow \emptyset$                      // stack of the triangles left to S symbols
repeat
  $t$.mark $\leftarrow true$                                // mark current triangle
  $v \leftarrow t$.apex                                     // orient the triangle from its apex
  if $v$.mark $= false$ then                                // C triangle
    write vertex ($v$)                                      // encode the geometry of v
    $v$.mark $\leftarrow true$                              // mark the vertex
    write symbol (C)                                        // encode the clers code: C
    $t \leftarrow t$.right                                  // spiral traversal to the right
  else if is boundary($t$.right) or $t$.right.mark then     // right triangle visited
    if is boundary($t$.left) or $t$.left.mark then          // E triangle
      write symbol (E)                                      // encode the clers code: E
      check handle ($t$)     // check if it is the left triangle of a $\mathbf{S}^h$ triangle
      repeat
        if $Sstack = \emptyset$ then                        // end of compression
          return                                            // exit the external repeat loop
        end if
        $t \leftarrow Sstack$.pop                           // pop the S stack
      until not $t$.mark                                    // skip left of a handle $\mathbf{S}^h$ triangle
    else                                                    // R triangle
      write symbol (R)                                      // encode the clers code: R
      check handle ($t$)     // check if it is the left triangle of a $\mathbf{S}^h$ triangle
      $t \leftarrow t$.left                                 // break in spiral traversal: to the left
    end if
  else if is boundary($t$.left) or $t$.left.mark then       // L triangle
    write symbol (L)                                        // encode the clers code: L
    check handle ($t$)       // check if it is the left triangle of a $\mathbf{S}^h$ triangle
    $t \leftarrow t$.right                                  // spiral traversal to the right
  else                                                      // S triangle
    write symbol (S)                                        // encode the clers code: S
    if is boundary($v$) then                                // boundary $\mathbf{S}^b$ triangle
      write boundary ($t$)                                  // encode boundary
      $t$.mark $\leftarrow -\#_\mathrm{S}$                  // mark for the handle data
    else                                                    // normal S or handle $\mathbf{S}^h$ triangle
      $t$.mark $\leftarrow \#_\mathrm{S}$                   // mark for the handle data
    end if
    $Sstack$.push ($t$.left)                                // push the left triangle on the S stack
    $t \leftarrow t$.right                                  // spiral traversal to the right
  end if
until true                                                  // infinite loop
```

Algorithm 5 compress($\mathcal{S}$): compress separately each component of $\mathcal{S}$

```
1: $\mathfrak{b}^* \leftarrow 0$    // counts number of components with boundary
2: for all vertices $v \in \mathcal{S}$ do    // reset marks
3:   $v$.mark ← is boundary ($v$)    // mark boundary vertices
4: end for
5: for all triangles $t \in \mathcal{S}$ do    // compress components with boundary first
6:   if not $t$.mark and is boundary($t$) then    // not boundary or already encoded
7:     write boundary ($t$)    // encode boundary
8:     traverse($t$)    // component compression
9:     $\mathfrak{b}^* \leftarrow \mathfrak{b}^* + 1$    // one more component with boundary
10:  end if
11: end for
12: for all triangles $t \in \mathcal{S}$ do    // compress the other components
13:   if not $t$.mark then    // not already encoded
14:     $t$.mark ← $true$    // mark P triangle
15:     for all vertices $v \in \partial t$ do    // encode the 3 vertices of the P triangle
16:       write vertex ($v$)    // encode the geometry of v
17:       $v$.mark ← $true$    // mark the vertex
18:     end for
19:     traverse($t$.right)    // component compression
20:   end if
21: end for
22: write($handle, \mathfrak{b}^*$)    // write the number of components with boundary
```

Algorithm 6 check handle(t): check if triangle t is left to a $\mathbf{S}^h$ triangle

```
1: if not is boundary($t$.right) and $t$.right.mark $\notin \{true, false\}$ then    // handle $\mathbf{S}^h$ triangle to the right
2:   write($handle, t$.right.mark $- \#^+_{\mathbf{RE}}$)    // write the handle data
3: end if
4: if not is boundary($t$.left) and $t$.left.mark $\notin \{true, false\}$ then    // handle $\mathbf{S}^h$ triangle to the left
5:   write($handle, t$.left.mark $- \#^-_{\mathbf{LE}}$)    // write the handle data
6: end if
```

Algorithm 10 read geometry($Cstack'$): decompress the geometry

```
1: while $Cstack' \neq \emptyset$ do    // traverse the stack
2:   $t \leftarrow Cstack'$.pop ()    // pop the next element of the C stack
3:   if $t \geqslant 0$ then    // not a boundary triangle
4:     read vertex($t$)    // read a new vertex
5:   else    // boundary triangle
6:     read boundary($t$)    // read a new bounding curve
7:   end if
8: end while
```

Algorithm 7 decompress(streams): decompress separately each component

```
1: repeat
2:     $s - t^{\epsilon} \leftarrow$ read($handle$)                         // read handle data
3:     if $s > 0$ then                                                      // handle $\mathbf{S}^h$ symbol
4:         glue($s, t, \epsilon$)        // glue the handle on side $\epsilon$ before the decompression
5:     else                                                                 // boundary $\mathbf{S}^b$ symbol
6:         glue($-s, t, ^-$)     // close the bounding curve before the decompression
7:     end if
8: until end of file($handle$)                             // passed the last couple of data
9: $\mathfrak{b}^* \leftarrow s$   // last handle data counts number of components with boundary
10: stack $Cstack \leftarrow \varnothing$              // stack of the $\mathbf{C}$ and boundary $\mathbf{S}^b$ triangles
11: wrap($\mathfrak{b}^*, Cstack$)                                  // wrap using the clers stream
12: fast zip($Cstack$)                          // closes the stars of the primal remainder
13: read geometry($Cstack$)                         // reads the geometry of the surface
```

Algorithm 9 fast zip($Cstack$): decompress one primal remainder

```
1: stack $Cstack' \leftarrow \varnothing$         // reverse copy of the $\mathbf{C}$ stack for the geometry
2: while $Cstack \neq \varnothing$ do                                         // traverse the stack
3:     $t \leftarrow Cstack$.pop ()                   // pop the next element of the $\mathbf{C}$ stack
4:     $Cstack'$.push ($t$)                                                // copy the $\mathbf{C}$ stack
5:     if $t \geqslant 0$ then                                        // not a boundary triangle
6:         close star($t$)                             // close the star of the next vertex
7:     end if
8:     $Cstack \leftarrow Cstack'$                          // returns the copy of the $\mathbf{C}$ stack
9: end while
```

Algorithm 8 wrap($\mathfrak{b}^*$, $Cstack$): decompress the dual trees

```
1:  $\#_2 \leftarrow 0$                                                  // initialisation
2:  stack $Sstack \leftarrow \varnothing$                    // stack of the triangles left to S symbols
3:  repeat                                                          // components loop
4:    if $\mathfrak{b}^* > 0$ then                                   // component with boundary
5:      $\mathfrak{b}^* \leftarrow \mathfrak{b}^* - 1$; $t \leftarrow \varnothing$                 // first boundary
6:      $Cstack$.push $(-\#_2)$     // push the boundary triangle for the geometry
7:    else                                          // component with an empty boundary
8:      $t \leftarrow \#_2$                                                // P triangle
9:      $Cstack$.push $(t)$                     // push the first triangle for the zip
10:     for all vertices $v \in \partial t$ do   // decode the 3 vertices of the P triangle
11:       read vertex $(v)$                              // decode the geometry of $v$
12:     end for
13:     $t \leftarrow t$.right                          // spiral traversal to the right
14:     $\#_2 \leftarrow \#_2 + 1$                                         // initialisation
15:   end if
16:   repeat                                              // decompress one component
17:     glue$(t, \#_2)$        // glue the next triangle eventually to the boundary
18:     $s \leftarrow$ read symbol $(clers)$                        // reads the next symbol
19:     if $s = \mathbf{C}$ then                                            // C triangle
20:       $Cstack$.push $(\#_2)$                   // push the C triangle for the zip
21:       $t \leftarrow t$.right                   // orient the new triangle to the right
22:     else if $s = \mathbf{R}$ then                                       // R triangle
23:       $t \leftarrow t$.left                     // orient the new triangle to the left
24:       tryclose star $(t$.apex$)$                    // eventually zip the right edge
25:     else if $s = \mathbf{L}$ then                                       // L triangle
26:       $t \leftarrow t$.right                   // orient the new triangle to the right
27:     else if $s = \mathbf{S}$ then                                       // S triangle
28:       if not $t$.right.mark then      // not a handle or boundary S symbol
29:         $Sstack$.push $(\#_2$.left$)$        // push the S triangle for the next E
30:       else if is boundary$(\#_2)$ then                        // boundary triangle
31:         $Cstack$.push $(-\#_2)$          // push the boundary triangle for the
            geometry
32:       end if
33:       $t \leftarrow t$.right                   // orient the new triangle to the right
34:     else if $s = \mathbf{E}$ then                                       // E triangle
35:       tryclose star $(t$.apex$)$       // eventually zip the right and left edges
36:       if $Sstack = \varnothing$ then                         // end of the component
37:         break                                          // exits the component loop
38:       end if
39:       $t \leftarrow Sstack$.pop $()$             // pop the next element of the S stack
40:     end if
41:     $\#_2 \leftarrow \#_2 + 1$                                         // next triangle
42:   until true                                                     // infinite loop
43: until end of file$(clers)$                                // end of the clers stream
```

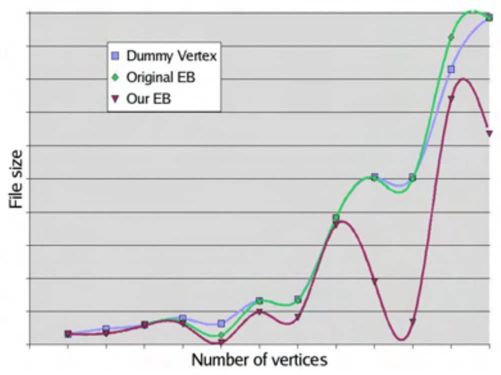
IV.3 Performances

We presented in this chapter the fundamental concepts of connectivity–driven compression. In particular, we focused on an extension of the Edgebreaker algorithm, which handles manifold surfaces of arbitrary topology. The complexity of the compression and the decompression are both linear in execution time and memory footprint, independently of the maximal number of the active elements during the execution. However, the decompression still requires two passes, which makes it harder to stream.

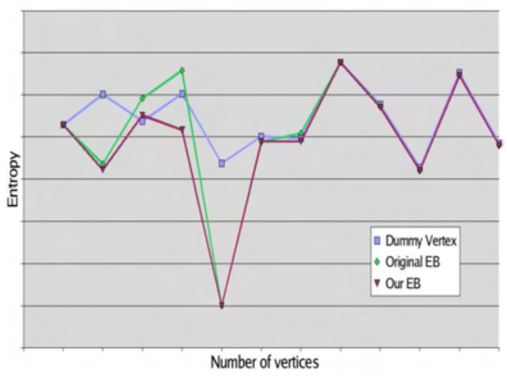
There are various ways of representing a geometrical object, even for simplicial surfaces. For specific type of meshes, some algorithms show better performances than other ones. This distinction is one of the main shifts from the MPEG compression [le Gall 1991] to the MPEG–4 one [Pereira and Ebrahimi 2002], which for example encodes differently human faces than landscapes. Although it is difficult to distinguish with precision classes of meshes and to predict exactly the behaviour of compression algorithms on these, we will try to get an intuition of which characteristics of a mesh are well suited for connectivity–driven compression schemes, and in particular for the Edgebreaker.

(a) Compression Rates

Our experimental results for the Edgebreaker are recorded on Table IV.2 and Figure IV.11. We compared with the original Edgebreaker implementation with the Huffman encoding of [King and Rossignac 1999] and the border handling of [Rossignac and Szymczak 1999], and our encoding with the simple arithmetic coder of [Martin 1979]. Our experimental results are always better than the original Edgebreaker, mainly due to the arithmetic coding. However, the entropy of our codes is always better than the other implementations of Edgebreaker, as shown on Figure IV.11(b). A compression ratio of a few bits per vertex, or even less, is a general order for efficient connectivity–driven compression schemes.



(a) Size of the compressed file vs complexity of the model.



(b) Entropy vs complexity of the model.

Figure IV.11: Comparison of the final size and entropy: for the range encoder, those parameters depends more on the regularity than on the size of the model, but our algorithm really enhances the previous results.

Model	$\|\#_0\|$	$\|\#_2\|$	**Dum**	**Ori**	**Ours**	$\frac{\textbf{Ori}}{\textbf{Ours}}$	$\frac{\textbf{Dum}}{\textbf{Ours}}$
sphere	1 848	926	3.39	3.39	3.45	0.98	0.98
violin	1 508	1 498	3.16	2.21	2.25	0.98	1.41
pig	3 560	1 843	3.26	3.24	3.13	1.03	1.04
rose	3 576	2 346	3.37	2.95	2.64	1.12	1.28
cathedral	1 434	2 868	2.25	1.00	0.19	5.27	11.86
blech	7 938	4 100	3.25	3.18	2.40	1.33	1.35
mask	8 288	4 291	3.19	3.12	1.93	1.62	1.65
skull	22 104	10 952	3.51	3.51	3.30	1.06	1.06
bunny	29 783	15 000	3.36	3.34	1.27	2.62	2.64
terrain	32 768	16 641	3.03	3.00	0.40	7.43	7.51
david	47 753	24 085	3.45	3.85	3.07	1.25	1.12
gargoyle	59 940	30 059	3.28	3.27	2.11	1.55	1.55

Table IV.2: Comparative results on different models drawn on Figure IV.13. 'Dum' stands for the dummy vertex method to encode meshes with boundaries [Rossignac and Szymczak 1999, King and Rossignac 1999], and 'Ori' stands for the original Edgebreaker [Rossignac 1999], and 'Ours' for the algorithm introduced here, with the simple arithmetic coder of [Martin 1979]. The size of the compressed symbols (columns 'Dum', 'Ori' and 'Ours') is expressed in bit per vertex. Our algorithm has a compression ratio in weighted average 2.5 better than the other two. The 'sphere' model has the same encoding in all the above algorithms, but the range coder used has a lower performance since there are few symbols to encode. The 'cathedral' model is the output of an architecture modelling program, which is almost unstructured: all the connected components are pairs of triangles.

(b) Good and bad cases

Topology–dependent applications. For our extended Edgebreaker, the separate handle data informs directly the application of the topology of the mesh. Many simple parameterisations, texturing or remeshing applications work only for closed surfaces without handle. The handle data can be used to call a preprocessing step for simplifying the topology before using these kind algorithms. For the Edgebreaker, this handle data is not an overhead, since encoding the handle and boundary **S** symbols as a true/false code on the clers string is in the best case logarithmic as we saw in Section II.2(c) *Statistical Modelling*, which is equivalent to the handle data.

Regular connectivity. The valence coding of [Touma and Gotsman 1998, Kälberer *et al.* 2005] encodes particularly well meshes where the vertices have a uniform valence. This can be obtained by subdivision [Loop 1987, Velho and Zorin 2001] or remeshing [Alliez *et al.* 2003, Alliez *et al.* 2003]. Remeshing can be done also to improve the Edgebreaker compression using

	Edgebreaker	Valence coding
Topology–dependent	$+++$ [Lewiner *et al.* 2004]	$---$
Regular valence	$---$	$+++$ [Touma and Gotsman 1998]
Lossy connec.	$+$ [Attene *et al.* 2003]	$+++$ [Alliez and Desbrun 2001]
Self–similar connec.	$+++$ [Rossignac 1999]	$+$ [Kälberer *et al.* 2005]
Irregular connec.	$+$ [King and Rossignac 1999]	$+++$ [Castelli and Devillers 2004]
Geometric connec.	$+++$ [Coors and Rossignac 2004]	$+$ [Lee *et al.* 2003]
Geometry prediction	$+$ [Lewiner *et al.* 2005]	$+++$ [Cohen–Or *et al.* 2001]
Low resource use	$+++$ [Rossignac *et al.* 2001]	$---$

Table IV.3: Good and bad cases for the two main connectivity–driven compression schemes.

the Swingwrapper of [Attene *et al.* 2003]. Without these regularisations, valence coding based algorithms have better performance when the connectivity is locally regular, whereas the Edgebreaker performs better on irregular meshes or meshes with a global regularity, such as those obtained by subdivision algorithms or with some self–similar connectivity. Meshes with a very irregular connectivity would be better encoded by enumeration methods of [Poulalhon and Schaeffer 2003, Castelli and Devillers 2004].

Regular geometry. The geometry of the mesh is not directly considered in connectivity–driven compression, and therefore geometry–based compression will outperform these schemes for the connectivity compression of meshes with a regular geometry. However, the geometry can be used to predict the connectivity, which works specifically when the geometry is regular. This has been done for the valence coding in [Alliez and Desbrun 2001, Lee *et al.* 2003] and in [Coors and Rossignac 2004] for the Edgebreaker.

Geometry prediction. Geometry prediction uses already decoded vertices to estimate the next vertex to be decoded, asserting that the geometry is locally regular. For connectivity–driven schemes are usually based on the parallelogram predictor of [Touma and Gotsman 1998]. It can be enhanced by using more than one parallelogram to estimate the new position, as described in [Cohen–Or *et al.* 2001]. This is particularly well adapted to the valence coding since the traversal can be adapted to the prediction. For the Edgebreaker, the parallelogram can be distorted to adapt to local mean curvature of the surface, as in [Lee *et al.* 2003], or to torsion and curvature of the primal remainder, as described in [Lewiner *et al.* 2005] and on Figure IV.12.

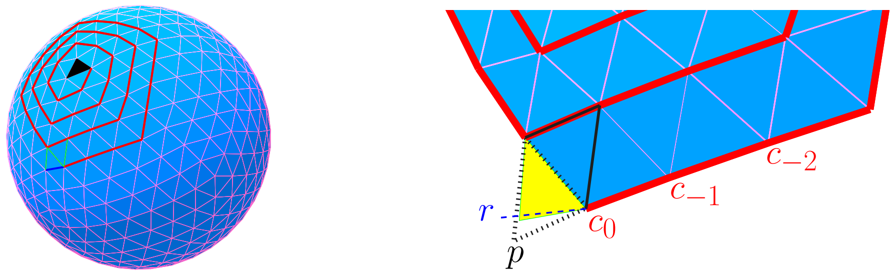


Figure IV.12: The Edgebreaker cuts the compressed surface along a curve in the space. An extrapolation of this curve is used to enhance the parallelogram predictor. The predictor uses the parallelogram predictor to guess the distance from the last vertex of the curve, and rotates this estimation according to the approximating curve.

Low resource applications. The Edgebreaker uses a deterministic traversal, independent of geometry considerations. Although this is less flexible for geometry prediction enhancements, it gives a very simple algorithm. Moreover, compared to the valence coding schemes that needs to maintain sorted active boundaries along compression and decompression, the Edgebreaker just needs a stack of past **S** symbols. The Edgebreaker thus requires much less memory for the execution, and spares a constant sort, which can become expensive. More generally, connectivity–driven compression schemes are easy to implement and quick to execute.

The above results are roughly summarised on table IV.3, as a general appreciation from the author.

(c) Extensions

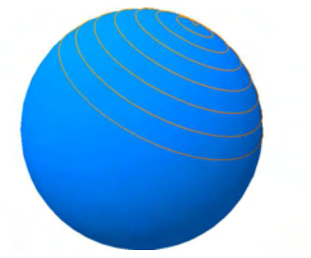
Non–simplicial meshes. Connectivity–driven compression schemes are easier on simplicial meshes, since the dual graph has a constant valence. Most of the mesh compression algorithms for polytope surfaces can be interpreted as a simplicial encoding preceded by a triangulation of each face. This triangulation is done in a canonical way from the traversal, and the decoder just need to know the degree of the triangulated faces. For example, the valence coding can be extended by encoding simultaneously the vertex valences and the face degrees, as in [Alliez and Desbrun 2001], and the Edgebreaker codes can be combined in a predictable way using the codes of [Kronrod and Gotsman 2001].

Non–manifold meshes. Extending these methods to non–manifold meshes directly is a hard task. The usual method consists in cutting the non–manifold surface into manifold pieces, using the techniques of [Guéziec *et al.* 1998], encoding the manifold parts as separate components, and then encoding the cut operations that were performed. The encoding of cut operations can be done directly as in the handle data, or more carefully by propagating the curves formed by the non–manifold edges.

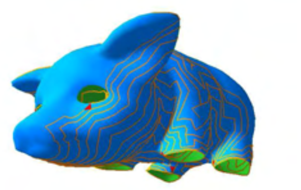
Higher dimensions. For solid meshes, the Edgebreaker compression has been directly extended to tetrahedral meshes in [Gumhold *et al.* 1999, Szymczak and Rossignac 2000], and the valence coding has been extended in [Isenburg and Alliez 2002]. The principles are the same, but the encoding needs some extra information to complete the intermediate dimension between the spanning tree and the remainders. This extra information has necessarily some expensive parts to encode, similar to the handle **S** symbols that are necessary to glue distant parts of the traversal. Minimising this extra information is an NP–hard problem, as proved in [Lewiner *et al.* 2004]. For higher dimensions, the combinatory of mesh connectivity makes it difficult to find a concise set of symbols for coding, or a good statistical model for them as was done for surfaces in [King and Rossignac 1999]. However, for high codimensions, the connectivity remains simple while the geometry can be efficiently predicted. Seen from the other side, this means that for low codimension, geometry–based coding can be very efficient, which is where isosurface compression outperforms any connectivity–based compression, as we will see in the next chapter.

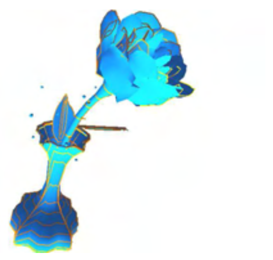
Robustness. The Edgebreaker is robust in the sense that it handles general manifold surfaces. However, it is not particularly robust with a noisy transmission, where the clers codes can be altered. In that case, the grammar inherent to these codes can be used to detect transmission errors, but not directly to correct them.

Deformable meshes. For animation purposes, the Edgebreaker can be used directly to compute the deformed mesh when its connectivity is constant, and using for example [Sorkine *et al.* 2003] to interpolate the geometry. Local changes in the connectivity can be further encoded using the explicit identification of vertices and triangles provided by the Edgebreaker, similarly to the description of [Vieira *et al.* 2004].

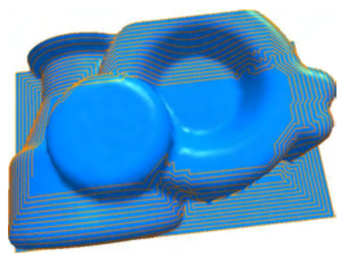

(a) sphere

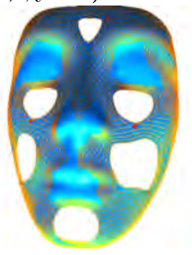
(b) violin (135 comps, 138 bdries)

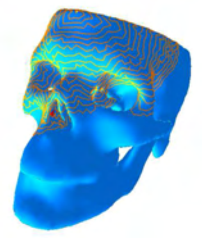

(c) pig (6 bdries)

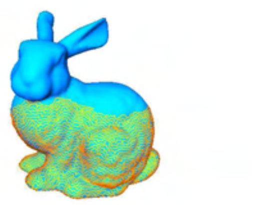

(d) rose (51 comps, 64 bdries, $\chi = 0$)

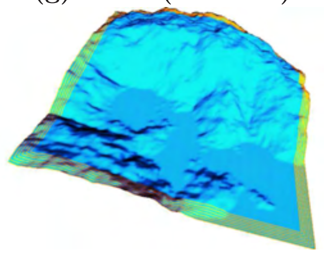
(e) cathedral (717 comps)

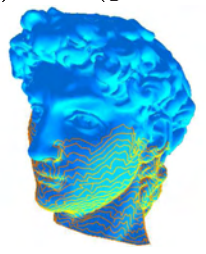

(f) blech


(g) mask (7 bdries)


(h) skull (genus 51)


(i) bunny (5 bdries)


(j) terrain


(k) david

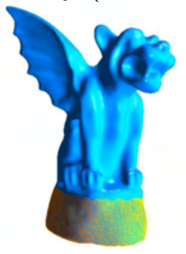

(l) gargoyle

Figure IV.13: Some of the models used for the experiments, with the beginning of Edgebreaker's dual spanning tree: The 'violin' has 135 components and 138 boundaries. The 'rose' has 51 components, genus 1 and 64 boundaries. The cathedral has 717 components with boundary. The 'mask' has 7 boundaries. The 'skull' has genus 51. The bunny has 5 boundaries.

