Progressive and Random Accessible Mesh Compression

Adrien, Enam Maglo

ECOLE CENTRALE DES ARTS
ET MANUFACTURES
"ECOLE CENTRALE PARIS"

PHD THESIS

in candidacy for the degree of

Doctor of Ecole Centrale Paris
Specialty : Computer Science

Defended by

Adrien Maglo

Progressive and Random Accessible Mesh Compression

prepared at Ecole Centrale Paris, MAS Laboratory

Jury :

Chairman:	Pr. Mohamed Daoudi	LIFL, Institut Mines-Télcom, Télécom Lille1, France
Reviewers:	Dr. Marc Antonini	Laboratoire I3S, Sophia-Antipolis, France
	Dr. Raphaëlle Chaine	LIRIS, CNRS, Université Lyon 1, France
Examiners:	Pr. Florent Dupont	LIRIS, CNRS, Université Lyon 1, France
	Dr. Frank Ledoux	CEA, DAM, DIF, France
Invited member:	Dr. Pierre Alliez	INRIA Sophia-Antipolis - Méditerranée, France
Advisors:	Dr. Céline Hudelot	Laboratoire MAS, Ecole Centrale Paris, France
	Dr. Ian Grimstead	Cardiff School of Computer Science & Informatics, Cardiff University, United Kingdom
	Pr. Marc Aiguier	Laboratoire MAS, Ecole Centrale Paris, France

Abstract

3D virtual models are today of common usage in various domains such as computational simulation, medical imaging, computer-aided design, entertainment, digital heritage and e-commerce. One way to represent 3D virtual object is to use meshes. A surface (or volume) mesh is a collection of adjacent elementary polygons (or cells) that together constitute a piecewise approximation of a represented surface (or volume). This representation is today ubiquitous. Specific chips dedicated to the efficient rendering of images with such representations are today integrated into many devices, from the smartphone to the high performance computer cluster.

In all the application domains that use meshes, the need for precision has never ceased to rise, thus leading to the generation of very large models. For instance today, numerical simulations can use meshes that contain several billions of cells. The storage of large models consumes a lot of disk space and their transmission over the network can take a lot of time in limited bandwidth conditions. Besides, the visualization and more generally the processing of large meshes are also problematic because they consume a lot of computation time and memory. Accessing to large meshes on a device with low resource need some 3D adaptation strategies.

The thesis presented in this manuscript is that 3D mesh compression is an efficient method for 3D data adaptation. The first aim of 3D mesh compression is to reduce the size of an input model. As a consequence the data can better fit storage and network constraints. Single-rate algorithms simply restore during the decompression the initial mesh. However, alternative types of algorithms enable the decompression of several versions of the input mesh depending on the decoded parts of the compressed data. Thus, progressive compression techniques embed a multiresolution structure inside the compressed data. During the decompression, a coarse version of the input mesh is progressively refined as more data is decompressed. Random-accessible techniques allow to decompress only requested parts of the mesh. Finally progressive random accessible techniques allow to decompress any part of the mesh at any level of detail. With these different types of algorithms, mesh compression can adapt 3D mesh data to save bandwidth, memory, computation time and increase the user interactivity.

Previous work on progressive mesh compression focused on triangle meshes but meshes containing other types of faces are commonly used. Therefore, we first propose a new progressive mesh compression method that can efficiently compress meshes with arbitrary face degrees. Its compression performance is competitive with approaches dedicated to progressive triangle mesh compression.

Progressive mesh compression is linked to mesh decimation because both applications generate levels of detail. Consequently, we propose a new simple volume metric to drive the polygon mesh decimation. We apply this metric to the progressive compression and the simplification of polygon meshes.

We then show that the features offered by progressive mesh compression algorithms can be exploited for 3D adaptation by the proposition of a new framework for remote scientific visualization.

Progressive random accessible mesh compression schemes can better adapt 3D mesh data to the various constraints by taking into account regions of interest. So, we propose two new progressive random-accessible algorithms. The first one is based on the initial segmentation of the input model. Each generated cluster is compressed independently with a progressive algorithm. The second one is based on the hierarchical grouping of vertices obtained by the decimation. The advantage of this second method is that it offers a high random accessibility granularity and generates one-piece decompressed meshes with smooth transitions between parts decompressed at low and high levels of detail. Experimental results demonstrate the compression and adaptation efficiency of both approaches.

Résumé

Les modèles 3D virtuels sont aujourd'hui couramment utilisés dans de nombreux domaines d'application tels que la simulation numérique, l'imagerie médicale, la conception assistée par ordinateur, l'industrie du divertissement, la sauvegarde numérique du patrimoine ainsi que le commerce électronique. Les maillages sont souvent utilisés pour représenter des objets 3D virtuels. Un maillage surfacique (ou volumique) est un ensemble de polygones (ou de cellules) élémentaires adjacents qui, ensemble, constituent une approximation par partie de la surface (ou du volume) représentée. Les maillages sont aujourd'hui omniprésents dans toutes les applications qui nécessitent le rendu d'objets 3D. Des composants électroniques dédiés au rendu d'images à partir de maillages sont intégrés dans de nombreux terminaux, de l'ordiphone au calculateur haute performance.

Dans tout les domaines d'applications qui utilisent des maillages, les besoins de précision ne cessent d'augmenter. Cela engendre la génération de modèles de plus en plus importants. Par exemple aujourd'hui, des applications de simulation numérique peuvent utiliser des maillages qui contiennent plusieurs milliards de mailles. Or le stockage de gros modèles consomme beaucoup d'espace disque. Leur transmission sur un réseau à faible bande passante peut prendre beaucoup de temps. De plus, la visualisation interactive et plus généralement le traitement de maillages de grande taille sont problématiques car ils requièrent beaucoup de temps de calcul et de mémoire. Des stratégies d'adaptation 3D sont donc nécessaires pour accéder à un important maillage à partir d'un périphérique à faible ressources.

La thèse présentée dans ce manuscrit voit la compression de maillages comme une méthode efficace pour l'adaptation des données 3D. Le premier but de la compression de maillages est en effet de réduire la taille des données du modèle d'entrée. Les données peuvent ainsi être stockées et transmises plus aisément. Les algorithmes de compression à simple taux restaurent uniquement, au moment de la décompression, le maillage initial. Cependant d'autres types d'algorithmes permettent de décompresser plusieurs versions du maillage initial en fonction de la partie des données qui est décodée.
Ainsi, les algorithmes de compression progressive embarquent dans les données compressées une structure de donnée multirésolution. Durant la décompression, une version grossière du maillage initial est progressivement raffinée au fur et à mesure que des données supplémentaires sont décompressées.
Les techniques de compression avec accès aléatoire permettent de décompresser juste les parties du maillage requises par l'utilisateur.
Enfin, les techniques de compression progressive avec accès aléatoire permettent de décompresser différentes parties du maillage à différents niveaux de détail.
La compression de maillages permet donc, grâce à ces différents types d'algorithmes, d'adapter les maillages 3D pour économiser de la bande passante, de la mémoire et du temps de calcul. Elle permet aussi d'augmenter la capacité de l'utilisateur d'interagir avec les données.

Les travaux de l'état de l'art en compression progressive de maillages se sont concentrés sur les maillages triangulaires. Mais les maillages contenant d'autres types de faces sont aussi couramment utilisés. Par conséquent, nous proposons en premier lieu une nouvelle méthode de compression progressive qui peut efficacement encoder des maillages avec des faces de degrés arbitraires. Ce nouvel algorithme atteint un niveau de performance comparable aux algorithmes progressifs dédiés à la compression de maillages triangulaires.

La compression progressive est liée à la décimation de maillages car ces deux applications génèrent des niveaux de détail. Par conséquent, nous proposons une nouvelle métrique volumique simple pour guider la décimation de maillages polygonaux. Nous appliquons cette métrique à la compression progressive et la simplification de maillages polygonaux.

Nous montrons ensuite que les possibilités offertes par les algorithmes de compression progressive peuvent être exploitées pour adapter les données 3D en proposant un nouveau cadre applicatif pour la visualisation scientifique distante.

Les algorithmes de compression progressive avec accès aléatoire peuvent mieux adapter les données 3D aux différentes contraintes en prenant en compte les régions d'intérêt de l'utilisateur. Nous proposons donc deux nouveaux algorithmes progressifs avec accès aléatoire. Le premier algorithme est basé sur une segmentation préliminaire du maillage d'entrée. Chaque grappe est ensuite compressée de manière indépendante par un algorithme de compression progressif. Le second algorithme est basé sur le groupement hiérarchique des

sommets obtenu par la décimation. L'avantage de cette seconde méthode est qu'elle offre une forte granularité d'accès aléatoire et génère des maillages décompressés en une pièce avec des transitions lisses entre les parties décompressées à un faible et un haut niveau de détail. Des résultats expérimentaux démontrent l'efficacité des deux approches en terme de compression et d'adaptation.

Acknowledgements

There are of course many people I should thank for having made possible this work. I will try here to not forget some of them.

The first person to thank is of course my advisor Céline Hudelot who trusted me to start this thesis. She gave me a lot of autonomy, guided me excellent advises and introduced me to great people. Thus, I would like also to thank Ian Grimstead for having offered me the opportunity to work at Cardiff university under his supervision and Pierre Alliez for the very rich experience of co-authoring a paper with him. I am also thankful to Marc Aiguier for accepting to co-direct my research and Pascale Le Gall for her advices.

The work described in this thesis was developed in the context of the ANR Collaviz project. It has been a pleasure to collaborate with all its members, especially with the people of LIRIS laboratory in Lyon: Ho Lee, Guillaume Lavoué, Florent Dupont and Çağatay Dikici.

I would like also to thank the other members of my PhD jury that I did not mention until here: Raphaëlle Chaine, Marc Antonini, Frank Ledoux and Mohamed Daoudi. I really appreciated their questions and remarks to improve my work.

I thank also all my former colleagues and friends at the MAS laboratory of the Ecole Centrale Paris: Konstantin Todorov for our talks and his wide culture, Clément Courbet for his strong views and having shown me the way, Nicolas James for our geek discussions, Bilal Kanso for teaching me about Lebanon, Adith Perez for teaching me about Columbia and latin america, Nesrine Ben Mustapha and Dong Bui for our funny table football plays, Sofiane Karchoudi and Marc-Antoine Arnaud for their humor, Amel Znaidia for her determination, Walid Hachicha for his good work, Emmanuelle Gallet for having remembered me at the end of my thesis the motivation of the starting PhD student, Jose Pablo Escobedo for his friendly company, Anthony Grisey for his good mood and last but not least Hichem Bannour for a lot of things... I have also a thought about all the other members of the laboratory I have been mixed with, especially Annie Gloméron, Sylvie Dervin for their always efficient logistical support and the director Frédéric Abergel for maintaining a good working atmosphere inside the laboratory.

I would also like to thank all the people I met at the Cardiff School of Computer Science & Informatics at Cardiff university: Yaser Alosefer for his limitless motivation and our rich discussions, Ioan Petri for his sense of humor and Rafael Tolosana Calasanz, Ricardo J. Rodríguez, Raquel Plumed Ferrer, Jose Angel Bañares for bringing some Spanish sun in Cardiff and the great time spent together.

This acknowledgement section would not be complete without mentioning my family: my parents Patrice and Patricia, my little sisters Léna and Célia. It was awesome to grow and evolve among them.

Of course completing this thesis would not have been possible without the unconditional support and the patience of the cold pow marmot.

To Cécile, Lucette, Marie-Josée and Théophile.

Contents

List of Tables

List of Figures

Introduction

Context

Computers have often been used to host and process, through the data, representations of real or virtual worlds. In the computer science history, some of the first use cases of computers were the simulation of natural or artificial processes (trajectory computation, stress computation inside a structure...). Today, computers are still used for this purpose, but many other applications have emerged such as office automation or entertainment. Another important usage of computers is related to communication, data and information sharing, with the huge development of the word wide computer network, the Internet.

As vision and hearing allow us to capture and then to interpret our environment, some data representations and algorithms have been developed to allow computers to process audio, images and dynamic images through videos. We refer to this field as multimedia processing. The analog physical quantities of the real world have to be digitalized to fit the constraint that computers can only process binary signals. The development of multimedia applications allowed the user to watch images and videos and listen to sounds thanks to their computer. The content was, however, brought to the user under the same exact form as during its creation. Quickly, the users have shown the desire to interact with the data, for instance by changing the image or video viewpoints in order to move into the scene. A new domain then emerged: the **virtual reality**. Computer were here used to produce successive images of a saved scene according to the viewpoint defined by the interactive orders of the user. The users could thus navigate inside virtual environments.

A three dimensional (3D) representation of objects was required to enable the rendering of images of a scene from different viewpoints. 3D surface meshes were quickly adopted for this purpose. A 3D surface mesh consists in a collection of adjacent elementary polygons called faces that together constitute a piecewise approximation of the represented surface. Like images, meshes can be either static or dynamic if they vary or not with time. In this thesis, we will only deal with static meshes. Several ways exist to produce 3D surface meshes. They can be either created directly by designers on a computer (synthetic meshes) or reconstructed from data coming from 3D scanners (reconstructed meshes). These capture devices reconstruct a 3D mesh from the acquisition of multiple points belonging to the object to represent. The Figure 1 depicts examples of synthetic and reconstructed 3D meshes.

The 3D surface mesh representation became popular because it was simple and could easily be processed by computers. Electronic chips dedicated to the rendering of images from 3D meshes were designed. They are today called Graphical Processing Units (GPUs) and are present in most personal computers and smartphones. Application Programming Interfaces (APIs) such as OpenGL[1] and Microsoft Direct3D[2] have been developed to allow software to use the GPU to render 3D data.

During the same time, computational simulation applications developed the usage of massive volume meshes. Volume meshes consist in a collection of adjacent elementary polyhedron called cells that together constitute a piecewise approximation of the represented volume. Indeed, when simulating physical phenomena, some equations cannot be solved analytically. To find an approximative solution, one must resort to discretization through meshes. The equations are then numerically solved on the elementary cells until the convergence is obtained. If the studied phenomenon has a 3D nature, one must resort to volume meshes.

[1] http://www.khronos.org/opengl
[2] http://msdn.microsoft.com

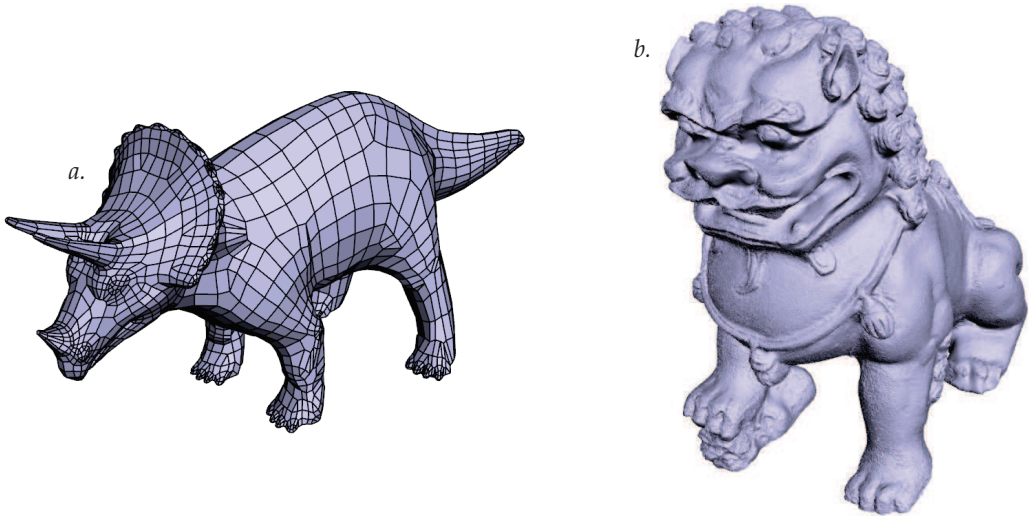


Figure 1: 3D mesh examples. *a.* A synthetic mesh. *b.* A reconstructed mesh.

In this thesis, we will mostly study 3D surface meshes. They are today used in numerous application domains for the representation and the visualization of 3D objects.

- In **Computed-Aided Design** (CAD), the engineers can visualize their models in 3D and interact with them through their surface mesh representations (see Figure 2). For example, Catia[3] from Dassault Systèmes is a well known software suite that allows to design complex industrial products with 3D views.

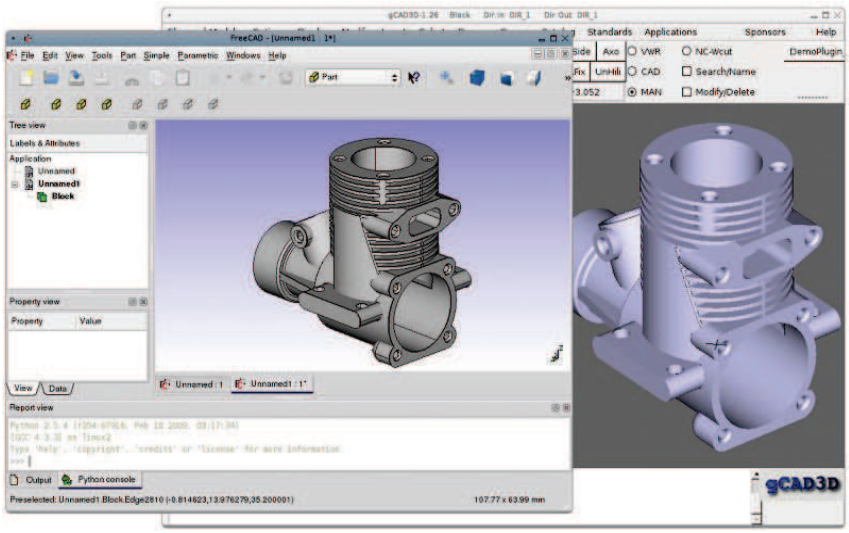


Figure 2: Modeling a 3D part with a CAD software. FreeCAD - www.freecadweb.org

Industrial objects acquired with a 3D scanner can be reverse engineered so as to find their design primitives. Injected in CAD software, the virtual representations of the objects can then be modified.

The users have commonly access to virtual reality through their computer screen. Immersive display environments, such as caves and image walls, were also developed to put the user at the center of a

[3] http://www.3ds.com/products/catia/welcome/

virtual world. Some installations have even been equipped with a stereo vision system, thus providing to the user the feeling that the virtual scene has a depth. These systems have many advantages in the context of product development. For instance, they allow to partially test a car without any costly physical prototyping.

- Some **3D medical imaging** methods allow to build 3D mesh representations of the organs of a patient to ease the diagnosis. They also enable the simulation of medical examinations or surgical gestures. For example in [Kühnapfel et al., 2000], the authors present a virtual reality training system for endoscopic surgery.

- The **digital capture of heritage** enables the convenient access to artwork from anywhere at anytime. It is therefore also easier to have access to great pieces of history and to conduct research on it. Koller et al. exhibit the challenges behind the creation of digital archives of 3D cultural heritage models [Koller et al., 2010].

- **E-commerce** applications have also found benefits in offering to customers interactive virtual replicas of sold products. Companies such as PackshotCreator[4] already propose automatic systems that can take pictures of product from different viewpoints. On the product web page, the customer can then interactively change the viewpoints of the displayed image to watch the product from different view points.

  With the development of the WebGL technology[5], which enables the inclusion of interactive 3D content inside web pages, future e-commerce web sites may include mesh representations of their products. They indeed offer more varied rendering features and user interactions.

  Some augmented reality frameworks have also been developed to allow customers to virtually try clothes [Hilsmann and Eisert, 2009] or shoes [Eisert et al., 2007]. These systems can display the 3D model of the tested product on the captured image of the customer.

- The **entertainment industry** is also a big 3D mesh producer. Indeed, meshes are used to represent the environments and the characters in 3D video games and animation movies (see Figure 3). In this domain, a lot of research is conducted to answer to the commercial need to generate games and animation movies as physically realistic as possible.

Figure 3: A frame of the Big Buck Bunny animation movie - Blender Foundation | www.blender.org. The characters of the movie were modeled with 3D meshes.

- Regarding **computational simulation**, the results of the computations are often visualized under the form of surface meshes with, for example, colors at nodes or cells that represent the computed values.

[4] http://www.packshot-creator.com/
[5] http://www.khronos.org/webgl/

This application is called scientific visualization. An example of scientific visualization mesh is shown on Figure 4.

The work described in this thesis was developed in the context of the Collaviz project [Dupont et al., 2010]. The aim of the project was to design a collaborative platform for the remote scientific visualization. It had to allow scientists working on remote sites to share, discuss and interact with simulation results. The output of the simulations performed on a high performance computer cluster were pushed to a common central server. Then, each of the members of the collaborative visualization session could ask for a specific post-processing operation, which produced a 3D mesh. This mesh was then transmitted to each of the clients to enable the interactive visualization. The users could then collaborate by sharing common annotations or viewpoints on the 3D data and discuss through a chat system. The Collaviz project was decomposed into 8 sub-projects. The aim of the sub-project in which this thesis took place was to develop solutions for the **remote scientific visualization of meshes**. Consequently, we built tools to stream and visualize efficiently such models. The Collaviz platform was designed to allow the users to access their data under different network and terminal conditions. This is why, as explained in the following, we had to resort to **3D data adaptation**.

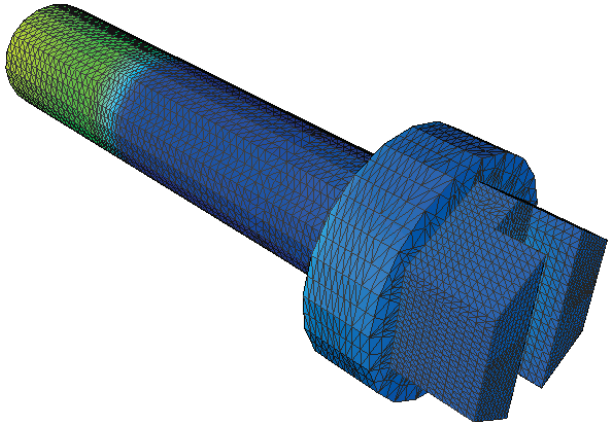


Figure 4: An example of a scientific visualization mesh. Colors at vertices allow to visualize the temperature on the surface of the object.

In all the previously described application domains, the needs of precision have never ceased to rise. For example, the meshes generated for the simulation have an increasing number of cells in order to get results that are close as possible to what is happening in the real process. For other fields such as digital heritage, the precision of 3D capture devices is also improving, thus producing larger data.

As a 3D surface mesh is a discrete approximation of a surface composed of polygons, increasing the precision means increasing the number of polygons. The more polygons a mesh contains, the bigger the mesh data is. The application domains, like remote scientific visualization for the Collaviz project, require this data to be stored and sometimes transmitted over networks. Data storage and transmission have an economic cost. Some meshes are today so big that their storage and transmission represent a significant cost. Mesh compression can help to reduce these costs.

The diversity of the computing devices used to visualize, or more generally process large meshes, is also challenging. A large scientific visualization mesh that needs a graphic computer cluster to be visualized cannot be directly rendered on a smartphone. Between these two systems, there is a tremendous difference of available storage, central memory, CPU and GPU resources. The same problem arises for the network. The amount of available bandwidth of an optic fiber connexion and the bandwidth of a DSL connexion are not of the same order. Yet, the user may want to access his data in all these different conditions. To allow the universal access, one must resort to **adaptation**.

3D data adaptation consists in transforming an input set of 3D objects into alternative modalities to achieve the best visualization experience according to network characteristics, device capabilities and user

preferences. Such adaptation mechanism allows, for the remote user, an optimal streaming of the 3D scene. In the next section, we explain why we think that mesh compression can stand as an adaptation solution for 3D data.

Mesh compression

3D mesh compression aims at encoding an input model with less bytes than its original representation. The complete decompression operation restores a mesh that is identical (lossless compression) or close to the input model (lossy compression). By reducing data size, mesh compression allows an easier storage and a faster transmission, which is useful in low resource conditions. Mesh compression therefore adapts 3D mesh data to storage and transmission constraints. Besides, some types of algorithms can give access during the decompression to different versions of the input mesh. These versions can either be different levels of detail or different parts of the original mesh. Consequently, they contain less polygons than the input model and can be more easily visualized on devices with low computational resources. Mesh compression in these cases adapts 3D mesh data to visualization constraints. Nevertheless, compression must be used keeping in mind that compression and decompression are complex operations that are time consuming. Sometimes, the positive effects produced by the reduction of data size can have negative effects in term of computation time.

We classify mesh compression algorithms into four different types, depending on the features they offer during the decompression.

Single-rate algorithms (see Figure 5 *a*) are the most simple schemes. They build a compact representation of an input mesh. The decompression algorithm generates a mesh that is identical to the input model or that slightly differs. The main motivation of these approaches is the storage and simple transmission.

Progressive algorithms (see Figure 5 *b*) embed in the compressed data a multiresolution representation of the input mesh. The decompression algorithm reconstructs successive levels of detail as more data is decoded. Progressive schemes are interesting for the remote visualization since the user do not have to wait for the full data to be downloaded and decompressed to visualize the model. An other advantage of this type of algorithms is that it is possible to select the best level of detail to display according to the device rendering capabilities, the network constraints or the visualization viewpoint.

With the two previous types of algorithms, the full data must be downloaded and decompressed to access a specific region of interest of the input mesh at the original resolution. In such case, the efficiency of these approaches is therefore strongly limited. By downloading and decompressing the full model while the user only requires a part, network and computational resources are wasted. Besides, the user must wait for data he does not need. The two next families of mesh compression algorithms can partially answer to this problem.