V
Level Set Compression

Connectivity–driven compression schemes provide general methods with good performances in average. However, for specific cases, their compression ratios are far away from those of dedicated techniques. This chapter describes the specific case of isosurface compression, and more generally level sets compression, where compression ratio can achieve fractions of connectivity–driven ones for the connectivity *and* the geometry (from 8–10 bits per vertex to only 1 or 2). This mixture with geometry is the key to reach these performances.

Level sets. A *level set* is the preimage $f^{-1}(0)$ of the singleton $\{0\}$ by a continuous function $f : \mathbb{X} \to \mathbb{R}$. The domain $\mathbb{X}$ of f is generally an Euclidean space $\mathbb{R}^d$. If f is continuously differentiable, this implies from Sard's theorem [Arnold 1981] that the level set is generically a manifold of dimension $d-1$ with an empty boundary. In the discrete setting, the sampling $\mathfrak{f}$ of f will be called the *scalar data*, and the points $\mathbf{x}$ of $\mathbb{X}$ where $\mathfrak{f}$ is defined will be called the *sample points*. The discrete level set then depends on the sampling and on the interpolation. We chose one particular interpolation, detailed in Section V.1 *Level Set Extraction*.

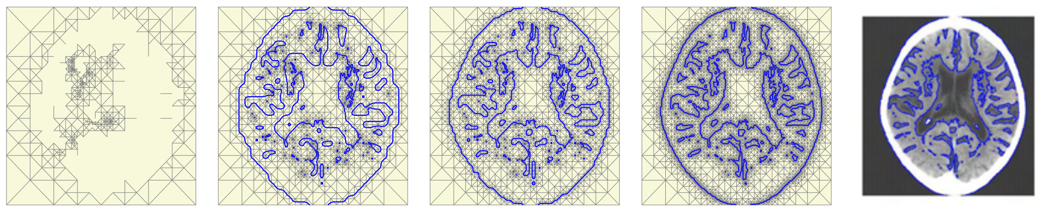

(a) 764 bytes (b) 853 bytes (c) 1089 bytes (d) 2011 bytes (e) 5369 bytes

Figure V.1: Topology controlled extraction of a Computerized Tomography image of the cortex, and progressive compression.

Level curve compression. The first level sets to be encoded were contour curves ($d = 2$), especially for cartographic data and shape classification. When the scalar data is an image, and the sample points are limited to the pixels of that image, the contour can be compressed using derivatives of the chain code of [Freeman 1974], or as a standard bidimensional signal as in [Langdon and Rissanen 1981, Bossen and Ebrahimi 1997]. When the contour is more precise than pixel quantisation, the vector displacement inside that pixel can be compressed using [Craizer *et al.* 2002*1, Safonova and Rossignac 2003]. These methods can be extended to hierarchical representations, as in [Lopes *et al.* 2002*1], and then to progressive compression schemes, as done in [le Buhan and Ebrahimi 1997] by a hierarchical representation induced by a multiresolution of the image.

Isosurface applications. Isosurfaces ($d = 3$) are widely used in many fields, from medical imaging [Lorensen and Cline 1987] to computer graphics [Parker *et al.* 1998] and surface reconstruction [Davis *et al.* 2002], through geophysical modelling [Tavares *et al.* 2003] and scientific visualisation [Bajaj *et al.* 1998]. For example, medical scan techniques such as computerised axial tomography and magnetic resonance imaging measure physical quantities sampled on a semiregular tridimensional grid. Also for scientific simulations, partial differential equations are usually resolved using level set methods, as those of [Sethian 1999], which also result in sampled functions over a tridimensional grid. This variety of applications resulted also from a large panel of techniques to manipulate isosurfaces [Bloomenthal *et al.* 1997]. These isosurfaces can be structured into meshes by the usual Marching Cubes method of [Lorensen and Cline 1987] and its extensions [Nielson and Hamann 1991, Montani *et al.* 1994, Lewiner *et al.* 2003*3], the dual contouring of [Ju *et al.* 2002] and simplicial methods [Velho 1996, Treece *et al.* 1999].

Isosurface compression. In most of isosurface applications, the visual result is only a surface extracted from the scalar data as an interpolation of $\mathfrak{f}$. For example, the cortex corresponds to only a specific X–ray scintillation inside the scan of the whole head. In that case, looking only at this cortex allows discarding most of the scalar data, as the reconstructed cortex is contained only in a limited part of the head. This motivates specific techniques to compress isosurfaces by encoding only the essential portion of the scalar data. Moreover, these techniques show better performances than generic mesh compression methods. Although [Saupe and Kuska 2002] and [Yang and Wu 2002] already developed specific compression schemes for isosurface, their extensive encoding based on Marching Cubes' configurations have been quickly bested by the JBIG compression of the scalar data, and then by [Taubin 2002] who extended this arithmetic encoding with a context in all three dimensions of scalar data. Then, [Boada and Navazo 2001] proposed to encode only the specific part of the scalar data containing the isosurface, in a progressive manner based on an octree decomposition of the scalar data. A small improvement of this work proposed in [Lee *et al.* 2003] to complete the compression with a final encoding of the geometry at the finest level of detail.

Simplicial level set compression. We will introduce now a general method to compress level sets in any dimension, which is competitive with the state–of–the–art for contour curves [le Buhan and Ebrahimi 1997] and isosurfaces [Lee *et al.* 2003]. This method was first introduced in [Lewiner *et al.* 2004*2, Lewiner *et al.* 2004*4, Lewiner *et al.* 2005*3], and can be used for direct or progressive compression, as on Figure I.5. It is based

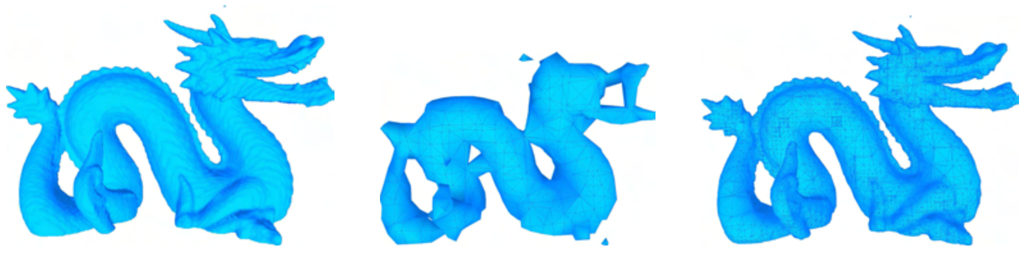


Figure V.2: Isosurface multiresolution representation can minimise the distortion induced by progressive compression.

on the regular binary multi–triangulations (RBMT) of [Mello *et al.* 2003], that we presented in Section III.3(c) *Multi–Triangulations*, which allows a greater adaptability than the octree of [Boada and Navazo 2001, Lee *et al.* 2003]. The RBMT is built on the sample points, and its simplification induce a multiresolution on the isosurface, which provides simple mechanisms to control the geometry, distortion and topology over the compression process, as on Figure V.1, Figure I.4, and Figure V.2. The compression is performed as a traversal, guided by the sign of $\mathfrak{f}$ on the sample points encountered.

V.1 Level Set Extraction

We will structure the scalar data as a RBMT, where each vertex coincides with a sample point of $\mathfrak{f}$. This is the case of regular grids, terrains, stratigraphical grids or discrete maps of functions. The RBMT can be automatically generated starting with a triangulation of the hypercube, and then mapped onto $\mathbb{X}$ by a continuous function.

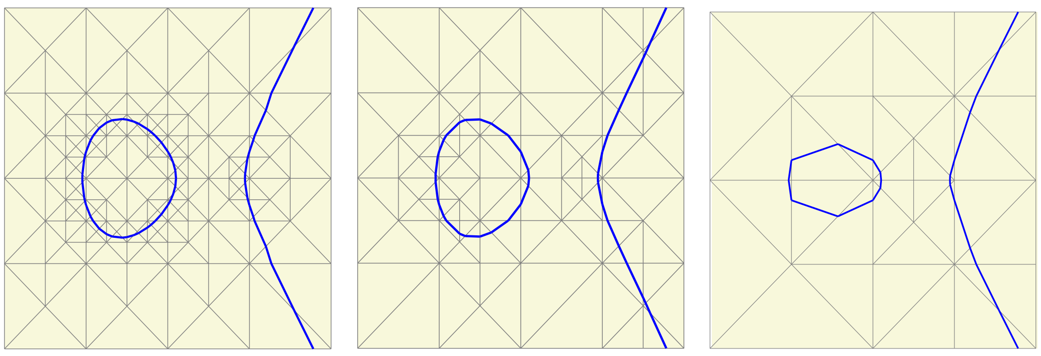


Figure V.3: A singular contour curve extracted in order to preserve: V.3(a) the curvature, V.3(b) the distortion and V.3(c) the topology.

The whole process of compression starts with the full scalar data, which corresponds to the finest level of the RBMT. The level set at that level will be called the *original* level set. Then, the RBMT is progressively simplified, and at each step, a simplification that would induce a too high distortion or a topological change can be prevented, as on Figure V.3. The simplification thus proceeds by non–local simplifications until reaching what we will call the coarsest level of the RBMT according to the distortion and topology criteria we used to simplify. Then, we encode this coarsest level directly, and encode successively the refinements of each level of the RBMT.

Extraction and compression thus work in opposite ways, and the quality of the multiresolution extraction will correspond to the quality of the progressive decompression. This whole process can be used by parts, since the direct encoding can be used directly at the finest level, leading to a competitive direct compression scheme for level sets. Moreover, the only extraction has direct applications into multiresolution representations of level sets with guarantees, as detailed in this section.

(a) Simplicial Interpolation

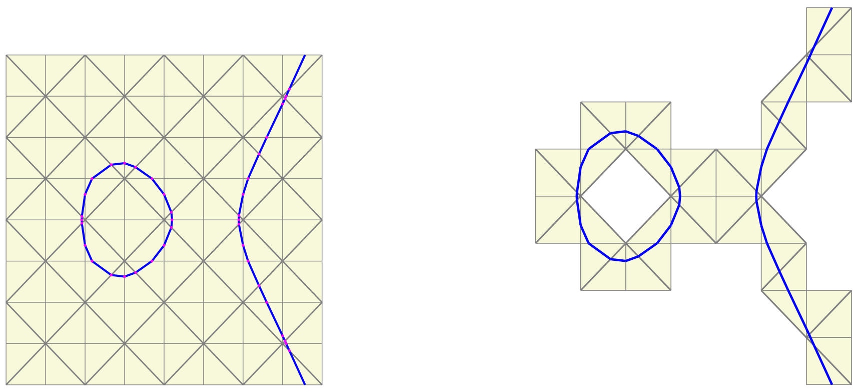


Figure V.4: The tubular neighbourhood of a contour curve.

Tubular neighbourhood. For a fixed resolution K of the RBMT, a vertex v is said to be *positive* if the scalar data $\mathfrak{f}$ is positive at the corresponding sample point, and *negative* otherwise. Then, a simplex of K incident to a positive and a negative vertex is called a *crossing simplex*. The collection of the crossing simplices of K will be called the *tubular neighbourhood* $\mathbb{U}_K(\mathfrak{f})$ of the level set of $\mathfrak{f}$ in K: $\mathbb{U}_K(\mathfrak{f}) = \{\sigma \in K : \mathfrak{f}(\sigma) \ni 0\}$. Figure V.4 shows an example of such tubular neighbourhood.

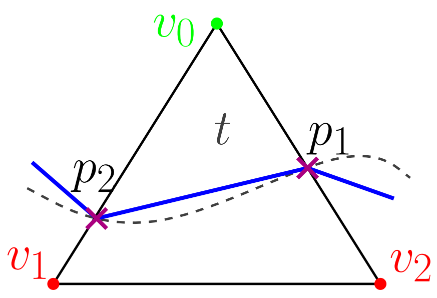


Figure V.5: Interpolation of the contour cruve.

Linear interpolation on edges. We intend to create a mesh $\mathcal{M}_K$ representing the level set, generalising the method of [Velho 1996]. The geometry of this mesh, i.e. the position of its vertices, will be computed by linear interpolation of $\mathfrak{f}$. Thus, each vertex v of $\mathcal{M}_K$ belongs to a crossing edges e of K. If we denote $\partial e = \{w_-, w_+\}$, then the position of v is given by the following linear equation, schematized on Figure V.5:

$$v = \lambda \cdot w_- + (1 - \lambda) \cdot w_+ \qquad \textit{with} \qquad \lambda = \frac{\mathfrak{f}(w_+)}{\mathfrak{f}(w_+) - \mathfrak{f}(w_-)} \qquad .$$

Level set meshing. The level set is also interpolated linearly inside each simplex. This corresponds to intersecting a simplex with a codimension 1 hyperplane. This hyperplane contains the vertices of $\mathcal{M}_K$, since they are linearly interpolated on the edges. In general, there exists more than n crossing edges for inside an n–simplex, and there will therefore be more than one maximal simplex of $\mathcal{M}_K$ inside a crossing simplex of K. For arbitrary dimensions, these simplices can be computed directly as the Cartesian product of two simplices as in [Gelfand *et al.* 1994, p. 246] or by triangulating the convex hull of the vertices of $\mathcal{M}_K$ of each crossing simplex, for example using the Delaunay triangulation of Section III.3(b) *Delaunay Triangulation*. For contour curve, there is exactly two vertices of $\mathcal{M}_K$ inside a crossing triangle, and the level set is meshed by linking these vertices by an edge. In the case of a grey–scale image, this corresponds to a linear subpixel interpolation, as the purple points of Figure V.4. For isosurfaces, the level set is triangulated using the look–up table of [Velho 1996], schematized on Figure V.6.

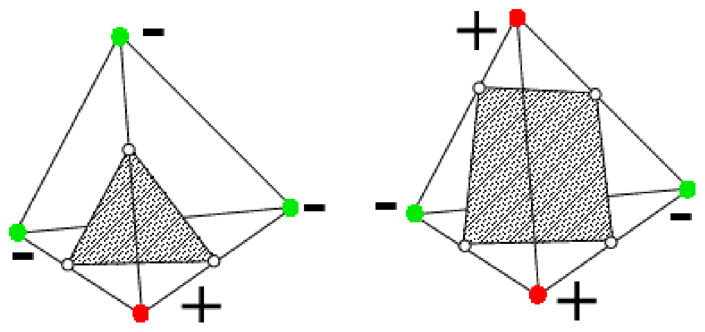


Figure V.6: Simplicial meshing of an isosurface.

(b) Geometry and Distortion Control

Controlled simplification. The Algorithm 3: simplify of Section III.3(c) *Multi–Triangulations* provided a simple way to simplify a resolution of the RBMT while preserving its regularity. Here, we want to simplify the resolution of the RBMT to induce a controlled simplification of the level set, as for example the simplifications of Figure V.3 and Figure V.7. The controlled is obtained by testing for each local simplification if its consequences on the level set are suitable or not. For example, we can simplify the RBMT while preserving the level set, by preventing any local operation on its tubular neighbourhood. This can be implemented by successive calls

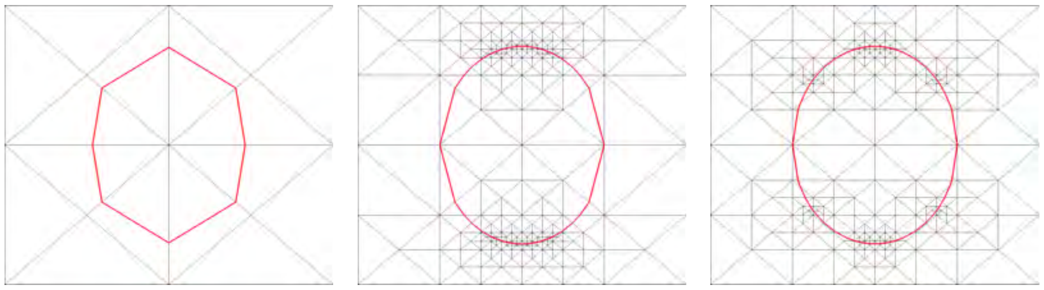

(a) no control (b) curvature (c) distortion

Figure V.7: An ellipse extracted without control, and controlling: V.7(b) its curvature, V.7(c) its level of distortion.

to Algorithm 3: simplify, but Algorithm 11: global simplify gives a faster implementation. In this algorithm, w.level refers to the level of the vertex in the RBMT. Figure III.16 was generated using such a criterion on the different

Algorithm 11 global simplify($\mathfrak{f}$) : global simplification preserving the tubular neighbourhood of the level set of $\mathfrak{f}$

```
1:  repeat                                                   // recurse on levels
2:    changed ← false                                    // to stop the repeat loop
3:    for all $w \in K_{(0)}$ do                                     // vertices of K
4:      for all $w' \in \mathrm{st}(w)$ do                        // vertices in the star of w
5:        if $w$.level > $w'$.level or $\mathfrak{f}(w) \cdot \mathfrak{f}(w') \leqslant 0$ then // w is not the lowest
          vertex or ww' is crossing
6:          continue[2]                                   // cannot simplify it: next w
7:        end if
8:      end for
9:      weld($w$) ; changed ← true                              // locally simplify w
10:   end for
11: until changed                                                    // stabilised
```

steps of Figure III.14.

Geometry control. The crossing criterion $\mathfrak{f}(w) \cdot \mathfrak{f}(w') \leqslant 0$ of line 5 of Algorithm 11: global simplify can be modified to simplify further the level set inside its tubular neighbourhood $\mathbb{U}_K$. The simplest case would be to simplify any vertex of the RBMT having a level superior to a given threshold. This gives a tubular neighbourhood with constant size of simplices. Since a flat curvature of the level set means that it is well approximated by an affine plane, regions where the estimated curvature is low can be simplified with small visual distortion. The curvature of a level set can be simply estimated by local differences of the scalar data around it, which makes it easy to modify Algorithm 11: global simplify for these kind of geometric control. Instead of curvature, view–dependent and ray–casting strategies can be used for visualisation, or specific curvatures for parameterisation and texturing applications.

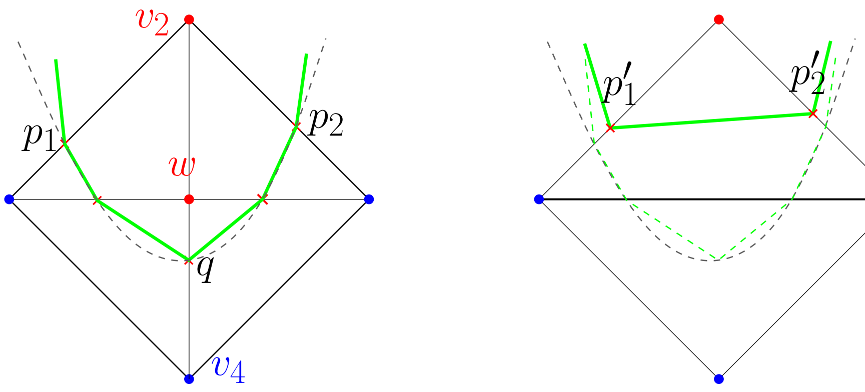


Figure V.8: The simplification of w induces a distortion that can be measured as $\max\{d(q,p'_1);d(q,p'_2)\}$.

Distortion control. The performance of progressive compression schemes is usually measured in terms of rate/distortion curves. Therefore, we can tune our simplification algorithm in order to control the distortion of the simplified level set $\mathcal{M}_K$ compared to the original one $\mathcal{M}_{K_0}$. An example for curves is drawn on Figure V.8. The distortion is usually measured as the *semi–Hausdorff distance* [Cignoni *et al.* 1998*1], which is the maximal distance of a point of $\mathcal{M}_{K_0}$ to the closest point of $\mathcal{M}_K$. This distortion induced by the local simplification of a vertex w in K can be estimated by the maximal distance between the portions of the original level set and the portion of the level set after the simplification lying in the star of w. Since the w needs to be the lowest vertex of its star to be simplified, the star of w in K has always the same connectivity for a given dimension, which allows a generic implementation of that test as a small modification at line 5 of Algorithm 11: global simplify. In particular for coding, this distance can take into account the quantisation error of the decoder for $\mathcal{M}_K$, as done on Figure V.8, in order for the encoder to compute exactly the final distortion of the whole compression scheme.

(c) Topology Control

Generic topology control. Similarly, the topology of the level set can be easily controlled during the extraction process. Consider the local simplification of a vertex w, and denote e the edge of the simplified mesh containing w. If the level set is a topological disk in the star of w, the local simplification does not change its topology. This test can be partially done by computing the Euler–Poincaré characteristic, which differs from 1 if the part of the level set is not a disk. Moreover, if the edge e is not crossing, while its two subedges were crossing before the simplification, the simplification can induce a topological change as on Figure V.9.

Topology control for contour curves. This test can be simplified and certified for contour curves. If we denote $\partial e = \{v_l, v_r\}$, there are only two prohibited simplifications, and similar ones on the boundary, as described on Figure V.9 and Algorithm 12: topology :

(1)

(2)

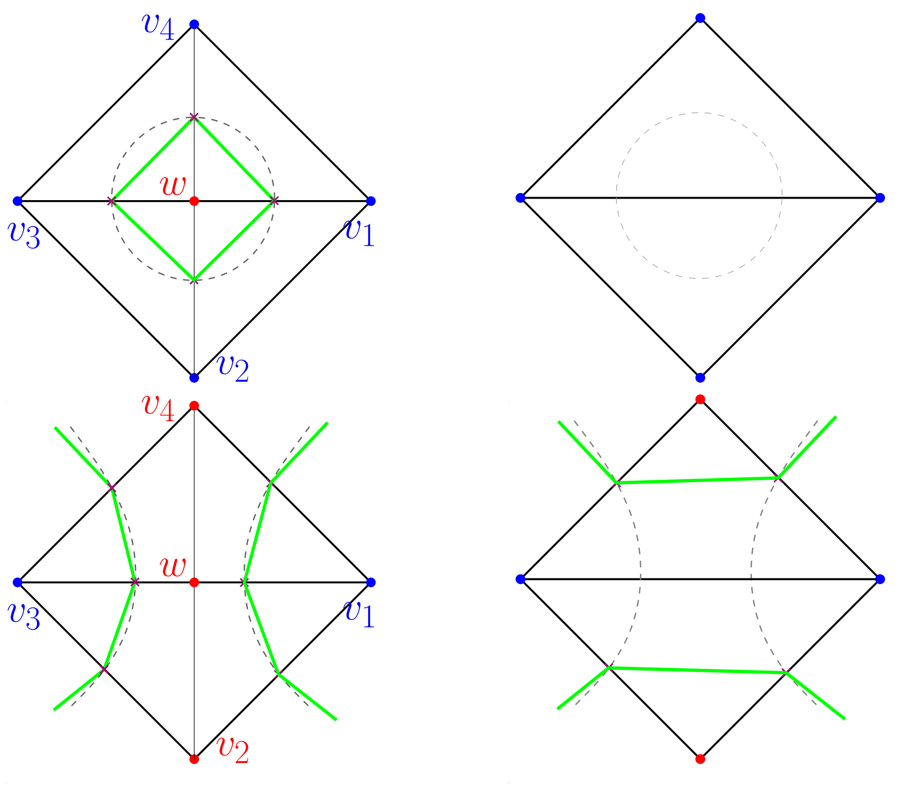


Figure V.9: There are two cases where the simplification of w induces a change on the topology: (1) destruction of a connected component, (2) exchange of local connected components.

1. The destruction of a connected component occurs when all the four edges incident to w are crossing.

2. The separation of a connected component in two or the merge of two connected components happen when the subdivision edge $e = \{v_l, v_r\}$ is not crossing $(\mathfrak{f}(v_l) \cdot \mathfrak{f}(v_r) > 0)$, while its subedges were crossing $(\mathfrak{f}(v_l) \cdot \mathfrak{f}(w) \leqslant 0)$.