Random accessible algorithms (see Figure 5 *c*) allow to decompress only requested parts of the input mesh. The decompression algorithm restores an extracted part of the input model. The data that codes the others parts of the mesh are ignored. These methods enable the access to models that do not fit into the device main memory. However, the user does not have any overview of the not selected parts.

Progressive random accessible algorithms (see Figure 5 *d*) combine the features of progressive and random accessible algorithms. Different parts of the input model can be decompressed at different levels of detail, thus allowing view-dependent decompression. The decompressed mesh offers a good visualization experience at a given viewpoint with an affordable cost in term of transmitted data and number of polygons.

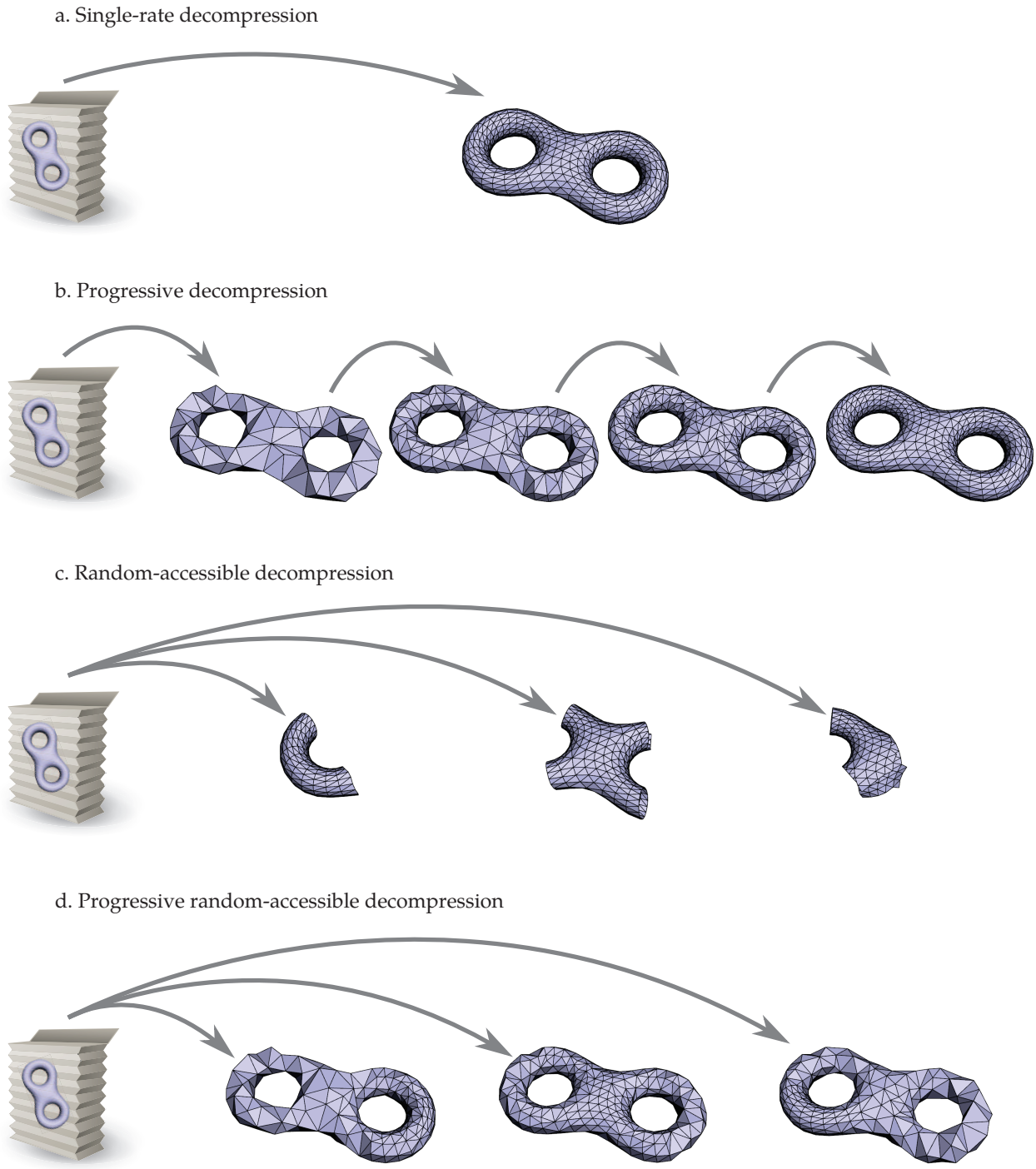


Figure 5: The four types of mesh compression algorithms and the modalities they offer at decompression time.

Contributions

The work described in this thesis deals with mesh compression as a method for 3D adaptation. Consequently, we focused on progressive and random-accessible techniques. After having studied the related literature, we developed a progressive polygon mesh compression algorithm, a metric for polygon mesh decimation, an adaptation framework for remote visualization and two progressive random accessible algorithms. We designed each of our schemes keeping in mind the following qualities:

High compression performance. The data produced by the compression algorithms should be as small as possible to allow the efficient storage and transmission of the compressed models.

High rate-distortion performance. With a minimum amount of data, the decompression algorithm should output a mesh that is as close as possible to the original model.

High random-access granularity. The developed random-accessible schemes should be able to decompress the smallest requested parts of the input mesh with the least possible overhead.

Low processing times. The compression and the decompression algorithms should be as fast as possible to not extend too much the time needed to access to the mesh.

Memory efficient. The compression and the decompression algorithms should be able to run with a low amount of memory. More particularly, the decompression algorithm should be able to run on mobile devices that have few resources.

Generic. The proposed approaches should handle all the different types of mesh topologies or be easily adaptable to do so.

Parallel. As the architecture of modern computers is evolving towards many computing cores, the compression and decompression algorithms should be able to perform as many steps as possible in parallel.

Easy to implement. The implementation of the proposed algorithms relying on well-known third-party libraries should be as easy as possible.

Existing progressive mesh compression schemes have exclusively been designed to compress surface meshes only composed of triangle faces. Nevertheless, surfaces meshes composed of other types of polygons are commonly used. We therefore designed a new progressive mesh compression scheme called **PPMC** (Progressive Polygon Mesh Compression). It compresses manifold polygon meshes with arbitrary face degrees. Starting from a simple scheme based on a new polygonal decimation operator, we proposed connectivity prediction schemes, a wavelet formulation and an adaptive quantization method to increase the rate-distortion performance of the algorithm. As progressive mesh compression is linked to mesh decimation, we also proposed a new simple volume metric to drive the simplification of polygon meshes.

Progressive mesh compression algorithms such as PPMC allow to adapt 3D data to storage, transmission and visualization constraints. However, they must be used in conjunction with an adaptation framework that chooses which level of detail, must be downloaded, decompressed and displayed to the user according to the various constraints. In the context of the Collaviz project, we proposed an adaptation framework for the remote scientific visualization of meshes. This framework was designed to optimize the user remote interactive visualization experience by taking into account constraints coming from the network (bandwidth), the visualization device (memory, computational capabilities and screen resolution) and the user (distance to the model).

Nevertheless, with progressive mesh compression algorithms, to access to a specific part of the mesh at the finest resolution, the full model must be downloaded and decompressed. A better efficiency would be possible if the decompression could adapt to an additional parameter: the user region of interest. Progressive random accessible algorithms allow to download and decompress any part of the input mesh at any level of detail. The decompression algorithm can refine on-demand the regions of interest and keep the others at a coarse level of detail. This enables view-dependent decompression. Such schemes adapt the number and the location of the decompressed polygons according to what the user wants to see.

We first designed a progressive random accessible triangular mesh compression scheme based on an initial segmentation of the input model. We named it **PRAM** (Progressive Random Accessible Mesh). The generated clusters are compressed independently by a progressive algorithm. With an appropriate segmentation, this approach is suited to view-dependent decompression but it cannot combine a high granularity random-access with a high compression performance. The closing of holes between partially decompressed clusters remains also problematic.

We therefore designed a second progressive random accessible triangular mesh compression scheme named **POMAR** (Progressive Oriented Mesh Accessible Randomly). It allows a much finer random access granularity. With this approach, no initial segmentation is performed. The decompression algorithm generates one piece decompressed models with a smooth transition between regions decompressed at high and low levels of detail. All the decompressed triangles come directly from the decimation and therefore respect its metric.

Overview

In the first two chapters, we recall some preliminary knowledge and previous work about meshes and single-rate mesh compression.

- The Chapter 1 introduces preliminary knowledge about 3D meshes and data compression. It defines a mesh and what are its elements and properties. After a description of general data structures to store meshes, it provides the general principles of data compression.
- The Chapter 2 is a state-of-the-art about single rate mesh compression. Starting from the pioneering work, we review all relevant approaches proposed for the compression of static meshes.

The next four chapters are dedicated to previous work and our contribution about progressive mesh compression, mesh decimation and mesh adaptation.

- The Chapter 3 is a state-of-the-art about progressive mesh compression.
- The Chapter 4 describes PPMC, our polygon progressive mesh compression algorithm. Experimental results are provided to demonstrate the performance of the approach.
- The Chapter 5 is about polygon mesh simplification. It presents our new simple volume metric that can drive the decimation of polygon meshes. It also show how this metric can in some cases improve the rate-distortion performance of PPMC.
- The Chapter 6 demonstrates the utility of progressive mesh compression algorithms in a context of remote scientific visualization. It describes a new simple adaptation framework that manages the multiple constraints of this application.

The last three chapters deal with previous work and our contribution about progressive random accessible mesh compression.

- The Chapter 7 is a state-of-the art about random accessible and progressive random accessible mesh compression.
- The Chapter 8 describes PRAM, our first progressive random accessible mesh compression algorithm based on the segmentation of the input mesh. Experimental results demonstrate the efficiency of the approach.
- The Chapter 9 deals with POMAR, our second progressive random accessible mesh compression algorithm. Experimental results are also included in this chapter.

Finally, this thesis ends by a general conclusion that outlines general perspectives for future work.

Chapter 1

Preliminaries on meshes and compression

1.1 Introduction

This chapter aims at providing preliminary knowledge to understand the rest of this thesis. This manuscript is about mesh compression. So the Section 1.2 describes what is a mesh in a practical and a formal way and what are its properties. The Section 1.3 then addresses the mesh storage in a computer memory trough specific data structures. Finally, the Section 1.4 introduces some general notions about data compression.

1.2 Definition of a mesh

1.2.1 Practical definition

A 3D mesh is a discrete representation of the surface or the volume of a 3D object. It can be seen as an unstructured grid or a graph with geometric properties. A mesh is actually a hierarchical assembly of different elements represented on Figure 1.1.

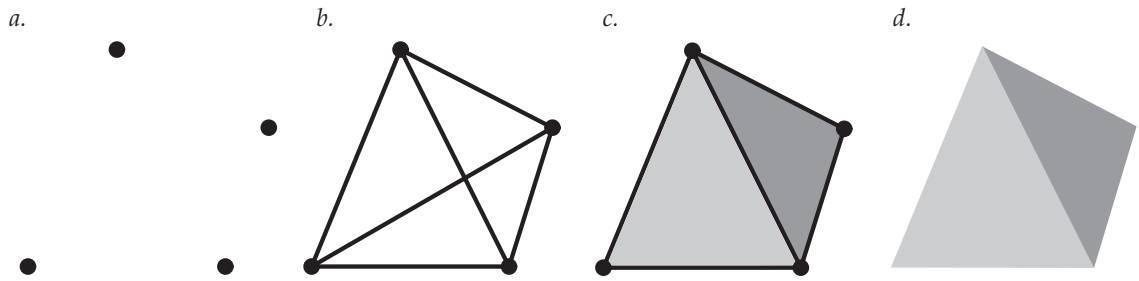


Figure 1.1: Elements of a mesh. *a.* A set of vertices. *b.* Vertices are connected two by two through edges. *c.* Closed paths of edges constitute faces. *d.* The interior of the surface defined by faces forms a cell.

- **Vertices** are the base elements of the mesh. They define a position in a common 3D Cartesian space.
- **Edges** are segments that connect two vertices of the mesh.
- **Faces** are polygons defined by a closed path of edges.
- **Cells** are polyhedron defined by a closed surface of faces.

The information contained in a mesh is often divided into three categories.

- The **geometry** information is the position of each vertex of the mesh in the 3D Cartesian space.
- The **connectivity** information (sometimes called topology or structure) describes the incidence relations between the mesh elements. It is commonly studied as a graph and as a consequence the vocabulary and the results of the graph theory can be applied to mesh connectivities.

  The Figure 1.2 depicts the geometry and the connectivity information of a simple mesh.
- The optional **attribute** information associates scalar or discrete properties to the mesh elements (vertices, edges, faces and cells) useful for the applications. For example, a color, defined by three floating-point numbers can be associated to each vertex or face. Textures coordinates, stored as two floating-point numbers per vertex, allows to apply a 2D texture to a 3D mesh by creating a mapping between the image of the texture and the surface of the mesh. Vertices normals, defined by three floating-point numbers, allow to render a mesh with lightening effects.

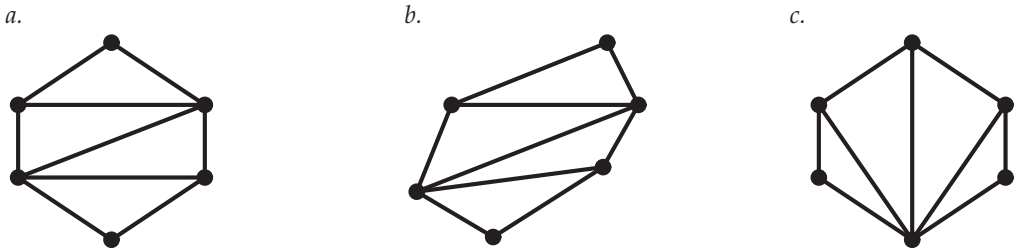


Figure 1.2: Illustration of the connectivity and the geometry of a mesh. *a.* A simple mesh. *b.* A mesh with the same connectivity as the first mesh but with a different geometry. *c.* A mesh with the same geometry as the first mesh but with a different connectivity.

Figure 1.3: Textured mesh with its associated 2D texture.

A 3D mesh can be either surfacic or volumic as show on Figure 1.5.

Surface meshes do not contain cells. They only define surfaces that can be closed or not. Most of the models used in video games and animation films are surfacic because only the exterior shape of the model is used to render them. A surface mesh is said to contain boundaries if some of its edges are adjacent to only one face as depicted on Figure 1.4.

Volume meshes have their inner volume filled by their cell elements. They are often used in 3D computational simulations. For instance, finite elements methods decompose a volume domain in elementary cells and apply iteratively the physic equations to all these elements until the convergence is obtained.

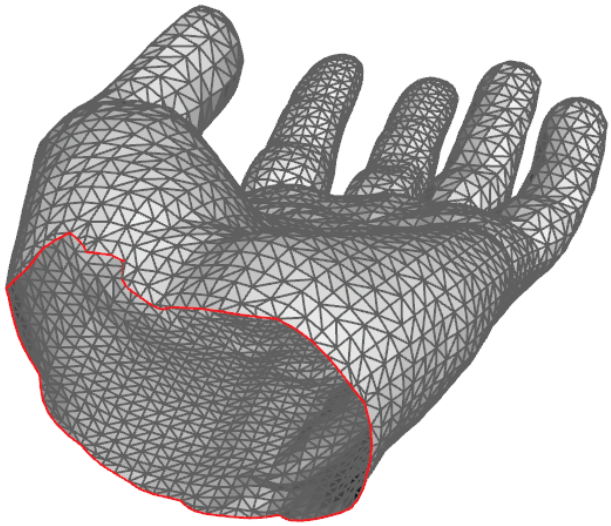


Figure 1.4: Surface mesh with a boundary. Edges in red are adjacent to only one face.

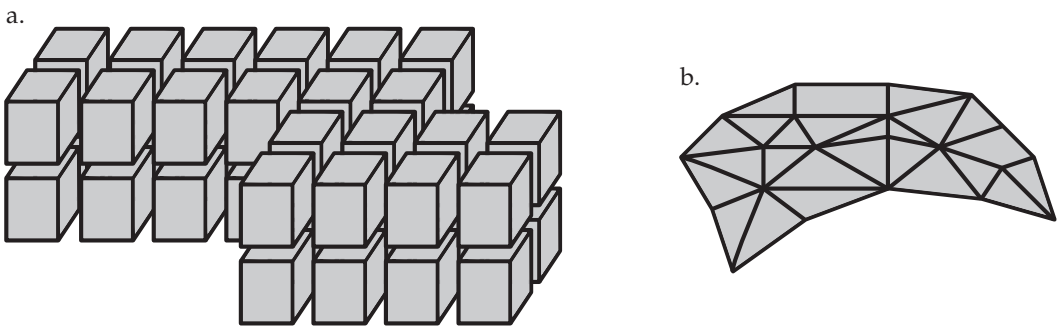


Figure 1.5: Volume and surface meshes. a. An exploded view of the cells of a volume mesh. b. A surface mesh.

1.2.2 Valence and regularity

In graph theory, the number of edges incident to a vertex is called the *valence* (or *degree*) of this vertex. For meshes, the number of edges of a face or the number of faces of a cell are also called valence (or degree) of this face or this cell. The faces can have a degree superior or equal to three. The common cases for surface meshes are degree three (triangle face) and degree four (quadrangle or quad face). In the same way, the common cases for volume meshes are cells with a degree equal to three (tetrahedron) or four (hexahedron).

Triangle or quadrangle faces may be privileged according to the context. For instance, a triangle mesh can better represent a spherical surface while quadrangle will more suit to the representation of a cylindrical surface. These properties are often used by mesh designers. By stating that triangle propose a linear approximation and quadrangle a bilinear approximation, the work of D'Azevedo [D'Azevedo, 2000] demonstrates the efficiency of these two types of polygons according to the shape of the surface to represent.

Some mesh connectivities are said to be *regular* because they are composed of the repetition of the same pattern. If only few elements of the mesh do not have a regular connectivity, then the mesh is said to be *semi-regular*. When the connectivity of the mesh does not contain at all regular structures, it is said to be *irregular*. Figure 1.6 shows examples of mesh connectivities with different regularities. The vertex degrees of a regular triangle mesh are always six. And the vertex degree of a regular quad mesh is always four.

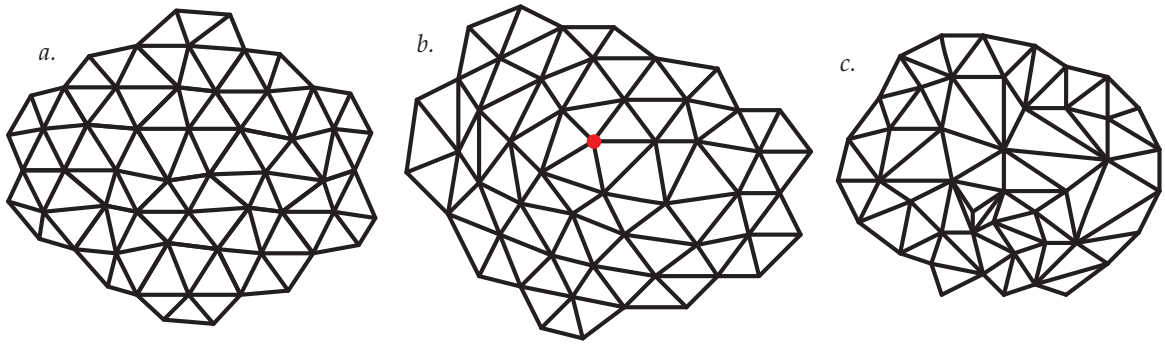


Figure 1.6: Regular, semi-regular and irregular meshes. *a.* A regular triangle mesh. The degree of each inner vertex is always 6. *b.* A semi-regular mesh. The red vertices has a valence of 5 while the other inner vertices have all a valence of 6. *b.* An irregular triangle mesh. The degree of each vertex is undefined.

1.2.3 Formal definition

The previous section provides a practical definition of a mesh. For most applications, this definition is sufficient to be able them. Nevertheless, when some topological properties of a mesh need to be defined, checked or manipulated, a formal definition becomes mandatory. In this subsection, we outline several ways of formally defining a mesh.

In the literature, 3D meshes have often been defined as *polytopal complexes* or *simplicial complexes* [Edelsbrunner, 2001, Edelsbrunner and Harer, 2010] through algebraic topology.

A polytope in the Mathematics literature, can reference several objects that are not equivalent. The mathematician Ludwig Schläfli [Schläfli, 1901] originally defines it by first stating that a 0-dimensional polytope is a vertex. A 1-dimensional polytope is an edge and bounds two 0-dimensional polytopes. A 2-dimensional polytope is a polygon and bounds n 1-dimensional polytopes. A 3-dimensional polytope is a cell and bounds m 2-dimensional polytopes, and so forth. Therefore, a mesh is a *polytopal complex*, an assembly of polytopes.

A Simplex or *k-simplex* refers to a polytope that is the convex hull of its $k-1$ elements. Thus, a 0-simplex is a vertex. A 1-simplex is an edge. A 2-simplex is a triangle. And a 3-simplex is a tetrahedron. A *simplicial complex* is built by gluing together simplices. What is called a triangular mesh is often a *simplicial complex* that has additional topological properties.

The work of Ledoux and Shepherd described in [Ledoux and Shepherd, 2010] focuses on hexahedral meshes. However, it starts with general definitions for any kind of meshes. Their definition of a mesh is not based on polytopes but on what they call *i-dimensional cells*. *Cells*, like polytopes, have a dimension: a vertex is a 1-dimensional cell, an edge is a 2-dimensional cell and so on. They define formally cell incidence and adjacency relations to finally propose their definition of a *n-dimensional topological mesh* or *n-mesh*, where n is the maximum dimension of the mesh cells.

1.2.4 Manifold property

A surface mesh is *2-manifold* if all its vertices have a neighborhood homomorphic to a closed or open fan, as shown on Figure 1.7. The neighborhood of a vertex is homomorphic to an open fan if its contains boundaries. The manifold property of a mesh is important as a lot of algorithms and data structures are only compatible with 2-manifold meshes.

The formal definition of the *manifold* property of Ledoux and Shepherd [Ledoux and Shepherd, 2010] states that a n-mesh is *manifold* if there exists, for each of its cells, a n-topological ball surrounding it inside the mesh. For Popović and Hoppe [Popović and Hoppe, 1997], a mesh is *2-manifold* if each of its vertex has

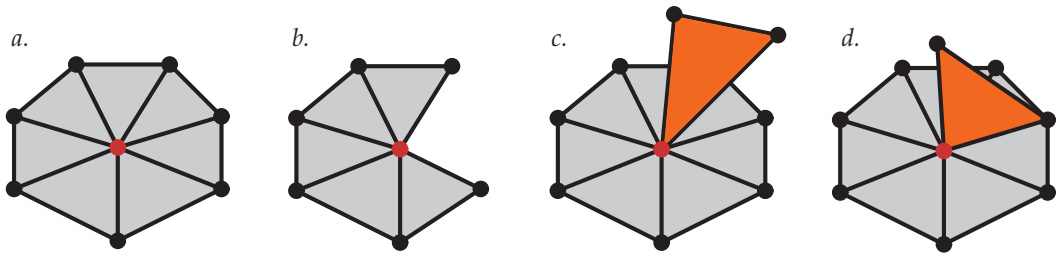


Figure 1.7: Manifold and non-manifold vertices. *a.* The red vertex is manifold because its neighborhood is equivalent to a closed fan. *b.* The red vertex is manifold because its neighborhood is equivalent to an open fan. *c.* & *d.* The red vertex is non-manifold because its neighborhood is not equivalent to an open or closed fan.

a neighborhood homomorphic to $\mathbb{R}^2$ or $\mathbb{R}^2_+$ where $\mathbb{R}^2_+ = \{x \in \mathbb{R}^2 : x_1 \geq 0\}$, which seems equivalent to the previous definition.

It is possible to associate an *orientation* to each face of a 2-manifold surface mesh. This orientation of a face corresponds to a chosen cyclic order of its vertices. The orientations of two adjacent faces are *compatible* if they are opposite. A mesh is said to be *orientable* if all the orientations of its faces are compatible (see Figure 1.8 a). In the other case, it is said to be *non-orientable* (see Figure 1.8 *b*).

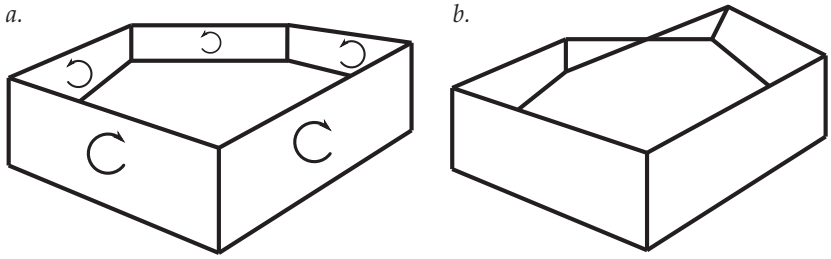


Figure 1.8: Mesh orientation. *a.* An orientable mesh: every faces have a compatible orientation. *b.* A non-orientable mesh: a simple version of the Möbius strip.

1.2.5 Euler characteristic

The Euler characteristic χ of an 2-manifold orientable mesh defines its topological type. It is equal to:

$$\chi = v - e + f,$$

where v, e and f are respectively the number of vertices, edges and faces of the mesh. A mesh is said to have genus g if it can be cut along $2g$ closed loops without disconnecting it. Intuitively, the genus is the number of handles of the mesh as illustrated on Figure 1.9.