Algorithm 12 curve topology(w) : test if the simplification of w would alter the topology of the contour curve (valence 4 case)

```
1: $(e_1, e_2, e_3, e_4) \leftarrow$ edges of $w$.star            // $e_2 \cup e_4$ is the subdivision edge
2: $(c_1, c_2, c_3, c_4) \leftarrow (e_1$.crossing, $e_2$.crossing, $e_3$.crossing, $e_4$.crossing$)$   // local topology
3: if $(c_1, c_2, c_3, c_4) =$ (true, true, true, true) then
4:     return false                 // (1) destruction of a connected component
5: end if
6: if $(c_1, c_2, c_3, c_4) =$ (true, false, true, false) then
7:     return false                 // (2) exchange of local connected components
8: end if
9: return true                                                 // simplification is safe
```

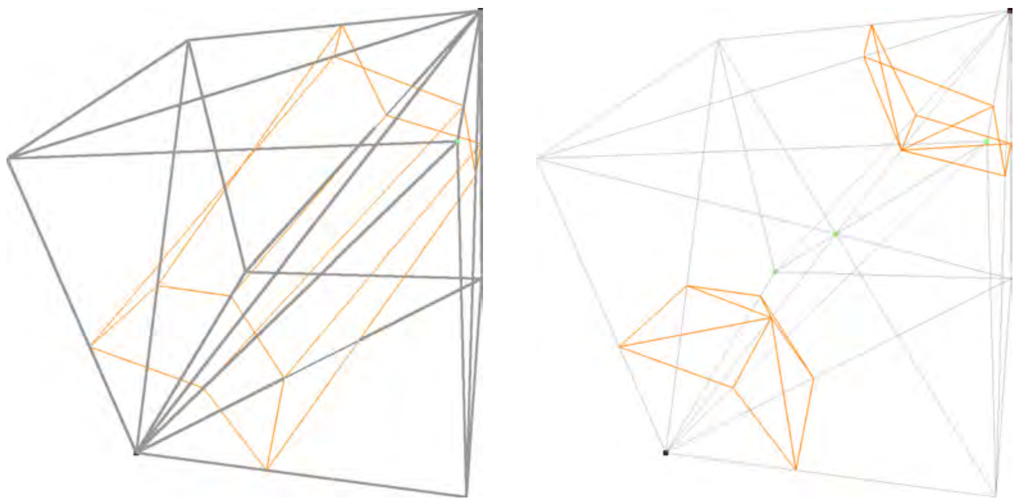
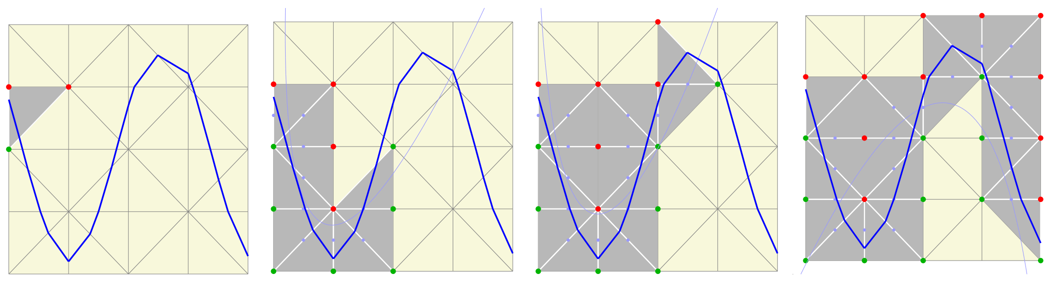


Figure V.10: The second prohibited case for isosurfaces.

Topology control for isosurfaces. For isosurfaces, the test can also be simplified and certified. If the refined edge e is crossed at most once, below or above the welded vertex w, the topology of the isosurface will not change during the weld. This is a sufficient condition, and covers the two cases for contour curves, as shown on Figure V.10. To obtain a necessary one, we compute, when the surface crosses e twice, the Euler characteristic χ of the isosurface in the star of w. The simplification is allowed only if $\chi = 1$.

V.2 Direct Encoding



(a) Location: level 4, 1, $L\ R\ L\ R$. $+--$. **(b) Vertex signs: $--++++++$.** **(c) 2 known vertices, then: $-+-$.** **(d) $----++-+$ $-End$.**

Figure V.11: Uniform encoding of the coarser resolution of a small sinusoid. The light curve is a second–order fitting of the decoder's points (in the middle of the crossing edges), and serves as geometrical predictor.

The whole compression process extracts the level set at the finest resolution, simplifies it to a coarser one, encodes this coarser resolution and the sequence of refinements operations. A direct compression scheme can be implemented by simply encoding the level set at a given resolution with the technique described in this section. This technique is also used for the compression of the coarse resolution, as an initial data for the successive refinements. When the tubular neighbourhood has constant size, as obtained on a regular grid or after the execution of the original version of Algorithm 11: global simplify, the direct encoding is simpler and more efficient.

The main idea is to encode the tubular neighbourhood $\mathbb{U}_K$ going along with the level set, from one crossing simplex to an adjacent one of $\mathbb{U}_K$. Similarly to the connectivity–driven compression, this allows good prediction mechanisms, since the local structure of the level set can be supposed to be smooth. The algorithm encodes first the localisation of an unvisited crossing simplex σ_0, and the signs of its vertices. From this initial simplex, it follows the level set by a traversal of its dual graph in a depth–first–search manner, encoding the sign of each unvisited vertex encountered. When the traversal is done, it continues on the next connected component. Notice that only the vertices of $\mathbb{U}_K$ have their sign encoded, leaving our algorithm almost independent of the initial size of the scalar data.

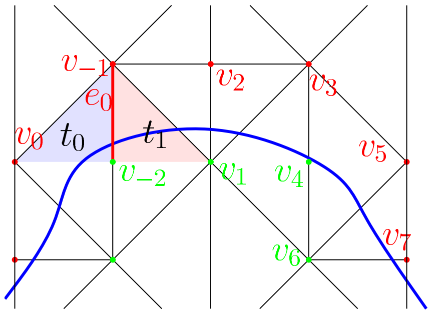


Figure V.12: Coarser resolution compression: the traversal goes from t_0 to t_1 through gate e_0, and encodes the sample point v_1.

Figure V.12 illustrates the idea by the use of a contour curve. The RBMT is spatially adapted to the contour curve. Once the initial triangle is detected, we encode in a certain sequence only the scalar value of the vertices on the tubular neighbourhood of the curve.

(a) Localisation

Algorithm 13 send localise(σ_0) : send localisation of σ_0

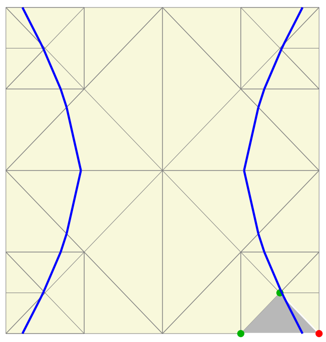
```
1: $\sigma \leftarrow \sigma_0$                                  // temporary variable
2: stack $RLstack \leftarrow \emptyset$          // stack of the positions of the ancestors of $\sigma_0$
3: while $\sigma$.parent $\neq \emptyset$ do                    // not reached the root
4:     $stack$.push($\sigma$.parent.left $= \sigma$)      // stores as Left (true) or Right (false)
5:     $\sigma \leftarrow \sigma$.parent                                  // one level up
6: end while
7: send(position ($\sigma, roots$))    // encodes which root of the RBMT binary tree
8: while $RLstack \neq \emptyset$ do                              // pops the stack
9:     send bit($top$.push)                  // encodes the Left or Right symbol
10: end while
```

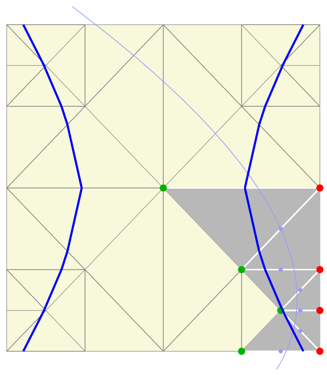
The location of the initial simplex σ_0 can be encoded using the binary tree inherent to the RBMT, as done by Algorithm 13: send localise. The root of the tree contains only a few simplices: 2 for a regular bidimensional grid, 6 for the tridimensional one. The root ancestor of σ_0 can therefore be encoded by its index with few bits. Then, knowing the level l of the simplex to encode, it can be localised using a sequence of l symbols Left and Right, as on Figure V.11(a), Figure V.13(a) and Figure V.13(d).

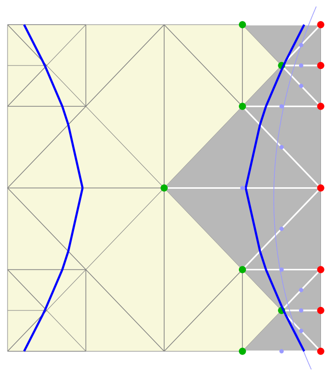
Then σ_0 and all its vertices are marked as visited, and their signs are encoded. Although the location is cheap to encode this way ($\log(\#_{roots}) + \log(l) + 2 \cdot l + 1$ bits), it is one of the most expensive parts of the compressed data. Since this has to be encoded for each connected component, topology preserving extraction for the last simplifications can enhance the compression ratio.

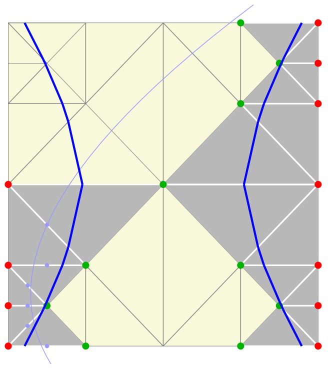
(b) Uniform Encoding

Once the signs of the vertices of the initial simplex σ_0 have been encoded, its crossing $n-1$–faces are known to the decoder. The crossing $n-1$–faces τ_i are then pushed into a stack. These faces are ordered by the level of their vertices in the RBMT, so that the decoder is synchronised with the encoder.


(a) Location: level 5**,** 0**,** $L\ R\ L\ L\ L$**.** $- + +$**.**


(b) Vertex levels and signs: $> - = - < + < - < +$**.**


(c) $= + > - > + > - = - < +$**.**


(d) level 5**, trig** 0**,** $L\ L\ L\ L\ L$**.** $- + +$**.** $> - = - < + < - <$**.**

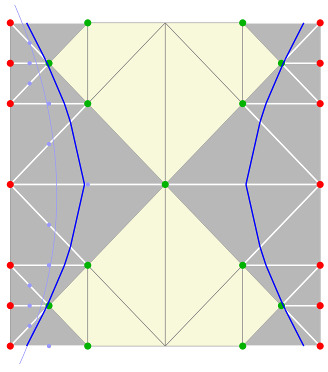

(e) $= + > - > + > - = - < +$*End***.**

Figure V.13: Adaptive encoding of the coarser resolution of a small hyperbola.

Then, the stack is popped and the first crossing $n-1$–faces of σ_0 will play the role of the gate τ_0, as described in Algorithm 14: receive coarse uniform. The other simplex σ_1 incident to the gate τ_0 is also a crossing simplex. The

Algorithm 14 receive coarse uniform : decodes the coarser level uniformly

```
1: stack $Gstack \leftarrow \emptyset$ ; $levelset \leftarrow \emptyset$   // traversal stack and neighbourhood
2: while $\sigma \leftarrow$ receive localise do   // new connected component
3:   $\sigma$.mark $\leftarrow$ true ; $\partial\sigma \cap K_{(0)}$.mark $\leftarrow$ true   // mark $\sigma$ and its vertices
4:   repeat   // traverse a connected component
5:     for all $\tau^{n-1} \in \partial\sigma$ do   // look for gates in order
6:       $\sigma' \leftarrow \mathrm{st}\,(\tau^{n-1}) \setminus \{\sigma\}$   // retrieves the next candidate
7:       if not $\sigma'$.mark and $\tau^{n-1}$.crossing then   // crossing new face
8:         Gstack.push($\tau^{n-1}, \sigma'$)   // push the next gate onto the stack
9:       end if
10:    end for
11:    repeat   // retrieve a new gate
12:      if $Gstack = \emptyset$ then   // empty stack
13:        break   // next component
14:      end if
15:      $(\tau, \sigma) \leftarrow Gstack$.pop   // next gate
16:    until not $\sigma$.mark
17:    $w \leftarrow \sigma \backslash \tau$   // apex of the new simplex
18:    if not $w$.mark then   // $w$ not visited yet
19:      $w$.mark $\leftarrow$ true ; $\mathfrak{f}(w) \leftarrow$ receive bit()   // decodes the sign of $w$
20:    end if
21:    $levelset \leftarrow \{levelset, (\tau, \sigma')\}$   // add to the decoded levelset in order
22:  until $Gstack = \emptyset$   // empty stack
23: end while
24: return $levelset$   // the ordered levelset is used for the refinements
```

algorithm encodes the sign of its apex, i.e. the vertex w_1 not contained in τ_0, so that the decoder knows its crossing faces. Both σ_1 and w_1 are marked as visited. The algorithm then pushed the crossing n–1–faces of σ_1, and continues the traversal by popping the next gate on the stack. This way, the traversal of the connected component of the level set only ends when the stack is empty.

Actually, a gate is valid only when the simplex σ_1 it reaches has not been visited yet, and the sign of the vertex w_1 is encoded only when it has not been marked (which can occur even when σ_1 has not been visited). This guarantees to encode exactly one sign bit per sample point, with an overhead of a few bits per connected components used for the localisation procedure.

(c) Adapted Encoding

The above algorithm can be easily modified to encode the tubular neighbourhood when it is composed of simplices of different levels. In that case, the algorithm also encodes the level of the current simplex σ_1 during the traversal. The decoder will read the required level for σ_1 and subdivide or simplify σ_1 if necessary. Since we defined the RBMT in order to maintain a difference of at most one level between adjacent simplices in Section III.3(c) *Multi–Triangulations*, the decoder will have to subdivide or simplify an unvisited simplex at most once (after line 16 of Algorithm 14: receive coarse uniform), and the encoder will have to send only one out of 3 symbols: $' =' $ when the levels match, $' >' $ or

$' <'$ when the levels differ. The encoding of the signs is done similarly to the uniform one.

This guarantees to encode exactly one bit per vertex of the tubular neighbourhood $\mathbb{U}_K$, plus one symbol of $\{=, <, >\}$ per simplex of $\mathbb{U}_K$, with an overhead of a few bits per connected components, used for the localisation procedure, as done on Figure V.13.

V.3 Progressive Encoding

Successive refinements reverse the multiresolution extraction, allowing a progressive adaptation of the tubular neighbourhood to the level set, using each time smaller simplices. The algorithm can encode the signs of the new vertices of $K' \leftsquigarrow K$ created by the subdivisions of all simplices of $\mathbb{U}_K$, or it can first specify which simplices to subdivide according to their order in the coarse level encoding, and then send the sign of the new vertices inserted, as done on Figure V.15. The refinements can also be performed by encoding the position of the level set directly inside $\mathbb{U}_K$, which becomes cheaper in high dimensions and allows sharing information with close–by level sets.

(a) Combinatorial Refinement

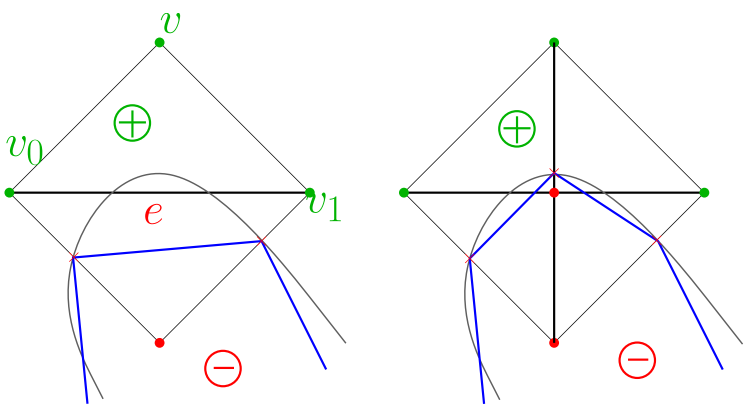
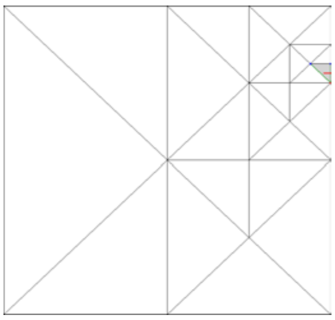
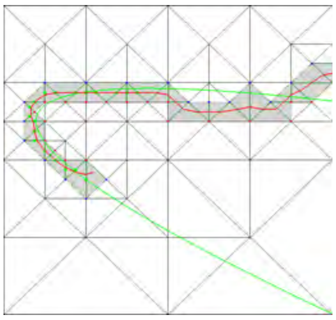


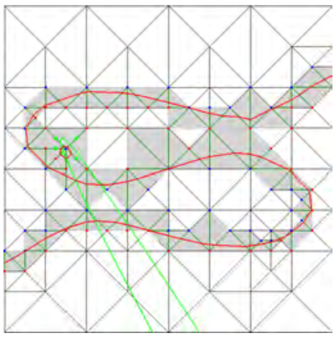
Figure V.14: A subdivision can extend the tubular neighbourhood of the level set if the sign of $vv_0 > 0$ and $vv_1 > 0$.

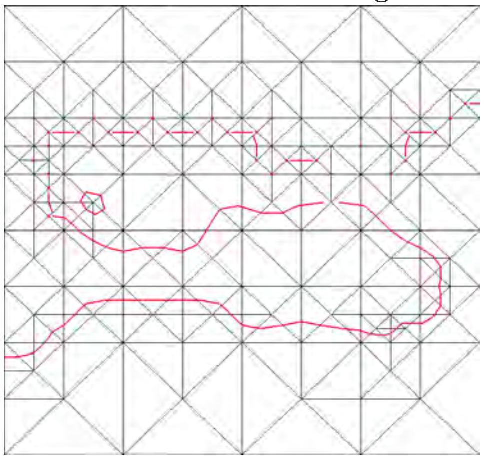
The uniform refinement simply subdivides first all the simplices of $\mathbb{U}_K$, and then encodes the signs of the vertices of $\mathbb{U}_{K'}$ belonging to the former tubular neighbourhood $\mathbb{U}_K$ created by the subdivision. The order of the new vertices is induced by the traversal used for the coarser resolution encoding.

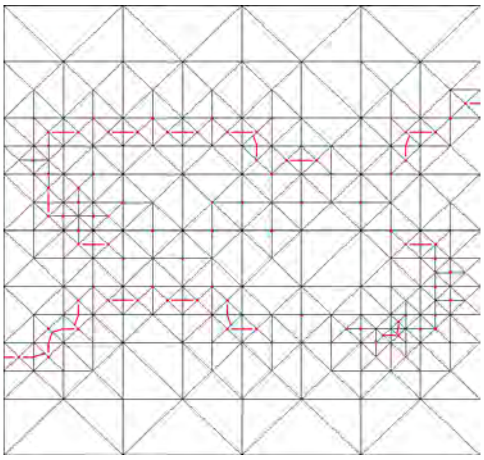
When subdividing a simplex crossing the contour curve, the position of the subdivided simplices with respect to the contour curve is determined by the sign of the new vertex introduced on the subdivision edge. This sign is encoded during the progression. However, this subdivision can locally extend the tubular neighbourhood: $\mathbb{U}_{K'} \not\subset \mathbb{U}_K$, as on Figure V.14. This case occurs when the new vertex w is inserted inside a non–crossing face at the boundary of $\mathbb{U}_K$. In that case, $\mathbb{U}_{K'}$ is extended to enclose the new vertex w and its star, but the signs of the star do not need to be encoded, since they are either

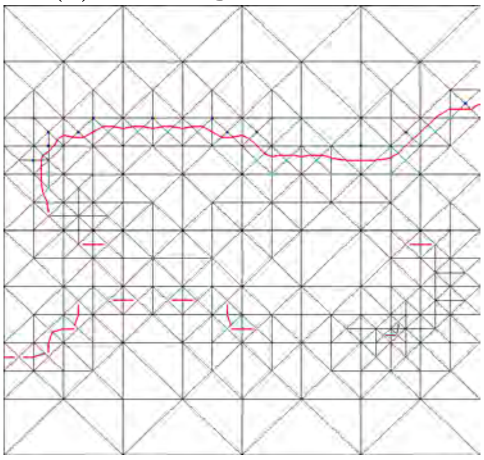

(a) Location.


(b) Adaptive decoding...


(c) ...with prediction


(d) Decoding refinements...


(e) ...the curve is incomplete


(f) Decoding signs of new vertices...

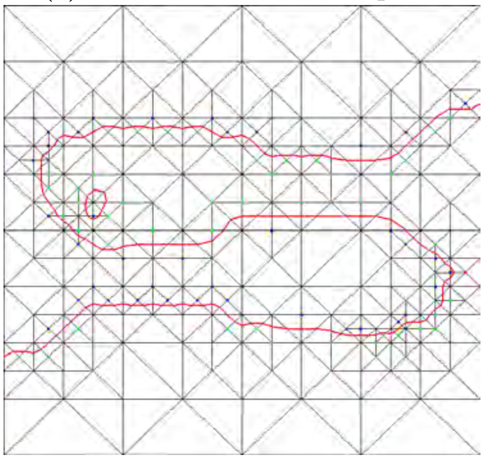

(g) ...until completing the curve.

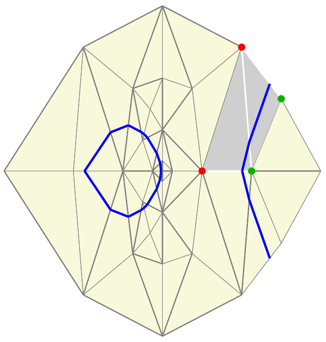
Figure V.15: Direct decoding, followed by a progressive refinements of a fish curve.

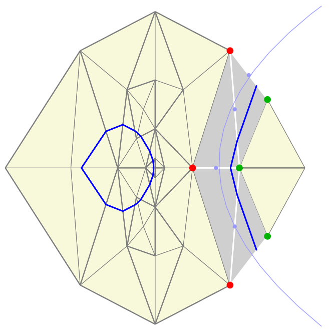
already encoded or opposed to the sign of the w. Otherwise, there would have been an uncoded connected component at the former level of detail.

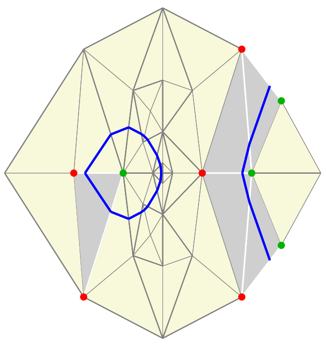
Our method also allows an adaptive progression, creating smaller simplices where the level set is more complex, and leaving bigger simplices where it is simple. There are only a fraction of simplices of the tubular neighbourhood $\mathbb{U}_K$ that can be refined: only the one with the lowest level. For each of these simplices, taken in the order of the encoding returned by Algorithm 14: receive coarse uniform, we encode a Refine or Keep code on one bit. When a Keep symbol is encoded, the simplex will not be considered for future refinements, keeping it as a leaf on the binary tree of the RBMT. The simplices for Refine codes are refined, and the sign of the newly inserted vertices are encoded in the same way as for uniform refinement. This procedure is repeated until all simplices are marked with a Keep code.

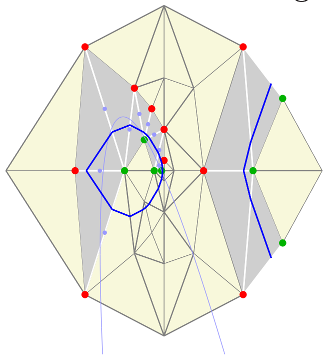
(b) Extrinsic Geometry Refinement

Once the tubular neighbourhood $\mathbb{U}_K$ of a resolution level is encoded, only one bit for each vertex of $\mathbb{U}_K$ is known to the decoder. Therefore, the position of each vertex v of the level set mesh is *a priori* set to the middle point of its crossing edge e. This can be improved by sending a more precise value of $\mathfrak{f}$ on the endpoints of e.


(a) Location: level 6, 1, $L\ L\ L\ L\ L\ R.\ -++.\ <-$.


(b) Vertex levels and signs: $= - > +$.


(c) level 5, **trig** 0, $L\ L\ L\ L\ L$.


(d) $-+-.\ = - = - > + > - = - < + = -$.

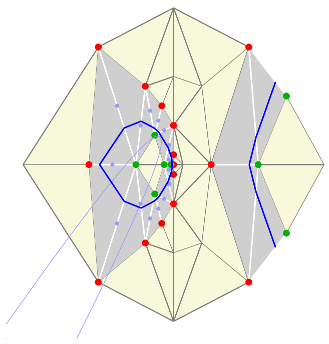

(e) $> - = - = - =< + = - > - = + << End$.

Figure V.16: Adaptive encoding of the coarser resolution of a cubic: irregular tessellation can reduce the distortion.

The input scalar data regularity can actually influence the quality of the compression, as illustrated on Figure V.16. For specific applications such as human face contour compression in images, the RBMT can be mapped onto a non–regular model given by a few set of parameters, such as the mean and variances of the original level set, for example an elliptic–radial initial distribution of the scalar data for human faces.

Geometry and combinatorial refinements are concurrent and complementary. It is not clear to us yet how much information each one adds at different levels. *A priori*, geometry refinements are more efficient in high dimensions, where there are many simplices for only a few scalar points.

(c) Level Sets Extensions

The scalar value given by the extrinsic geometry refinements can be used to encode other level sets corresponding to small translations $\mathfrak{f}'(w) = \mathfrak{f}(w) + \varepsilon$ of $\mathfrak{f}$. By decoding ε, we can deduce the sign of many vertices for $\mathfrak{f}'$ before any new transmission. This feature can be very useful in medical applications, where the contrast of measure planes varies inside the grid. The algorithm to complete the level set at the same resolution as $\mathfrak{f}$ is then almost identical to Algorithm 14: receive coarse uniform.

V.4 Tuning and Results

We presented in this chapter a novel approach for level set compression, which works for arbitrary dimension. The following sections show results for isocontour and isosurfaces, and comparisons with the state–of–the–art method for isosurface [Lee *et al.* 2003]. Note that our compression and decompression has, in the worst case, a linear complexity with respect to the size of the input data (the image or the volume for isocurve and isosurfaces), both in execution time and memory footprint. However the expected execution time rather depends on the size of the level set. We used the arithmetic coder of Section II.2 *Arithmetic Coding* for the final coding, with an implementation based on the one used by [Lee *et al.* 2003].

(a) Prediction and Statistical Models

Direct encoding. The sign of the vertex being decoded can be predicted by approximating the decoded level set and extrapolating this approximation. For example for contour curves, we fit a parabola on the last 20 decoded curve vertices using [Lewiner *et al.* 2004*1]. For the decoder, the contour vertices are at the midpoints of each crossing edge, and the interpolated parabola on these edges midpoints passes closer to one of the two unknown edges the triangle on top of the stack, as shown on Figure V.17. The algorithm predicts that this edge will be the next crossing edge. Moreover, the distance to the central vertex allows defining 3 different cases, each of which generates a specific probability model through the context mechanism described at Section II.2(c) *Statistical Modelling*. This prediction succeeds in average in 80% of the cases, as described in [Lewiner *et al.* 2005] and on table V.1. We complete this prediction with an order 0 arithmetic coder.

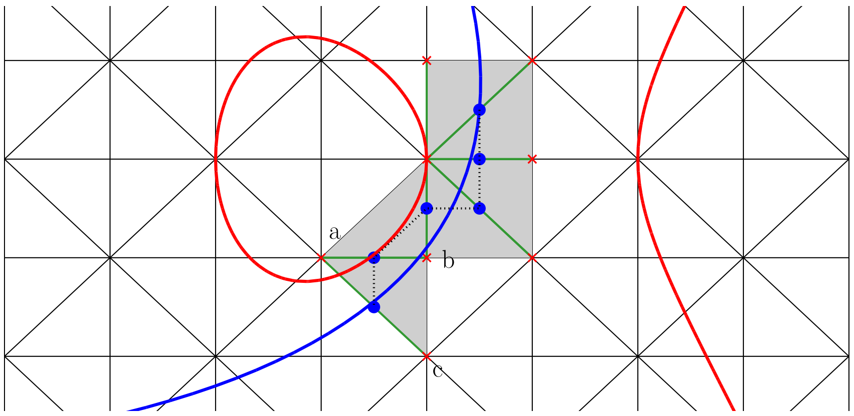


Figure V.17: "Singular" curve: the predictor must guess whether the curve will traverse edge ac or bc, approximating the real curve (in red) by a parabola (blue curve) using only the points known to the decoder (in blue), i.e. midpoints of the edges crossing the curve.

Adaptive codes. The encoding schemes we introduced here goes along with the level set to compress. Therefore, the next symbols to encode are closely related to the symbols just encoded before. This fact gives a priori more accurate probability models for the context–based arithmetic encoding. The context model can be enhanced with a good initialisation. For example, we can give more probability to the $=$ symbol. Moreover, the encoder does not encode a RBMT grid vertex twice, although it visits it more than once. This multiple visit can be used to update the probability model of the context, even without encoding the corresponding symbol. This accelerates the learning delay of the arithmetic coder, without any cost.

Combinatorial refinements. For the combinatorial refinements, we distinguish two cases for the vertex w created during the refinement process: either w is on the boundary of $\mathbb{U}_K$, or inside $\mathbb{U}_K$. On the boundary, the general case is not to extend $\mathbb{U}_K$. Inside $\mathbb{U}_K$, the prediction is made using the parabola fitting of [Lewiner *et al.* 2004*1, Lewiner *et al.* 2005]. We used an order 2 arithmetic coder for the refinements.

Extrinsic geometry. The extrinsic geometry can be predicted by smoothing. However, for the contour curves case, since we already used the parabola fitting with success, we use it again for the prediction of the geometry: the reconstruction of the curve can be enhanced by a smoothing operation, moving the intersection of the curve with the simplicial grid away from the edge midpoint. This should reduce the distortion of the decoded curve. In order to perform this operation on–the–fly, the approximation used by the predictor is directly used to compute the new intersection, as on Figure V.18. This smoothing behaves well when the curve is regular and well sampled, as shown on table V.1). We used an order 1 arithmetic coder for the geometry transmission. For the general case, the probability model is initialised with a Gaussian around zero.

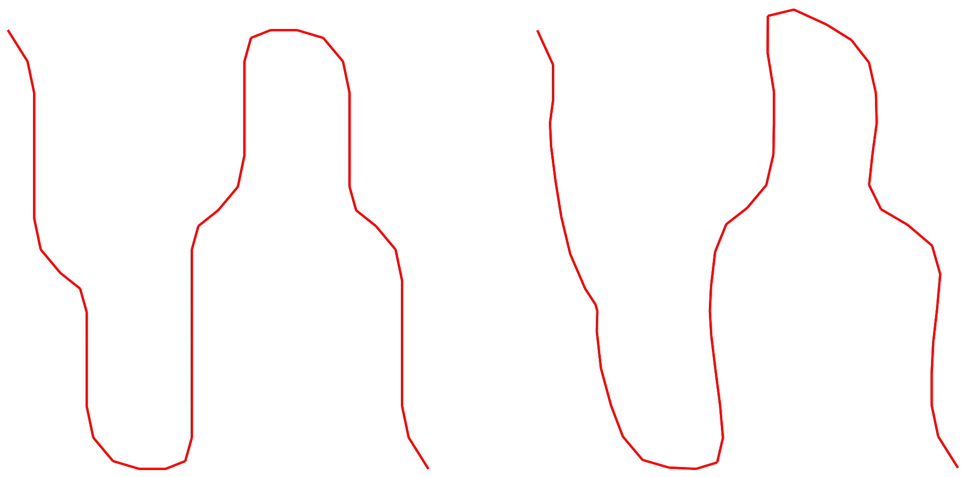


Figure V.18: A decompressed implicit sinoide without smoothing (left) and with the predictor on–the–fly smoothing (right).

Curve	Predict success	Size (bits) w/out	with	gain	Distortion (10^{-3}) w/out	with	gain
bicorn	80.3 %	1479	1014	46 %	2.26	2.52	–10 %
circle	80.6 %	1658	1106	50 %	2.38	2.20	8 %
cubic	81.4 %	566	324	75 %	3.48	3.99	–13 %
ellipse	82.5 %	1430	920	55 %	2.03	1.99	2 %
singular	77.9 %	1501	1099	37 %	1.43	1.36	5 %
hyperbola	80.2 %	1648	1164	42 %	1.53	1.39	10 %
sinoide	84.5 %	2645	1612	64 %	0.91	0.76	20 %

Table V.1: Results of the predictor for the contour curves case: for each one of the implicit curves, the number of successes over the erroneous guesses of the predictor is around 80%. The gain in the size of the compressed curve is half of the compressed size. The smoothing induced by the predictor reduces the distortion for most of the case. All the results were obtained as means of the same model over various resolution of the grid, from 50 to 3000 vertices, with $q = 84$.

(b) Direct

The methods we introduced here are flexible and can be used in many ways, and resulted in nice rate/distortion curves, as on Figure V.22. For contour curves in images, we started from the complete triangulation of the pixels. For implicit curves, we sent in–between two levels of detail a part of the geometry (2 bits). For any dimensions, single rate encoding is, as usual, a little more efficient than progressive encoding.

(c) Progressive

The different controls on the adaptation of the multiresolution, such as those of Figure V.19, can have a significant cost, as shown on Figure V.22(a). For example, on regular models such as the elevation curve of the Sugar Loaf drawn on Figure V.20, the distortion and topology controls provide nicer results to the eyes at the beginning of the compression than uniform refinements, while

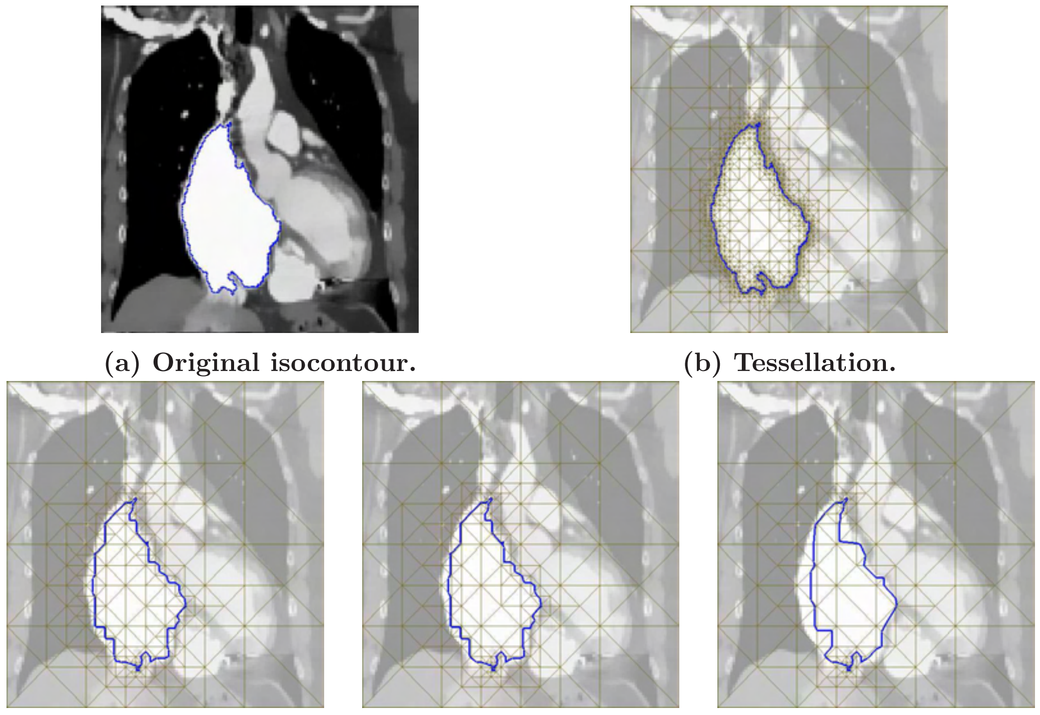

(a) Original isocontour. (b) Tessellation.
(c) Curvature. (d) Distortion. (e) Topology.

Figure V.19: Extraction of a contour curve with different controls.

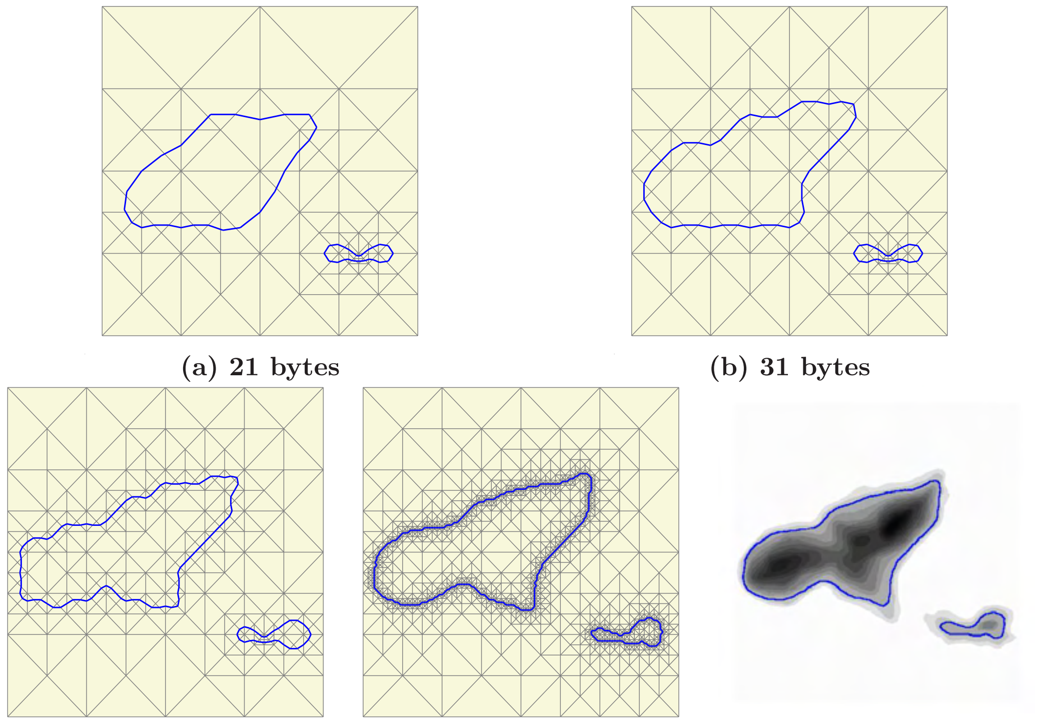
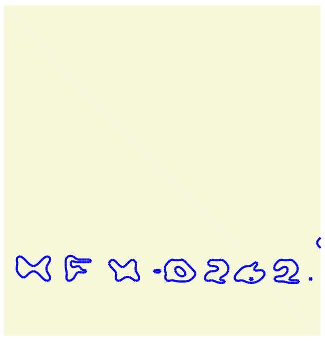
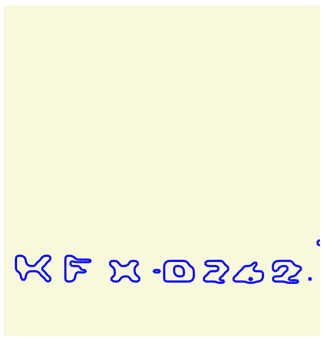

(a) 21 bytes (b) 31 bytes
(c) 46 bytes (d) 156 bytes (e) 690 bytes

Figure V.20: Progressive compression of an elevation curve of the Sugar Loaf. The tessellation is adapted to the curve to minimize the distortion and to preserve the topology.

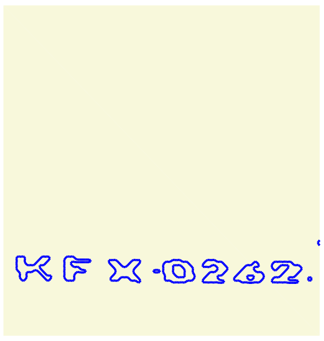
finally resulting in similar performances. Topology control means an extra cost depending on the complexity of the model: compare the car number plate



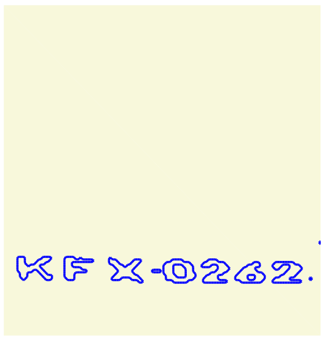
(a) 97 bytes



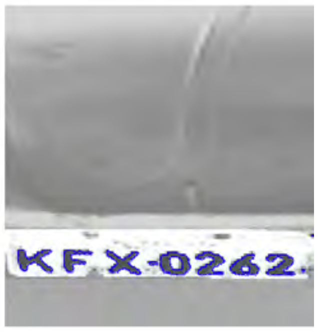
(b) 129 bytes



(c) 164 bytes

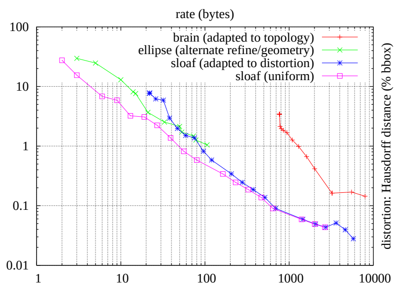


(d) 223 bytes

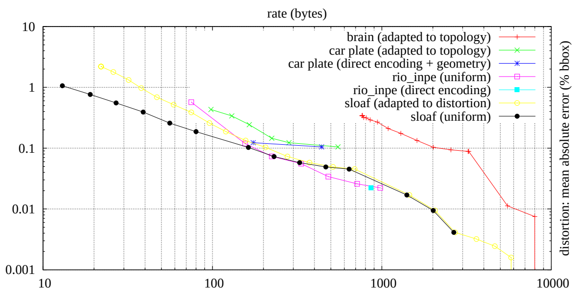


(e) 283 bytes

Figure V.21: Progressive compression of a car number plate (with enhanced contrast), topology controlled: direct encoding (173 bytes) and progressive encoding (283 bytes) both with good compression ratio.



(a) Semi-Hausdorff distance.



(b) Mean absolute error.

Figure V.22: Compression results: on complex models, topology control is mode expensive with progressive encoding.

of Figure V.21 with the brain model of Figure V.1 on Figure V.22. However, the rate/distortion performances have a similar evolution for all methods and models, as shown on Figure V.22(a). For simple curves, the geometry encoding seems a good alternative to refinements for regular models such as the ellipse on Figure V.22(a), and for higher dimensions.

Octree [Lee *et al.* 2003]				**S**implicial				
	level	rate	dist		level	rate	dist	
							brute	smooth
				Coarse	15	220	11.01	6.27
Comb	5	191	68.16	Comb	16	349	4.15	4.34
Comb	7	1 563	9.06	Comb	17	651	3.19	3.29
Comb	8	4 854	4.44	Comb	18	934	2.70	2.72
				Comb	19	1 462	2.02	1.97
				Comb	20	2 637	1.69	1.59
				Comb	21	3 816	1.49	1.41
				Comb	22	5 908	1.20	1.08
				Comb	23	10 595	0.98	0.87
				quantisation (bits)				
Geom	30%	21 138	2.79	Geom	2	18 966	0.91	
Geom	100%	53 706	0.52	Geom	4	34 060	0.27	
				Geom	8	78 096	0.04	
				Geom	16	158 566	0.04	
				Geom	20	201 548	0.04	

Table V.2: Comparison of the octree–based uniform refinement strategy from [Lee *et al.* 2003] and our method with the same strategy on the horse model.

For isosurfaces, [Lee *et al.* 2003] use the dual marching method [Ju *et al.* 2002] to reduce the amount of geometry to encode (only one vertex per crossing cube), which force the geometry to be sent at the end. We tested the same kind of methods for the comparisons of Table V.2, with uniform refinements and the geometry encoding as a final step.

Figure V.23 shows our rate/distortion curves for other models with the same strategy, and Figure I.5 and Figure V.24 illustrates it. Figure V.25 shows the rate/distortion ratios for different distortion threshold and different geometry quantisation. The threshold is well respected and the rate/distortion distributions converge to an optimum.

But since our method is more flexible, we experienced using distortion control for multiresolution encoding, by alternating refinement and geometry encoding pass. This is illustrated on Figure V.26, where the base mesh is encoded as in the previous single rate application, and the refinements are sent adaptively. At each level, we encode the full geometry of the new vertices of the tubular neighbourhood (8 bits). The resulting distortion converges quickly in that case, which goes in the direction of the assertion of Section V.3(b) *Extrinsic Geometry Refinement* on the efficiency of geometry coding in higher dimensions.

(d) Extended applications

This method already compresses level sets in any dimension. It could detect transmission errors during the initial coarse encoding better

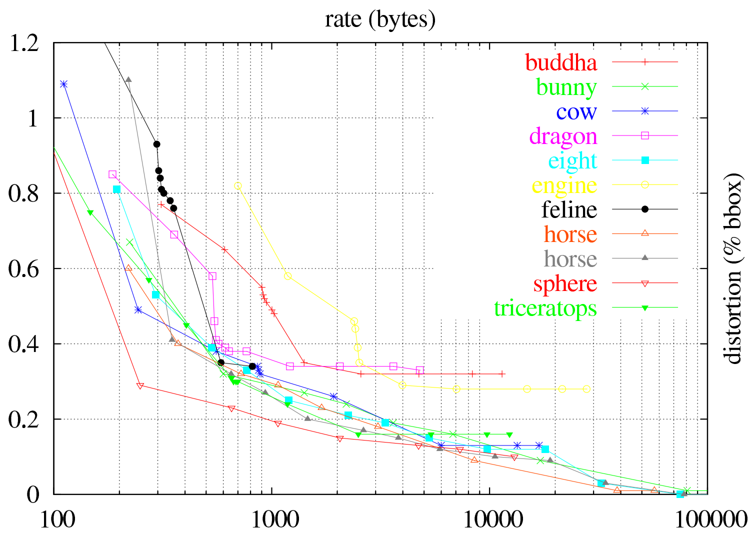


Figure V.23: Rate/distortion curves on various models

than [Lee *et al.* 2003], since the level set traversal would probably not close if the transmitted signs are altered. However, it would not be straightforward to correct these errors. In the refinement pass, errors would probably not be detected since the refinements are mainly local and the global topology can be preserved independently.

This method naturally extends to codimension higher than one, for example curves in space, by transmitting several signs for each RBMT vertex of the tubular neighbourhood. For animation applications, in particular for physical simulations where level sets are commonly used for integrating partial differential equations (PDE), our level set compression can be used directly in two ways. First, a deformed level set corresponds to a level set with one more dimension (the time). This compression can be expensive, particularly in memory, and would not use the coherence of the level set during its evolution. The second option would be to use the method described at Section V.3(c) *Level Sets Extensions*, to encode the level set of each instant as a local deformation of the one at the previous instant. In particular for PDE integration, the proper structure of the PDE can be used to improve the prediction mechanisms.

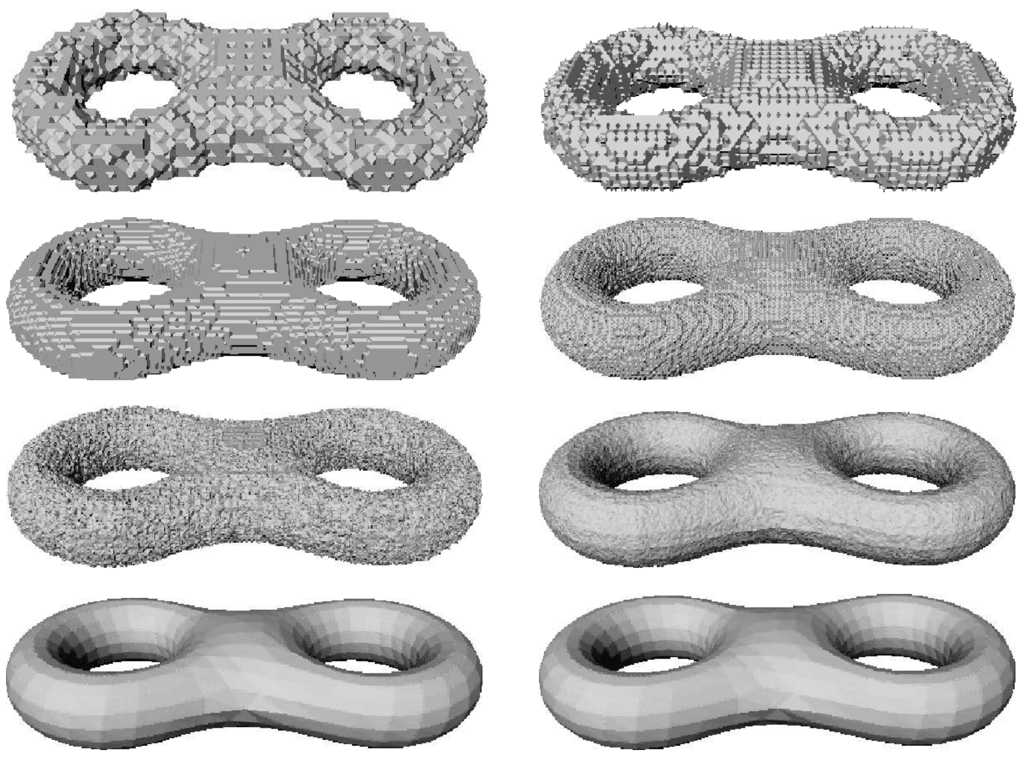


Figure V.24: Decompression of the eight model as a 257^3 regular sampled data. The first images correspond to the binary multi–triangulation refinement, followed by the geometric refinements.