It can be proven that the Euler characteristic is also equal to:

$$\chi = 2(s - g) - b,$$

where s is the number of connected components of the mesh and b is its number of boundary loops. Consequently, the following equation can be written for an orientable 2-manifold mesh with one connected component and no boundary:

$$v - e + f = 2 - 2g. \tag{1.1}$$

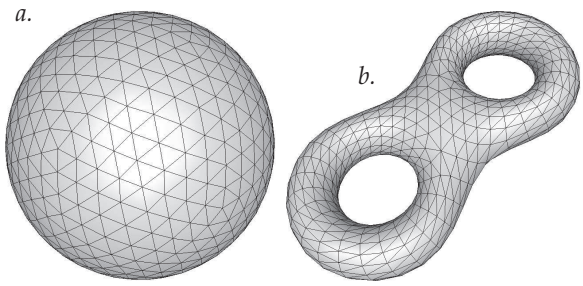


Figure 1.9: Mesh genus. *a.* A mesh with a genus 0. *b.* A mesh with a genus 2.

In the case of a triangular mesh, each face has 3 edges and each edge is incident to two faces. If we start with one triangle face and we add to the other side of its edges new faces, we have created three times the previous number of faces and two times the previous number of edges. Therefore, the following relation can be written:

$$3f = 2e. \tag{1.2}$$

By substitution of Equation 1.2 in Equation 1.1, the following relations can be obtained:

$$v = \frac{4 - 4g + f}{2}, \tag{1.3}$$

$$e = 3(v - 2 + 2g). \tag{1.4}$$

If the genus of the mesh is low and it has a significant number of faces, the equation 1.3 becomes:

$$f \approx 2v.$$

So, a 2-manifold orientable triangle mesh has about twice more faces than vertices.

Each edge is incident to two vertices. It is counted when calculating the degree of both vertices. So, the average degree of a vertex in an 2-manifold orientable triangle mesh is equal to:

$$AverageDegree = 2e/v$$

Integrating the Equation 1.4, this expression becomes:

$$AverageDegree = 6(v - 2 + 2g)/v$$

Therefore, by considering that the mesh has a significant number of vertices and a low genus, we can conclude that the average degree of the vertices of a 2-manifold orientable triangle mesh is **6.**

1.3 Storing a mesh

As meshes are virtual objects processed by computers, data structures are needed to store them in memory or disk. Criteria to choose a mesh data structure are its compactness, the types of meshes it can store and the ability to perform incidence queries on the mesh elements. This section describes three basic mesh data structure. For a short review on data structures that allow random-accessible and progressive access, we refer the reader to the Section 7.3.3.

1.3.1 Indexed data structure

A lot of mesh file formats (OFF, PLY, OBJ, VRML, X3D) are based on an indexed data structure. The first part of an indexed data structure is composed of the list of all the vertex coordinates. This list stores the mesh geometry but also introduces an ordering of the mesh vertices. Vertices get their number identifier from their position in the list. The second part of the data structure describes the connectivity of the mesh. Each line is related to one face and lists all its vertices in a cyclic order. For volume meshes, the faces of each cell can also be stored in the same way. The Figure 1.10 shows an example of an indexed data structure that stores a surface mesh.

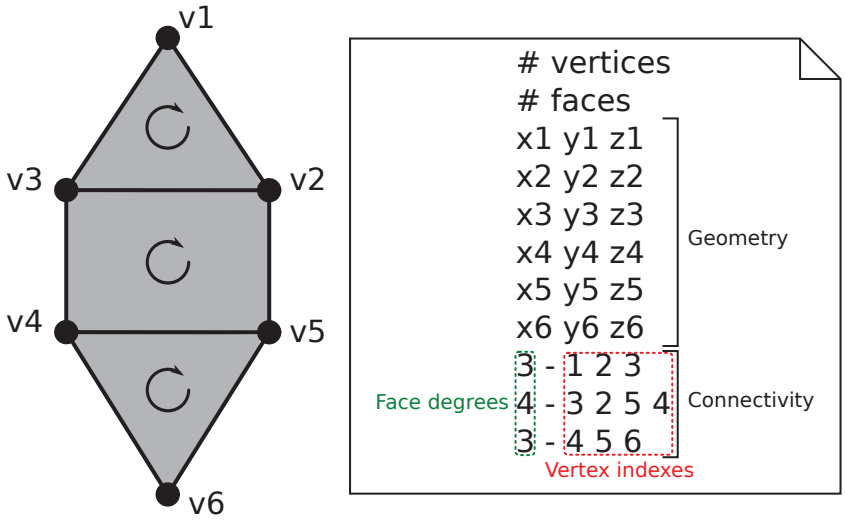


Figure 1.10: Indexed mesh data structure.

This data structure has the advantage of being simple. It can also store any type of mesh. However, it does not allow incidence queries from one element of a mesh.

1.3.2 Winged-edge data structure

The winged-edge data structure [Baumgart, 1975] can only store 2-manifold orientable polygon meshes. Its reference element is the edge. Each edge of the mesh is stored in a table (the edge table) with:

- its start and end vertices that define the orientation of the edge,
- its left and right face,
- its previous and next adjacent edges on its left face,
- its previous and next adjacent edges on its right face.

A separate table (the vertex table) stores each vertex with one of its incident edge and its position in the space. At last, each face is stored in the face table with one of its incident edge. The Figure 1.11 depicts a winged-edge data structure.

This data structure allows to perform incidence queries from a specified vertex, edge or face. For example, it is possible to retrieve all the neighbor vertices of one given vertex without going over the whole tables of elements. Nevertheless, it is not very efficient in term of storage size.

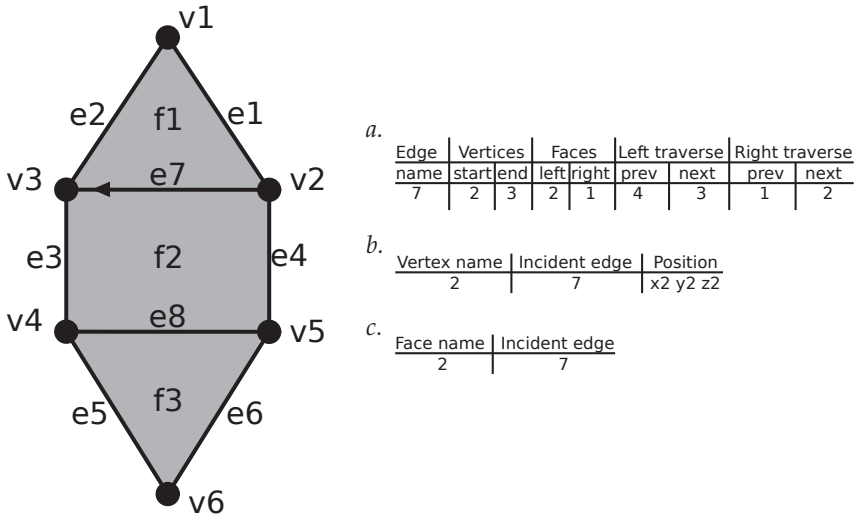


Figure 1.11: Winged-edge mesh data structure. *a.* The edge table with an example entry. *b.* The vertex table with an example entry. *c.* The face table with an example entry.

1.3.3 Halfedge data structure

The *halfedge data structure* or *doubly connected edge list* [Muller and Preparata, 1978] can also only store 2-manifold orientable polygon meshes. It is a simplified version of the winged-edge data structure that offers the same functionalities. Its base element is the half-edge: one oriented side of an edge. A table stores for each halfedge of the mesh:

- the next halfedge in the same face,
- the vertex it points to,
- the face it belongs to,
- its opposite halfedge that belongs to the same edge.

A separate table stores for each face one incident halfedge. An other one stores for each vertex its position and one incident halfedge. The Figure 1.12 depicts an halfedge data structure.

The advantage of the halfedge data structure over the winged-edge data structure is its compactness. It can store the same types of meshes and allows the same incidence queries with fewer element identifiers.

1.4 Data compression

As this thesis deals with mesh compression, this section introduces general definitions and principles about data compression. They are the basis of the algorithms presented in the following chapters.

In computer science, data compression consists in encoding an information in fewer bits than its original representation. Data compression can be either lossless or lossy. Lossless compression allows to restore the original data without any alteration after the decompression. Lossy compression retrieves an altered version of the information. Distortion has been introduced to further reduce the size of the data. Data compression is used to reduce the data storage and transmission costs. It also sometimes raises the comfort of the user by shortening the processing times.

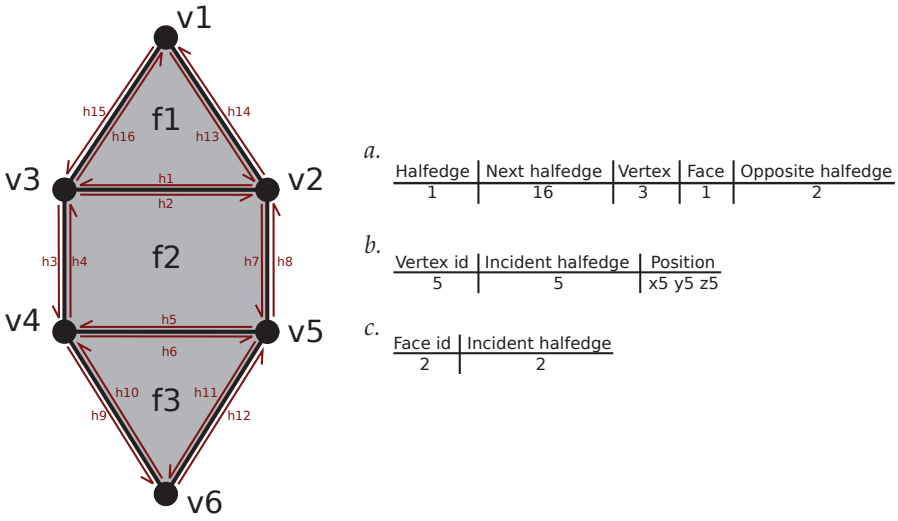


Figure 1.12: Halfedge mesh data structure. *a.* The halfedge table with an example entry. *b.* The vertex table with an example entry. *c.* The face table with an example entry.

1.4.1 Shannon entropy

The *Shannon entropy* [Shannon, 1948] of a signal characterizes its average unpredictability or its information content. The Shannon entropy of a signal X with n symbols $\{x_i : i = 1, ..., n\}$ can be expressed in bits as:

$$H(X) = -\sum_{i=1}^{n} p_{x_i} log_2(p_{x_i}),$$

where p_{x_i} is the probability of appearance in the signal of the symbol x_i. The Shannon entropy of a signal stands for the theoretical limit of the compression rate a lossless encoder can achieve.

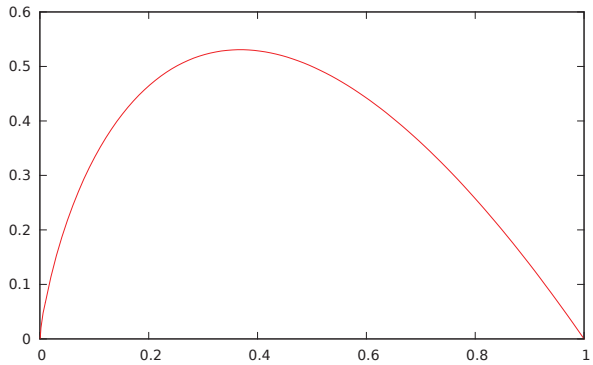


Figure 1.13: Curve of the function $f(x) = -xlog_2(x)$. The function extended to 0 is equal to 0.

Looking at the shape of the function $f(x) = -xlog_2(x)$ (Figure 1.13), it can be deduced that the Shannon entropy of a signal is low when this signal has a small number symbols with a biased distribution of probabilities. To minimize the entropy, symbols must be very frequent (high probability) or very little frequent (low probability). The common method to transform a signal into an other one that has a smaller entropy is to use *predictions*. The encoder makes an assumption on the next value to encode in function of previous values already encoded. It then encodes the difference between the prediction and the actual value.

If the prediction function is well chosen, the distribution of the residuals is concentrated around 0 and has a small number of different values. This allows an entropy coder to compress efficiently.

1.4.2 Entropy encoding

Entropy encoding is a variable length encoding technique that exploits the entropy of a signal to represent it with fewer bits than its original representation. We present in this section some popular schemes.

Huffman encoding [Huffman, 1952] is one of the most well-known entropy encoding technique. It has been widely used for sound, image and video compression. The encoder assigns to each symbol a sequence of bits, which length depends on the probability of the symbol. Symbol with a high probability are encoded with less bits than symbols with a low probability. Sequences of bits are chosen to be *prefix-free*: this means that a sequence cannot be the prefix of an other one. This allows the decoder to know the size of each sequence during the decoding. This technique has the advantage of being easily implementable. Furthermore, the compression and decompression algorithms are usually very fast. Nevertheless, as a symbol is always encoded with an integer number of bits, the optimal limit of the compression rate cannot be always reached.

Arithmetic encoding [Rissanen, 1976, Rissanen and Langdon, 1979, Witten et al., 1987] does not code a symbol with a fixed number of bits. The principle of this technique is to code a signal with a real number from the interval $[0, 1[$. The compression starts by subdividing the interval $[0, 1[$ in one interval per symbol. The width of these intervals depends on the probability of their corresponding symbols. The interval of the current symbol to encode is selected to be subdivided as previously described for the encoding of the next symbol. At the end of this progressive subdivision process, when there is no more symbol to encode, a real number belonging to this interval represents the whole signal. Arithmetic encoding is generally slower than Huffman encoding, but by not being limited to integer sequence of bits, it allows to get close to the Shannon entropy, the theoretical limit of the compression rate.

Range encoding [Martin, 1979] is very similar to arithmetic encoding. The difference is that the interval $[0, 1[$ is changed to an integer interval $[0, n^n[$. Its interest is that its implementation is often faster than arithmetic coding. Besides, it is claimed as being patent free. This means that range encoding can be used in a system without having to pay any license fee. It is often, however, slightly less efficient than arithmetic encoding.

The entropy coders need a symbol probability table to encode a signal. There is four different types of models which allow to build these tables:

- **Static models** hardcode the probability table inside the encoder and decoder.
- **Semi static models** get the symbol probabilities with a first pass on the signal. Then a second pass encodes the symbols with the probabilities obtained with the first pass. The probability table must then be provided with the compressed data.
- **Adaptive models** dynamically modify the probabilities tables each time a symbol is encoded. If the encoder and the decoder starts with an equiprobable model, just the number of different symbols has to be transmitted with the compressed data. However, the compression performance of the first symbols is reduced.
- **Quasi static models** dynamically modify the probabilities tables each time K symbols have been encoded. The principle is otherwise the same as with adaptive models.

Chapter 2

State of the art on single-rate mesh compression

2.1 Introduction

Single-rate mesh compression consists in encoding an input model with fewer bytes than its original representation. The decompression then decodes the compressed data to output a mesh that is identical to the input model or that slightly differs. As it reduces data size, single-rate mesh compression is therefore a way to **adapt 3D mesh data** to network bandwidth and storage constraints.

Compressing meshes is different than compressing other types of multimedia data such as sound, images or videos. The common point between sound, images and videos is that their structure is known in advance by the encoder and the decoder. For instance, an image is a two-dimensional signal. It is always structured as a grid of pixels. Image compression consists in compressing the three color components of each pixel. The traversing order is fixed inside the encoder and the decoder. Some image compression schemes use the *scanline* order, which traverses the pixel rows from the left to the right, starting from the top row to the bottom one. A mesh is, however, not necessarily regularly structured. Its connectivity is completely unknown to the encoder before the compression. So, besides having to code the vertex positions, as the pixel colors would be coded for an image, a mesh encoder must encode the structure, which is the connectivity.

Since the end of the 1990's, mesh compression has been an active research topic, pushed by the development of the applications related to 3D. Very good reviews about mesh compression, describing the fundamental approaches, have been published in 2005 [Alliez and Gotsman, 2005, Peng et al., 2005]. More recently, Avilés and Morán proposed a short state-of-the-art article about static mesh compression [Aviles and Moran, 2008]. But since 2005, no complete reviews including the novel advances in the domain have been published. The state-of-the-art chapters of this thesis aim at providing a list as most complete as possible of the relevant mesh compression approaches starting from the pioneering techniques and going to the most recent work.

This chapter focuses on the single-rate compression of static meshes. As the problem of mesh compression is often decomposed in two parts, **connectivity** and **geometry** encoding, this chapter is organized as follow. The Section 2.2 deals with the encoding the connectivity information of a mesh. The Section 2.3 addresses the encoding of the geometry. The fourth part lists some strategies developed to encode large models. Finally the fifth part describes some algorithms that do not restore the initial connectivity. The chapter ends with a conclusion that proposes perspectives for future work on this topic.

2.2 Connectivity compression

This section aims at describing the different approaches developed to encode the connectivity of 3D meshes. The general principle behind all these techniques is to perform a traversal of the mesh elements and emit symbols depending on the encountered configurations.

2.2.1 Triangle strip encoding

The first needs for compact mesh representations have actually been motivated by rendering applications that needed efficient mesh representation to quickly transfer the data between different memory units of the computer.

The rendering of a significant 3D mesh is a task that requires high computational capabilities. It is possible to perform all the required operations on the CPU. Nevertheless when the complexity of models increases, this solution becomes rapidly inadequate to achieve interactive frame rates. This problem has motivated the development of specific hardware. The GPUs are designed to perform the mesh rendering operations quickly in parallel. Specific API such as OpenGL or DirectX have been designed as portable ways to program GPUs. Efficient solutions to transfer the mesh data from the central memory to the GPU memory became of prime interest. Memory transfer in a computer is indeed a costly operation that consumes a lot of CPU cycles. So, a good method to transfer mesh data can significantly decrease the rendering time. Triangle strips and triangle fans are mesh representations that have been used for this purpose.

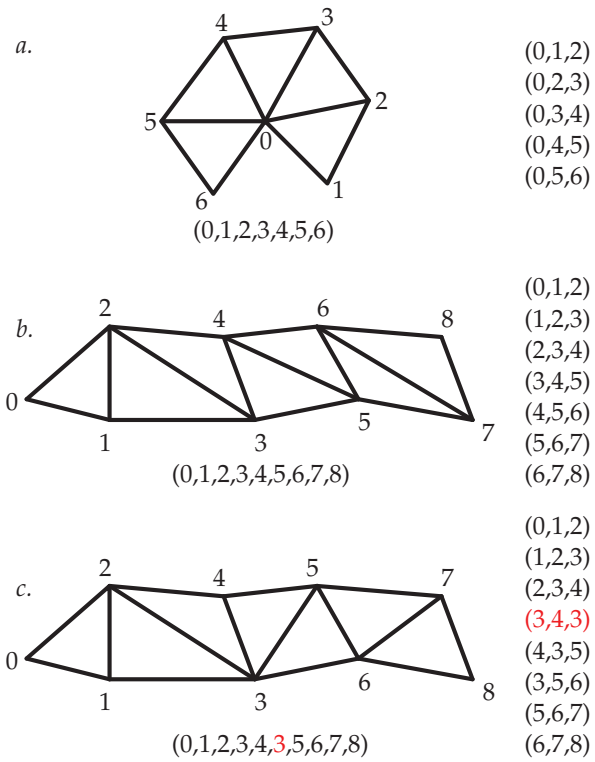


Figure 2.1: A triangle fan, a triangle strip and a generalized triangle strip. The numbers inside the parenthesis correspond to the indices of each triangle. *a.* A triangle fan. *b.* A triangle strip. *c.* A generalized triangle strip. The dummy triangle is in red and its corresponding added vertex is also in red.

A triangle fan is a sequence of vertices that defines a set of triangles sharing a common center vertex. The Figure 2.1 *a* shows an example triangle strip.

A triangle strip is a sequence of vertices where each added vertex defines with the two previous vertex of the strip a new triangle. The Figure 2.1 *b* shows an example of a triangle strip.

These methods are much more efficient than the indexed representation that requires three vertex indices to code each triangle. Indeed, after having encoded the first triangle, a new vertex index codes the connectivity of a new triangle. The mesh connectivity encoding with triangle strips and triangle fans can significantly reduce the data size.

In a triangle strip, to form a new triangle, the vertices are always taken in the same order: $\{n-2, n-1, n\}$. Generalized triangle strips do not always respect this order so as to generate longer strips. However, to preserve the triangle strip structure, a dummy triangle is added when the standard order is not preserved as shown on Figure 2.1 *c*. This dummy triangle is flat, therefore not rendered. Even if it costs one vertex, it is cheaper than the start of a new strip that costs two vertices. If the generalized triangle strip is long enough, the ratio between the number of triangles and the number of vertices is close to 1.

Optimal generation of triangle strips on a mesh was demonstrated as being an NP-complete problem [Evans et al., 1996a]. Consequently, strategies with heuristics must be employed [Evans et al., 1996b, Xiang et al., 1999]. Yet, these methods are still computationally intensive. So, it may be interesting to store a mesh with its stripification information.

One of the first work in mesh compression is the generalized triangle mesh format of Deering [Deering, 1995], which is based on generalized triangle strips. Deering noted that, in a generalized triangle strip, many of the interior vertices are repeated twice. So, instead of encoding twice their positions traversing the mesh, he proposed to push them in a 16 positions queue and refer to them by their location in the queue when needed.

Chow [Chow, 1997] designed an alternative method based on generalized triangle meshes. His approach allows to reuse more vertices than the original approach of Deering [Deering, 1995]. The algorithm starts from a mesh boundary and creates the first strip with the triangles incident to this boundary. It then removes these triangles and starts the second iteration with the new boundary. The algorithm knows a significant part of the new strip vertices because the new strip is incident to the boundary encoded at the previous iteration.

Bajaj et al. [Bajaj et al., 1999] proposed to compress triangular meshes with a layered decomposition. Vertex layers are vertex strings that cannot intersect each others. In most cases, they are separated by a distance of one edge between them, forming concentric circles. The triangles between the vertex strings form the triangle layers. Triangle layers contain triangle strips and fans, useful for the rendering. In this scheme, the connectivity data is composed of the layout of the vertex layers and their triangle strips and fans. This method compresses the mesh connectivity at about 1.5 to 6 bits per vertex. Besides, it can process non-manifold meshes.

2.2.2 Spanning tree encoding of planar graphs

As stated in Section 1.2.1, the connectivity of a mesh can be modeled as a graph. For surface meshes, the vertices are the nodes of the graph that are linked through edges to form faces. This analogy between meshes and graphs explains why some well-known results from the graph theory can be applied to the compression of mesh connectivity.

Tutte first proposed formula that enumerate planar triangulations [Tutte, 1962] and planar maps [Tutte, 1963]. He answered to the following question. How many combinatorially distinct rooted planar maps (with a set face orientation) are there with n edges or with k vertices or with l faces? Tutte's enumerations of planar triangulations [Tutte, 1962] and planar maps [Tutte, 1963] allow to calculate what was later called the Tutte's entropy. This entropy, approximately equal to 3.25 bits per vertex, stands for an upper bound of the entropy of any arbitrary surface triangular mesh connectivity. This result is, however, only theoretical. Tutte did not propose any compression scheme able to encode an arbitrary planar triangulation at a compression rate equal to this entropy. Nevertheless, Tutte's entropy value has been later used to prove the optimality of several mesh connectivity encoding schemes [Alliez and Desbrun, 2001b].

Turán [Turán, 1984] was one of the first to propose a practical method to encode in a compact representation a planar unlabeled graph. His encoding algorithm starts by numbering all the graph vertices. It then builds a spanning tree on the graph that connects all the vertices from a chosen root node. A depth-first traversal of the spanning tree is performed to construct a cyclic sequence of vertices (Figure 2.2 a). The edges of the graph not belonging to the previous spanning tree are listed (Figure 2.2 b). The generation of the symbol list transforms the cyclic sequence of vertices previously obtained ((see Figure 2.2 c) by:

- putting a '+' symbol when the current vertex directs away the root node of the spanning tree,
- putting a '-' symbol when the current vertex directs towards the root node,
- putting a '(' symbol if the next vertex is the first vertex of an edge not belonging to the spanning tree,
- putting a ')' symbol if the next vertex is the second vertex of an edge not belonging to the spanning tree.

Finally, each previously described symbol is replaced by its corresponding 2 bit code to obtain a compact representation of the graph. This connectivity encoding methods requires at maximum 12 bits per vertex.

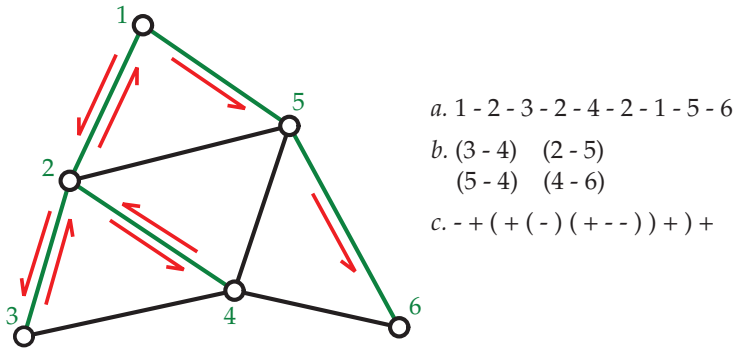


Figure 2.2: Turán's succinct representation of graphs. The edges belonging to the spanning tree are in green while the others are in black. The root node is the node number 1. *a.* The cyclic sequence of vertices obtained by following the red arrows. *b.* The list of edges not belonging to the spanning tree. *c.* The generated symbol list.