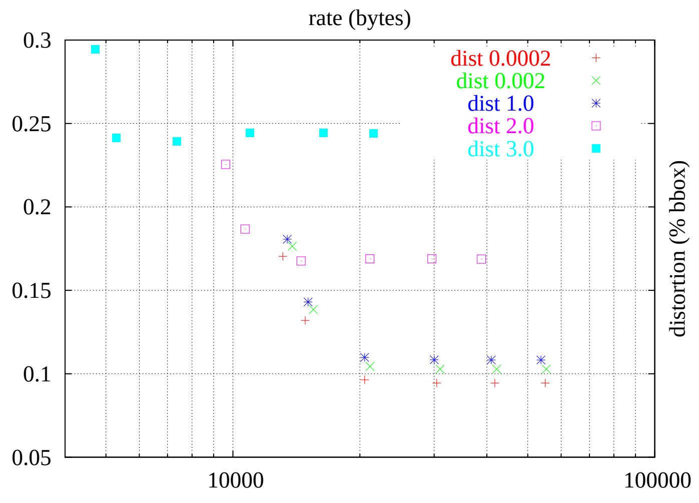
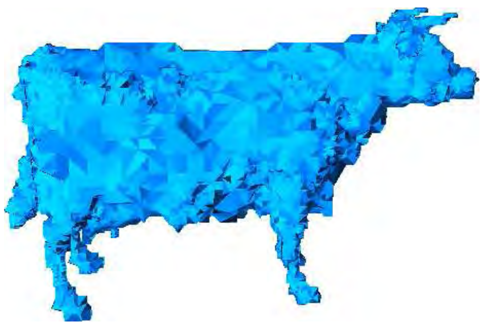
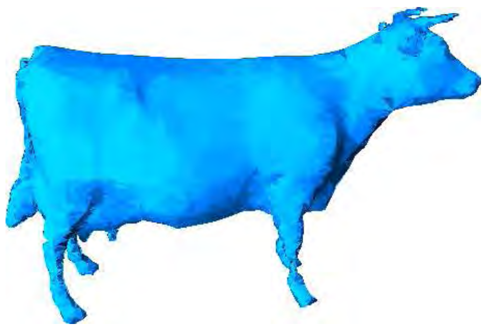
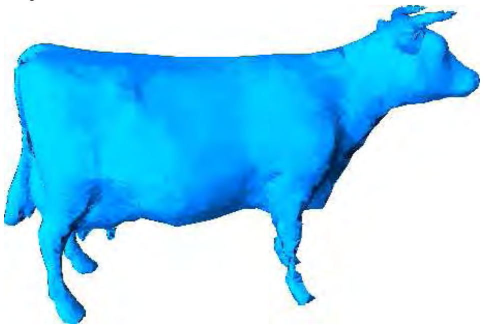


Figure V.25: Rate/distortion curves of single rate encoding of the happy buddha model, with different distortion thresholds and geometry quantisations


(a) base mesh : threshold 3.0, 2258 bytes


(b) base mesh with geometry: 5439 bytes


(c) first refinement: threshold 0.2, 7353 bytes

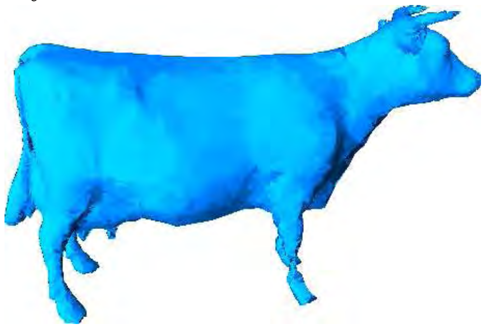

(d) geometry of the first refinement: 1135 bytes

Figure V.26: Progressive encoding according to the distortion, with alternate refinement/geometry levels.

VI GEncode

Among the mesh compression algorithms, different schemes compress better specific categories of model. In particular, we saw in the last chapter that geometry–driven approaches have outstanding performances on isosurfaces. It would be expected these algorithm to also encode well meshes reconstructed from the geometry, or optimised by a geometric re–meshing. These meshes adapted to geometry are usually expensive to compute, either in the reconstruction case, as for example with [Amenta *et al.* 2001] or the remeshing one as in [Alliez *et al.* 2003, Alliez *et al.* 2003]. Moreover, their geometry is sometimes known independently of their connectivity, such as scanning results, and the only encoding of their connectivity can be useful in these contexts.

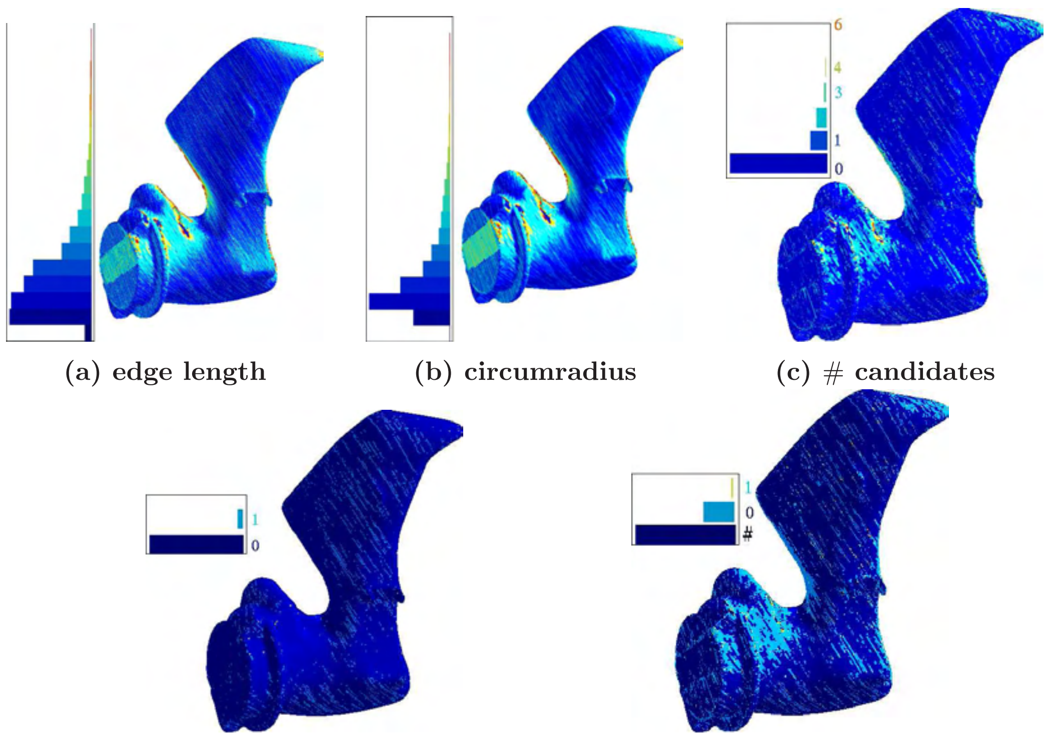
We will present now GEncode, first introduced in [Lewiner *et al.* 2005*1], which is a direct mesh compression scheme that compresses the connectivity of these meshes at almost zero–cost. This scheme is very general, and deals with simplicial complexes of arbitrary dimension in arbitrary ambient space, even if the complex is non–pure, non–manifold or non–orientable. Compression results for surfaces are competitive with existing connectivity–driven compression schemes.

VI.1 Purposes

(a) Focus on Geometrical Meshes

We will focus on *geometrical meshes*, i.e. meshes whose connectivity can partly deduced from its geometry, as for reconstructed scans or remeshed models, and meshes of high dimension or with a complex topology. In these cases, connectivity–driven compression approaches should be less efficient on the connectivity encoding. On one side, the connectivity of geometrical meshes can be decoded from the geometry, and thus the connectivity does not even need to be encoded. On the other side, connectivity codes become exponentially complex with the dimension, while geometry–driven connectivity codes handle gracefully complex topology, as we will detail now. These particularities were already stated in [Gandoin and Devillers 2002], which was the first geometry–driven progressive compression scheme.

Moreover, we would like our scheme to be predictable and flexible. More precisely, since this scheme is adapted to a specific category of meshes, as connectivity–driven compression or level sets compression wer adapted to other categories, we would like to know by a simple test if our algorithm will



(a) edge length (b) circumradius (c) # candidates

(d) quantised criterion (e) candidate position

Figure VI.1: Compression of a scanned mechanical piece: once the geometry is decoded, the decoder attaches a triangle $e \star w$ to the active bounding edge e. The vertex w is identified by the circumradius $\rho(e \star w)$ of the original triangle, divided by the length of e in order to reduce the entropy. The position of w in the list of vertices with the same quantised criterion is then encoded.

be performing before the whole compression. Furthermore, depending on the mesh type, i.e. computer aided design meshes, finite element method supports, scanned meshes, remeshed models…, the algorithm should use different *geometric criterion* to be compatible with the mesh generation properties. We will focus on the *Ball Pivoting criterion* since it is easy to understand and to implement in any dimension, but the algorithm works with any geometric criterion.

In the case of scanned models, the reconstruction algorithm computes the connectivity of the surface only from the geometry of its vertices. Therefore, a geometry–driven approach using the same reconstruction algorithm should not waste any byte in connectivity. The proposed scheme achieves this, by encoding a mesh based on a local geometric criterion.

(b) Zero Cost for Reconstructible Meshes

Compression overview. The proposed encoding scheme works similarly to reconstruction algorithms, although it encodes continuously the differences of the original and reconstructed mesh. It first encodes the geometry of each vertex, in any ambient space, using a simple *octree coding* that is a synthesis of both [Gandoin and Devillers 2002, Botsch *et al.* 2002], as for example on Figure I.6. The algorithm then encodes the connectivity of an n–manifold propagating from an initial cell an advancing front triangulation, attaching at each step an n–cell $\tau^{n-1} \star \{w_j\}$ to a bounding $(n-1)$–cell τ^{n-1}.

Deviation encoding. The difference between the original and reconstructed mesh is that w is not always the one that minimises the *geometric criterion* $\mathcal{G}\left(\tau^{n-1}, w\right)$ of the reconstruction procedure. This difference is encoded by the position of w in a list of *candidates*. These candidates are chosen from a quantised bound on $\mathcal{G}\left(\tau^{n-1}, w\right)$.

Entropy for reconstructible meshes. To minimise the entropy (with the definition of Section II.1(b) *Information Theory*), that list is ordered by the geometric criterion. Therefore, when the mesh is reconstructible by the given geometric criterion, the candidate is always the first one and the quantisation generates only one class. Therefore, the symbols to encode are always the same for that case, and the message have a zero entropy, which is what we intended.

General meshes. When the geometric criterion corresponds to the construction of the mesh, the entropy is almost zero. Otherwise, general meshes require to find an adapted geometric criterion in order to encode the deviation between the reconstruction and the original model as described above. From a geometrical point of view, usual meshes are not so general, since their simplices are relatively well shaped and connect mainly neighbour vertices. Their local geometry is therefore relatively simple, which motivates looking for parametric geometric criterion to fit the mesh geometry.

Extensions. From the connectivity point of view, the algorithm can be easily extended. The generic algorithm does not need to be modified when elevating the dimension or the codimension. For non–simplicial meshes, the generic attachment can require more than one vertex. The first one is encoded as previously, and the other ones are selected in order to belong to the same hyperplane, and in increasing order for the geometric criterion. For non–manifold n–meshes, many n–cells can be attached to the same $(n-1)$–cell of the active border. For non–pure meshes, the encoding performs successively for each dimension, completing progressively the highest dimensional structure with lower ones.

(c) Independent Geometry Encoding

Paradigm. For the GEncode algorithm, the geometry is supposed to be known before the connectivity is encoded. This can be the case for a practical application or for artificial sampling. However, for generic cases, the separate encoding of the geometry turns out to be expensive with actual techniques. Mixed step of geometry/connectivity encoding could greatly improve the overall compression ratio, but this is still ongoing work. Moreover, since point sets encoding is still a new area, the cost of the geometry encoding should reduce in a close future. For scanned objects, the points are usually aligned on parallel planes, similarly to medical imaging reconstruction [Boissonnat and Geiger 1993], in which case the geometry encoding can be improved. For generic case, the geometry can be encoded as a general multidimensional signal, using for example the optimal coding of [Craizer *et al.* 2002*1], or by some space subdivision techniques.

-	Lower vs Higher		Lower vs Higher vs Ours		
nv	Lower	Higher	Lower	Higher	Ours
0 - 49	81%	19%	0%	12%	88%
50 - 99	100%	0%	60%	0%	40%
100 - 199	95%	5%	14%	4%	82%
200 - 499	100%	0%	40%	0%	60%
500 - 999	93%	7%	67%	0%	33%
1000 - 1999	86%	14%	67%	0%	33%

Table VI.1: Percentage of models on which each geometry encoded performed best, grouped by model size: the lower node efficient method of [Gandoin and Devillers 2002] performs better on bigger models, particularly when compared only to the higher node efficient method of [Botsch *et al.* 2002]. Our method achieves integrating the benefits of [Botsch *et al.* 2002], and partially of [Gandoin and Devillers 2002].

Space subdivision encoding. We used the last option, with a synthesis on [Gandoin and Devillers 2002] and [Botsch *et al.* 2002], which uses a simple tree coding technique. This techniques works for vertices with an arbitrary number of coordinates, allowing encoding meshes of arbitrary codimension. As in [Gandoin and Devillers 2002] and [Botsch *et al.* 2002], the space is divided with a particular binary space partition where each separator is perpendicular to the axis (i.e. a octree): the axis alternates from one level to the other (X,Y,Z,X,Y... in $\mathbb{R}^3$), and the separator is always positioned at the barycenter of the cell represented by the node, as for Figure VI.2, Figure VI.3 and Figure VI.4. The subdivision is performed until each node contains only one vertex.

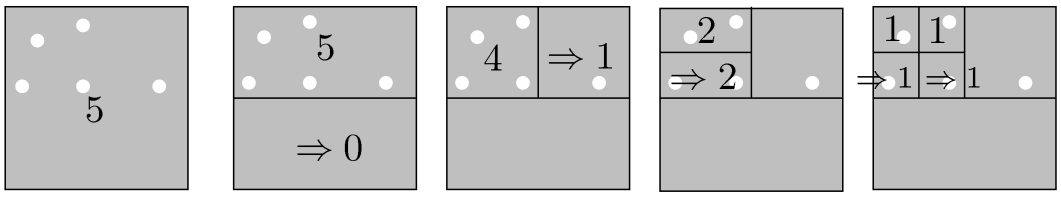


Figure VI.2: Geometry encoding of [Gandoin and Devillers 2002]: the encoder encodes 5 on 32 bits, then 5 on $\lceil \log_2(6) \rceil$ bits, then 4 on $\lceil \log_2(6) \rceil$ bits. The right vertex position is then encoded. Then 2 is encoded on $\lceil \log_2(5) \rceil$ bits, 1 on 1 bit and 1 bit again. Then each remaining vertex is encoded.

Lower nodes efficient. In [Gandoin and Devillers 2002], each node of the binary space partition is encoded by the number of vertices $\#v_l$ its left children contains. The number of vertices of the other node $\#v_r$ is simply deduced by difference from the number of the known vertices of the parent's node $\#v_f$. This technique wastes many bits at the beginning of the encoding, as the number of nodes must be encoded on $\lceil log_2(\#v_f + 1)\rceil$ bits, as for the example of Figure VI.2. When there is only one vertex per node, is is encoded with 1 bit per level, which is optimal.

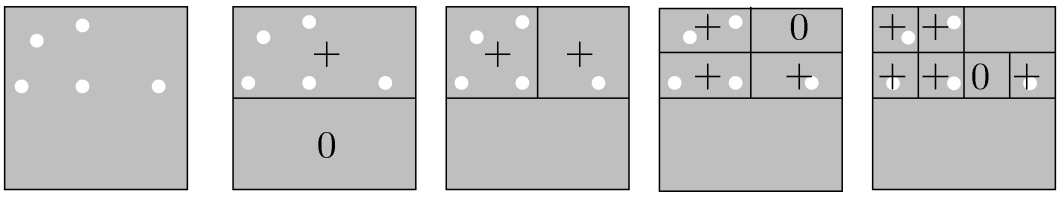


Figure VI.3: Geometry encoding of [Botsch *et al.* 2002]: the encoder encodes the following sequence: +0, ++, ++, 0+, ++, ++, 0+**. Then follow** 0+ **and** +0 **to reach the the desired number of bits.**

Higher nodes efficient. In [Botsch *et al.* 2002], 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 vertexs, as for the example of Figure VI.3. Note that at least one child must contain a vertex, as the parent did. The encoding stops at a predefined level. This method spends more bits at the end of the encoding, since the decoder doesn't 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 [Gandoin and Devillers 2002] for the last part.

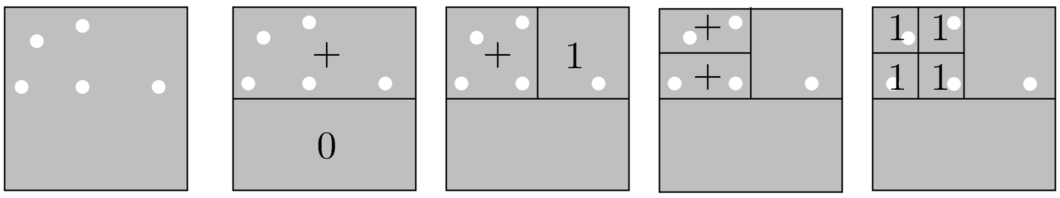


Figure VI.4: Our geometry encoding: the encoder encodes the following sequence: +0, +1 **and the right vertex is fully encoded. Then** ++, 11, 11. **The position of each remaining vertex is then encoded.**

Synthesis. Our technique takes 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 emptys, as for the example of Figure VI.4. With this encoding, the encoder detects when there is only one vertex in a node, and then uses the technique of [Gandoin and Devillers 2002]. The symbols do not have the same probability and the coder takes that into account. 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. For example, the 1986 vertices of the solid sphere of Figure I.6, left, were compressed by [Gandoin and Devillers 2002] with 3433 bytes, by [Botsch *et al.* 2002] with 3877 bytes and by our method with 3429 bytes. The 192 vertices of the Cartesian product of a sphere and a circle of Figure I.6, right, were compressed by [Gandoin and Devillers 2002] with 649 bytes, by [Botsch *et al.* 2002] with 980 bytes and by our method with 646 bytes. The coloured lines represent part of the traversal of the mesh by the connectivity encoder.

VI.2 GEncode schemes

(a) Advancing front compression

Any mesh can actually be constructed by a sequence of general attachments, as defined in Section III.2(a) *Euler Operators*. This property is intensively used in surface reconstruction with advancing front triangulation algorithms [Medeiros *et al.* 2003], and will be the base of our algorithm. The mesh is built step by step, each step consisting in attaching a cell to the boundary of the last step, called the *active boundary*. The cell attached is the join of a cell τ^{n-1} of the active boundary with some vertices w_i. For example for triangulated surfaces, the triangle attached is the join of an edge of the active boundary with one vertex. Since the geometry of all the vertices is known, the problem is to identify which vertices w_i are used for the attachment, and we will identify each w_i by its position j inside a list of *candidates*: $w_j \in \{v_1, \ldots, v_k\}$. This process is implemented by Algorithm 15: gencode and Algorithm 16: gdecode. For these algorithms, we suppose that there is at most one cell spanning a given set of vertices.

(b) Optimisations and Extensions

Generality of GEncode. The above algorithms are actually very general, and can be simplified for specific cases. In its present forms, it handles simplicial complexes and polytopes of arbitrary dimension, embedded in any codimensional space. The complex can be orientable or not, and have any kind of topology. It can respect the manifold or pseudo–manifold criterion or not. We will see how each of these restrictions can simplify the algorithm, and improve the final compression ration. Such improvements do not change the nature of the GEncode, but accelerates its execution. For example, we can push only the unmarked $n-1$–cells of the frontier of the attached cells at lines 33 and 29 of Algorithm 15: gencode and Algorithm 16: gdecode respectively, in order to maintain a smaller priority queue and to avoid the test of lines 7 and 6 of Algorithm 15: gencode and Algorithm 16: gdecode respectively.

Algorithm 15 gencode(): encodes one component of a pure n–polytope.

```
1: $Gqueue \leftarrow \varnothing$ // priority queue of the (n−1)–cells of the active boundary
2: $\sigma^n \leftarrow$ initial_cell () // first cell of the component
3: encode_first ($\sigma^n$) // encode the first cell by its geometry
4: $Gqueue$.push ($\partial\sigma^n$) // initial active boundary
5: while $Gqueue \neq \varnothing$ do // main loop
6:    $\tau^{n-1} \leftarrow Gqueue$.pop // the active (n−1)–cell of highest priority
7:    if $\tau^{n-1}$.mark then // already encoded
8:       continue // try next
9:    end if
10:   for all $\sigma^n \in \dot{\mathrm{st}}(\tau^{n-1})$ do // look for an uncoded cell incident to $\tau^{n-1}$
11:      if $\sigma^n$.mark then // already encoded
12:         continue // try next
13:      end if
14:      $\{w_1, \ldots, w_m\} \leftarrow$ apex ($\sigma^n, \tau^{n-1}$) // vertices to be encoded
15:      encode ($m$) // encode the complementary degree of the cell
16:
17:      sort ($\{w_1, \ldots, w_m\}, \mathcal{G}$) ; $g \leftarrow \mathcal{G}(\tau^{n-1}, w_1)$ // minimal apex for $\mathcal{G}$
18:      $[\mathcal{G}_{\min}, \mathcal{G}_{\max}] \leftarrow$ quantise ($g$) // quantisation interval for g
19:      encode ($\mathcal{G}_{\min}, \mathcal{G}_{\max}$) // encode the quantised criterion
20:      $\{v_1, \ldots, v_k\} \leftarrow$ candidates ($\tau^{n-1}, [\mathcal{G}_{\min}, \mathcal{G}_{\max}]$) ) // look for candidates
21:      sort ($\{v_1, \ldots, v_k\}, \mathcal{G}$) // optimal candidate first
22:      encode ($i : w_1 = v_i$) // encode the position of $w_1$ as a candidate
23:
24:      $[\mathcal{G}_{\min}, \mathcal{G}_{\max}] \leftarrow$ quantise ($\{w_2, \ldots, w_m\}$) // quantisation of $\mathcal{G}$ for the
         other vertices
25:      encode ($\mathcal{G}_{\max}$) // encode the quantised criterion
26:      for all $j \in [\![2, m]\!]$ do // encode the other vertices of the cell
27:         $\{v_1, \ldots, v_k\} \leftarrow$ candidates ($\tau^{n-1}, \{w_1, w_{j-1}\}, [\mathcal{G}_{\min}, \mathcal{G}_{\max}]$) ) // look
            for candidates
28:         sort ($\{v_1, \ldots, v_k\}, \mathcal{G}$) // optimal candidate first
29:         encode ($i : w_j = v_i$) // encode the position of $w_j$ as a candidate
30:      end for
31:
32:      $\sigma^n$.mark $\leftarrow$ true // mark the encoded cell
33:      $Gqueue$.push ($\partial\sigma^n$) // update the active boundary
34:   end for
35:   $\tau^{n-1}$.mark $\leftarrow$ true // mark the encoded active boundary cell
36:   encode ($-1$) // end of the edge star
37: end while
```

Algorithm 16 gdecode(): decodes one component of a pure n–polytope.