The algorithm of Keeler and Westbrook [Keeler and Westbrook, 1995] is also based on the encoding of a spanning tree with additional information for the edges not belonging to the spanning tree. The authors state that their method can encode an unlabeled planar triangulations in about 4,6 bits per vertex with their scheme. Chuang et al. [Chuang et al., 1998] later proposed a more efficient scheme based on a canonical ordering of the graph vertices in the spanning tree and multiple types of parenthesis.

2.2.3 Spanning tree encoding of meshes

The influence of the previously described planar graph encoding techniques can be felt on some of the first 3D mesh encoding techniques. For instance, the *topological surgery* algorithm of Taubin and Rossignac [Taubin and Rossignac, 1998] encodes a triangular mesh with two spanning trees.

A *vertex spanning tree* is first generated on the mesh (Figure 2.3 *a*). The sections of the spanning tree that only contain vertices connected to two other vertices of the tree are called *vertex runs*. In the article, some approaches are proposed to maximize the length of vertex runs. The vertex spanning tree is encoded by a depth first algorithm. Starting from a root leaf vertex, it encodes for each run, its length, a bit indicating whether the run has right sibling and a bit indicating whether the run ends with a leaf vertex. The vertex spanning tree is also used to encode the position of the mesh vertices as explained in Section 2.3.2.

As with the planar graph encoding techniques, the edges not belonging to the vertex spanning tree need also to be stored. That is why a topological surgery is performed on the mesh. The mesh is cut along the edges belonging to the vertex spanning tree. This generates a decomposition of the mesh in triangle strips (Figure 2.3 *b*). The dual graph of the cut mesh is the *triangle spanning tree*. It is encoded in the same way as the vertex spanning tree. The length of its runs and one bit per run telling if the run ends with a leaf triangle are stored.

The topological surgery algorithm allows to compress the connectivity of a manifold triangle mesh with about 2.5 to 7 bits per vertex.

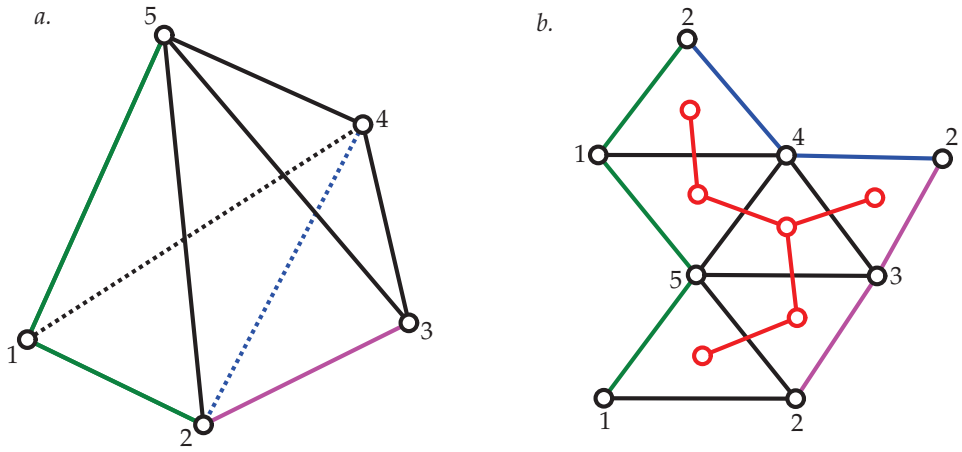


Figure 2.3: Geometric compression through topological surgery. *a.* The input mesh. The colored edges represent the runs of the vertex spanning tree. *b.* The input mesh is flattened after being cut along the edges of the vertex spanning tree. The triangle spanning tree is shown in red.

Diaz-Gutierrez et al. [Diaz-Gutierrez et al., 2005] proposed an original approach that encodes a genus-0 mesh with two vertex spanning trees, always distant of one edge, and additional information to define the triangles between the two trees. One advantage of this representation is that it embeds a hierarchyless multiresolution structure (through the two trees collapsible edges) and a stripification. Experimental results report connectivity compression rates of about 1.5 bits per vertex on average.

The algorithm of Li and Kuo [Li and Kuo, 1998a] encodes the connectivity of a triangle mesh with its dual graph. Each node of the dual graph is incident to three edges. So, the encoding traversal has just to perform a breadth-first traversal of the mesh dual graph and outputs one binary symbol per edge telling if this edge is connected to an already visited node or not.

2.2.4 Triangle traversal encoding

Vertex spanning trees do not contain all the mesh edges. They do not represent the whole mesh connectivity. That is why, in the previously described encoding methods, edges that do not belong to the vertex spanning tree are encoded in a separate step. A triangle spanning tree can, however, capture all the mesh faces and therefore represent the full connectivity. That is why some algorithms were proposed to encode a mesh with a region growing approach by generating a triangle spanning tree. To generate such trees, the algorithm iteratively processes the mesh triangles with a breadth first traversal. This means that the next triangles to process are always adjacent to already processed triangles and the first triangles that were added to the neighbor triangle list are processed first. As shown on the Figure 2.4, at each step of the compression, some faces have already been traversed and other not. There can have one or several closed edge borders between these two types of faces.

The Cut-Border machine [Gumhold and Straßer, 1998] follows this face traversal strategy. It extends the border formed by an initial triangle by iteratively traversing adjacent triangles. 7 different symbols code

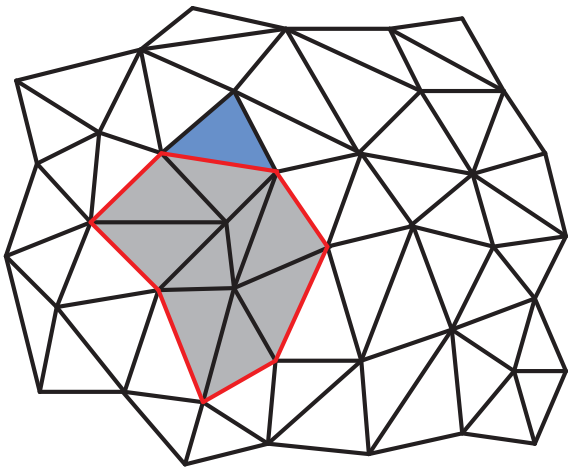


Figure 2.4: Encoding a mesh with a triangle traversal. The already traversed faces are in gray. The next face to process is in blue. The edge border between processed and not processed faces is in red.

whether the extension of the border on a new triangle was done by inserting a new vertex or not. If no vertex insertion is performed, the symbol describes how the new adjacent triangle is connected to the border. These symbols also code the closing of a border, its split or the union of two distinct borders. The scheme can compress 2-manifold triangle meshes. Experimental results reports connectivity compression rates of about 4 bits per vertex.

The Edgebreaker algorithm of Rossignac [Rossignac, 1999] encodes the connectivity of triangular meshes by iteratively nibbling the faces. It traverses the mesh faces by entering through one of their edges called the gate. The mesh connectivity is encoded by a code that describes the patch configuration of the vertex v belonging to the current triangle X in front of the active gate. The patch of v is the set of faces adjacent to v. There are five different configurations as shown on Figure 2.5. X is then removed from the mesh. The next gate points to the triangle adjacent to X that is on the right of the active gate. If there is no such triangle, the next gate is popped out from a stack. An example of encoding traversal is shown on Figure 2.6. An extension of the algorithm is proposed to handle non manifold meshes and meshes with a genus superior to 0. Experimental results report connectivity compression rates of about 4 bits per vertex.

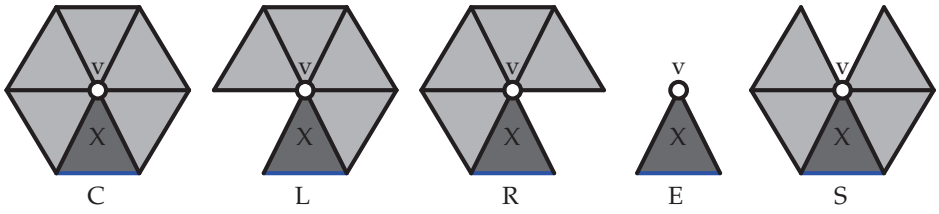


Figure 2.5: The five patch configurations of the Edgebreaker algorithm. v is the patch center vertex and X is the current triangle. The active gate is the blue edge. C: there is a complete triangle fan around v. L: there are missing triangles at the left of the active gate. R: there are missing triangles at the right of the active gate. E: v is only adjacent to X. S: there are missing triangles elsewhere than the left or the right of the active gate.

Some Edgebreaker derived schemes were proposed to guarantee a worst-case coding cost of 3.67 bits per vertex [King and Rossignac, 1999a] and then 3.55 bits per vertex [Gumhold, 2000]. Decoding algorithms with linear time and space complexities were proposed [Isenburg and Snoeyink, 2000b, Rossignac and Szymczak, 1999]. Szymczak et al. [Szymczak et al., 2001] optimized the original scheme to encode meshes with a high regularity. This methods has a worst-case coding rate of 1.62 bits per vertex

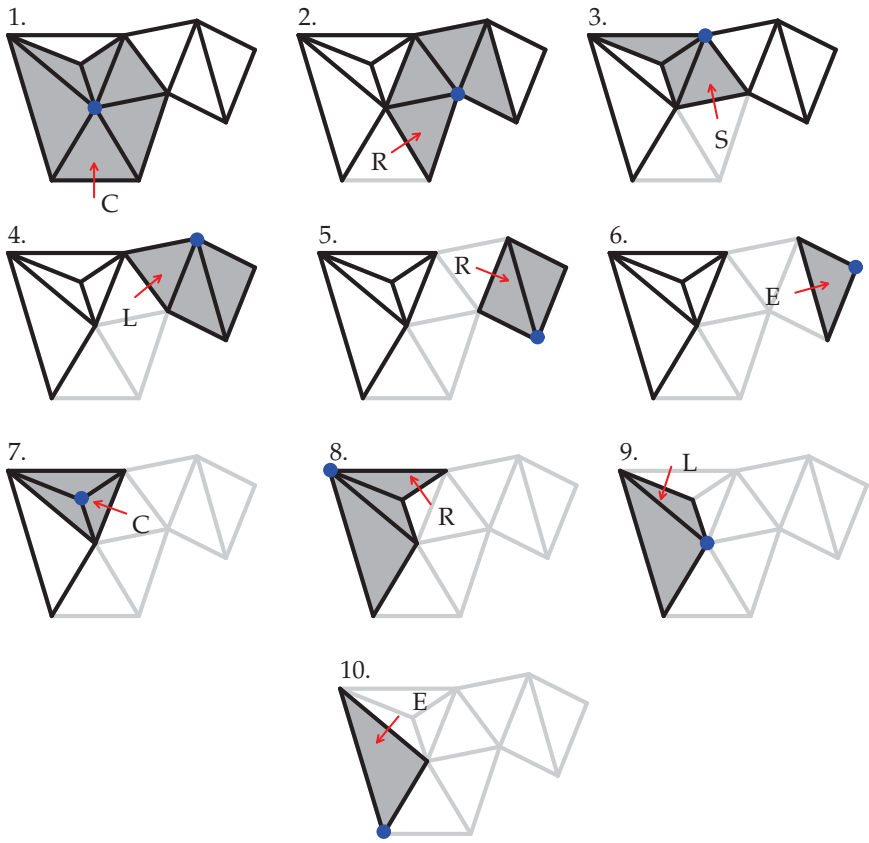


Figure 2.6: An encoding traversal of the Edgebreaker algorithm. For each step the active gate is in red, the current patch is in gray and its vertex v is blue. At step 3, 6, 7, 9 and 10 the gate is popped out the stack because the previous current triangle did not have a right triangle. The connectivity is encoded by the following symbol list: C R S L R E C R L E.

for large regular meshes. Coors and Rosignac [Coors and Rossignac, 2004] added a connectivity prediction method based on the mesh geometry. They claim to obtain, on average, connectivity compression rates that are below 0.75 bits per vertex. Gumhold [Gumhold, 2005] proposed to optimize a Markov model, for which each Edgebreaker symbol is a state, in order to design an asymptotic optimal arithmetic coder for the Edgebreaker encoder. Ying et al. [Ying et al., 2010] designed an algorithm to select the next edge gate to proceed in order to minimize the number of 'S' symbols. They also built a table that ranks the Edgebreaker symbols depending on the number of traversed faces connected to the gate vertices. This *code-mode*, which stands for a connectivity prediction method, allows after a highly improved entropy coding (about 2.2 bits per vertex on average). Jong et al. [Jong et al., 2005] replaced the symbols E and S of the origin Edgebreaker encoder [Rossignac, 1999] by two new symbols: J and Q. The symbol Q encodes at the same time two triangles that share the same new vertex. The symbol J allows to skip the current new vertex to avoid boundary splitting overhead. Liu et al. [Liu et al., 2007] later improved this encoding method by using also a code-mode. They built a probability table to predict the current symbol in function of the previous one.

The angle-analyser scheme [Lee et al., 2002] encodes the connectivity of a triangular mesh with five symbols depicted on Figure 2.7. This algorithm also iteratively traverses the mesh triangles through one of their edges called the gate. If the vertex in front of the gate has not yet been visited, a 'C' symbol is generated. If the front vertex has already been visited a 'CW' or 'CCW' symbol is generated depending on its location (right or left) of the edge of the new triangle that has not already been visited. If their is no front vertex, the gate is on a boundary, a 'S' symbol is generated. Finally, if the front vertex has already

been conquered but the two other edges of the new triangle were not, a 'J' symbol following by an offset to locate the vertex is generated. The next gate to proceed is chosen in order to maintain the border between conquered and not conquered regions of the mesh the most convex as possible. This algorithm is reported to compress manifold mesh connectivity at 1.5 bits per vertex on average. Lee and Park [Lee and Park, 2005] later showed that this rate can be slightly reduced by using contexts for the arithmetic coder based on the angle between successive gates.

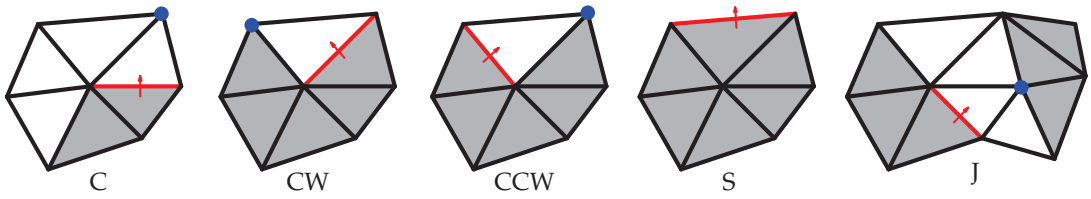


Figure 2.7: The five symbols of the Angle-Analyser encoder. The conquered triangles are in gray. The gate is in red and the front vertex is in blue.

2.2.5 Valence encoding

Triangle spanning trees can be built on a mesh by starting from an initial triangle and iteratively expending its boundaries. A symbol is generally generated for each newly processed triangle. The insertion of the new triangle can add or not a new vertex to the processed region. But as seen in Section 1.2.5, a manifold triangle mesh contains approximately twice less vertices than triangles. So, an algorithm that focuses on the insertion of new vertices and generates one symbol per vertex produces less symbols. Consequently, it may lead to a better connectivity compression performance. One way to describe vertex connectivities is to encode their valences. Schemes that exploit this idea are called valence-driven approaches.

The pioneering valence-driven approach is the algorithm of Touma and Gotzman [Touma and Gotsman, 1998]. The encoder first adds a dummy vertex to each boundary loop of the mesh to close the connectivity. The encoder then starts from an initial triangle and pushes its vertices in the *active list*. Then, it pops a *focus vertex* from the active list and pushes all its adjacent not processed vertices in the active list. The valence of all these vertices is encoded. The algorithm then continues with another focus vertex popped out the active list. In some cases, the algorithm needs to code active list split or merge operations with special symbols like the Cut-Border machine [Gumhold and Straßer, 1998]. This process is illustrated on Figure 2.8. In typical meshes, the valences of the vertices are concentrated around 6. Therefore, the generated list of vertex valences can be very efficiently compressed by an entropy coder. This algorithm can compress the connectivity of manifold triangle meshes at about 2.0 bits per vertex. It is still today seen as one of the most efficient compression method.

Alliez and Desbrun [Alliez and Desbrun, 2001b] proposed some modifications of the original Touma and Gotsman algorithm to further reduce the compression rates. Their method minimizes the number of split codes by selecting the next focus vertex in the active list in function of the number of not conquered edges in its neighborhood. They also improved the coding of the split operations with a prediction method based on the mesh geometry. Finally, only one dummy vertex is used when a mesh with several boundaries is encoded. Experimental results report connectivity compression rates up 18% lower than the results of the Touma and Gotsman encoder with irregular models. The authors demonstrated that their method upper bound compression rate for an arbitrary mesh matches Tutte's entropy [Tutte, 1962] (see Section 2.2.2). Thus, they claimed having demonstrated the optimality of valence-based approaches to encode the connectivity of manifold triangle meshes. This does not mean that valence-driven approaches always outperforms other coders. It is possible to construct pathologic examples where other coders perform better. But this means that it uses no more bits than needed to encode a connectivity [Isenburg, 2002].

The Freelence encoder [Kälberer et al., 2005] has a slightly different approach. Instead of directly encoding the valence of vertices, it codes the number of not conquered edges incident to the processed vertex. This method goes along with a geometry-driven mesh traversal to keep the active list as convex as possible

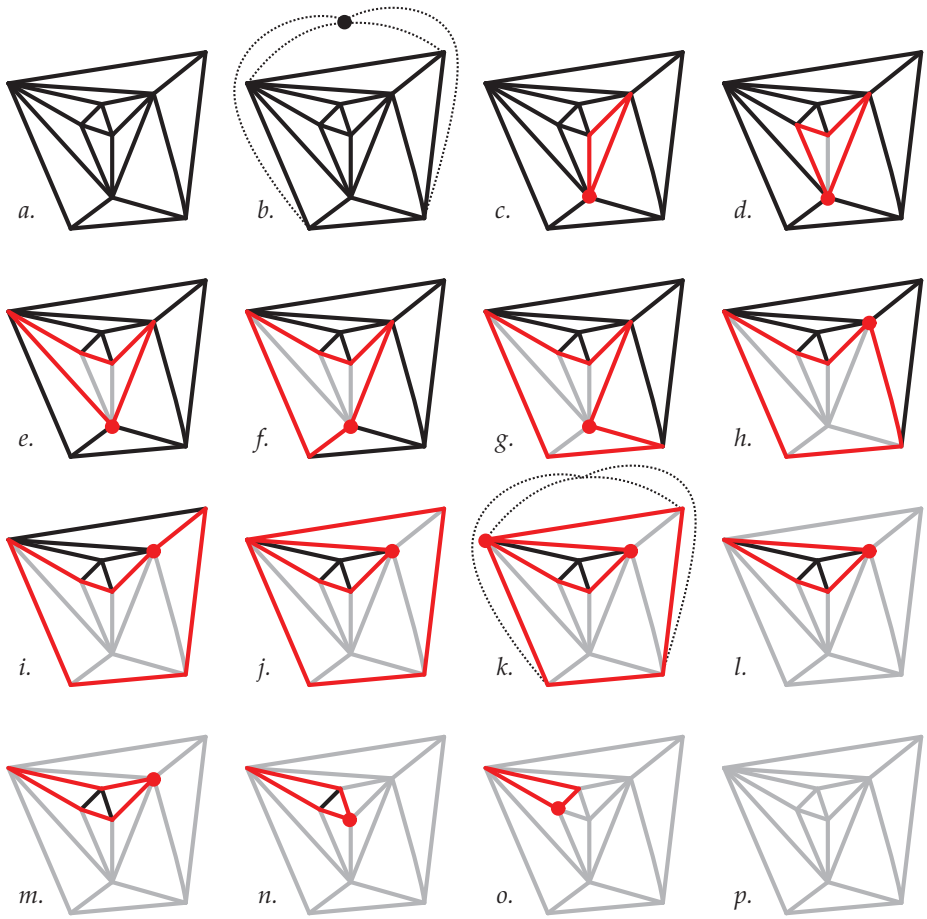


Figure 2.8: An encoding traversal of the Touma and Gotsman algorithm. The *active list* is represented by the red edges. The *focus vertex* is the red vertex. *a.* The initial mesh. *b.* A dummy vertex is added to close the mesh. *c.* The three vertices of the initial triangle are added to the active list and encoded. One focus vertex is selected. *d-g.* Neighbor vertices are added to the active list and encoded. *h.* The focus vertex is moved. *i.* A new vertex is encoded and added to the active list. *j.* The active list is split into two. *k.* The focus vertex is moved. The dummy vertex is encoded. *l.* The first active list is completed. The next focus vertex is taken from the second active list. *m.* A new vertex is encoded and added to the active list. *n.* The focus vertex is moved. *o.* A new vertex is encoded and added to the active list. The focus vertex is moved. *p.* The second active list is completed. The encoding is finished.

like the angle-analyzer encoder [Lee et al., 2002]. This algorithm yields an average improvement of 35% compared to the Alliez and Desbrun approach [Alliez and Desbrun, 2001b] for the compression of manifold triangle mesh connectivities.

Mamou et al. [Mamou et al., 2009] proposed an extended valence approach to encode non-manifold triangle meshes. This algorithm partitions a non-manifold triangle mesh into a set of triangle fans, a set of triangles that share a common central vertex. For each triangle fan, the encoded information is: its configuration code among 10 (boundary, faces visited or not), its degree and additional data for cases that would correspond to a 'merge' configuration of the Touma and Gotsman algorithm.

2.2.6 Compressing polygon meshes

Triangles are the most common types of faces inside meshes but not the only one. The modeling of meshes with other types of polygons has also its interest (see Section 1.2.1). As a consequence, the compression of polygon meshes has also been studied in the literature.

An indirect approach to deal with polygon meshes consists in first triangulating the mesh polygon faces before resorting to an existing method restricted to triangle meshes. While being simple at first glance, this approach is not efficient when caring about restoring the initial connectivity. It requires encoding an extra information to remove the edges added during triangulation, and conceptually adds more connectivity information by adding more edges than necessary. The theoretical result of Tutte's entropy [Tutte, 1963] states that the entropy of a planar graph is expressed in bits per edge (see Section 2.2.2). This result confirms the intuition that adding extra edges increases entropy and hence that the size of the compressed triangulated mesh is in general superior to that of the compressed original mesh. Consequently, specific schemes targeted to the compression of polygon mesh connectivity were proposed.

The algorithm of King et al. [King et al., 1999] generalizes the Edgebreaker algorithm [Rossignac, 1999, Rossignac and Szymczak, 1999] to allow the compression of quad mesh connectivity. Its principle is to split each quadrangle into two triangles that are on the same Edgebreaker traversal sequence. This leads to efficient compression of quad meshes connectivity (from 0.25 to 0.85 bits per vertex) because some combinations of the Edgebreaker algorithm become impossible.

The *Face Fixer* algorithm [Isenburg and Snoeyink, 2000a] compresses the connectivity of manifold polygon meshes with arbitrary face degrees using a face traversal of the mesh. The encoder generates one symbol per edge. Experimental results yields connectivity compression rates ranging from 1.7 to 2.9 bits per vertex. The connectivity encoder of Kronrod and Gotsmam [Kronrod and Gotsman, 2000] codes the degree of each face of the mesh and its relation to the active border. The authors report their method as being slightly less efficient than [Isenburg and Snoeyink, 2000a] but easier to describe and implement. Liu and Wu [Liu and Wu, 2006] later showed that much better rates can be obtained by entropy coding the generated symbols.

Isenburg [Isenburg, 2002] and Khodakovsky et al. [Khodakovsky et al., 2002] independently proposed valence-based approaches to encode the connectivity of manifold polygon meshes. Inspired by the work of Touma and Gotsman [Touma and Gotsman, 1998], their approaches encode both the valence of vertices and faces with a face traversal of the mesh. Khodakovsky et al. [Khodakovsky et al., 2002] demonstrated the optimality of such schemes by proving that the entropy of the two valence lists matches Tutte's entropy for planar graphs [Tutte, 1963]. For both methods, experimental results range approximately from 0.8 to 2.6 bits per vertex, thus improving over the Face Fixer scheme [Isenburg and Snoeyink, 2000a].

Krivograd et al. [Krivograd et al., 2006] proposed an encoder adapted to the compression of quad meshes connectivities. Based on a quad traversal through halfedges, it generates four main different symbols. Experimental results shows that it is slightly more efficiency (10 percent) than the encoder from [Isenburg, 2002] to compress the connectivity of quad meshes with more than 20000 vertices.

2.2.7 Compressing volume meshes

The compression of volume meshes has also been studied in the literature because they are used a lot for the simulation of physic phenomenon.

Szymczak and Rossignac [Szymczak and Rossignac, 1999] proposed a spanning tree approach to encode tetrahedral meshes. The compressed format is composed of a tetrahedral spanning tree string and a folding spanning tree. This second tree codes the folding operations on the tetrahedral spanning tree edges needed to restore the initial mesh. This scheme encodes a tetrahedral mesh with about 7 bits per tetrahedral. Gumhold et al. [Gumhold et al., 1999] proposed a border extension approach based on the original Cut-Border machine [Gumhold and Straßer, 1998]. But here, instead of being a curve composed of edges, the border is a surface composed of triangles. Ten different symbols encode the connectivity. The connectivity of the tested tetrahedral meshes is compressed at less than 2.4 bits per tetrahedron.

Isenburg and Alliez [Isenburg and Alliez, 2002a] investigated the compression of hexahedral volume meshes with a valence-driven approach. It encodes the mesh edge degrees, their number of incident faces, during an iterative traversal of all the mesh hexahedra. The obtained average compression rate is 1.5 bits per hexahedron for the tested models. The hexahedral mesh connectivity compressor of Krivograd et al. [Krivograd et al., 2008] first compresses the mesh boundary with the quad mesh connectivity encoder from [Krivograd et al., 2006]. Then, the interior of the mesh is compressed with six different symbols and vertex degrees. Experimental results show that this approach compresses better (up to 50%) big meshes with a low genus compared to the scheme of Isenburg and Alliez [Isenburg and Alliez, 2002a]. However, smaller models are compressed worse (up to 45%).

Lindstrom and Isenburg [Lindstrom and Isenburg, 2008] proposed an original approach to compress hexahedral mesh connectivity. Their method encodes the vertex indices from the input file indexed data structure. In this file, the hexahedra are described by a table with 8 columns of vertex indices. The principle is to compress independently each column of the table. Indeed, if the vertices are coherently ordered, a context of few indices can accurately predict the next value. The prediction residual are then coded with byte-aligned variable length coding. The obtained string is later compressed by a standard *gzip* data compression algorithm. Even if this algorithm is not as efficient as state-of-the-art approaches, it has numerous advantages. Among them, it is fast and memory efficient. It processes non-manifold meshes without any adaptation. It is easy to implement: it does not need any particular mesh data structure and it heavily relies on the freely available data compression algorithm *gzip*.

Prat et al. [Prat et al., 2005] described a generic algorithm to compress the connectivity of any kind of manifold meshes (surface or volume, orientable or not, with arbitrary face or cell degrees). They based their scheme on a generalized map data structure containing a single type of primitive elements called darts. Connectivity relations between these elements called involutions are also stored. Even if this method is not competitive in term of compression rates with specialized approaches, its main advantage is its genericity.

2.3 Geometry compression

As seen in the Section 2.2.2, the first mesh compression approaches were inspired by work on compact encoding of planar graphs and focused on the encoding of the mesh connectivity. Yet, the geometry also takes up a significant part of the mesh data, often the biggest. Consequently, efficient geometry compression schemes are also very important.

Usually the compression of the geometry of a mesh begins with the *quantization* of all the coordinates of its vertices. Then, during each compression iteration, the position of an encoded vertex is predicted from its already encoded neighbors. If the *prediction* is accurate, the prediction error or residual, the difference between the predicted position and the real position of the inserted vertex is small. So, it can be later efficiently entropy-coded.

2.3.1 Quantization

In input files, or in the memory of computers, the vertex coordinates in the space are often represented by 3 IEEE 32-bit floating-point numbers. As reported by Deering in [Deering, 1995], this representation allows positions "to span the known universe: from a scale of 15 billion light years, down to the radius of sub-atomic particles". The precision provided by such a representation is not needed for most applications. So quantization can significantly reduce the quantity of data to encode without any identifiable quality loss.

Scalar quantization consists in transforming the floating-point number positions into integer positions. The mesh bounding box is partitioned into a 3D grid. The number of cells per axis depends on the maximum integer that can be coded with the number of quantization bits. The size of each cell can be either uniform or non-uniform. Each vertex of the mesh is moved to the center of the cell it belongs to. The integer position is then composed of the three index coordinates of the cell. The Figure 2.9 illustrates this process.

Most of the geometry encoders that go along with the previously described well-known connectivity compression schemes [Deering, 1995, Taubin and Rossignac, 1998, Rossignac, 1999, Touma and Gotsman, 1998] use a uniform scalar quantization. The number of quantization bits usually ranges from 8 to 16. Lossy compression is therefore applied to the mesh geometry contrary to the connectivity.

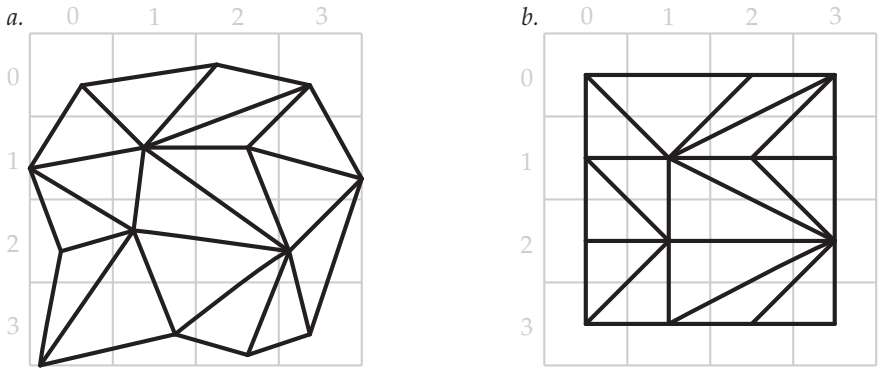


Figure 2.9: Uniform scalar quantization example of a 2D mesh. *a.* The mesh before the quantization. *b.* The mesh after a 2 bits quantization. Vertices are moved to the center of the cells. Some triangle disappear because they become flat.

Bajaj et al. [Bajaj et al., 1999] and then Lee et al. [Lee et al., 2002] proposed to encode the vertex positions with three angles. For the angle-analyzer encoder [Lee et al., 2002] two internal angles and one dihedral angle are computed. Instead of being performed on global vertex coordinates, the quantization is performed on these local angles. By applying different quantizations to the different angles, this method can achieve a better rate-distortion performance. The quantization ranges can be set before the compression. An alternative method is to determine bounding ranges for all the vertex positions encoded in the local frames with a first mesh traversal. These positions are locally quantized and encoded in a second step. Lee and Park [Lee and Park, 2005] proposed to locate the vertices within 4 different range sizes as they noticed that very few vertices are located in the biggest range. To encode the position of the vertex within a range, the ranges are more or less subdivided depending on their size. The position of the vertex is therefore encoded by the type of the range and the subcell number.

Vector quantization is an alternative technique that divides the set of points to quantize into arbitrary shaped groups. Quantization cells are no longer cuboids. Their shape can better adapt to the data. Each group has a representative point. All of these points constitutes the *codebook* that must be saved with the compressed data. For mesh geometry compression, it was not applied directly to vertex positions as with scalar quantization but it was applied to the prediction residual vectors. Vector quantization has experimentally demonstrated its ability to achieve better rate-distortion performance than scalar quantization technique [Lee and Ko, 2000, Chou and Meng, 2002, Bayazit et al., 2007, Chen et al., 2010, Meng et al., 2010]. However, the determination of the quantization cells can lead to intensive computations. All of these schemes, except [Lee and Ko, 2000], use vector quantization with parallelogram prediction (see Section 2.3.2). Some approaches [Li et al., 2007, Lu and Li, 2008] dynamically build codebooks for different qualities by just varying a parameter.

2.3.2 Prediction

The first mesh compression approaches [Deering, 1995, Chow, 1997] only used *delta prediction*. This method assumes that two vertices close in the mesh connectivity graph are close in the 3D space. So, the position of the next vertex to encode is predicted to be the position of the previous vertex. The delta, the vector between the two positions, is then encoded. Bajaj et al. [Bajaj et al., 1999] uses a *second order predictor* that encodes the differences between consecutive delta predictions.

The topological surgery algorithm [Taubin and Rossignac, 1998] uses *linear prediction*. The position of the next vertex to encode is predicted to be a linear combination of the K previous vertices in the vertex spanning tree. The K parameters of the linear function minimize the mean square prediction error over the mesh. Their values are stored in the compressed file to be available to the decoder.

Besides valence-driven connectivity encoding, Touma and Gotsman also introduced *parallelogram prediction* [Touma and Gotsman, 1998]. Like valence-driven connectivity encoding, parallelogram prediction is a founding idea that has inspired many later schemes. The compression algorithm introduces a new vertex with a triangle from one edge. The new vertex predicted position forms a parallelogram with the two edge vertices and the third vertex of the opposite triangle (Figure 2.10 *a*). However, this prediction assumes incorrectly that the whole mesh is planar. A better prediction could be obtained by taking into account the crease angle between the two triangles. So the authors proposed to estimate this angle with the crease angles between the reference triangle and its adjacent triangles already encoded. Experimental results show that triangle mesh geometry can be compressed at about 8.5 bits per vertex with a 8 bit quantization.

The *dual parallelogram prediction* [Sim et al., 2003] uses the average position given by two parallelogram predictions whenever it is possible (Figure 2.10 b). Dual parallelogram can be used in about 75 percent of the cases. It results to slight improvements over *parallelogram prediction*.

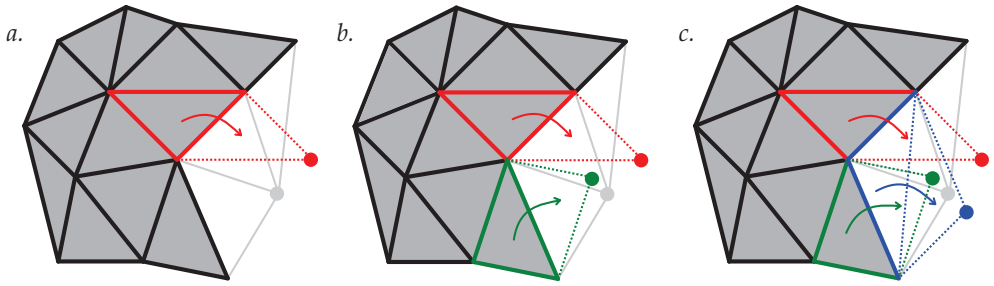


Figure 2.10: Parallelogram prediction. The already encoded part is in gray. *a.* Simple parallelogram prediction. *b.* Dual parallelogram prediction. The predicted position is the average of the two parallelogram prediction. *c.* Freelence prediction. The predicted position is the average of the three parallelogram prediction.

Isenburg and Alliez [Isenburg and Alliez, 2002b] showed that parallelogram prediction can also be used for the geometry compression of polygon meshes with arbitrary face degrees. As polygons are in most cases fairly planar and convex, they noticed that predictions within polygons (Figure 2.11 *b*) are more accurate than predictions across polygons (Figure 2.11 *a*). By performing as many as possible inner predictions, this method can reduce the compression rate by 10 to 40 percent. Isenburg et al. [Isenburg et al., 2005a] later presented a generalization of the parallelogram prediction for arbitrary polygons. Missing vertex positions of a polygon are predicted with weights computed for polygons of different degrees with different known vertices. These weights come from a Fourier decomposition of polygons where the highest frequencies are set to 0 to ensure that the polygon are nicely shaped.

The Freelence coder [Kälberer et al., 2005] uses the combination of three parallelogram predictions to encode the geometry of triangular meshes: two standard ones, as with dual parallelogram prediction, and a new one applied across a virtual edge joining the two outer vertices (Figure 2.10 *c*).

Cohen-or et al. introduced *multiway prediction* [Cohen-Or et al., 2002]. The principle is to predict the position of new vertices with as many as possible parallelogram predictions based on already encoded vertices (see Figure 2.12). The model to compress is first segmented. The position of the boundary vertices are explicitly encoded. Then the algorithm builds a rough approximation of the mesh. It starts from boundary vertices and iteratively computes the multiway prediction of inner vertices. A relaxation approach tries to keep the edge lengths uniform. Finally a smoothing operation is applied to vertex positions. The difference between the current positions of vertices and their real position is compressed. In a similar way, the geometry predictor of Gumhold and Amjoun [Gumhold and Amjoun, 2003] performs as many as possible

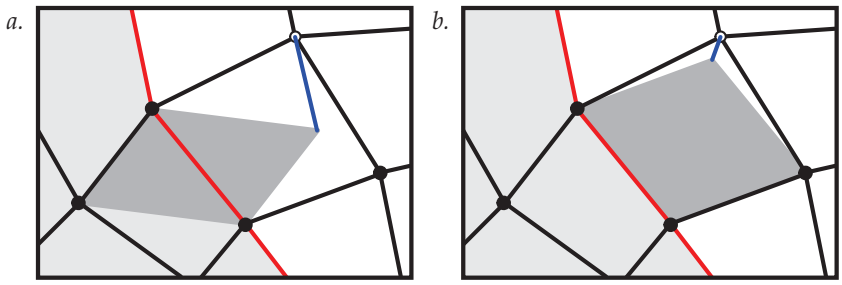


Figure 2.11: Parallelogram prediction for quadrangles. *a.* Across quadrangles. *b.* Within quadrangle. The encoded region of the mesh is in light gray. The known vertices are filled in back. The prediction error is represented by the blue segment.

parallelogram predictions to determine the tangential components of the prediction. However, to estimate the normal component, encoded by a dihedral angle, the encoder fits a high order surface to the already encoded part of the geometry. The predicted position is therefore at the intersection of the higher order surface with the circle defined by the tangential components. Ahn et al. [Ahn et al., 2006] also use as many as possible parallelogram predictions. They also added their own dihedral angle prediction scheme that averages all the neighboring dihedral angles.

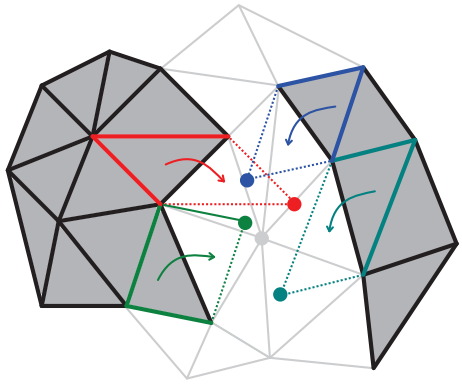


Figure 2.12: Multiway prediction. The encoded parts of the mesh are in gray. The predicted position is the average of the four parallelogram prediction.

Recently, Váša and Brunett [Váša and Brunnett, 2013] proposed weighted parallelogram prediction methods, which weights are calculated in function of the vertex valences. The three proposed weight calculation methods have an increasing computational cost. Experimental results report that residuals have a computed Shannon entropy up to 20 lower than with standard parallelogram prediction.

Courbet and Hudelot [Courbet and Hudelot, 2011] used a Taylor expansion to determine prediction weights for various prediction stencils. This method allows to theoretically determine the best weights for the parallelogram prediction [Touma and Gotsman, 1998]. It proves that the Freelence weights [Kälberer et al., 2005] work better than the dual parallelogram prediction weights [Sim et al., 2003] and determines better weights for the polygon prediction [Isenburg et al., 2005a].

Regarding volume meshes, Gumhold et al. [Gumhold et al., 1999] resorted to delta coding to encode the geometry of tetrahedral meshes. Isenburg and Alliez [Isenburg and Alliez, 2002a] compressed the geometry of hexahedral meshes with intra and inter hexahedron parallelogram predictions.

2.3.3 Geometry-driven compression

For a compressed mesh, the size of the geometry information is often superior to the size of the connectivity. Nevertheless for most algorithms, the encoding is driven by the mesh connectivity. So, the idea of focusing on geometry encoding before connectivity has emerged in the mesh compression community.

Kronrod and Gotsman [Kronrod and Gotsman, 2002] designed a compression scheme where the mesh encoding is driven by the geometry. This approach builds a mesh cover tree that contains all the mesh vertices. This tree, which defines a particular traversal of the mesh vertices, is generated to minimize the parallelogram prediction errors needed for each vertex position. The geometry compression becomes up to 50 percent more efficient at the cost of a slightly higher connectivity compression rate. This approach particularly benefits to CAD models, which often have a non-smooth geometry.

The tetrahedral mesh compression scheme of Chen et al. [Chen et al., 2005] is also driven by the geometry. It tries to build an optimal traversal tree that minimizes the prediction errors of the new vertex positions. The used predictor is the generalization of the parallelogram prediction for tetrahedral meshes.

Shikhare et al. [Shikhare et al., 2001] pushed the geometry-driven compression idea one step further. Their scheme tries to find repeated geometric patterns inside 3D models. The recognized patterns can be either components, regions within components or region across components. This approach particularly benefits to large CAD or digital heritage models. Cai et al. [Cai et al., 2009] later proposed a similar scheme that achieves slightly better performance. They included scaling transformations for the repeated patterns.

The connectivity encoding of the previous schemes is guided by the mesh geometry. But the mesh geometry and connectivity are compressed at the same time and their data is interleaved in the compressed stream. Lewiner et al. [Lewiner et al., 2005, Lewiner et al., 2006] with the GEncode algorithm proposed an alternative geometry-driven encoding technique. The mesh geometry is first compressed completely independently from the connectivity by encoding a kd-tree decomposition of the quantized space. Then, a surface reconstruction algorithm iteratively attaches new triangles to the border of the mesh by selecting a new vertex among candidates around. The reference to the right vertex among the candidates is encoded. This algorithm can compress any kind of triangle meshes. This approach is very competitive for the compression of tetrahedral meshes [Lewiner et al., 2006] compared to the Grow & Fold [Szymczak and Rossignac, 1999] and streaming [Isenburg et al., 2006] approaches.

In a similar idea, Chaine et al. [Chaine et al., 2007, Chaine et al., 2009] proposed a mesh connectivity compression scheme based on surface reconstruction. For the decompression, this technique assumes that the geometry has already been decoded. For both the encoding and decoding, a Delaunay triangulation is generated from the point sets. Then a convection algorithm, sometimes constrained by encoded orders, restores the initial connectivity of the input mesh. The connectivity of meshes generated with similar techniques is therefore encoded at a very low cost.

2.3.4 Compressing floating-point positions

As described above, most of the mesh compression algorithms quantize the coordinates of the vertices. But some applications may require the exact restoration of the floating-point coordinates. Isenburg et al. [Isenburg et al., 2005b] investigated the lossless compression of floating-point geometry. The floating-point coordinates are broken into sign, exponent, and mantissa components. The prediction errors of these components are compressed independently with different arithmetic contexts. This method is able to compress the geometry of a mesh at about 35 bits per vertex.

2.4 Handling large meshes

Some models are too large to fit into the main memory of computers. As swapping (using a part of the hard disk as an extension of the main memory) is generally not an efficient solution, *out-of-core* schemes were designed to compress such models. Out-of-core algorithms allow to process a mesh without fully loading it

in the main memory of the computer. The different parts of the mesh are dynamically loaded and unloaded depending on the currently processed region of the mesh.

In [Ho et al., 2001], the authors proposed to partition the input model. Each part is then compressed independently in-core. The Edgebreaker compression algorithm [Rossignac, 1999] compresses the connectivity. Additional symbols are integrated to stitch the parts together during the decompression. For the geometry compression the parallelogram prediction [Touma and Gotsman, 1998] is used. In the same idea, Ueng [Ueng, 2003] proposed to compress large tetrahedral meshes connectivity by dividing them into blocks called octans with an octree data structure based on the geometry. Each octan is compressed in-core by encoding tetrahedral strips.

Isenburg and Gumhold [Isenburg and Gumhold, 2003] proposed an out-of-core mesh data structure for the compression of large polygon meshes. This data structure is built on a segmentation of the input mesh. However, contrary to the previous out-of-core approaches, the clusters are compressed together with a *streaming* approach. During the compression, the data structure dynamically loads the mesh clusters that are on the active list of the Touma and Gotsman encoder [Touma and Gotsman, 1998] and unload them when they are no longer needed. The decompression is performed with a small memory footprint composed only of the active list. The authors claim that the obtained compression rates are about 25 percent lower and the decompression speeds about 100 times faster than Ho et al. scheme [Ho et al., 2001].

Following their idea of processing meshes with a streaming approach, Isenburg and Lindstrom [Isenburg and Lindstrom, 2005] proposed a general I/O efficient streaming format for meshes. This format interleaves indexed triangles and vertices with extra information describing when mesh elements are introduced and finalized. Therefore, the application keeps only in memory the small active part of the mesh currently being processed. The authors propose several strategies to reorder the meshes indices in a layout compatible with streaming.

This framework has been the basis of several streaming compression techniques such as [Isenburg et al., 2005c] for triangle meshes, [Isenburg et al., 2006] for tetrahedral meshes and [Courbet and Isenburg, 2010] for hexahedral meshes. Compared to their non-streaming counterparts, these techniques have in general equivalent geometry compression rates but higher connectivity compression rates. Yet, the I/O efficiency of the streaming approach allows them to compress large meshes quickly with a very low memory footprint.

2.5 Compression and remeshing

Most of the compression schemes do not alter the connectivity of the mesh. But some applications do not require the restoration of the initial connectivity after the decompression. In this case, the compression rates can be further improved by exploiting this new degree of freedom.

Szymczak et al. [Szymczak et al., 2002] proposed a remeshing algorithm that produces piecewise regular meshes. The input mesh is first segmented into approximately flat regions called reliefs. Each relief belongs to one of six different directions. They are then resampled over a regular hexagonal grid defined by three families of parallel lines. All the original vertices, except the boundary ones, are collapsed. A stitching algorithm reforms a closed mesh. This scheme is associated with an Edgebreaker compression algorithm optimized for regular meshes for the connectivity. The tangential and normal components of the geometry are compressed independently. The Swingwrapper scheme [Attene et al., 2003] produces also a semi-regular meshes. It aims at generating isosceles triangles to encode only one quantized dihedral angle when their inserted vertex is encoded by an Edgebreaker encoder.

2.6 Conclusion

Pioneering work on single-rate mesh compression focused on connectivity encoding. Schemes based on triangle strips allowed to encode efficiently the mesh connectivity. The aim was to speed up the rendering by transferring quickly data between the CPU and GPU memory. Work on compact graph coding later

inspired approaches based on the encoding of vertex spanning trees. Better compression rates were obtained by compressing face spanning trees rather than vertex spanning trees. The best compression performances were obtained by valence-driven schemes, which optimality was proven. Seeing that the size of the geometry is often much more significant than the size of the connectivity, the community started getting more interested in developing efficient geometry predictors and quantization methods. We summarized in the Table 2.1 what we judged as the main single-rate mesh compression approaches.

Algorithm	Connect. comp. rates (bpv)	Compress polygon meshes	Embed strips	Remarks
Deering [Deering, 1995]	11	no	yes	
Topological surgery [Taubin and Rossignac, 1998]	2.5 to 7	no	no	
Cut border machine [Gumhold and Straßer, 1998]	4.4 on avg.	no	no	
Edgebreaker [Rossignac, 1999] [Gumhold, 2000]	3.55 max.	no	yes	
Touma & Gotsmam [Touma and Gotsman, 1998]	2.3 on avg.	no	no	Introduced parallelogram prediction for geom. coding
Valence polygonal [Khodakovsky et al., 2002] [Isenburg, 2002]	1.8 on avg. for poly. meshes	yes	no	Proven near-optimality of the connect. coding
Valence coder [Alliez and Desbrun, 2001a]	2.1 on avg.	no	no	Proven optimality of the connect. coding

Table 2.1: Summary of the main single-rate compression algorithms approximately ranked by their connectivity compression performance.

Looking at the entries of the Table 2.1 and more generally the references of this chapter, the period 1998-2003 seems to have been the golden age of mesh compression. It is during this period than most of the main approaches were proposed. Among them, the Touma-Gotsman algorithm [Touma and Gotsman, 1998] has left its strong mark as it introduced two major concepts: valence-driven encoding for connectivity encoding and parallelogram prediction for geometry encoding. It is still seen today as one of the most efficient method.

We see the future of single-rate mesh compression directed by the following points.

Geometry compression. While no major progress seems to have been done lately on connectivity encoding because the optimality of valence encoding was proven, we think that there is still some perspectives for the geometry compression. Efficient mesh geometry compression is an important challenge as it often constitutes the bigger part of the compressed data.

Genericity. Some of the already proposed approaches can already compress meshes with specific connectivities but we think that an algorithm that can process meshes of arbitrary topology (arbitrary genus, surface or volume, manifold or not, with borders or not...) at a low cost may be of great interest for the industry. The proposal of such algorithm could help the process of standardization already begun in the MPEG or X3D communities.

Parallelism. As the architecture of modern computers is evolving from one core too many cores, the design of parallel mesh compression algorithms may be the future trend to cope with the non-stopping increase of data size. This work has actually already begun. A straightforward way to parallelize a mesh compression algorithm can be to segment the input mesh and assign one thread to one cluster. The compression and decompression are then run independently on each chart. Zhao et al. [Zhao et al., 2012] followed this approach in their encoder based on the Edgebreaker scheme [Rossignac, 1999]. Recently

Meyer et al. [Meyer et al., 2012] proposed a more innovative method. They built a new mesh compression scheme based on generalized triangle strips encoded in parallel with *scan-operations*. Both approaches did not, however, address the problem of parallel geometry compression.

The next chapter is a state-of-the-art about progressive mesh compression. Contrary to single-rate algorithms, progressive schemes build a multiresolution representation of the input model useful for the transmission and the adaptation of the data.

Chapter 3

State of the art on progressive mesh compression

3.1 Introduction

Compression is a efficient tool for mesh storage and transmission because its aim is to reduce the data size. To access to a compressed model, single-rate algorithms presented in the Chapter 2 require the full data to be downloaded (in a transmission context) and decompressed. If the input model is large, these operations can take a significant time. More interactivity can be provided by allowing the user to first access to a coarse version of the compressed model, called the base mesh. Then, as more data is received and decompressed, the base mesh is progressively refined until the input model is restored. The generated multiresolution structure is also useful to adapt the mesh resolution to the device capabilities. For instance, a high performance graphic station may not render the same level of detail as a smartphone.