```
1: $Gqueue \leftarrow \varnothing$ // priority queue of the $(n-1)$–cells of the active boundary
2: $\sigma^n \leftarrow$ decode_first () // decode the first cell of the component
3: $Gqueue$.push $(\partial\sigma^n)$ // initial active boundary
4: while $Gqueue \neq \varnothing$ do // main loop
5:   $\tau^{n-1} \leftarrow Gqueue$.pop // the active $(n-1)$–cell of highest priority
6:   if $\tau^{n-1}$.mark then // already encoded
7:     continue // try next
8:   end if
9:   while true do // uncoded simplices incident to $\tau^{n-1}$
10:     $m \leftarrow$ decode () // decode the complementary degree of the cell
11:     if $m = -1$ then // no new incident simplices
12:       break // next cell of the active boundary
13:     end if
14:
15:     $\mathcal{G}_{\max} \leftarrow$ decode () // decode the quantised criterion
16:     $\{v_1, \ldots, v_k\} \leftarrow$ candidates $(\tau^{n-1}, [\mathcal{G}_{\min}, \mathcal{G}_{\max}])$ ) // look for candidates
17:     sort $(\{v_1, \ldots, v_k\}, \mathcal{G})$ // optimal candidate first
18:     $i \leftarrow$ decode () ; $w_1 \leftarrow v_i$ // decode the position of $w_1$ as a candidate
19:
20:     $(\mathcal{G}_{\min}, \mathcal{G}_{\max}) \leftarrow$ decode () // decode the quantised criterion
21:     for all $j \in [\![2, m]\!]$ do // encode the other vertices of the cell
22:       $\{v_1, \ldots, v_k\} \leftarrow$ candidates $(\tau^{n-1}, \{w_1, w_{j-1}\}, [\mathcal{G}_{\min}, \mathcal{G}_{\max}])$ ) // look
          for candidates
23:       sort $(\{v_1, \ldots, v_k\}, \mathcal{G})$ // optimal candidate first
24:       $i \leftarrow$ decode ; $w_1 \leftarrow v_i$ // encode the position of $w_j$ as a candidate
25:     end for
26:
27:     $\sigma^n \leftarrow$ attach $(\tau^{n-1}, \tau^{n-1} \star \{w_1, \ldots, w_m\})$ // attach the new cell
28:     $\sigma^n$.mark $\leftarrow$ true // mark the decoded cell
29:     $Gqueue$.push $(\partial\sigma^n)$ // update the active boundary
30:   end while
31:   $\tau^{n-1}$.mark $\leftarrow$ true // mark the decoded boundary cell
32: end while
```

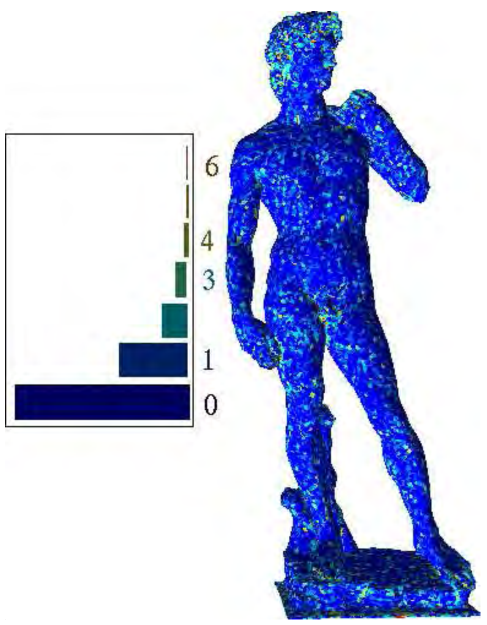
Simplicial meshes. For simplicial meshes, we know that each n–simplex has $n+1$ vertices. Therefore the m symbol is always equal to 1 at lines 15 and 10 of Algorithm 15: gencode and Algorithm 16: gdecode respectively. Moreover, the coding of the extra vertices $\{w_2, w_{m-1}\}$ can be avoided, which suppresses lines 16 to 31 of Algorithm 15: gencode, and lines 19 to 26 of Algorithm 16: gdecode. For generic purposes, this can be implemented with the same algorithms by incorporating this information in the probability model of the m symbols, either at initialisation or progressively, as described in Section II.2(c) *Statistical Modelling*.

Manifolds. If the mesh is a pseudo–manifold with an empty border, we know that each $n-1$–cell is incident to exactly two n–cells. Therefore, there is no need to encode the -1 symbol at lines 36 and 11 of Algorithm 15: gencode and Algorithm 16: gdecode respectively, nor to include the loops starting at lines 10 and 9 of Algorithm 15: gencode and Algorithm 16: gdecode respectively, that wait for this symbol. Furthermore, if we know the original mesh is manifold, we can remove from the list the candidates that would create a non–manifold mesh, i.e. the vertices that are not in the active boundary.

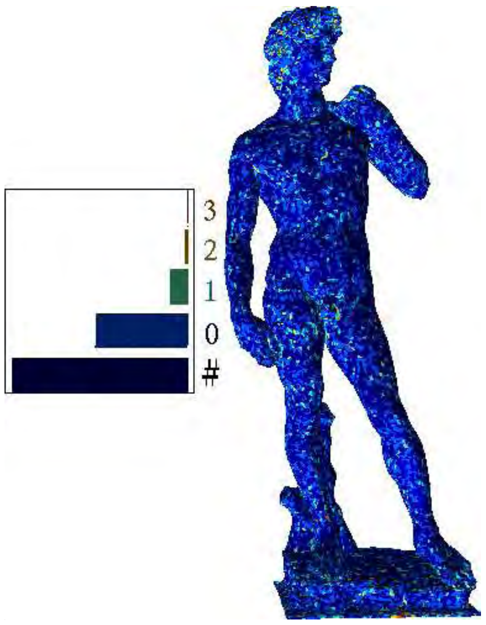
Non–pure complexes. The above algorithms can be easily extended to handle multiple components and non–pure elements by calling Algorithm 15: gencode and Algorithm 16: gdecode many times. For the first case, the algorithm is called once per component using the function initial_cell at line 2 of Algorithm 15: gencode, and a by encoding a bit signal Stop/Continue at the end of the procedure. Non–pure elements of dimension $p < n$, i.e. p–cells not face of any $(p+1)$–cell, can actually be considered as pure elements of the p–skeleton $K_{(p)}$, as defined at Section III.1(a) *Simplicial Complexes*. The algorithm then encodes the pure part K^{pure} of the polytope with Algorithm 15: gencode, and then successively encodes the pure parts $K^{pure}_{(p)}$ of the lower skeletons $K_{(p)}$, starting with the boundary of $K^{pure}_{(p+1)}$ as active boundary, and considering each cell of $K^{pure}_{(p+1)}$ as already encoded. This involves at most $n-1$ calls to Algorithm 15: gencode or Algorithm 16: gdecode.

(c) Candidates selection

Quantisation. The decoder has to choose which vertices w of the encoded mesh was adjacent to τ^{n-1}. This information is encoded by the index of w inside a list of candidates. This list could be the list of all vertices, but that would require $O(\log(\#_0))$ bits for each vertex, which is too expensive. In order to minimise the number of elements of the candidate list and the time used to create that list, the geometric criterion $\mathcal{G}(\tau^{n-1}, w)$ of w is quantised and encoded. This quantisation is a tradeoff: On one hand, if the quantisation is brutal, there will be many candidates, which means a long time to generate the list and an expensive encoding of the position of w in the list of the candidates, as show the histograms of Figure VI.5. On the other hand, if the quantisation is too refined, the quantised geometric criterion will be expensive



(a) # candidates



(b) candidate position

Figure VI.5: The quantisation of the geometrical criterion affects the entropy of the candidate position.

to encode. Therefore, a given geometric criterion must be associated with a adapted quantisation scheme.

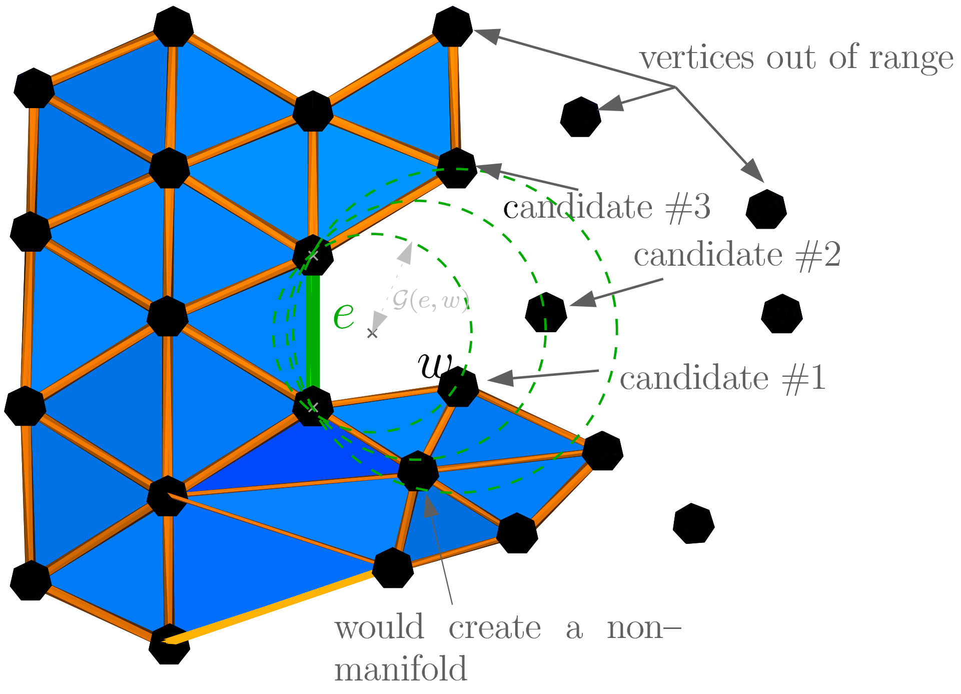


Figure VI.6: Candidate selection from a given geometric criterion (here the circumradius).

Ordering. The quantised geometric criterion is just an interval $[\mathcal{G}_{\min}, \mathcal{G}_{\max}]$ that contains $\mathcal{G}(\tau^{n-1}, w)$. The decoder then enumerates all the candidates

$v_i \notin \tau^{n-1}$ that could fit in this interval : $\mathcal{G}(\tau^{n-1}, v_i) \in [\mathcal{G}_{\min}, \mathcal{G}_{\max}]$, and order this list of candidates by $\mathcal{G}$: $\mathcal{G}(\tau^{n-1}, v_i) < \mathcal{G}(\tau^{n-1}, v_{i+1})$, as shown on Figure VI.6. For typical meshes, this ordering lowers drastically the entropy, since the vertex w is usually one of the first ones, and the entropy coder can reflect this using predefined probabilities, as described in Section II.2(c) *Statistical Modelling*.

Non–simplicial meshes. In the implementation, the binary space partition created by the encoding of the geometry is used to accelerate the search for candidates, when the range of the geometric criterion can be bound in the ambient space. However, the number of candidates for the other vertices $\{w_2, w_{m-1}\}$ can be further reduced : On one hand, these vertices belong to the hyperplane containing $\tau^{n-1} \star w$, and not to the hyperplane supporting τ^{n-1}. This criterion translates directly on the binary space partition, and greatly accelerates the search. On the other hand, the vertices w_j are taken in increasing order of $\mathcal{G}$, which means that the lower bound of the quantisation interval is reduced at each step.

VI.3 Geometric Criteria

For reconstruction purposes, the vertex w chosen to join the simplex τ^{n-1} minimises a geometric criterion $\mathcal{G}(\tau^{n-1}, w) = \mathcal{G}_{\min}(\tau^{n-1})$. For example, the Ball Pivoting Algorithm of [Bernardini *et al.* 1999, Medeiros *et al.* 2004] uses the *circumradius* of the simplex $\tau^{n-1} \star w$ as geometric criterion: $\mathcal{G}(\tau^{n-1}, w) = \rho(\tau^{n-1} \star w)$.

This geometric criterion is a fundamental piece of the GEncode scheme, since it must predict the local rules to deduce the connectivity from the geometry. In this section, we will describe which are the important elements of a geometric criterion, and detail these by the specific example of a criterion derived from the Ball Pivoting Algorithm. We will conclude by some indications to a practical scheme for deducing a *ad hoc* criterion for a given mesh.

(a) Generic Formulation

Purpose. Consider a fixed mesh K. A geometric criterion is a real valued function $\mathcal{G}(\tau^{n-1}, w)$ depending on an active $(n-1)$–simplex τ^{n-1}, a candidate vertex w and eventually any vertex of the complete mesh and the already decoded mesh. The geometry criterion is adapted to K if, for a fixed active simplex τ^{n-1}, this function assigns the lowest possible values to the apex w of an n–simplex $\tau^{n-1} \star w$ of K. This would characterise the mesh local connectivity from its geometry, but for practical use in our GEncode scheme, the criterion must satisfy other

Quantisation/enumeration tradeoff. In order to avoid generating huge lists of candidates, the geometric criterion is quantised into a few classes. This quantisation should optimise the tradeoff between the number of candidates generated and the number of classes. A simple class repartition mechanism would be an exponential increase of the class size, giving more precision for the more frequent elements. However, this should be specified for each geometric criterion.

Spatial extension. The search for candidates can be accelerated using the binary space partition created by the encoding of the geometry described at Section VI.1(c) *Independent Geometry Encoding*. To do so, a range of the geometric criterion should correspond to a spatial restriction, in order to look for candidates in only a specific part of the space. This generates a significant acceleration of the algorithm.

Traversal strategy. The geometric configuration of a simplex σ^n differs if looking at it from an apex w ($\sigma^n = \tau \star w$) or another one w' ($\sigma^n = \tau' \star w'$), as shown on Figure VI.8. The geometric criterion can perform better if entering in a simplex σ^n by a specific face τ^{n-1}. That is why the active boundary is implemented by a priority queue in Algorithm 15: gencode and Algorithm 16: gdecode. The geometric criterion can thus be enhanced by a specific priority for the faces of the active boundary.

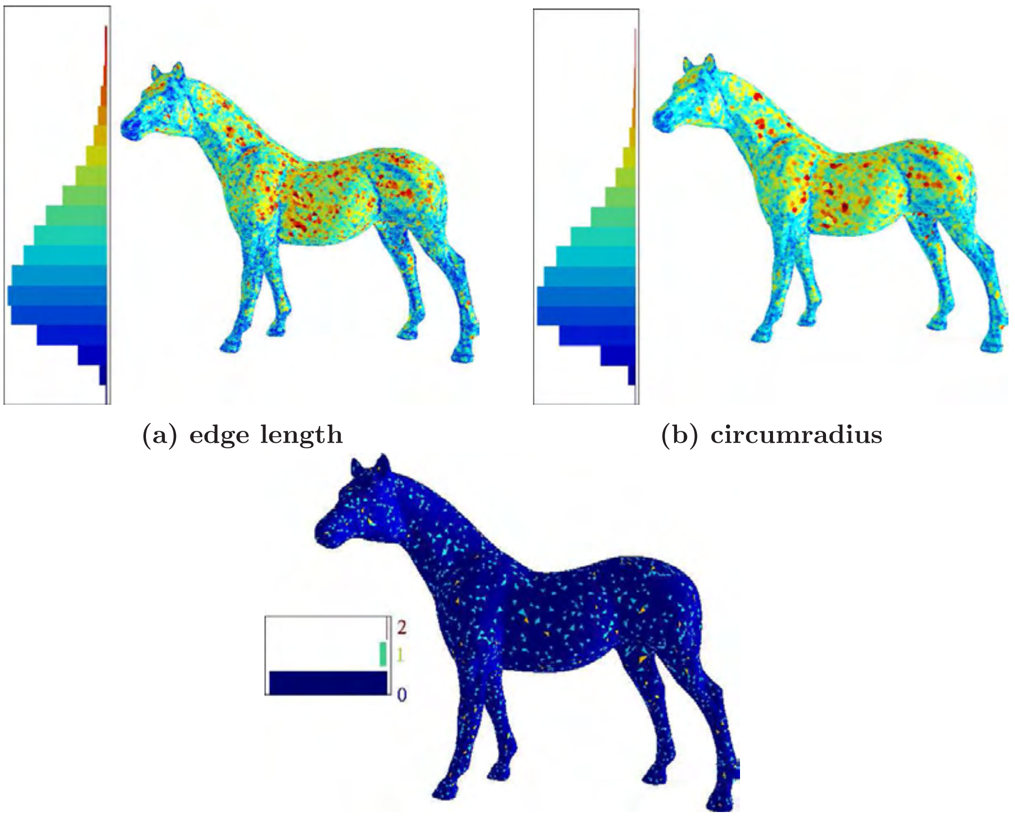
(b) Ball–Pivoting and Delaunay–Based Reconstruction

Delaunay–based meshes. In order to illustrate the GEncode scheme with a practical geometric criterion, we will present a criterion derived from the simple and efficient reconstruction algorithm of [Bernardini *et al.* 1999]: the Ball Pivoting described at Section III.3(b) *Delaunay Triangulation*. For meshes that were reconstructed with this algorithm, or with similar Delaunay–based reconstruction algorithms as [Boissonnat and Geiger 1993, Lopes *et al.* 2000, Chaine and Bouakaz 2000, Boissonnat and Cazals 2002, Cohen–Steiner and Da 2002, Nonato *et al.* 2005], this criterion will be particularly efficient.

Ball Pivoting criterion. For surfaces, the Ball Pivoting algorithm always attaches to an active boundary edge τ^1 the vertex w that minimizes the circumradius $\rho(t)$ of the triangle $t = \tau^1 \star w$. This can be extended to higher dimensions, as was done in [Medeiros *et al.* 2004]. The circumradius can be computed in high dimensions using derivations of the Cayley–Menger determinant [Blumenthal 1970]. In practise, this computation is expensive, and can be substituted by the volume $|\sigma|$ of $\sigma = \tau^{n-1} \star w$, using directly the Cayley–Menger determinant.

Quantisation. The quantisation can actually be improved if we consider the circumradius divided by the length of the active face as show the histograms of Figure VI.7 :

$$\mathcal{G}\left(\tau^{n-1}, w\right) = \frac{\rho\left(\tau^{n-1} \star w\right)}{|\tau^{n-1}|} \quad .$$



(a) edge length **(b) circumradius**

(c) quantised radius

Figure VI.7: Whereas the circumradius of a simple mesh has a high entropy, its circumradius divided by the edge length has a very low entropy and can be efficiently quantised. However, when some of the edges are linked to a distant vertex, as in Figure VI.9, their quantisation becomes expensive.

As $\rho\left(\tau^{n-1} \star w\right)$ is the circumradius, $\mathcal{G}\left(\tau^{n-1}, w\right) > \frac{1}{2}$. Then, from $\frac{1}{2}$, the criterion is quantised in an exponential manner, with speed k that, experimentally, was optimum around 7:

$$\mathcal{G}_{\max}\left(\tau^{n-1}, w\right) = \left\lceil log_2\left(\mathcal{G}\left(\tau^{n-1}, w\right) - \frac{k}{2}\right)\right\rceil, \qquad k = 7.$$

Spatial bound. Moreover, with this quantisation, the candidates can be searched only in a ball centred of radius proportional to $\mathcal{G}_{\max}$, centered at the midpoint of τ^{n-1}. The proportion depends on the dimension of the ambient space.

Traversal priority. These kinds of optimisations actually depend of which face of the active boundary is chosen at each step. In particular for our criterion will be better quantised if the volume of the active face is bigger, as it optimises the entropy of $\mathcal{G}$, conforming the examples of Figure VI.8. This gives a direct key for the priority of the queue representing the active boundary.

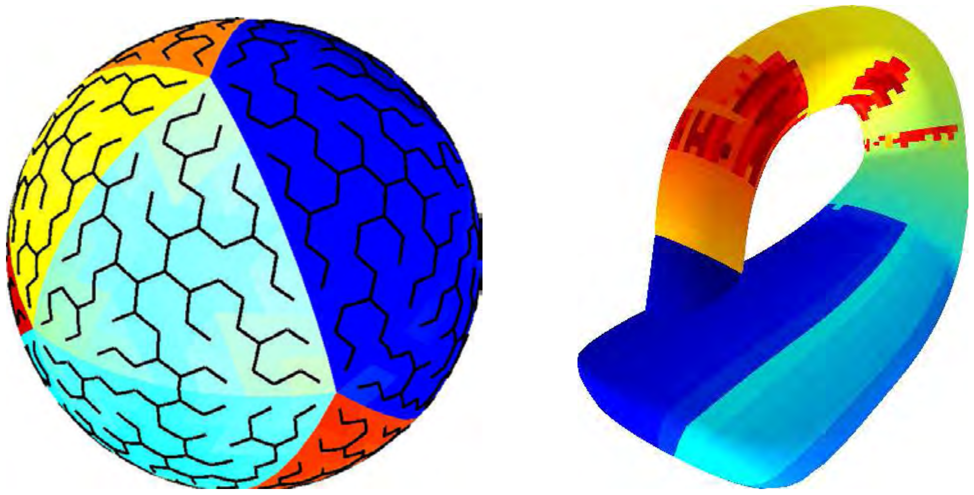


Figure VI.8: Traversal of a sphere and of a Klein bottle models, from cold to hot colours: good orders can improve the compression.

(c) Parametric Criterion

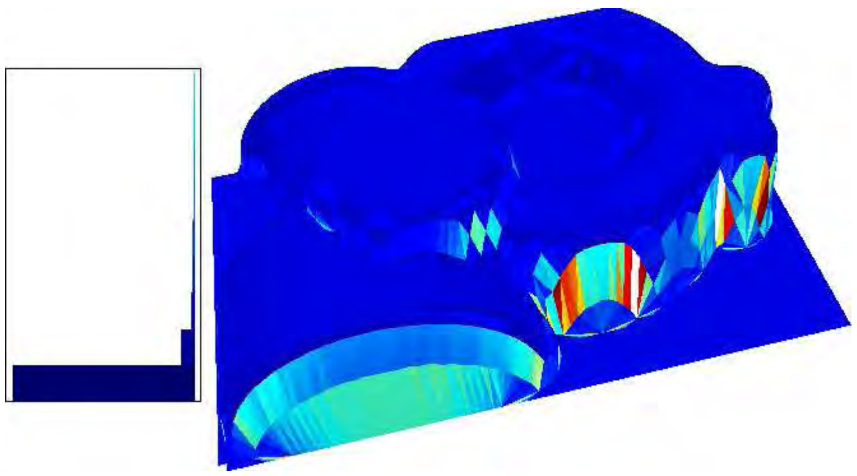


Figure VI.9: The circumradius encoding can be expensive for meshes generated with a criterion different from $\mathcal{G}$.

Mesh–dependent automatic tuning. The ball pivoting criterion that we just introduced is efficient for usual meshes of Computer Graphics. However, it is obviously not universal, and in particular not adapted to Computer Aided Design meshes, as the one of Figure VI.9. More generally, each mesh would have a specific criterion, ideally requiring only one quantisation class and assigning to the right candidate a local minimum. This kind of criterion can be arbitrarily complex to describe, and its transmission to the decoder can be even more expensive than the encoding with a simpler criterion. This results in another tradeoff between increasing the complexity of the criterion to get closer to an ideal criterion that would encode the mesh at zero cost, and reducing its complexity to avoid an expensive transmission of the criterion.

Parametric criterion. Actually, for a given mesh and eventually a given traversal of the mesh, we can create a sampling of the criterion $\mathcal{G} : \mathbb{R}^{nd} \times \mathbb{R}^{d} \to \mathbb{R}$ by assigning to each gate τ^{n-1} one sample per valid vertex w of the mesh. A vertex is valid if it does not create a degenerated cell, and for the manifold case, if it belongs to the active boundary during the traversal. The gate is considered as a simplex by eventually omitting some vertices, in order to simplify the sampling task. The domain of the criterion is actually projected in order to put the gate τ^{n-1} at the origin and aligned with the axes. The real value associated with the sample (τ^{n-1}, w) is somehow arbitrary, and can be 0 if the vertex is the apex of a valid cell of the mesh, and 1 otherwise. This value can be scaled by the distance of the vertex to the gate in order to get a better fit, and far–away vertices can be omitted to accelerate the process. The criterion is then a parametric fit of these samples, for example using [Pratt 1987]. For example, a low–degree polynomial or rational fraction can be used, which would encompass the Ball Pivoting criterion.