The main focus of progressive mesh compression schemes has always been to achieve the best rate-distortion performance. The generated levels of detail must be as close as possible to the input model while being coded with the lowest possible number of bits. Consequently, the performance of progressive mesh compression algorithms is not only measured with final compression rates but also with rate-distortion curves. In the literature, the *Hausdorff* distance or the *Root Mean Square* (RMS) distance have often been chosen as distortion measures. Moreover, they were implemented in the well-known METRO tool [Cignoni et al., 1998].

Most of the progressive mesh compression algorithms generate levels of detail thanks to decimation operators. These operators iteratively simplify the input model by removing vertices, faces and edges and locally remeshing the surface. Progressive mesh compression is therefore highly related to mesh simplification. Metrics adapted to the decimation operators have also been proposed to drive the mesh simplification or to forbid the removal of important vertices during the progressive compression.

This chapter is a review on progressive mesh compression algorithms. We classified the progressive mesh compression algorithms into two categories.

Connectivity-preserving schemes restore during the decompression the connectivity of the input model. We listed them in the Section 3.2.

Connectivity-oblivious schemes resort to remeshing to encode an input model with a higher compression performance. We listed them in the Section 3.3.

The chapter ends with a conclusion that proposes perspectives for future work in this research field.

3.2 Connectivity-preserving schemes

The compression schemes described in this section restore the compressed model with no connectivity modification when the full data is decompressed.

3.2.1 Edge collapse and vertex split

Hoppe first introduced the concept of *progressive meshes* (PM) [Hoppe, 1996]. The idea is to incrementally decimate a mesh using the edge collapse operator (see Figure 3.1) driven by an optimization procedure that minimizes an energy function. The energy function includes a term that aims at minimizing the squared distance between the simplified and input meshes. A second term aims at regularizing the mesh faces. The third and fourth terms preserve the scalar and discontinuous attributes of the original mesh. The compressed representation consists of the base mesh followed by all parameters required for the incremental reverse operations, called *vertex splits*. Hoppe encoded the connectivity of a vertex split by storing the index of the vertex v_s and approximately five bits to select the vertices v_l and v_r among the vertices adjacent to v_s (see Figure 3.1). For the geometry encoding, the vertex positions are globally quantized and encoded through delta prediction. The main advantage of this scheme is its high multi-resolution granularity, together with the possibility to perform selective refinement during decoding. Such granularity is achieved at the cost of low compression rates: in the order of 37 bits per vertex (bpv) with 10 bits quantization.

Popović and Hoppe in [Popović and Hoppe, 1997] generalized the PM representation to arbitrary simplicial complexes. They introduced the *generalized vertex split* and its inverse, the *vertex unification* operations for simplices of any dimension (see Section 1.2.3). With this representation, an arbitrary triangulation requires about 50 bpv with a 10 bit quantization.

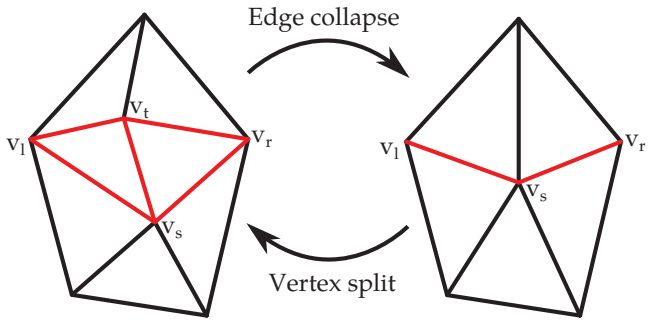


Figure 3.1: Edge collapse and vertex split operations. The edge collapse merges the vertex v_t with the vertex v_s. The vertex split inserts the vertex v_t and the two triangles with the red edges.

In order to come closer to compression rates of single-rate methods, some methods were proposed to encode the vertex split operations in batches. Taubin et al. [Taubin et al., 1998] build a progressive mesh compression scheme inspired by the single-rate *topological surgery* algorithm [Taubin and Rossignac, 1998] (see Section 2.2.3). Their *progressive forest split* representation encodes a manifold triangular mesh with a base mesh and a sequence of *forest split* operations. The forest split operation, during the decompression, consists in cutting the mesh through several sets of connected edges (the trees), filling the generated holes with triangles and relocating the vertices. The connectivity is encoded by marking the cut edges and the way the holes are retriangulated. After a tree split has been performed, the vertex position are smoothed and then restored to their original position thanks to an encoded geometry correction.

Pajarola and Rossignac [Pajarola and Rossignac, 2000] proposed to perform as many as possible edge collapse operations to generate of a new level of details. Vertex splits are then encoded by building a vertex spanning tree on the mesh and marking each split vertex. The cut edges of the vertex splits are encoded by their indices in the sorted list of the edges incident to the split vertex. This list starts from the edge that links the split vertex to its parent in the spanning tree and contains the other adjacent

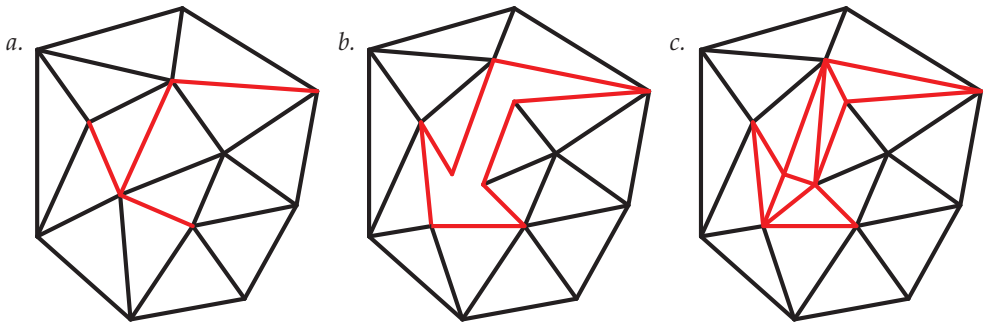


Figure 3.2: Progressive forest split. *a.* The input mesh. *b.* Cut along the edge of the tree in red. *c.* A finer level of detail is obtained after the retriangulation of the holes.

edges added in a clockwise order as shown on Figure 3.3 *a.* The distortion introduced by an edge collapse is measured with a variation of the *Quadric Error Metric* [Garland and Heckbert, 1997]. To improve the geometry encoding, they designed a new prediction scheme inspired by the Butterfly subdivision scheme [Zorin et al., 1996]. In [Pajarola and Rossignac, 2000], the vertex positions are uniformly quantized but Li et al. [Li et al., 2006] later demonstrated that the use of vector quantization (see Section 2.3.1) improves the rate-distortion performance of the algorithm.

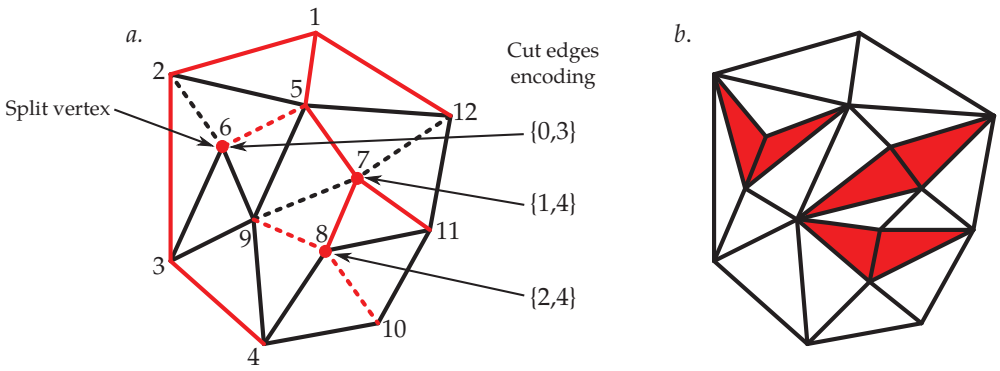


Figure 3.3: Encoding of compressed progressive meshes. *a.* The vertex spanning tree is represented by the red edges. The doted edges represent the cut edges. *b.* Level of detail obtained after the vertices were split. The red faces were added by the split operations.

Karni et al. [Karni et al., 2002] adapted the Pajarola and Rossignac scheme [Pajarola and Rossignac, 2000] to design a progressive compression scheme which enables the fast rendering of all the LODs with vertex buffers. Vertex buffers have been introduced by the graphic hardware manufacturers to accelerate rendering with the reuse of vertex data through generalized triangle strips (see Section 2.2.1). The first step of the proposed algorithm is to create an efficient vertex rendering sequence composed of series of incident vertices. The mesh is then decimated by collapsing edges along this sequence.

Better compression ratios are achieved with these approaches (about 30 bpv [Taubin et al., 1998] and 22 bpv [Pajarola and Rossignac, 2000] for 10 bits quantization). Nevertheless the multiresolution granularity is impacted compared to the PM representation.

More recently Cellier et al. [Cellier et al., 2012] described a parallel simplification algorithm based on edge collapse. It consists in searching and performing in parallel, for each point of the mesh, the collapse that has the lowest cost in its 2-ring neighborhood. The refinement approach is suitable for mesh compression and streaming. Vertex splits operations are encoded in batches in order to double the number of vertices per level of detail. They are described by the valence at which the two inserted vertices will become candidates

for a split, the indices of the split edges and one bit specifying how the new vertices are connected to the generated triangles. The authors report connectivity compression rates of about 15 bits per vertex.

3.2.2 Vertex removals

Other progressive compression schemes use vertex removals instead of edge collapses. Li and Kuo [Li and Kuo, 1998b] pioneered a method based on vertex removal followed by a local patch retriangulation. The connectivity is encoded with a local index which specifies the patch pattern and a global index which locates this pattern in the whole mesh. The geometry data is encoded with a barycentric error prediction. The authors also pioneered the idea of adapting the vertex quantization along the transmission of the LOD.

Cohen-Or et al. [Cohen-Or et al., 1999] used the same decimation mechanism. However, they grouped the vertex removal into batches to generate discrete levels of detail. For the generation of a new level, all the patches of the removed vertices must be independent. The patches are then identified by assigning the same color to triangles belonging to the same patch. Two adjacent patches must of course have a different assigned color and the triangles that do not belong to a patch have a null color assigned. The authors remarked that, in most cases, 3 colors are enough that can be coded with 2 bits per triangle (see Figure 3.4). A deterministic *Z-triangulation* of the patches is also proposed to encode the patch pattern with only one bit per triangle. The triangles inside the Z have the 1 bit assigned while the others have the 0 bit. Compression rates competitive with single rate techniques can be achieved with this method (about 23bpv with 12 bits quantization).

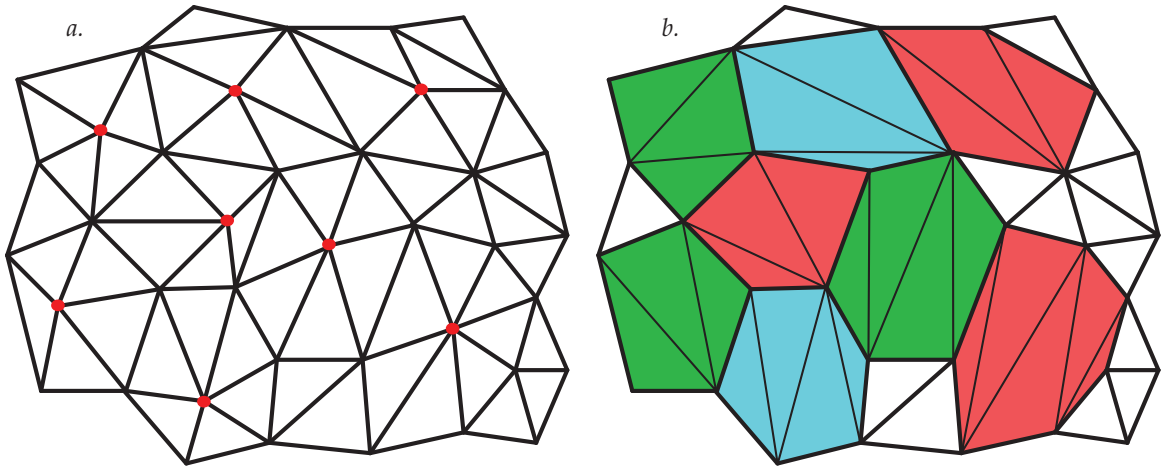


Figure 3.4: Progressive compression with colored patches. *a.* The red vertices are selected to be removed. *b.* The red vertices were removed. Patches have been remeshed and assigned a color among three.

Alliez and Desbrun [Alliez and Desbrun, 2001a] proposed what could be seen as a progressive version of the Touma-Gotsman single-rate encoder [Touma and Gotsman, 1998]. At each iteration, the mesh is decimated by two deterministic mesh traversals. The *decimating conquest* removes vertices with a valence inferior or equal to 6. Considering the border vertices, only vertices with a valence of 3 or 4 are removed. A deterministic retriangulation strategy fills the created holes. The *cleansing conquest* removes only valence 3 vertices. The mesh connectivity is encoded through the valence of the removed vertices plus one null patch code to encode triangles not belonging to patches. Border vertices are encoded with specific symbols that are different from valence 3 and 4 inner vertices. As the patches are retriangulated in a deterministic way set by the mesh traversal, the decompression, which performs the same traversal, can restore the initial connectivity with only these integer codes. As valence approaches have demonstrated their high efficiency to compress single-rate meshes (see Section 2.2.5), this technique improves significantly progressive connectivity encoding compared to previous approaches. The geometry is encoded through the patch barycentric error prediction in a local Frenet frame. To prevent the decimation of important vertices, the author proposed

a metric based on the volume of the patch to decimate. The obtained compression rates are about 21 bpv with 12 bit quantization.

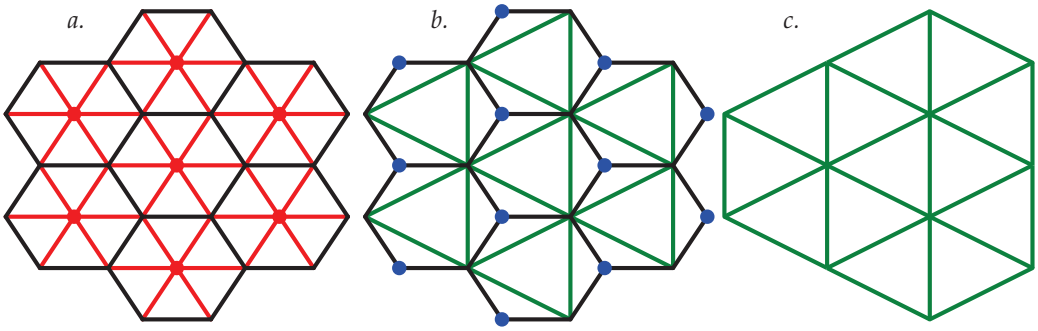


Figure 3.5: Decimation of a regular mesh by the Alliez-Desbrun progressive encoder. a. The red vertices and the red edges of the input mesh will be removed by the decimating conquest. b. The deterministic triangulation fills the holes with the green edges. The blue vertices will be removed by the cleansing conquest. c. The obtained decimated mesh.

Since valence-driven encoding of the mesh connectivity has demonstrated its efficiency for single-rate and progressive mesh compression, the original Alliez-Desbrun has inspired a lot of numerous derived schemes that have improved its rate-distortion performance.

For instance, Cheng et al. [Cheng et al., 2006, Gang et al., 2010] forbid during the decimation conquests the removal of some vertices called *anchors*, detected by their principle curvatures. Experiments show that keeping these vertices in all levels of detail allows to improve the rate-distortion performance.

Lee et al. [Lee et al., 2012] demonstrated that the rate-distortion trade-off of the Alliez-Desbrun coder is improved by using an adaptive quantization method. The idea consists in interleaving decimation operations with global quantization operations. The decimation operations are the classic Alliez-Desbrun decimation and cleansing conquests while the quantization operations are encoded with Peng and Kuo approach [Peng and Kuo, 2005] (see section 3.2.3). The decision of performing a decimation or a quantization operation can be made with a costly optimal approach that computes both meshes and chooses the one with the smallest distortion. The authors also show that this optimal behavior can be approximated with a parameterized formula. They obtain both better R-D performances and compression rates (about 1bpv improvement with a 12 bit quantization).

Ahn et al. [Ahn et al., 2011] proposed another improvement of the Alliez-Desbrun coder through an optimized mesh traversal to maximize the number of removed vertices per decimation step. A curvature prediction is also used for encoding the geometry. The latter shares the general idea of spectral methods (see. Section 3.2.6) because a topology-based *Karhunen-Loève transform* concentrates the distribution of geometry residuals. The residuals are entropy coded with a bit plane coder. Decimation conquests are interleaved with the transmission of bit planes to improve the R-D performance. They significantly improve the compression rates of the Alliez-Desbrun coder (about 4 bpv reduction with a 12 bit quantization).

Lee et al. [Lee et al., 2011] significantly improved the geometry compression (up to 60%) by introducing two new prediction methods. The *dual ring* prediction aims at having the same barycentric prediction for one vertex and its one-ring neighbor vertices. The *minimum mean square error* prediction constructs a linear predictor based on vertices having a topological distance of 1 or 2 with the predicted vertex. The algorithm first segments the input level of details by an algorithm based on the mesh connectivity. Then, the most efficient method is chosen to encode each cluster.

To improve the connectivity encoding, Kim et al. [Kim et al., 2011] proposed to predict the valence of the current patch inserted vertex during the decoding. With the geometry information, each patch possibility corresponding to one valence value is tried. The position of the inserted vertex is provided by the encoded geometry information. Then all the possibilities are ranked by measuring the regularity of the generated

triangle. The predicted possibility is the one with the highest regularity. For each level of details, the frequencies of each valence value is also used for better prediction.

3.2.3 Geometry-driven progressive mesh compression

As with single-rate mesh compression (see Section 2.3.3), the idea of focusing on geometry compression before connectivity has also been tested for progressive mesh compression.

In the scheme of Gandoin and Devillers [Gandoin and Devillers, 2002], the vertex positions are stored in a kD-tree by cell subdivision. The algorithm starts by constructing a cell that contains all the mesh vertices. Its width is equal to 2^q where q is the number of quantization bits. The number of vertices inside the cell is encoded. This cell is successively split in the three axis directions. For each split, the number of vertices contained in one sub-cell is entropy coded. The number of vertices of the second generated cell can be calculated with the number of vertices of the parent cell. After the three subdivisions of the root cell, the obtained child cube cells are recursively split in the same way and the vertices become more and more accurately localized. The subdivision process stops once the precision set by q is reached. The Figure 3.6 illustrates this process in 2D. To encode the connectivity, cells and their vertices are remerged through edge collapse and vertex unification operations. The reverse vertex splits and generalized vertex splits are encoded with prediction mechanisms based on the geometry.

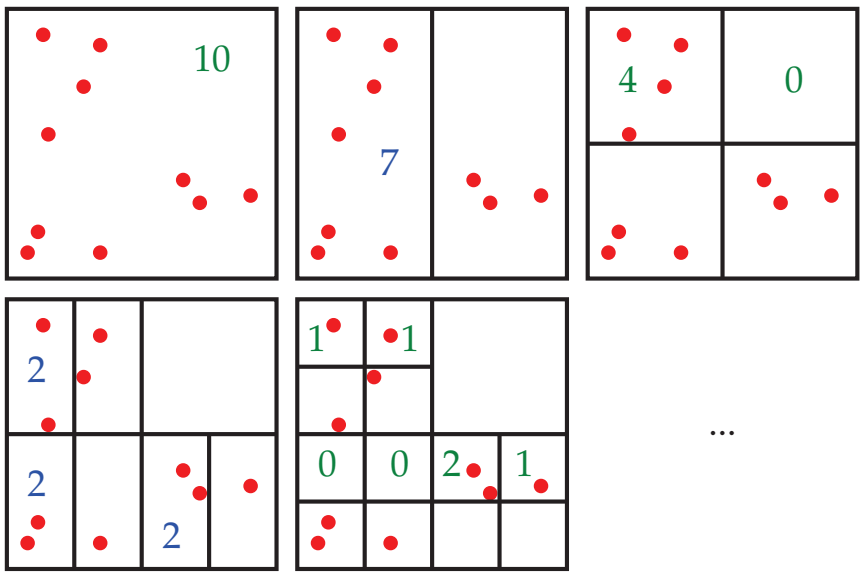


Figure 3.6: Progressive geometry encoding of the Gandoin-Devillers algorithm in 2D. For each step, the emitted symbols are the numbers that represent the number of vertices in each cell.

Peng and Kuo also developed a geometry-driven progressive mesh compression algorithm [Peng and Kuo, 2005] based on an octree data structure. After a subdivision, instead of encoding the number of vertices inside each cell as in [Gandoin and Devillers, 2002], the number of non-empty child cells is encoded. Then, the neighbor vertices allow to predict which cells may be non-empty. Rate-distortion performance is improved by prioritizing the subdivision of important cells. The connectivity of the mesh is encoded through vertex splits. The number of vertices connected to the two vertices generated by the split, called *pivot vertices* is encoded. Then a prediction algorithm based on the geometry determines the most probable candidates. Triangle meshes are compressed at about 15 bpv with a 12 bit quantization. Tian et al. [Tian et al., 2012] later proposed an alternative method to predict non-empty cells based on smoothness measure. They also rank the cell to subdivide only with their valence.

Hongnian et al. [Hongnian et al., 2009] used a binary tree to progressively encode the geometry information of a mesh. The mesh bounding box is divided into cells, which contains an unique vertex. To construct the binary tree, the mesh bounding box is recursively split sequentially in the three axis directions and the child cell are numbered accordingly. After a subdivision, a symbol among 3 is emitted to code if the left, right or both cells contain one vertex. After each subdivision, the connectivity transformation is encoded through vertex splits and generalized vertex splits. The simple encoding scheme saves the list of the edges between the generated vertices and their neighbors.

One advantage of these progressive compression methods based on space subdivision is that they can compress arbitrary simplicial complexes. Besides, they achieve very good compression rates with manifold meshes. However, their rate-distortion performance suffers at low rates due to the low quantization. No control of the important vertices removal is permitted. The mesh simplification cannot be driven with a metric contrary to connectivity-guided approaches.

3.2.4 Wavelet for irregular meshes

Wavelet frameworks are traditionally reserved for semi-regular connectivities as explained in Section 3.3.1 but Valette et al. proposed a subdivision scheme that can generate irregular meshes [Valette and Prost, 2004a]. They later built a progressive mesh compression algorithm based on this framework: Wavemesh [Valette and Prost, 2004b]. The initial mesh is progressively decimated with the subdivision scheme tailored to irregular meshes. The connectivity data is composed of all face subdivision operations. A triangle face can be subdivided into 4, 3, 2 faces or be unchanged.

The connectivity is encoded with three types of data. One bit per edge is encoded to tell if a new vertex is inserted to split an edge (see Figure 3.7 *a*). In the case where a face is subdivided into 3 faces, 1 bit is required to indicate the orientation of generates faces (see Figure 3.7 *b*). Finally, as some connectivity configurations do not allow to merge faces with the proposed subdivision schemes, the encoder performs some edge flips. Therefore 1 or 2 supplementary bits are needed in the case a face is subdivided into two so as to encode edge flips (see Figure 3.7 *c*). The geometry is encoded through a wavelet lifting scheme. The positions of the inserted vertices are predicted as being at the middle of their parent edges. A wavelet geometrical criterion was proposed to control the removal of sharp vertices.

The obtained compression rates are slightly better than those from [Alliez and Desbrun, 2001a] (about 19bpv with 12 bits quantization). Nevertheless, as the Wavemesh decimation scheme removes 75% of the mesh vertices at each decimation, while the Alliez and Desbrun decimation scheme [Alliez and Desbrun, 2001a] removes only 33% of the vertices, the Wavemesh representation generates less levels of detail.

Lee et al. [Lee et al., 2013] improved the compression performance of the original Wavemesh scheme by 16.9% on average. To improve the connectivity encoding, they proposed methods based on mixture of Gaussian probability models to predict the inserted vertices, the face directions and the edge flips from the geometry. For the geometry encoding, they divide the new vertices of a level of detail into three groups. To encode the vertex positions of a later group, the encoder uses the positions of the already encoded group(s). Vertex positions are predicted with the dual-ring prediction scheme from [Lee et al., 2011]. The residuals are encoded in a local coordinate frame like in [Alliez and Desbrun, 2001a] with a bit plane coder.

3.2.5 Progressive compression through reconstruction

Valette et al. [Valette et al., 2009] cast the progressive mesh compression problem as a mesh generation problem in their incremental parametric refinement framework. The encoder starts from a coarse version of the initial mesh generated by the simplification scheme from [Garland and Heckbert, 1997]. The base mesh is incrementally refined by splitting the longest edge. The position of the inserted vertex corresponds to the position of one vertex of the original mesh. Positions are adaptively quantized. The number of quantization bits depends on the level of refinement. At each refinement step, the triangulation is modified by means of edge flips to satisfy a local Delaunay property or to fix connectivity drifts. This process is summarized on

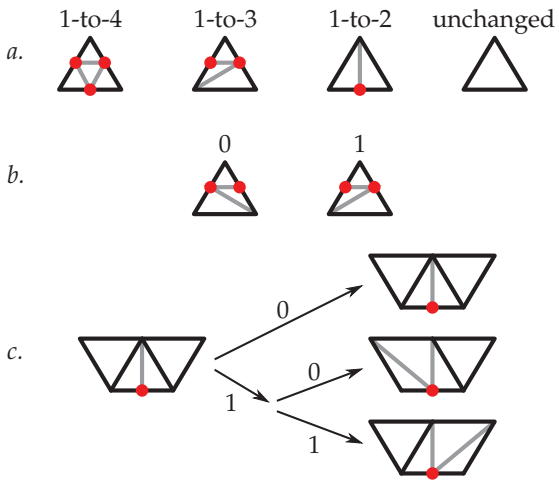


Figure 3.7: Connectivity encoding with the Wavemesh scheme [Valette and Prost, 2004b]. *a.* The four types of face subdivisions. Red vertices are added to split the edges. *b.* In the case a face is subdivided into 3 faces, one bit encodes the direction of the faces. *c.* In the case a face is subdivided into 2, to restore an initial connectivity modified to merge faces, one bit encodes if an edge needs to be flipped. If yes, an other bit codes which edge has to be flipped.

Figure 3.8. When all vertices of the original mesh have been inserted, the initial connectivity is restored by flipping edges guided by a flip distance heuristic. A bit per edge is encoded to tell if an edge must be flipped. This algorithm is known to compress efficiently (about 15bpv with 12 bits quantization) and achieves good rate-distortion performances. The complete connectivity restoration process is, however, not guaranteed to succeed.

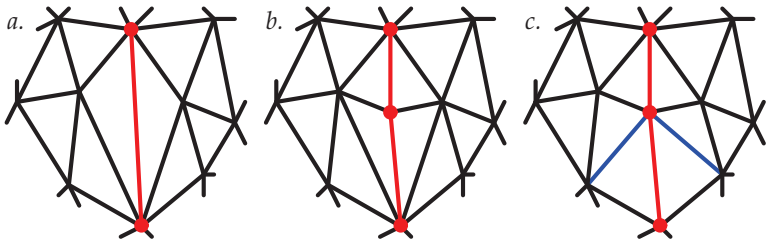


Figure 3.8: Incremental parametric refinement [Valette et al., 2009]. *a.* The longest mesh edge in red is selected for refinement. *a.* The red edge is split into two. *b.* The two blue edges are flipped to satisfy a local Delaunay property.

The idea of computing the best decimated version of an initial mesh has been recently further investigated [Peng et al., 2010]. The algorithm starts from the initial mesh vertex set and recursively splits it into several child subsets using generalized Lloyd algorithm [Lloyd, 1982]. Each time a new vertex subset is generated, a representative vertex of this set is selected to be close to the geometric center of the set and to have high curvature. In this hierarchy, the number of children of a set is encoded. The offsets between a representative and its parent representative are predicted in a cylindric frame and adaptively quantized. The connectivity is encoded through vertex splits with prediction of pivot vertices. This algorithm yields compression rates at about 16 bpv with a 12 bit quantization.

3.2.6 Geometry compression with the Laplacian operator

Fourier analysis has been commonly used for sound and image compression. Projecting the data into the frequency domain and encoding the low frequencies often allows to retrieve during the decompression a quality approximation of the original signal with few data. Karni and Gotsman [Karni and Gotsman, 2000] proposed to apply this technique to compress mesh vertex positions. To project the coordinates from the space domain to the frequency domain, the encoder uses the mesh *Laplacian operator*. The Laplacian matrix L of a mesh with n vertices is a $n \times n$ matrix defined by:

$$L_{i,j} = \begin{cases} 1 & \text{if } i = j \\ -1/degree(v_i) & \text{if } v_i \text{ and } v_j \text{ are adjacent} \\ 0 & else \end{cases}$$

The eigenvectors of L form an orthogonal basis of $\mathbb{R}^n$ and their associated eigenvalues are considered as frequencies. The projections of each coordinate component vectors on the basis vector are the mesh spectrum. The underlining principle is that if the geometry is smooth enough, its spectrum will be concentrated around low frequencies. So the spectral coefficients of low frequencies, after being quantized and entropy coded, are sufficient to build a good approximation of the initial mesh. In practice, spectral compression achieves excellent R-D performance with few coefficients. Incidentally, Ben-Chen and Gotsman [Ben-Chen and Gotsman, 2005] proved that spectral compression is optimal for certain classes of geometric mesh models as it is equivalent to principal component analysis on these classes. Mahadevan [Mahadevan, 2007] replaced the laplacian bases by diffusion wavelet bases and showed their ability to better represents an input mesh with the same number of basis functions.

As eigenvector decomposition is a high complexity operation ($O(n^3)$), the mesh must be segmented in regions of about 500 vertices in [Karni and Gotsman, 2000]. The computation becomes realist but is still very heavy. A later approach [Karni and Gotsman, 2001] proposed to use a fixed basis derived from a regular connectivity mapped to the irregular connectivity of the mesh. The hope is that the average energy will still be concentrated on low frequencies. The compression becomes less efficient but the decompression is accelerated since the eigenvector decomposition is not anymore computed. Bayazit et al. [Bayazit et al., 2010] reduced the computational complexity of the mapping method. They also used a bit plane encoder for the spectral coefficients as also presented in [Konur et al., 2008]. Cayre et al. [Cayre et al., 2003, Rondao-Alface et al., 2003] showed that rate-distortion gains can be obtained by introducing overlap between the segmented regions.

Mamou et al. [Mamou et al., 2010] devised an algorithm that computes the Laplacian matrix of a mesh. The mesh is then approximated by solving a heat equation with a minimal set of control points. The vertex locations are encoded as residuals from the approximation given by the heat equation. The connectivity is encoded by the Touma and Gotsman single-rate encoder [Touma and Gotsman, 1998]. This scheme achieves an excellent compression ratio (about 10 bpv with a 12 bit quantization). However, its high complexity due to solving the heat equation is a significant drawback.

3.2.7 Polygon meshes

Most of the approaches described above can only compress triangular meshes. For some of them, simple extension were proposed to compress meshes with arbitrary face degrees.

As stated in Section 2.2.6, a preliminary triangulation allows to compress polygon meshes with an existing method restricted to triangle meshes. Additional data has then to be encoded to restore the initial connectivity after the decompression. Taubin et al. [Taubin et al., 1998] followed this approach to extend the progressive forest split algorithm. Li et al. stated also that their compression scheme [Li and Kuo, 1998b] can be used on polygonal meshes. However, they provide no details about how the algorithm would be adapted. It can also be noticed that the experimental results presented in [Taubin et al., 1998] and [Li and Kuo, 1998b] do not compare favorably with recent state of the art techniques (see Section 3.2.1 and Section 3.2.2).

In more recent work [Peng and Kuo, 2005], Peng and Kuo discussed the progressive compression of polygon meshes by an octree coder (see Section 3.2.3). As their algorithm can compress arbitrary connectivity between vertices, it is possible to modify the face construction algorithm to reconstruct polygon faces. The mesh connectivity is encoded through vertex splits and efficient prediction of pivot vertices (see Section 3.2.3). By definition, pivot vertices are connected to the two vertices of the edge generated by a vertex split, but there is no pivot vertex when the adjacent faces of this edge are not triangles. This encoding scheme is therefore optimized to compress triangle meshes and is not best suited to polygon meshes.

3.2.8 Volume meshes

As far as we know, very few work have been proposed for the progressive compression of volume meshes. Pajarola et al. [Pajarola et al., 1999] designed a progressive compression algorithm for the connectivity of tetrahedral meshes. To generate the levels of detail, batches of independent edge collapses are performed. The connectivity is encoded by marking with one bit the vertices that must be split during the decompression and identifying the cut-faces around a split-vertex. An alternative approach can be to use the geometry-driven approaches [Gandoin and Devillers, 2002, Peng and Kuo, 2005] (see Section 3.2.3) as they can compress arbitrary simplicial complexes. They, however, do not integrate the notion of cells.

3.2.9 Compression of attribute data

The compression of mesh associated properties (see Section 1.2.1), like colors, plays a secondary role in state of the art compression schemes. However, mesh properties can often be of greater size than the other data flows of a mesh and can carry a great part of the relevant information, in particular in scientific visualization. Basically very few progressive mesh coders allow the encoding of color information; Cai et al. [Cai et al., 2007] and Cirio et al. [Cirio et al., 2010] proposed two geometry-based methods (respectively based on octree and kd-tree decompositions) that handle colors. However, these methods are limited by the poor quality of the intermediate models. Lee et al. [Lee et al., 2012] specifically adapted their connectivity-driven scheme to efficiently compress meshes with color attributes at vertices. The obtained visual quality of colored intermediate levels is much better than with geometry-based methods.

3.3 Connectivity-oblivious schemes

If the application do not require the restoration of the original connectivity of the mesh, this degree of freedom can be exploited to further reduce the compression rates by resorting to semi-regular remeshing or other kind of geometry representation. This section briefly enumerates approaches exploiting this idea.

3.3.1 Wavelet for semi-regular meshes

As for single-rate mesh compression (see Section 2.5), when the restoration of the initial mesh connectivity is not crucial, resorting to remeshing can further reduce the compression rates. In the general case, progressive mesh compression schemes based on wavelet start from a coarse irregular version of the input mesh and progressively refines it with a subdivision scheme that produces semi-regular meshes. After each subdivision, the vertices are moved to reduce as much as possible the distortion between the subdivided model and the input mesh. These delta displacements, which are the wavelet coefficients, are then encoded. Loop's subdivision [Loop, 1987] or the butterfly [Dyn et al., 1990] subdivision are often used for this purpose. The Figure 3.9 illustrates the subdivision of a mesh with such schemes.

In image coding, wavelet representation are known to decorrelate efficiently the original data. As a consequence, Khodakovsky et al. [Khodakovsky et al., 2000] proposed to also use wavelet for the compression of surfaces of arbitrary topology. However, this compression scheme cannot encode any connectivities. A remeshing of the input model is performed by the MAPS algorithm [Lee et al., 1998]. The wavelet transform, based on Loop's subdivision [Loop, 1987], replaces the original mesh with a coarsest irregular mesh

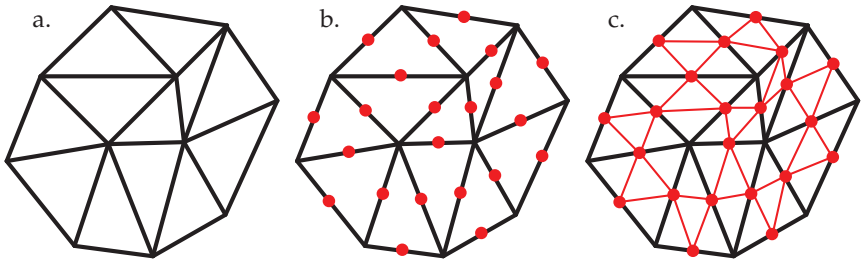


Figure 3.9: Butterfly or Loop's subdivision. a. The input mesh b. New vertices split all mesh edges. Their positions are computed with a formula given by the subdivision scheme c. Edges to link the added vertices are inserted to form the new faces. Old mesh vertices may be moved depending on the subdivision scheme.

and a sequence of wavelet coefficients expressing the difference between successive semi-regular levels of detail. The wavelet coefficients are represented in a local frame and later encoded with a zero-tree coder. This scheme was later improved [Khodakovsky and Guskov, 2003] thanks to the normal mesh representation [Guskov et al., 2000]. Normal mesh is a wavelet decomposition where the detail coefficients contain only one normal component wherever it is possible (above 90% of the cases) instead of three coefficients for standard subdivision. As there is only one wavelet coefficients to encode instead of three, normal mesh compression yields significantly better compression ratios.

Payan and Antonini [Payan and Antonini, 2002, Payan and Antonini, 2005, Payan and Antonini, 2006] allocate the bits across the wavelet sub-bands for the standard and normal mesh representations in order to improve the rate distortion performance. The proposed optimization framework varies the quantization of the wavelet coefficients to minimize the global reconstruction error. To improve the compression of normal meshes, Lavu et al. [Lavu et al., 2003] optimize locally the quantization of the normal component based on values previously encoded in the neighborhood. The encoder of Sim et al. [Sim et al., 2002] allocates bits to each cluster defined on the normal mesh representation [Guskov et al., 2000] in order to achieve the best rate-distortion performance. It also allows to allocate more bits to regions defined as important. In a similar idea, Zheng et al. [Zheng et al., 2004] proposed a scheme that focuses on the encoding of regions of interest.

Kammoun et al. [Kammoun et al., 2011, Kammoun et al., 2012] proposed to optimize the prediction scheme of lifting-based wavelet transforms (Butterfly and Loop). For each level of detail, the best parameters of the predictor are computed to minimize the set of detail coefficients. Experimental results show that, compared to classic wavelet decomposition, the root mean square distortion is slightly reduced (about 2%) at similar rate. Chourou et al. [Chourou et al., 2008] resorted to mesh segmentation in order to optimize the wavelet operators for each of the generated clusters. The parameters of lifting scheme prediction step are chosen to minimize the variance of the detail coefficients. Zhao et al. [Zhao et al., 2011] based their compression scheme on matrix-valued Loop's subdivision for better shape control. The encoder of Denis et al. [Denis et al., 2010] exploits the statistical dependencies between the intraband and composite wavelet coefficients to determine the best quantizers.

Chen et al. [Chen et al., 2008] proposed to compress an input surface by regularly remeshing it with quads. The quadrilateral subdivision splits each face into 4 new faces by inserting one vertex. The wavelet decomposition is formulated through a lifting scheme. Zero-tree coding is also used to encode the coefficients.

Li et Fan [Li and Fan, 2010] claimed that surfacelet transform (3D wavelet transform) better captures the features of 3D models. They compared the distortion of the reconstructed models. However, they did not provide any experimental compression rates.

Other mesh methods using wavelet transforms focus on mobile decompression [Ma et al., 2009] and the support of lossy transmission [Luo and Zheng, 2008]. The ideas behind are to reduce the number of computations needed for the decompression and to use error protection techniques to fit mobile computational capabilities and lossy networks.

As wavelet-based compression schemes resort to remeshing, it is not really possible to give absolute compression rates for compressed models. Indeed, neither the geometry (even with a set quantization) or

the connectivity are restored during the decompression. Nevertheless in general, wavelet-based algorithms provide a significantly better rate-distortion performance compared to their lossless counterparts.

3.3.2 Geometry image

Geometry image [Gu et al., 2002] is an original approach that compresses manifold meshes with a wavelet image compression scheme. To resample the input model over a regular 2D grid, the mesh is first cut in order to be homeomorphic to a disc. Then a parametrization function that maps the points of the cut mesh to the points of an unit square allows to compute x y z values for each pixel of the image. During the decompression, the geometry image pixels are used to build a triangle mesh approximation of the original mesh. However, the lossy compression leads to 'cracks' along the surface cuts. Hoppe and Praum [Hoppe and Praun, 2005] later proposed to construct the geometry image using a spherical remeshing approach. To unfold the sphere on to geometry image, they proposed a scheme based on a regular octahedron domain and an other scheme based on flattened octahedron domain. The geometry of these domain fits nicely with the use of spherical wavelet, thus avoiding boundary reconstruction issues.

Shi et al. [Shi et al., 2012] improved the compression of normal-map images since they are generally more difficult to compress due to their high variations. Their framework exploits the correlation between the normal-map image and the geometry image and between the three components of the normal-map image. Sander et al. [Sander et al., 2003] mapped the surface piecewise on several charts so as to reduce the distortion. Peyré and Mallat [Peyré and Mallat, 2005] investigated the compression of geometry images with bandelets. The aim of the bandeletization is to remove the correlation between high amplitude wavelet coefficients. Experimental results show that the distortion can be reduced by about 1,5 db compared to classical wavelet compression at similar rates.

Mamou et al. proposed a progressive mesh compression method based on b-splines and geometry images. After a segmentation of the input mesh, each patch are parameterized and approached by a b-spline surface. The B-Spline control points are quantized and encoded into three gray-scales geometry images (one per axis) with the progressive JPEG 2000 encoder. The connectivity of the patches is lossless encoded with the Touma and Gotsman algorithm [Touma and Gotsman, 1998]. Additional encoded information allow to recover the patch adjacency relations.

Ochotta and Saupe [Ochotta and Saupe, 2008] proposed an alternative image-based surface compression method. The input mesh is first partitioned. Each region is then projected on a plane. The resulting height fields are transformed into images and compressed with an adaptive wavelet coder. After the decompression, the partition are stitched in order to generate a close mesh. The obtained experimental results are similar to the results of the normal mesh approach [Guskov et al., 2000].

3.3.3 MeshGrid

Salomie et al. proposed a new surface representation, named MeshGrid [Salomie et al., 2004], based on a 3D connectivity wireframe and a reference grid defined in the 3D space. The grid vertices are not necessarily regularly spaced. The connectivity wireframe describes how the input surface goes through the 3D grid. One wireframe per level of detail is encoded. The 3D grid is compressed with a 3D wavelet coder. Munteanu et al. [Munteanu et al., 2010] later experimentally showed that local error control of the grid vertex positions based on an L-infinite distortion metric can improve the rate-distortion performance of the MeshGrid representation.

3.4 Conclusion

We summarized in the Table 3.1 what we judged as the main progressive compression approaches. We think that several points from this review should be highlighted.

- Progressive algorithms that do not restore the initial connectivity (wavelet-based approaches, geometry images...) lead to a much better rate-distortion performance compared with the lossless algorithms. This fact seems intuitive as, if we do not care about the connectivity, this leaves a larger margin to focus on the geometry encoding. Consequently, when the connectivity restoration is not crucial, such schemes should be preferred.

- Among lossless progressive algorithm, the geometry-driven approaches based on space subdivision provide very good final compression performance. However, the rate-distortion performance suffers at low rates due to the low quantization.

- Algorithms based on reconstruction methods yield very good rate-distortion performance.

Recently, Berjón et al. [Berjón et al., 2013] reviewed the different geometry metrics to compare static mesh compression algorithms. Their conclusion is that there is still a need for formal methodologies as in the image processing field. Metrics that well represent the human perception should be preferred to the traditional RMS or Hausdorff distances because the final destination of a compressed model is often to be shown to an human. Corsini et al. [Corsini et al., 2012] have written a very complete review on perceptual metrics for 3D meshes.

Lavoué [Lavoué, 2011] has proposed a multiscale perspective metric (MSDM2) based on Gaussian-weighted curvature. Subjective experiments with human observers demonstrated that this metric can better capture a voluntary introduced distortion than the RMS or Hausdorff distances. Comparing the rate-distortion curves of progressive mesh compression algorithm drawn with this metric yields different results. This fact is interesting as, before this work, progressive mesh compression algorithms were never evaluated with such metrics. Consequently, a complete study might change deeply the currently admitted ranking of algorithms.

As for single-rate mesh compression (see Section 2.6), we see the future of the research in progressive mesh compression in **genericity** and **parallelism**.

Genericity The processing of non-manifold triangle meshes has already been studied with geometry-driven approaches. A solution was also proposed for the progressive compression of tetrahedral meshes. But as far as we know, no scheme was proposed for the processing of polygon surface meshes. It was never studied as a standalone problem. Only extensions of schemes targeted to the progressive compression of triangle meshes were proposed. The algorithm we present in the Chapter 4 fills this gap.

Parallelism To our knowledge, no parallel progressive mesh compression algorithm have been proposed in the literature. As the modern computing architectures are evolving towards more and more computing cores, we believe that future work should address this problem. The progressive random-accessible approach described in Chapter 8 can compress and decompress in parallel all the generated clusters. This leads to a faster decompression compared to traditional progressive approaches.

Algorithm	Lossless connect. comp.	Total comp. rates (bpv)	Compress non-manifold meshes	Progr. granularity	Remarks
Progr. simplicial complexes [Popović and Hoppe, 1997]	yes	50 (10 bit)	no	5/5	
Progr. meshes [Hoppe, 1996]	yes	37 (10 bit)	no	5/5	
Progressive forest split [Taubin et al., 1998]	yes	30 (10 bit)	no	3/5	
Compressed progr. meshes [Pajarola and Rossignac, 2000]	yes	22 (10 bit)	no	3/5	
Colored patches [Cohen-Or et al., 1999]	yes	23 (12 bit)	yes	3/5	
Valence encoder [Alliez and Desbrun, 2001a]	yes	21 (12 bit)	no	3/5	
Wavemesh [Valette and Prost, 2004b]	yes	19 (12 bit)	no	1/5	Low number of levels of detail
Spectral compression [Karni and Gotsman, 2000]	yes	19 (12 bit)	no	3/5	No progr. coding of the connect.
Kd-tree coder [Gandoin and Devillers, 2002]	yes	19 (12 bit)	yes	2/5	High distortion at low rates
Octree coder [Peng and Kuo, 2005]	yes	15 (12 bit)	yes	2/5	Like above
Incremental parametric refinement [Valette et al., 2009]	yes	15 (12 bit)	no	5/5	Original connect. may fail to be restored
Geometry image [Gu et al., 2002]	no	-	no	-	Can generate cracks in decomp. models
Wavelet compression [Khodakovsky et al., 2000]	no	8 (eq. 12 bit)	no	3/5	Fits well to smooth and dense meshes
Normal meshes [Guskov et al., 2000]	no	6 (eq. 12 bit)	no	3/5	Like above

Table 3.1: Summary of the main progressive mesh compression algorithms. Approaches are approximately ranked by their compression performance.

Chapter 4

Progressive compression of manifold polygon meshes

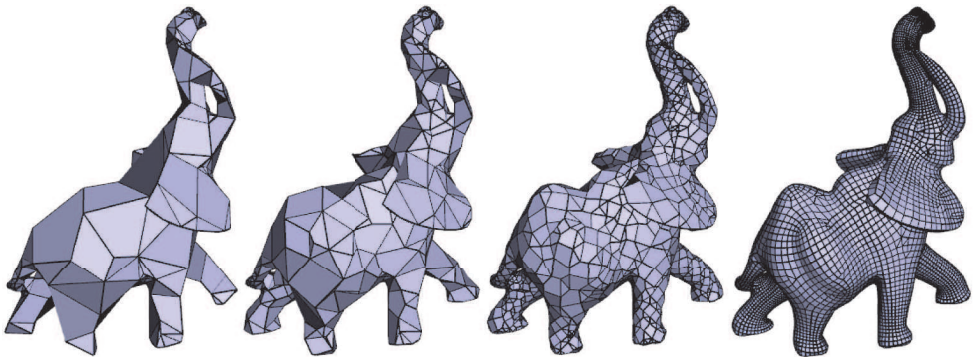


Figure 4.1: Levels of detail of the quadrangle elephant model generated by our progressive compression algorithm for polygon meshes.

4.1 Introduction

As described in the Chapter 3, progressive lossless mesh compression algorithms allow during decompression to first obtain a coarse version of the mesh. This first level of detail is then progressively refined as more data is transmitted and decompressed, until the input mesh is restored. The general goal of progressive mesh compression algorithms is to achieve the best rate-distortion performance in the sense that each decoded level of detail must be as close as possible to the original mesh with the minimum amount of decoded data.

Progressive mesh compression algorithms reduce the data size, which is useful in a context of storage and transmission. The fact that the user can have an access to the data without having fully downloaded and decompressed it is also interesting because it reduces the waiting time. Finally, the level of detail structure embedded in the compressed data allows to choose the best resolution that can be smoothly visualized on the user device. We therefore think that progressive mesh compression is an efficient tool for the **adaptation** of 3D meshes to storage and transmission constraints.

The Chapter 3 shows that previous work on progressive mesh compression has focused on the compression of triangle surface meshes. However, a significant number of carefully designed meshes are composed of polygon faces. In addition, many recent work focused on quad mesh processing. Moreover, for applications such as remote scientific visualization, meshes can contain not only triangular faces and the de-

compression must restore the initial connectivity. While some approaches have been proposed for single-rate compression [King et al., 1999, Isenburg and Snoeyink, 2000a, Kronrod and Gotsman, 2000, Isenburg, 2002, Khodakovsky et al., 2002] (see Section 2.2.6) or random-accessible compression [Courbet and Hudelot, 2009] (see Section 7.2), to our knowledge no specific approaches were proposed for their progressive compression until now. Only not efficient simple extensions of schemes targeted to the compression of triangle meshes were proposed [Taubin et al., 1998, Li and Kuo, 1998b, Peng and Kuo, 2005] (see Section 3.2.7). Following our objective of **genericity**, we therefore designed a new progressive compression scheme that can compress efficiently manifold polygon meshes. We named it **PPMC** for Progressive Polygonal Mesh Compression.

The content of this chapter can be summarized as follows.

- We first present a new simple **progressive polygon mesh** compression algorithm that compresses any 2-manifold mesh with arbitrary face degrees. Levels of detail are generated with a new decimation operator that combines vertex removal and local remeshing. The connectivity is encoded with two lists of boolean symbols: one for the faces with a removed vertex and one for the edges inserted by the remeshing. The geometry is encoded with a barycentric error prediction of the removed vertex coordinates.
- We then describe a **curvature prediction** method to improve the geometry coding and a **connectivity prediction** scheme based on the geometry to further reduce the size of connectivity.
- We propose a **wavelet formulation** of the geometry compression that contains a **lifting step**. After the decimation of one level of detail, the remaining vertices are moved in function of the position of the removed vertices. This formulation improves the rate-distortion performance of the encoder without increasing the final compression ratio.
- We also include the **adaptive quantization** algorithm from [Lee et al., 2012] that interleaves decimation traversals with global quantization operation. This method further improves the rate-distortion performance but increases the final compression rate.
- We demonstrate the efficiency of the proposed techniques by presenting **experimental results**.

4.2 Base algorithm

4.2.1 Compression

The proposed algorithm is based on mesh decimation. It is composed of four main steps that are successively repeated until the initial mesh M^n cannot be further simplified (see Figure 4.2).