Quantised class creation. The quantisation can be directly deduced by summing for each value of the criterion the position of the right apex for each gate inside the list of candidates of higher criterion. The quantisation can optimally allocate quantisation intervals to have the lowest average value inside each class.

Spatial division. We can compute the maximal distance of a valid vertex to the centre of a gate directly for each quantisation bound. This information can be transmitted directly since there are only a few quantisation intervals. Eventually, these distances are affected by the volume of the gate, which can be taken in account here, similarly to the Ball Pivoting criterion.

Traversal guiding. The traversal strategy is somehow harder to deduce automatically, mainly because the whole criterion depends on the traversal. Since the traversal should be guided by only the gates, we should start with one priority for a gate, such as the volume or the orientation, and draw the error of the criterion as a function of that priority function. Then, a simple transformation, such as a multiplication by -1 or a recentring on an interval $[a, b]$ $(x \mapsto (x - a)\,(b - x))$, can outline a lower error on higher value of the transformed priority. If the difference between low and high values is not significant, another priority could be used.

Practical use. This method is time consuming in practise, which prevents its direct use for each mesh to encode. However, it can be used to generate criteria for a specific class of meshes. For example, this process can be applied to a sequence of meshes generated for a specific finite element method. The resulting criterion would then be hard coded to form a specific compression algorithm for this kind of meshes. The only care to be taken concerns the spatial extension, which should either be guaranteed, or the algorithm should take in account escape codes to handle exceptions to the extension rules. Moreover, we believe that applying this method to a specific kind of meshes provides a nice way to understand better their geometry/connectivity relation.

VI.4 Results

	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
remeshed	18.882	1.269
all	19.089	1.717

Table VI.2: GEncode compression ratio for triangulated surfaces, in bits per vertex. These results are an average over 200 models grouped by category.

GEncode intends to compress better geometrical meshes. In the case of the Ball Pivoting criterion, this means that the simplices should be as equilateral as possible, or at least a subtriangulation of a Delaunay polyhedron. This is the case of reconstruction algorithm [Bernardini *et al.* 1999] or some techniques of remeshing [Alliez *et al.* 2003]. The results of Table VI.2 show that this behaviour stands in practice (we used the arithmetic coder of Section II.2 *Arithmetic Coding*). In particular, the remeshed models and the scans sculptures are better compressed than the other models, as for example the model of Figure VI.10. Although most of the results presented here are for surfaces, the algorithm has been implemented for any dimension and codimension. The only two features that were not implemented in high dimension are the non–pure compression and the manifoldness recognition (which is a NP–hard problem [Hass 1998]). For higher dimension, we lacked of geometrical models to make a valid benchmark. The compressor compares nicely to existing compression scheme, although it is able to compress a wider range of models. Table VI.3 details some comparisons with the Edgebreaker algorithm of Chapter IV **Connectivity–Driven Compression**. These comparisons were made using the parallelogram prediction of [Touma and Gotsman 1998] for the Edgebreaker, with the same quantisation for the vertices (12 bits per coordinate). We are aware that many techniques improved on this parallelogram prediction, in particular [Cohen–Or *et al.* 2001]. As the difference is tight, the performance of connectivity–driven algorithm could best this current version of GEncode for surfaces. However, we believe that there are many open directions to improve this new scheme that will maintain it competitive.

The GEncode compression is already robust to errors in the input model since it compresses any mesh. The connectivity compression is actually quite robust to noise in the transmission, since the attachment criterion is local and the decoder should generate an interpolation of the points. Moreover,

	$\#_2$	Edgebreaker		GEncode	
		Geometry	Connectivity	Geometry	Connectivity
mechanical	71150	*	*	15.860	0.822
bunny	34834	17.050	2.178	18.767	1.182
gargoyle	30059	20.992	2.110	18.379	0.894
david	24988	25.800	3.070	16.985	1.631
horse	19851	24.856	3.012	18.216	1.193
terrain	16641	17.093	0.400	13.505	3.557
fandisk	6475	19.555	2.254	21.519	2.630
klein	4120	*	*	22.110	3.436
blech	4102	21.552	2.400	20.721	4.550
sphere	642	27.975	3.450	24.685	0.051
rotor	600	31.667	3.693	24.078	6.438

Table VI.3: GEncode compression ratio on triangulated surfaces, compared with the Edgebreaker. The mechanical piece is not a manifold object, and the Klein bottle is not orientable, which prevented the Edgebreaker to work.

the number of candidates varies significantly along the mesh, and errors can thus be easily detected by a candidate position exceeding the number of candidates. Animated models could be handled similarly to the connectivity–driven compression, although the vertices displacement can be encoded directly on the spatial hierarchy in that case.

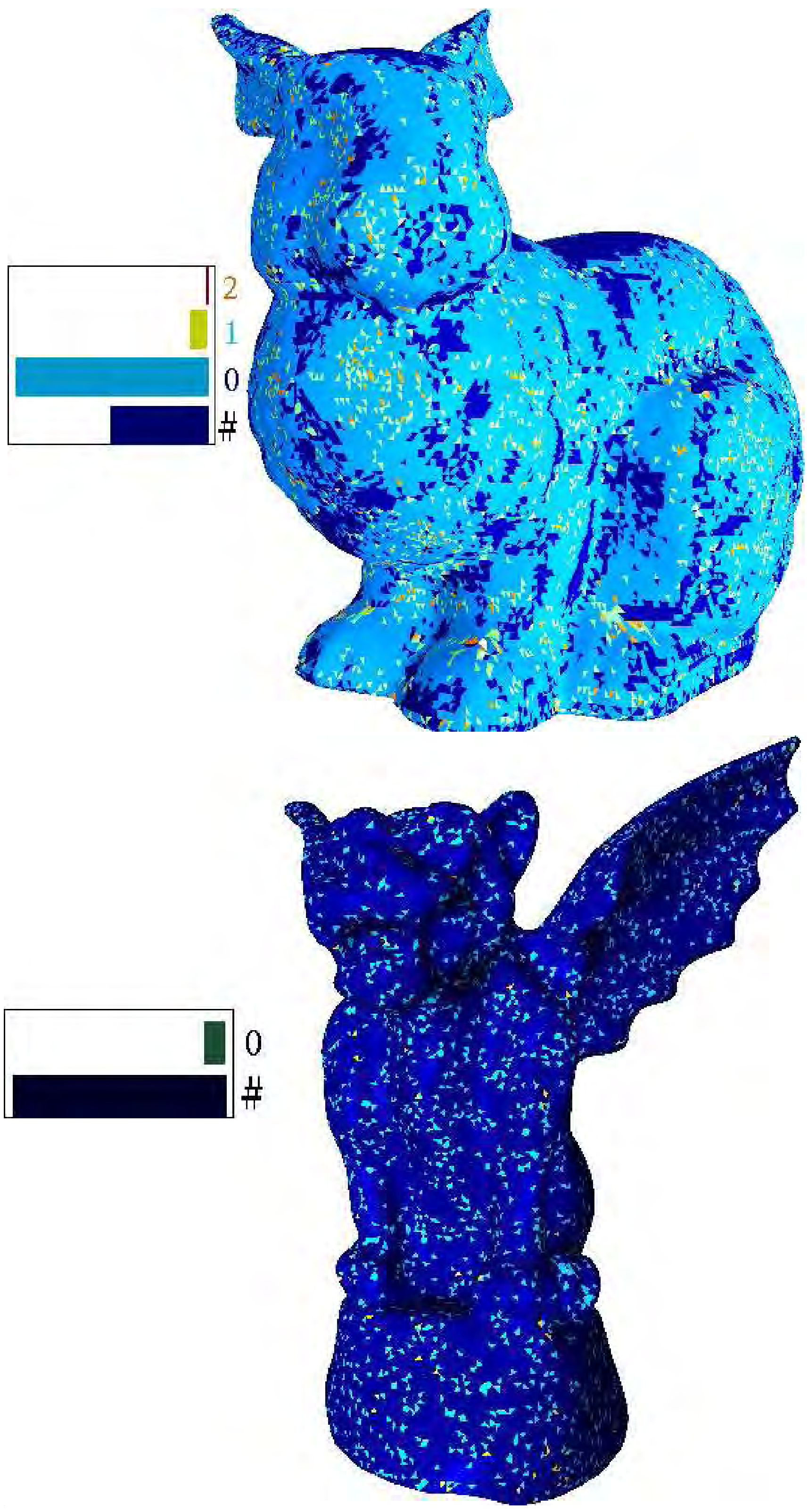


Figure VI.10: Candidate position for scanned models: the connectivity is encoded almost at zero rate: # is not transmitted, and thus almost only 0 codes are encoded.

VII
Conclusions and Future Works

This work proposed two new geometry–driven compression schemes for meshes in arbitrary dimension, while showing good performances for low dimensions compared to state of the art methods.

The first method is restricted to level sets, but provides both direct and progressive coding with a high controllability over the progression in the last case.

The mesh can be refined either by subdividing its ambient space or by transmitting more precise values of the implicit function defining the level set. These two methods are complementary and competitive, and a continuation of this work could study how to optimise their interaction in practise.

Moreover, a general model for the mesh can be translated into a specific ambient space subdivision. For example for medical images of a given organ, we can map a regular subdivision of a cube onto a space denser close to the model for that organ. Once again, the gain of such method could motivate a more precise study. However, the encoding of the level set can then represent more than just a compact image format, since it describes a certain deviation of the real level set with respect to the model.

The second method is a very general direct encoding scheme that handles any mesh of any topology, of arbitrary dimension and in any ambient space. GEncode originally aimed at encoding predictable mesh connectivity at almost zero cost. This goal is achieved if the prediction rule is known.

The strategy can be improved in many ways. First, the separate encoding of the geometry makes the whole compression ratio still a little higher than connectivity–driven approaches. On one side, the geometry encoding technique we used is still one of the first ones, and it can surely be improved by itself. On the other side, the mixing of geometry and connectivity coding usually results in more efficient programs, as we saw for level set compression. For example, this mixture could be implemented as an alternate process between connectivity and geometry coding, the result of one guiding the other.

A general method to fit the geometrical criterion to a specific class of mesh has been proposed. It can be directly used to improve the compression ratios for the given range of models, but the look for the best criterion actually generates much more information. In particular, this criterion can be used in the reverse direction to generate greedy reconstruction algorithms that would mimic the generation of the given class of meshes.

Moreover, this criterion represents part of the sampling paradigm used to generate the meshes, which is usually implicit. We believe that this relation between a given geometry and its discretisation can lead to a better understanding of discrete geometry. Designing coding schemes actually translates into isolating and representing clearly the most relevant information of the objects to encode. In that sense, it provides a nice framework to approach these essential questions.

These two geometry–driven methods are general, and compare nicely to existing surface compression schemes. In particular for isosurfaces, our method improved on state–of–the–art methods, which already outperformed generic surface compression schemes. For more general surfaces, GEncode compares nicely to the Edgebreaker. However, these two approaches are delicate to compare. On one side, the geometry coding of GEncode is more adapted to regular sampling in the space, and the one of the Edgebreaker relies on the regularity of the sampling *on* the surface. On the other side, the local coherence of the sampling is directly used in the contexts of the arithmetic coder for the octree coding, whereas this coherence is included in the parallelogram prediction scheme for the connectivity–driven schemes. The contribution these two steps of the compression (the symbol generation and their coding), remains delicate to measure in theory. During the preparation of this thesis, it appeared experimentally that on specific categories of models, the relative impact of each step becomes more visible: On manual designs, the sparse and even distribution of points makes it difficult to predict locally, while the statistical modelling of the arithmetic coder produces nice results. On the contrary for re-meshed models, the geometry is oversampled on the surface, leading to better local prediction compared to global statistics. This justifies the point of view of this work, which intends to provide specific compressing techniques for each category of models.

Bibliography

[Alexander 1930] J. W. Alexander. **The combinatorial theory of complexes**. *Annals of Mathematics*, 31:219–320, 1930. III.1(b), III.2(c)

[Alliez and Desbrun 2001] P. Alliez and M. Desbrun. **Valence–driven connectivity encoding of 3D meshes**. In *Computer Graphics Forum*, pages 480–489, 2001. IV.1(a), IV.3(b), IV.3(b), IV.3(c)

[Alliez *et al.* 2003] P. Alliez, D. Cohen–Steiner, O. Devillers, B. Levy and M. Desbrun. **Anisotropic polygonal remeshing**. In *Siggraph*. ACM, 2003. IV.3(b), VI

[Alliez *et al.* 2003] P. Alliez, É. Colin de Verdière, O. Devillers and M. Isenburg. **Isotropic surface remeshing**. In *Shape Modeling International*. IEEE, 2003. IV.3(b), VI, VI.4

[Amenta and Kullori 2002] N. Amenta and R. Kolluri. **The medial axis of unions of balls**. *Computational Geometry: Theory and Applications*, 20(1–2):25–37, 2001. III.3(b)

[Amenta *et al.* 2001] N. Amenta, S. Choi and R. Kolluri. **The Power Crust, unions of balls, and the medial axis transform**. *Computational Geometry: Theory and Applications*, 19(2–3):127–153, 2001. VI

[Arithmetic coding source] F. W. Wheeler. **Adaptive arithmetic coding source code**. `http://www.cipr.rpi.edu/~wheeler/ac/index.html`. II.2

[Armstrong 1979] M. A. Armstrong. **Basic topology**. McGraw–Hill, London, 1979. III.1(d), IV.1(c)

[Arnold 1981] V. I. Arnold. **Catastrophe theory**. Znanie, Moscow, 1981. V

[Attene *et al.* 2003] M. Attene, B. Falcidieno, M. Spagnuolo and J. Rossignac. **SwingWrapper: retiling triangle meshes for better Edgebreaker compression**. *Transactions on Graphics*, 22(4):982–996, 2003. IV.3(b), IV.3(b)

[Bajaj *et al.* 1998] C. L. Bajaj, V. Pascucci and D. Schikore. **Visualization of scalar topology for structural enhancement**. In D. Ebert, H. Hagen and H. Rushmeier, editors, *Visualization*, pages 51–58. IEEE, 1998. V

[Baumgart 1972] B. G. Baumgart. **Winged edge polyhedron representation**. Technical report, Stanford University, 1972. Technical Report AIM–179 (CS–TR–74–320). III

[Bentley 1975] J. L. Bentley. **Multidimensional binary search trees used for associative searching**. *Communications of the ACM*, 18(9):509–517, 1975. III.3(c)

[Bernardini *et al.* 1999] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva and G. Taubin. **The Ball–Pivoting Algorithm for surface reconstruction**. *Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999. III.3(b), VI.3, VI.3(b), VI.4

[Bloomenthal *et al.* 1997] J. Bloomenthal, C. L. Bajaj, J. Blinn, M.-P. Cani, A. Rockwood, B. Wyvill and G. Wyvill. **Introduction to implicit surfaces**. Morgan Kaufmann, San Francisco, 1997. V

[Blumenthal 1970] L. M. Blumenthal. **Theory and applications of distance geometry**. Chelsea, New York, 1970. VI.3(b)

[Boada and Navazo 2001] I. Boada and I. Navazo. **An octree isosurface codification based on discrete planes**. In T. L. Kunii, editor, *Spring Conference on Computer Graphics*, pages 187–194, Comenius University, Bratislava, 2001. V, V

[Boissonnat and Geiger 1993] J.-D. Boissonnat and B. Geiger. **Three dimensional reconstruction of complex shapes based on the Delaunay triangulation**. In R. Acharya and D. B. Goldgof, editors, *Biomedical Image Processing and Biomedical Visualization*, volume 1905, pages 964–975, SPIE, 1993. III.3(b), VI.1(c), VI.3(b)

[Boissonnat and Yvinec 1998] J.-D. Boissonnat and M. Yvinec. **Algorithmic geometry**. Cambridge University Press, 1998. III, III.3, III.3(b)

[Boissonnat and Cazals 2002] J.-D. Boissonnat and F. Cazals. **Smooth surface reconstruction via natural neighbour interpolation of distance functions**. *Computational Geometry: Theory and Applications*, 22(1–3):185–203, 2002. III.3(b), VI.3(b)

[Boissonnat and Oudot 2005] J.-D. Boissonnat and S. Oudot. **Provably Good Sampling and Meshing of Surfaces**. *Graphical Models*, 2005. Solid Modeling '04 special issue, to appear. III.3(b)

[Bossen and Ebrahimi 1997] F. Bossen and T. Ebrahimi. **A simple and efficient binary shape coding technique based on bitmap representation**. In *Acoustics, Speech, and Signal Processing*, pages 3129–3132, 1997. V

[Botsch *et al.* 2002] M. Botsch, A. Wiratanaya and L. Kobbelt. **Efficient high quality rendering of point sampled geometry**. In *Eurographics workshop on Rendering*, pages 53–64, 2002. VI.1(b), VI.1, VI.1(c), VI.3, VI.1(c), VI.1(c)

[Castelli and Devillers 2004] L. Castelli Aleardi and O. Devillers. **Canonical triangulation of a graph, with a coding application**. *INRIA preprint*, 2004. II.1(a), IV.1(a), IV.3(b), IV.3(b)

[Castelli *et al.* 2005*1] L. Castelli Aleardi, O. Devillers and G. Schaeffer. **Succinct representation of triangulations with a boundary**. In *Workshop on Algorithms and Data Structures*, volume 5608, pages 134–145. Springer, 2005. II.3(a)

[Castelli *et al.* 2005*2] L. Castelli Aleardi, O. Devillers and G. Schaeffer. **Dynamic updates of succinct triangulations**. In *Canadian Conference on Computational Geometry*, pages 135–138, 2005. II.3(a)

[Catmull and Clark 1978] E. Catmull and J. Clark. **Recursively generated B–spline surfaces on arbitrary topological meshes**. *Computer–Aided Design*, 10(6):350–355, 1978. III.3, III.3(a)

[Chaine and Bouakaz 2000] R. Chaine and S. Bouakaz. **Segmentation of 3–d surface trace points, using a hierarchical tree–based, diffusion scheme**. In *Asian Conference on Computer Vision*, volume 2, pages 995–1002, 2000. III.3(b), VI.3(b)

[Christopoulos *et al.* 2000] C. Christopoulos, A. Skodras and T. Ebrahimi. **The JPEG2000 still image coding system: An Overview**. *Transactions on Consumer Electronics*, 46(4):1103–1127, 2000. II.3(b)

[Cignoni *et al.* 1998*1] P. Cignoni, C. Rocchini and R. Scopigno. **Metro: measuring error on simplified surfaces**. *Computer Graphics Forum*, 17(2):167–174, 1998. V.1(b)

[Cohen–Or *et al.* 2001] D. Cohen–Or, R. Cohen and T. Ironi. **Multi–way geometry encoding**. *TAU Tech. Report*, 2001. IV.1(a), IV.2(b), IV.3(b), IV.3(b), VI.4

[Cohen–Steiner and Da 2002] D. Cohen–Steiner and T. K. F. Da. **A greedy Delaunay–based surface reconstruction algorithm**. *The Visual Computer*, 20(1):4–16, 2002. III.3(b), VI.3(b)

[Coors and Rossignac 2004] V. Coors and J. Rossignac. **Delphi: geometry-based connectivity prediction in triangle mesh compression**. *The Visual Computer*, 20(8–9):507–520, 2004. IV.3(b), IV.3(b)

[Craizer *et al.* 2002*1] M. Craizer, D. A. Fonini Jr and E. A. B. da Silva. **Alpha–expansions: a class of frame decompositions**. *Applied and Computational Harmonic Analysis*, 13:103–115, 2002. V, VI.1(c)

[Davis *et al.* 2002] J. Davis, S. Marschner, M. Garr and M. Levoy. **Filling holes in complex surfaces using volumetric diffusion**. In *Symposium on 3D Data Processing, Visualization, and Transmission*, 2002. V

[Deering 1995] M. F. Deering. **Geometry compression**. In *Siggraph*, pages 13–20. ACM, 1995. I, IV.1(a)

[Delone 1934] B. N. Delone. **Sur la sphère vide**. *Otdelenie Matematicheskikh i Estestvennykh Nauk*, 7:793–800, 1934. III.3(b)

[Desbrun *et al.* 1999] M. Desbrun, M. Meyer, P. Schröder and A. H. Barr. **Implicit fairing of irregular meshes using diffusion and curvature flow**. In *Siggraph*, pages 317–324. ACM, 1999. III.3(a)

[Devillers 2002] O. Devillers. **The Delaunay hierarchy**. *Foundations of Computer Science*, 13:163–180, 2002. Special issue on triangulations. III.3(c)

[Doo and Sabin 1978] D. Doo and M. Sabin. **Behaviour of recursive division surfaces near extraordinary points**. *Computer–Aided Design*, 10(6):356–360, 1978. III.3, III.3(a)

[Eastman 1982] C. M. Eastman. **Introduction to computer aided design**, 1982. Course Notes. Carnegie–Mellon University. III

[Finkel and Bentley 1974] R. A. Finkel and J. L. Bentley. **Quad trees: A data structure for retrieval on composite keys**. *Acta Informatica*, 4:1–9, 1974. III.3(c)

[Fleishman *et al.* 2003] S. Fleishman, I. Drori and D. Cohen–Or. **Bilateral mesh denoising**. *Transactions on Graphics*, 22(3):950–953, 2003. III.3(a)

[de Floriani *et al.* 1997] L. de Floriani, E. Puppo and P. Magillo. **A formal approach to multiresolution modeling**. In W. Straşer, R. Klein and R. Rau, editors, *Theory and Practice of Geometric Modeling*, pages 302–323. Springer, 1997. III.3(c)

[de Floriani and Hui 2003] L. de Floriani and A. Hui. **A scalable data structure for three–dimensional non–manifold objects**. In *Symposium on Geometry processing*, pages 72–82. Eurographics, 2003. III

[Freeman 1974] H. Freeman. **Computer processing of line drawing images**. *Computing Surveys*, 6(1):57–97, 1974. V

[Friedman *et al.* 1977] J. H. Friedman, J. L. Bentley and R. A. Finkel. **An algorithm for finding best matches in logarithmic expected time**. *Transactions on Mathematical Software*, 3(3):209–226, 1977. III.3(c)

[le Gall 1991] D. Le Gall. **MPEG: a video compression standard for multimedia applications**. *Communications of the ACM*, 34(4):46–58, 1991. II.3(b), IV.3

[Gandoin and Devillers 2002] P.-M. Gandoin and O. Devillers. **Progressive lossless compression of arbitrary simplicial complexes**. In *Siggraph*, volume 21, pages 372–379. ACM, 2002. I, VI.1(a), VI.1(b), VI.1, VI.1(c), VI.2, VI.1(c), VI.1(c), VI.1(c)

[Garland and Heckbert 1997] M. Garland and P. S. Heckbert. **Surface simplification using quadric error metrics**. *Computers & Graphics*, 31:209–216, 1997. III.3, III.3(a)

[Gelfand *et al.* 1994] I. M. Gelfand, M. Kapranov and A. Zelevinsky. **Discriminants, resultants and multidimensional determinants**. Birkhäuser, Boston, 1994. V.1(a)

[Glaser 1970] L. C. Glaser. **Geometrical combinatorial topology**. Van Nostrand Reinhold, New York, 1970. III.2(c)

[Guéziec *et al.* 1998] A. Guéziec, G. Taubin, F. Lazarus and W. P. Horn. **Converting sets of polygons to manifold surfaces by cutting and stitching**. In D. Ebert, H. Hagen and H. Rushmeier, editors, *Visualization*. IEEE, 1998. IV.1, IV.3(c)

[Guibas and Stolfi 1985] L. J. Guibas and J. Stolfi. **Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams**. *Transactions on Graphics*, 4:74–123, 1985. III

[Gumhold *et al.* 1999] S. Gumhold, S. Guthe and W. Straŝer. **Tetrahedral mesh compression with the Cut–Border machine**. In *Visualization*, pages 51–58. IEEE, 1999. IV.3(c)

[Gumhold and Amjoun 2004] S. Gumhold and R. Amjoun. **Higher Order Prediction for Geometry Compression**. In *Shape Modeling International*, pages 59–68. IEEE, 2003. IV.1(a)

[Hamming 1950] R. W. Hamming. **Error-detecting and error-correcting codes**. *Bell System Technical Journal*, 29(2):147–160, 1950. II.1(a)

[Hartley 1928] R. V. L. Hartley. **Transmission of information**. *Bell System Technical Journal*, 7:535, 1928. II, II.1(a)

[Hass 1998] J. Hass. **Algorithms for recognizing knots and 3–manifolds**. *Chaos, Solitons & Fractals*, 9(4–5):569–581, 1998. VI.4

[Hatcher 2002] A. Hatcher. **Algebraic topology**. Cambridge University Press, 2002. III, III.1, III.1(c), III.2(a), III.3(a)

[von Herzen and Barr 1987] B. von Herzen and A. H. Barr. **Accurate triangulations of deformed, intersecting surfaces**. In *Siggraph*, pages 103–110. IEEE, 1987. III.3(c)

[Hoppe *et al.* 1993] H. Hoppe, T. DeRose, T. Duchamp, J. A. McDonald and W. Stuetzle. **Mesh optimization**. In *Siggraph*, volume 27, pages 19–26. ACM, 1993. III.2(c), III.3(a)

[Hoppe 1996] H. Hoppe. **Progressive meshes**. In *Siggraph*, pages 99–108, New Orleans, Aug. 1996. ACM. III.2(c)

[Houston *et al.* 2005] B. Houston, M. B. Nielsen, C. Batty, O. Nilsson and K. Museth. **Gigantic Deformable Surfaces**. In *Siggraph Sketches & Applications*. ACM, 2005. II.3(a)

[Huffman 1952] D. A. Huffman. **A method for the construction of minimum redundancy codes**. In *I.R.E*, pages 1098–1102, 1952. II, II.1(b), II.1(b)

[Isenburg and Snoeyink 2000] M. Isenburg and J. Snoeyink. **Spirale reversi: reverse decoding of the Edgebreaker encoding**. In *Canadian Conference on Computational Geometry*, pages 247–256, 2000. IV.2

[Isenburg and Alliez 2002] M. Isenburg and P. Alliez. **Compressing hexahedral volume meshes**. In *Pacific Graphics*, pages 284–293, 2002. IV.1, IV.3(c)

[Isenburg and Gumhold 2003] M. Isenburg and S. Gumhold. **Out–of–core compression for gigantic polygon meshes**. *Transactions on Graphics*, 22(3):935–942, 2003. II.3(b)

[Ju *et al.* 2002] T. Ju, F. Losasso, S. Schaefer and J. Warren. **Dual contouring of hermite data**. In *Siggraph*, pages 339–346. ACM, 2002. V, V.4(c)

[Kälberer *et al.* 2005] F. Kälberer, K. Polthier, U. Reitebuch and M. Wardetzky. **Freelence — coding with free valences**. *Computer Graphics Forum*, 24(3):469–478, 2005. IV.1(a), IV.3(b), IV.3(b)

[Katanforoush and Shahshahani 2003] A. Katanforoush and M. Shahshahani. **Distributing Points on the Sphere**. *Experimental Mathematics*, 12(2):199–209, 2003. III.3

[King and Rossignac 1999] D. King and J. Rossignac. **Guaranteed 3.67v bit encoding of planar triangle graphs**. In *Canadian Conference on Computational Geometry*, pages 146–149, 1999. IV.2, IV.3(a), IV.2, IV.3(b), IV.3(c)

[Kobbelt 2000] L. Kobbelt. **$\sqrt{3}$ subdivision**. In *Siggraph*, pages 103–112. ACM, 2000. III.12(c), III.3(a)

[Kronrod and Gotsman 2001] B. Kronrod and C. Gotsman. **Efficient coding of nontriangular mesh connectivity**. *Graphical Models*, 63:263–275, 2001. IV.2, IV.3(c)

[le Buhan and Ebrahimi 1997] C. Le Buhan and T. Ebrahimi. **Progressive polygon encoding of shape contours**. In *Image Processing and its Applications*, pages 17–21, 1997. V, V

[Lage *et al.* 2005*1] M. Lage, T. Lewiner, H. Lopes and L. Velho. **CHE: a scalable topological data structure for triangular meshes**. Technical report, Pontifical Catholic University of Rio de Janeiro, 2005. II.3(a)

[Lage *et al.* 2005] M. Lage, T. Lewiner, H. Lopes and L. Velho. **CHF: a scalable topological data structure for tetrahedral meshes**. In *Sibgrapi*, pages 349–356, Natal, Oct. 2005. IEEE. II.3(a), III

[Langdon and Rissanen 1981] G. Langdon Jr and J. Rissanen. **Compression of black–white images with arithmetic coding**. *Transactions on Communications*, 29(6):858–867, 1981. V

[Lee *et al.* 2003] H. Lee, P. Alliez and M. Desbrun. **Angle-Analyzer: A Triangle-Quad Mesh Codec**. In *Eurographics*, volume 21(3), 2002. IV.3(b), IV.3(b), IV.3(b)

[Lee *et al.* 2003] H. Lee, M. Desbrun and P. Schröder. **Progressive encoding of complex isosurfaces**. *Transactions on Graphics*, 22(3):471–476, 2003. V, V, V.4, V.4(c), V.2, V.4(c), V.4(d)

[Lempel and Ziv 1977] A. Lempel and J. Ziv. **A universal algorithm for sequential data compression**. *Transactions on Information Theory*, 23(3):337–343, 1977. II, II.3(b)

[Lewiner *et al.* 2003*3] T. Lewiner, H. Lopes, A. W. Vieira and G. Tavares. **Efficient implementation of Marching Cubes' cases with topological guarantees**. *Journal of Graphics Tools*, 8(2):1–15, 2003. V

[Lewiner *et al.* 2004] T. Lewiner, H. Lopes, J. Rossignac and A. W. Vieira. **Efficient Edgebreaker for surfaces of arbitrary topology**. In *Sibgrapi*, pages 218–225, Curitiba, Oct. 2004. IEEE. I, IV.1(a), IV.2, IV.3(b)

[Lewiner *et al.* 2004*4] T. Lewiner, L. Velho, H. Lopes and V. Mello. **Simplicial isosurface compression**. In *Vision, Modeling and Visualization*, pages 299–306, Stanford, 2004. IOS Press. I, III.2, V

[Lewiner *et al.* 2004*2] T. Lewiner, L. Velho, H. Lopes and V. Mello. **Hierarchical isocontours extraction and compression**. In *Sibgrapi*, pages 234–241, Curitiba, Oct. 2004. IEEE. I, III.2, V

[Lewiner *et al.* 2004] T. Lewiner, H. Lopes and G. Tavares. **Applications of Forman's discrete Morse theory to topology visualization and mesh compression**. *Transactions on Visualization and Computer Graphics*, 10(5):499–508, 2004. IV.3(c)

[Lewiner *et al.* 2004*1] T. Lewiner, J. Gomes Jr, H. Lopes and M. Craizer. **Arc–length based curvature estimator**. In *Sibgrapi*, pages 250–257, Curitiba, Oct. 2004. IEEE. V.4(a), V.4(a)

[Lewiner *et al.* 2005*3] T. Lewiner, L. Velho, H. Lopes and V. Mello. **Extraction and compression of hierarchical isocontours from image data**. *Computerized Medical Imaging and Graphics*, 2005. accepted for publication. I, V

[Lewiner *et al.* 2005*1] T. Lewiner, M. Craizer, H. Lopes, S. Pesco, L. Velho and E. Medeiros. **GEncode: geometry–driven compression in arbitrary dimension and co–dimension**. In *Sibgrapi*, pages 249–256, Natal, Oct. 2005. IEEE. I, III.2, VI

[Lewiner *et al.* 2005] T. Lewiner, J. Gomes Jr, H. Lopes and M. Craizer. **Curvature and torsion estimators based on parametric curve fitting**. *Computers & Graphics*, 2005. IV.3(b), IV.3(b), V.4(a), V.4(a)

[Li and Vitanyi 1997] M. Li and P. M. B. Vitanyi. **An introduction to Kolmogorov complexity and its applications**. Springer, 1997. II, II.1(c), II.2(c)

[Lickorish 1999] W. B. R. Lickorish. **Simplicial moves on complexes and manifolds**. In *Kirbyfest*, volume 2 of *Geometry and Topology Monographs*, pages 299–320, 1999. III.2(c)

[Loop 1987] C. T. Loop. **Smooth subdivision surfaces based on triangles**. Master's thesis, *Department of Mathematics, University of Utah*, 1987. III.12(b), III.3(a), IV.3(b)

[Lopes 1996] H. Lopes. **Algorithm to build and unbuild 2 and 3 dimensional manifolds**. PhD thesis, *Department of Mathematics, PUC–Rio*, 1996. Advised by Geovan Tavares. III.2(b), III.2(b)

[Lopes and Tavares 1997] H. Lopes and G. Tavares. **Structural operators for modeling 3–manifolds**. In C. Hoffman and W. Bronsvort, editors, *Solid Modeling and Applications*, pages 10–18. ACM, 1997. III.2(b), III.2(b), IV.1(c)

[Lopes *et al.* 2000] H. Lopes, G. Nonato, S. Pesco and G. Tavares. **Dealing with topological singularities in volumetric reconstruction**. In P.-J. Laurrent, P. Sablonière and L. Schumaker, editors, *Curve and Surface Design*, pages 229–238, Saint Malo, 2000. Vanderbilt University Press. III.3(b), VI.3(b)

[Lopes *et al.* 2002] H. Lopes, J. Rossignac, A. Safonova, A. Szymczak and G. Tavares. **Edgebreaker: a simple compression for surfaces with handles**. In C. Hoffman and W. Bronsvort, editors, *Solid Modeling and Applications*, pages 289–296, Saarbrücken, Germany, 2002. ACM. I, IV.1(a)

[Lopes *et al.* 2002*1] H. Lopes, J. B. Oliveira and L. H. de Figueiredo. **Robust adaptive polygonal approximation of implicit curves**. *Computers & Graphics*, 26(6):841–852, 2002. V

[Lopes *et al.* 2003] H. Lopes, J. Rossignac, A. Safonova, A. Szymczak and G. Tavares. **Edgebreaker: a simple implementation for surfaces with handles**. *Computers & Graphics*, 27(4):553–567, 2003. IV.2, IV.2(c)

[Lorensen and Cline 1987] W. E. Lorensen and H. E. Cline. **Marching Cubes: a high resolution 3D surface construction algorithm**. In *Siggraph*, volume 21, pages 163–169. ACM, 1987. V

[Mäntylä 1988] M. Mäntylä. **An introduction to solid modeling**. Computer Science Press, Rockville, 1988. III, III.2(a)

[Martin 1979] G. Martin. **Range encoding: an algorithm for removing redundancy from a digitised message**. In *Video & Data Recoding*, 1979. IV.3(a), IV.2

[Medeiros *et al.* 2003] E. Medeiros, L. Velho and H. Lopes. **Topological framework for advancing front triangulations**. In *Sibgrapi*, pages 45–51, São Carlos, Oct. 2003. IEEE. III.3(b), VI.2(a)

[Medeiros *et al.* 2004] E. Medeiros, L. Velho and H. Lopes. **Restricted bpa: applying ball–pivoting on the plane**. In *Sibgrapi*, pages 372–379, Curitiba, Oct. 2004. IEEE. III.3(b), VI.3, VI.3(b)

[Mello *et al.* 2003] V. Mello, L. Velho, P. Roma Cavalcanti and C. Silva. **A generic programming approach to multiresolution spatial decompositions**. In H.-C. Hege and K. Polthier, editors, *Visualization and Mathematics III*, pages 337–360. Springer, Heidelberg, 2003. I, III.3(c), V

[Moffat *et al.* 1995] A. Moffat, R. Neal and I. H. Witten. **Arithmetic coding revisited**. In *Data Compression*, pages 202–211, 1995. II, II.2

[Montani *et al.* 1994] C. Montani, R. Scateni and R. Scopigno. **A modified lookup table for implicit disambiguation of Marching Cubes**. *The Visual Computer*, 10(6):353–355, 1994. V

[Munkres 1984] J. R. Munkres. **Elements of algebraic topology**. Addison-Wesley, Menlo Park, 1984. III, III.3(a)

[Newman 1926] M. H. A. Newman. **On the foundations of combinatorial analysis situs**. *Royal Academy*, 29:610–641, 1926. III.2(c)

[Nielson and Hamann 1991] G. M. Nielson and B. Hamann. **The asymptotic decider: resolving the ambiguity in Marching Cubes**. *Visualization*, pages 29–38, 1991. V

[Nonato *et al.* 2005] G. Nonato, A. Castelo, R. Minghim and H. Hideraldo. **Topological tetrahedron characterization with application in volume reconstruction**. *Journal of Shape Modeling*, 11(2), 2005. III.3(b), VI.3(b)

[Nyquist 1928] H. Nyquist. **Certain topics in telegraph transmission theory**. *Transactions of the American Institute of Electrical Engineers*, 47:617–644, 1928. II, II.1(a)

[Pachner 1991] U. Pachner. **PL homeomorphic manifolds are equivalent by elementary shellings**. *European Journal of Combinatorics*, 12:129–145, 1991. III.2(c)

[Parker *et al.* 1998] S. Parker, P. Shirley, Y. Livnat, C. Hansen and P.-P. Sloan. **Interactive ray tracing for isosurface rendering**. In D. Ebert, H. Hagen and H. Rushmeier, editors, *Visualization*, pages 233–238. IEEE, 1998. V

[Pascucci 2002] V. Pascucci. **Slow growing subdivision (SGS) in any dimension: towards removing the curse of dimensionality**. *Computer Graphics Forum*, 21(3):451–460, 2002. III.3(c)

[Pereira and Ebrahimi 2002] F. Pereira and T. Ebrahimi, editors. **The MPEG–4 Book**. Prentice Hall, Upper Saddle River, 2002. II.3(b), IV.3

[Poulalhon and Schaeffer 2003] D. Poulalhon and G. Schaeffer. **Optimal coding and sampling of triangulations**. In *ICALP*, pages 1080–1094, 2003. IV.2(b), IV.3(b)

[Pratt 1987] V. Pratt. **Direct least–squares fitting of algebraic surfaces**. *Computers & Graphics*, 21(4):145–152, 1987. VI.3(c)

[Puppo 1998] E. Puppo. **Variable resolution triangulations**. *Computational Geometry: Theory and Applications*, 11(3–4):219–238, 1998. III.3(c)

[Rissanen 1976] J. Rissanen. **Generalized Kraft inequality and arithmetic coding**. *IBM Journal of Research and Development*, 20:198–203, 1976. II, II.2

[Rossignac 1986] J. Rossignac. **Constraints in constructive solid geometry**. In *Workshop on interactive 3D graphics*, pages 93–110. ACM, 1986. III.3

[Rossignac 1999] J. Rossignac. **Edgebreaker: connectivity compression for triangle meshes**. *Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999. I, I, III.2, IV.1(a), IV.2, IV.2(a), IV.2, IV.3(b)

[Rossignac and Szymczak 1999] J. Rossignac and A. Szymczak. **Wrap & zip decompression of the connectivity of triangle meshes compressed with edgebreaker**. *Computational Geometry: Theory and Applications*, 14(1–3):119–135, 1999. I, IV.2, IV.2(b), IV.3(a), IV.2

[Rossignac *et al.* 2001] J. Rossignac, A. Safonova and A. Szymczak. **3D compression made simple: Edgebreaker on a corner–table**. In *Shape Modeling International*, pages 278–283. IEEE, 2001. II.3(a), III, IV.2, IV.3(b)

[Safonova and Rossignac 2003] A. Safonova and J. Rossignac. **Compressed piecewise–circular approximations of 3D curves**. *Computer–Aided Design*, 35(6):533–547, 2003. V

[Salomon 2000] D. Salomon. **Data compression: the complete reference**. Springer, Berlin, 2000. II, II.1(a), IV.2

[Saupe and Kuska 2002] D. Saupe and J.-P. Kuska. **Compression of isosurfaces for structured volumes with context modelling**. In *Symposium on 3D Data Processing, Visualization, and Transmission*, pages 384–390, 2002. V

[Sethian 1999] J. A. Sethian. **Fast marching methods and level set methods**. *Cambridge Monograph on Applied and Computational Mathematics*. Cambridge University Press, 1999. V

[Shannon 1948] C. E. Shannon. **A mathematical theory of communication**. *Bell System Technical Journal*, 27:379–423 and 623–656, 1948. II, II.1(a), II.1(b), II.1(b)

[Sorkine *et al.* 2003] O. Sorkine, D. Cohen–Or and S. Toledo. **High-pass quantization for mesh encoding**. In *Symposium on Geometry Processing*, pages 42–51. Eurographics, 2003. IV.1(a), IV.3(c)

[Szymczak and Rossignac 2000] A. Szymczak and J. Rossignac. **Grow & Fold: compressing the connectivity of tetrahedral meshes**. *Computer–Aided Design*, 32(8/9):527–538, 2000. IV.1, IV.3(c)

[Taubin 1995] G. Taubin. **A signal processing approach to fair surface design**. In *Siggraph*, pages 351–358. ACM, Aug. 1995. III.3(a)

[Taubin and Rossignac 1998] G. Taubin and J. Rossignac. **Geometric compression through topological surgery**. *Transactions on Graphics*, 17(2):84–115, 1998. IV.1(a), IV.1(b), IV.2

[Taubin *et al.* 1998] G. Taubin, W. P. Horn, F. Lazarus and J. Rossignac. **Geometry coding and VRML**. *Proceedings of the IEEE*, 86(6):1228–1243, 1998. IV.1(a)

[Taubin 1999] G. Taubin. **Geometric signal processing on polygonal meshes**. In *Eurographics State of the Art Report*, 1999. III.3(a)

[Taubin 2002] G. Taubin. **Blic: bi–level isosurface compression**. In *Visualization*, pages 451–458, Boston, Massachusetts, 2002. IEEE. I, V

[Tavares *et al.* 2003] G. Tavares, R. Santos, H. Lopes, T. Lewiner and A. W. Vieira. **Topological reconstruction of oil reservoirs from seismic surfaces**. In *International Association for Mathematical Geology*, Portsmouth, UK, 2003. Terra Nostra. V

[Touma and Gotsman 1998] C. Touma and C. Gotsman. **Triangle mesh compression**. In *Graphics Interface*, pages 26–34, 1998. I, IV.1(a), IV.3(b), IV.3(b), IV.3(b), VI.4

[Treece *et al.* 1999] G. Treece, R. Prager and A. Gee. **Regularised Marching Tetrahedra: improved iso–surface extraction**. *Computers & Graphics*, 23(4):583–598, 1999. V

[Tutte 1998] W. T. Tutte. **Graph theory as I have known it**. Oxford University Press, New York, 1998. II

[Velho 1996] L. Velho. **Simple and efficient polygonization of implicit surfaces**. *Journal of Graphics Tools*, 1(2):5–25, 1996. V, V.1(a)

[Velho 2001] L. Velho. **Mesh simplification using four–face clusters**. In *Shape Modeling International*, pages 200–208. IEEE, 2001. III.2(c)

[Velho and Zorin 2001] L. Velho and D. Zorin. **4–8 subdivision**. *Computer–Aided Geometrical Design, Special Issue on Subdivision*, 18(5):397–427, 2001. III.3(a), IV.3(b)

[Velho *et al.* 2003] L. Velho, H. Lopes, G. Tavares, E. Medeiros and T. Lewiner. **Mesh ops**. Technical report, Department of Mathematics, PUC–Rio, 2003. III.2

[Velho 2003] L. Velho. **Stellar subdivision grammars**. In *Symposium on Geometry Processing*, pages 188–199. Eurographics, 2003. III.3(a)

[Velho 2004] L. Velho. **A dynamic adaptive mesh library based on stellar operators**. *Journal of Graphics Tools*, 9(2), 2004. III.3(c)

[Vieira *et al.* 2004] A. W. Vieira, T. Lewiner, L. Velho, H. Lopes and G. Tavares. **Stellar mesh simplification using probabilistic optimization**. *Computer Graphics Forum*, 23(4):825–838, 2004. III.3(a), IV.3(c)

[Voronoi 1908] G. F. Voronoi. **Nouvelles applications des paramètres continus à la théorie des formes quadratiques**. *Journal für die Reine und Angewandte Mathematik*, 133:97–178, 1908. III.3(b)

[Wallace 1991] G. K. Wallace. **The JPEG still picture compression standard**. *Communications of the ACM*, 34(4):30–44, 1991. II.3(b)

[Weiler 1985] K. J. Weiler. **Edge–based data structures for solid modeling in curved–surface environments**. *Computer Graphics and Applications*, 5(1):21–40, 1985. III

[Yang and Wu 2002] S.-N. Yang and T.-S. Wu. **Compressing isosurfaces generated with Marching Cubes**. *The Visual Computer*, 18(1):54–67, 2002. V

[Zorin and Schröder 2000] D. Zorin and P. Schröder. **Subdivision for Modeling and Animation**. In *Siggraph course notes*. ACM, 2000. III.3(a)

Index

Summary of notations

$\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{X}$	points of a topological space $\mathbb{X}$
$\mathbb{R}$	set of the real number
$\mathbb{R}^n = \mathbb{R} \times \mathbb{R} \times \ldots \times \mathbb{R}$	Euclidean space of dimension n
$\\|\mathbf{x}\\| = \sqrt{\sum x_i^2}$	Euclidean norm of point $\mathbf{x}$
$\mathbb{B}^p \quad = \{\mathbf{x} \in \mathbb{R}^p : \\|\mathbf{x}\\| < 1\}$	unit ball in $\mathbb{R}^p$
$\mathbb{S}^{p-1} = \{\mathbf{x} \in \mathbb{R}^p : \\|\mathbf{x}\\| = 1\}$	unit sphere in $\mathbb{R}^p$
$\mathbb{N}$, $\mathbb{Z}$	sets of the natural and relative integers
$[\\![m, n]\\!] = \{m, m+1, \ldots, n\}$	integer interval
ρ, σ, τ	simplices
ρ^n, σ^n, τ^n	simplices of dimension n
v, w	vertices : simplex of dimension 0
e	edge : simplex of dimension 1
t	triangle : simplex of dimension 2
Δ	tetrahedron : simplex of dimension 3
$\tau^m \prec \sigma^n$, $\sigma^n \succ \tau^m$	τ^m is a face of σ^n, σ^n is incident to τ^m
$\partial\sigma = \{\tau, \tau \prec \sigma\}$	frontier of a simplex σ
$K = \{\sigma\}$	simplicial complex
$K^n = \{\sigma^p, p \leqslant n\}$	simplicial complex of dimension n
$\#_m(K) = \#\{\sigma^m \in K\}$	number of m–simplices of K
$\chi(K^n) = \sum(-1)^m \#_{n-m}(K^n)$	Euler–Poincaré characteristic of K
$K_{(m)} = \{\sigma^p \in K, p \leqslant m\}$	m–skeleton of K
$\sigma \star \tau = \text{hull}(\sigma \cup \tau)$	join of σ and τ
$\text{lk}(\sigma) = \{\tau \in K : \sigma \star \tau \in K\}$	link of σ
$\dot{\text{st}}(\sigma) = \{\sigma \star \tau, \tau \in \text{lk}(\sigma)\}$	open star of σ
$\text{st}(\sigma) = \dot{\text{st}}(\sigma) \ \cup \ \bigcup_{\rho \in \dot{\text{st}}(\sigma)} \partial\rho$	star of σ
$\partial K^n = \{\sigma^{n-1} : \#\,\text{lk}(\sigma^{n-1}) = 1\}$	boundary of a pure simplicial complex
$\mathcal{M}^d$	simplicial n–manifold
$\mathcal{S} = \mathcal{M}^2$	triangulated surface
$\chi(\mathcal{S}) = 2 - 2 \cdot \mathfrak{g}(\mathcal{S}) - \mathfrak{b}(\mathcal{S})$	genus and number of boundaries of $\mathcal{S}$
$K \preccurlyeq K'$, $K' \succcurlyeq K$	triangulation order: local refinement
$K \rightsquigarrow K'$, $K' \leftsquigarrow K$	RBMT order : non–local refinement
$\mathbb{U}_K(\mathfrak{f}) = \{\sigma \in K : \mathfrak{f}(\sigma) \ni 0\}$	tubular neighbourhood of $\mathfrak{f}^{-1}(0)$ in K