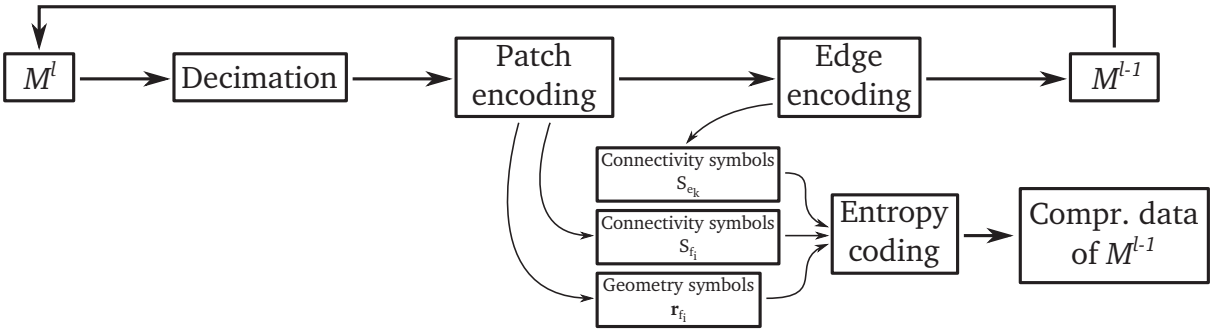


Figure 4.2: The four compression steps of the PPMC algorithm.

- The *decimation* step consists in applying a new decimation operator for polygon meshes, the *patch decimation* operator, to generate the mesh level of detail M^{l-1} from M^l. This operator removes vertices and adds new edges to the mesh.

- The *patch encoding* step builds the face symbol list S_f and the geometry residual list S_r. A deterministic face traversal of the mesh encodes the faces with a removed center vertex and the position of the removed vertices.
- The *edge encoding* step builds the edge symbol list S_e from the edge flags. A deterministic edge traversal encodes which edges have or have not been inserted.
- The *entropy coding* step compresses the S_f, S_r and S_e symbol lists.

In the following of the section, we detail the four steps.

Decimation step

The decimation step tries to simplify as much as possible a mesh LOD M^l to generate M^{l-1} using the patch decimation operator. This new operator combines the removal of one vertex and the insertion of edges. It can decimate polygon meshes with arbitrary face degrees.

A *patch* is a set of faces with a common center vertex. The decimation algorithm starts by randomly selecting a patch on the current level of detail. The degree of its center vertex v_j must be greater than two (see Figure 4.3(a)). It then creates a polygon f_i around v_j. To do so, it splits every faces around v_j that is not a triangle by inserting an edge between the two vertices of this face that are adjacent to v_j. We call this local remeshing operation *re-edging* (see Figure 4.3(b)). This operation aims at generating faces with low degree because they have more chance to be convex. The inserted edges, are marked. v_j is removed (see Figure 4.3(c)) and the residual

$$\mathbf{r}_{f_i} = \mathbf{p}_{v_j} - \mathbf{b}_{f_i}, \tag{4.1}$$

where $\mathbf{p}_{v_j}$ is the position of v_j and $\mathbf{b}_{f_i}$ is the barycenter of f_i vertices, is stored. The barycenter of the vertices of a face f_i is simply defined as:

$$\mathbf{b}_{f_i} = \frac{1}{|V_{f_i}|} \sum_{v_k \in V_{f_i}} \mathbf{p}_{v_k},$$

where V_{f_i} is the set of the f_i vertices. $\mathbf{r}_{f_i}$ represents the geometry data. f_i becomes a face of the mesh. It is marked as having a removed center vertex and can no longer be implied in a patch decimation operation in the current decimation step.

If v_j is a border vertex, the re-edging operation also creates a face with the three border vertices of the patch. The edge inserted to create this face is also marked as added. The stored residual becomes the difference between the position of v_j and the position of the center of this edge. When the algorithm decimates a mesh with boundaries, it considers that there is a fake face at the outer side of each boundary edge. The fake face f_i at the outer side of the inserted boundary edge is marked as having a removed center vertex. The decimation of a border patch is illustrated on Figure 4.4.

By definition, the patch decimation operation always removes one vertex v_j from the mesh. If v_j has t incident triangles, the variation of the number of faces in the mesh caused by the operation is equal to $1 - t$. This means that, if v_j is only incident to non-triangle faces, $t = 0$. One face is inserted to the patch but the degree of the other patch faces is decremented by one (see Figure 4.3). Other patch decimations on neighbor vertices may make these faces progressively disappear. If v_j is only incident to triangle faces, then no re-edging is needed and the number of faces in the mesh decreases (see Figure 4.5).

Patch decimation operations must preserve the manifold property of the mesh, so that if a patch border vertex shares an edge with more than 2 other patch border vertices, the patch is not decimated. The generated faces are not necessarily planar. This is not a problem since non-planar faces can be good local approximations. Concave faces may yet be problematic to render and may lead to further deteriorations during decimation. In our scheme, a face is said concave if its edges, projected on a plane directed by the face

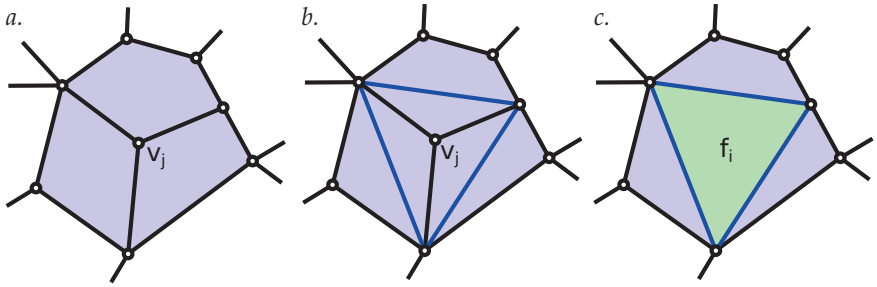


Figure 4.3: Decimation of a patch with non-triangle faces. *a.* The active patch is in blue. *b.* The faces are split by the re-edging operation with the inserted blue edges in order to create a new polygon around v_j. *c.* v_j is removed. f_i is marked as having a center vertex removed.

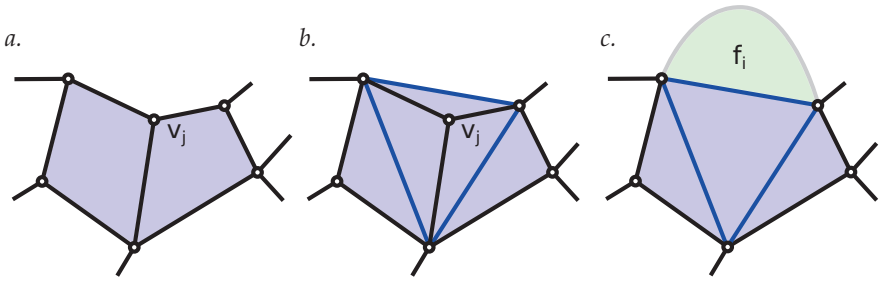


Figure 4.4: Decimation of a border patch. *a.* The active patch is in blue. *b.* The faces are split by the re-edging operation with the inserted blue edges in order to create a new polygon around v_j. A triangle face formed by the three border vertices is inserted *c.* v_j is removed. The fake face f_i on the boundary is marked as having a center vertex removed.

normal, form a concave polygon. A triangle face is by definition always convex. The normal of a non-planar face is computed with Newell's method [Tampieri, 1992].

Concave faces in a polygon mesh are difficult to render and, as a consequence, are often avoided. Our compression algorithm can be configured to only generate convex faces. In this case, the first decimation steps generate only convex faces. When it is no longer possible, the next decimation steps further simplify the mesh by allowing the generation of concave faces. The aim of these last decimation steps is to generate a minimal base mesh. During the decompression, the first LOD displayed to the user is the first that contains only convex faces.

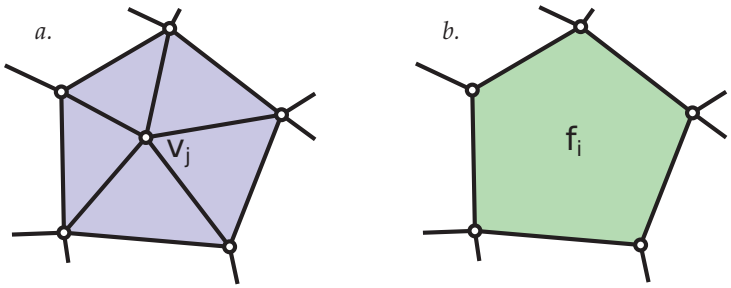


Figure 4.5: Decimation of a patch with triangle faces. *a.* The active patch is in blue. *b.* v_j is removed. f_i is marked as having a center vertex removed.

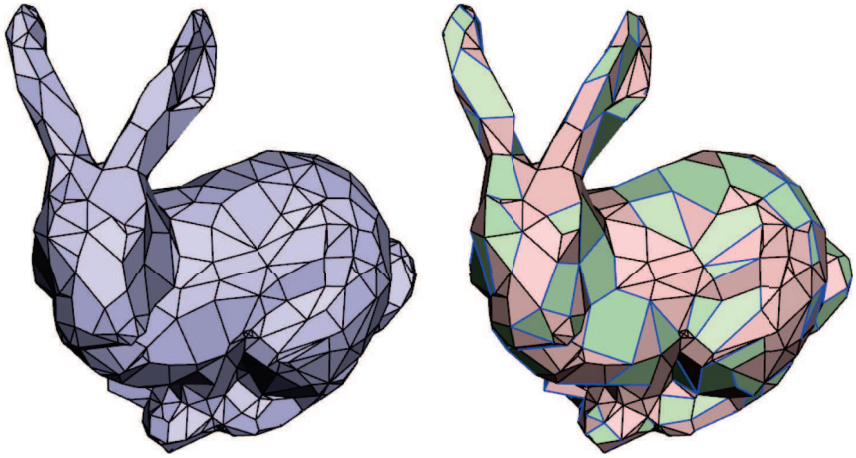


Figure 4.6: Decimation of an intermediate level of detail of the bunny model. Left: the initial level of detail. Right: the new level of detail after decimation. The inserted edges are depicted in blue. Faces with a removed vertex are depicted in green.

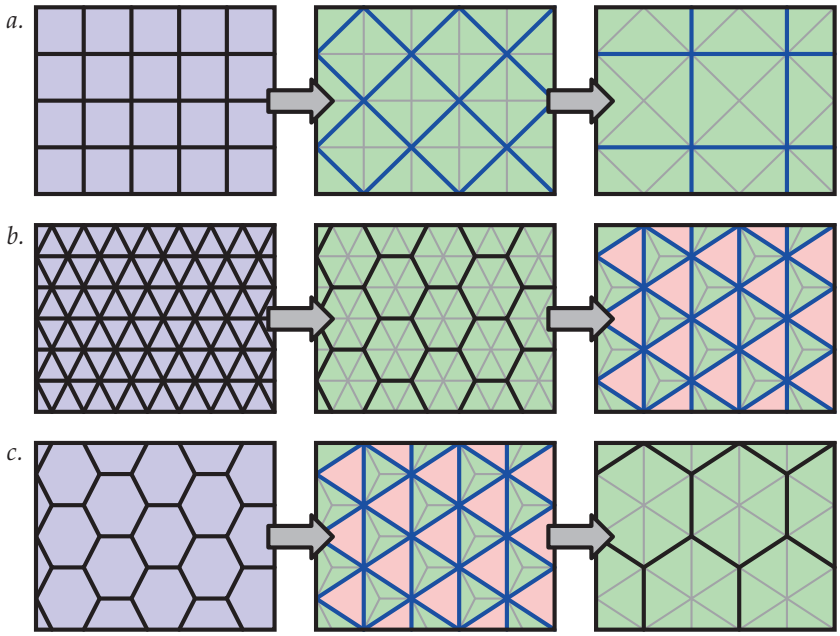


Figure 4.7: Examples of two successive decimations of regular connectivities. *a.* A regular grid is decimated into a larger regular grid pivoted of 45°. *b.* A regular triangle mesh is decimated into a regular hexagonal mesh. *c.* A regular hexagonal mesh is decimated into a regular triangle mesh.

In order to not too much deteriorate the shape of the input mesh, the decimation algorithm can also skip the decimation of important vertices through a volume-based metric such as the one proposed in the Chapter 5.

Once the current patch decimation is over, the algorithm attempts to create other patches with faces not marked as having a removed center vertex. The patch decimation order is not constrained. In our current implementation, the algorithm starts from a random seed vertex and progressively traverses the whole mesh by trying to generate new patches with adjacent vertices that do not belong to already marked faces. An

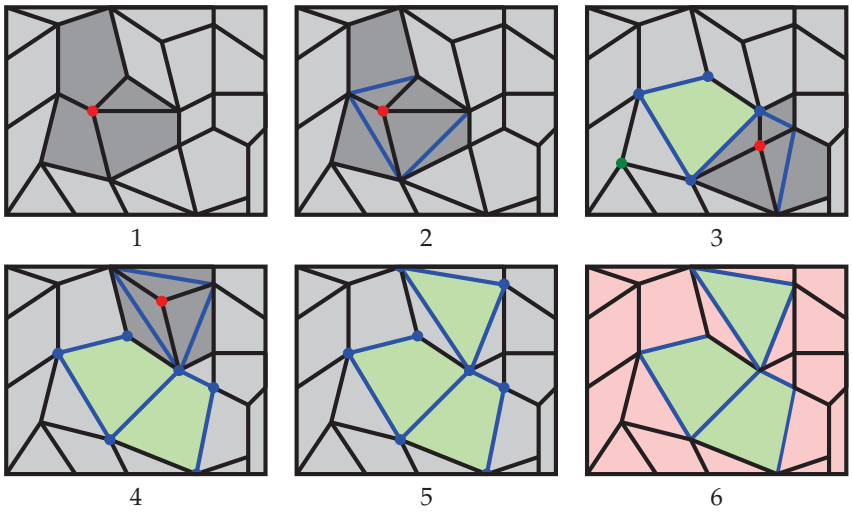


Figure 4.8: One example decimation step. 1. A seed vertex (in red) is chosen to form the dark gray patch. 2. The non-triangular faces of the patch are split by inserting edges (in blue) between their two vertices connected to the center vertex. 3-5. The patch center vertex is removed. The face generated by the vertex removal is marked as having a center vertex removed (in green). Its vertices are marked as visited (in blue). All its unvisited adjacent vertices are added to a FIFO queue in the counterclockwise order. A new patch center vertex (red) is popped out of the FIFO. If the decimation of the corresponding patch is invalid because, for example, it does not preserve the manifold property (the case of the green vertex at the step 3), the unvisited vertices adjacent to the current vertex are added to the FIFO and an other vertex is popped. 6. No more patches can be decimated. The current *decimation* step is finished. The results are: a set of faces with a center vertex removed (green), a set of faces without a removed center vertex (pink), a set of inserted edges (blue) and a set of original edges (black).

example is depicted in Figure 4.8. In the Section 5.3, we propose an alternative decimation algorithm guided by a volume metric.

A new mesh LOD M^{l-1} is obtained when no more patch decimation operations can be performed. The result of the decimation step is the simplified level M^{l-1} with a set of marked inserted edges and a set of marked faces with a removed center vertex (see Figure 4.6). Experimentally, about 30% of the vertices of a LOD are removed during a *decimation step*.

The decimation algorithm stops when no more *decimation* steps can be performed. The lowest LOD M^0 is called the *base mesh*. Its size depends on the complexity of the mesh but is generally less than 1% of the total file size. For the decimation of regular meshes, the results are roughly similar to those depicted in [Kovacevic and Sweldens, 2000]. As shown on the Figure 4.7, the *decimation* step preserves the regularity of the infinite regular structure it decimates. In practice however, the regularity gets progressively worse during decimation, as the meshes are not specifically designed to preserve regularity during decimation.

Patch encoding step

The *patch encoding* step produces the list S_f of Boolean face symbols s_{f_i} and the list S_r of the residuals r_{f_i}. s_{f_i} codes if f_i has a removed vertex or not. The algorithm uses a gate FIFO queue to perform a deterministic region growing traversal of the mesh. A gate is an edge between a traversed and an untraversed face. The first gate of the *patch encoding* step is specified for the whole mesh compression and hence can not be removed. When a face (or a fake face for meshes with boundaries) f_i is traversed, its symbol s_{f_i} is added to S_f. If s_{f_i} is *true*, we also add the projection of r_{f_i} in the local Frenet frame to S_r (see Figure 4.9). To avoid the post-quantization step mentioned in [Alliez and Desbrun, 2001a] and slightly reduce the entropy, we use the bijection proposed in [Lee et al., 2009] to transform the coordinates. This bijection corresponds to three

successive rotations plus rounding operations. The rotation angles are minimized to reduce the entropy of the residuals. The gates of f_i are then added to the queue in the counterclockwise order starting from the current gate. The next face to traverse is the one pointed by the next gate in the queue. The same traversal order is followed by the decoder during the decompression. An example of a *patch encoding* step is depicted in Figure 4.10.

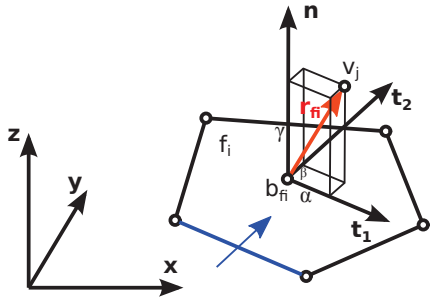


Figure 4.9: Encoding of the geometry residual $\mathbf{r}_{f_i}$ in the local Frenet frame ($\mathbf{t}_1$, $\mathbf{t}_2$, $\mathbf{n}$). $\mathbf{t}_1$ takes the direction of the gate edge (in blue). $\mathbf{n}$ is the patch normal. And $\mathbf{t}_2 = -\mathbf{t}_1 \wedge \mathbf{n}$.

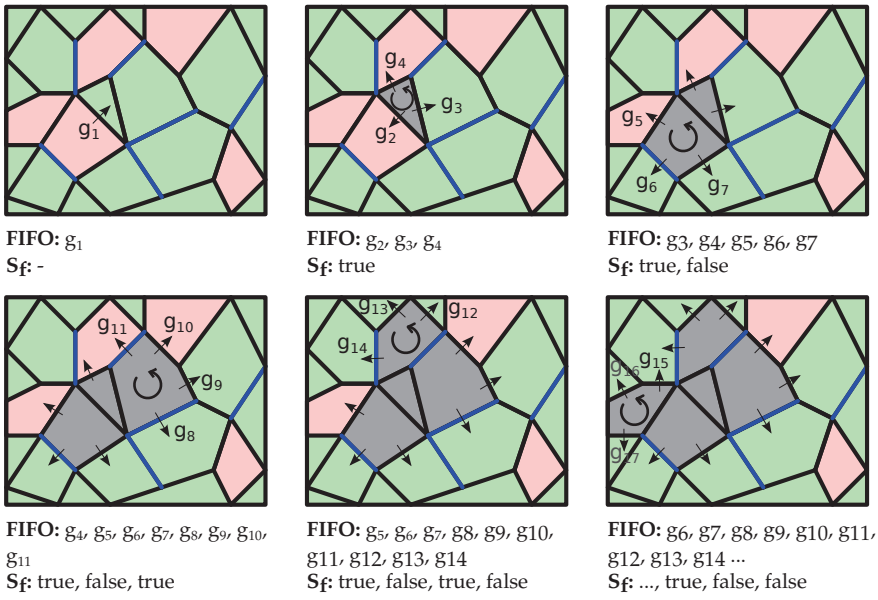


Figure 4.10: Example of a patch encoding traversal. Faces with, resp. without, a removed center vertex are depicted in green, resp. pink. Traversed faces are depicted in gray. The straight arrows represent the gates. The circular arrows represent the order of insertion of the gates into the FIFO.

Edge encoding step

To build the edge symbol list S_e, a deterministic full traversal of the mesh edges is performed using an edge FIFO queue. When an untraversed edge e_k is encountered, a symbol s_{e_k} is generated to code if e_k was inserted. Note that if e_k belongs to two faces that do not have a removed center vertex, it is inevitably original. Therefore, no symbol is generated in this case. The neighboring edges of one e_k vertex are then added to the queue. The next edge to traverse is extracted from the queue. An example of an *edge encoding* step is given by Figure 4.11.

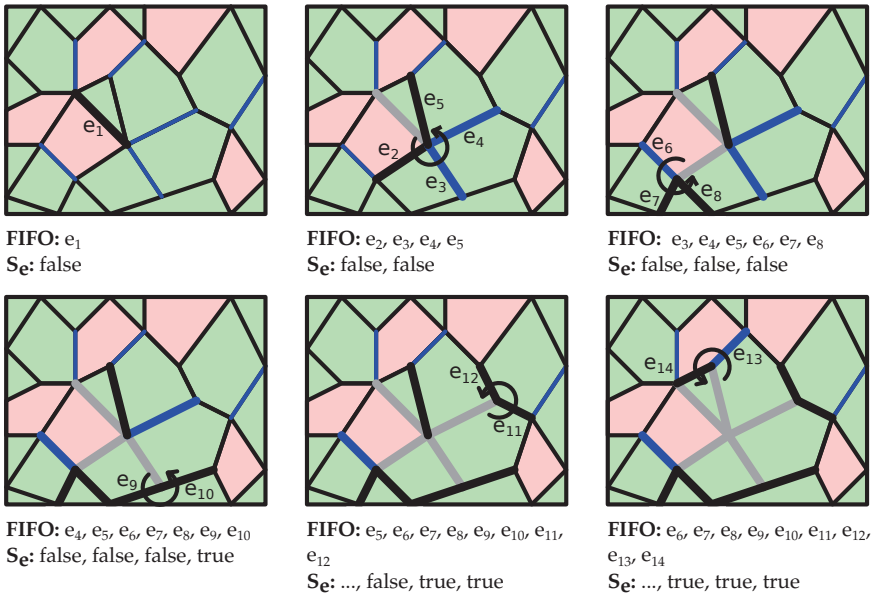


Figure 4.11: Example of an edge encoding traversal. The inserted edges are depicted in blue. The edges in the FIFO queue are depicted in bold. The traversed edges are depicted in gray. The circular arrows represent the order of insertion of the gates into the FIFO.

Entropy encoding

The previously described face and edge binary symbol lists S_f and S_e represent the connectivity data. The simplification process, depending on the LOD, may lead to biased binary distributions of their symbols (see Figure 4.14). Therefore, an entropy coder with one adaptive context per list is used to encode S_f and S_e.

The geometry data, the S_r list, is also entropy coded with two adaptive contexts: one for the tangential components of $\mathbf{r}_{f_i}$ and one for its normal component. In our current implementation, we used the range encoder from Michael Schindler [Schindler, 1998] (see Section 1.4.2). Figure 4.12 depicts the structure of the compressed data generated by our coder.

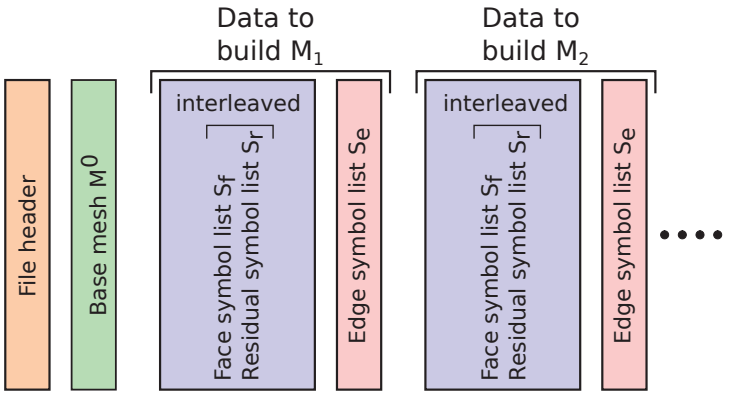


Figure 4.12: Structure of the compressed data generated by our coder.

4.2.2 Decompression

The mesh decompression starts by reconstructing M^0. Then, by applying the reverse operations of the *decimation* step, the successive LOD M^l are progressively restored to finally obtain M^n. Thus, for each LOD M^l, a first full traversal is performed to decode the faces with a removed center vertex. These vertices are then inserted at their original positions from the encoded geometry data. A second full traversal decodes and removes the edges that were not present in M^l.

4.3 Improving connectivity and geometry encoding

The scheme described above allows compressing and decompressing any 2-manifold polygon mesh. We describe next improvements to further reduce the size of the connectivity and geometry. They make our scheme competitive in term of compression ratio with triangle specialized approaches. The typical improvement is 0.3 bpv for geometry and 0.7 bpv for connectivity.

4.3.1 Predicting connectivity from geometry

As depicted in Figure 4.13, after one decimation step, the average area of the faces with a removed center vertex is greater than the average area of the other faces. This observation allows predicting connectivity from geometry. The prediction algorithm works as follows. During the patch encoding and decoding traversals, the average area of the two types of faces are progressively updated when new faces are traversed. If the area of the current face is closer to the average area of faces with a removed center vertex than to the average area of faces without a removed center vertex, then it is predicted as having a removed center vertex. Else, it is predicted as not having a removed center vertex. A binary symbol is generated to indicate if the prediction is verified. This symbol replaces s_{f_i} and is also later entropy coded. The binary distribution of this new symbol is for most cases more biased than the simple coding distribution.

For some connectivities, however, such as the decimation of a regular hexagonal mesh (see Figure 4.7(c)), the average area of the two types of faces can be equal. In this case, the type of a face can be predicted from the type of its neighbors. When the difference between the face type percentages is below a threshold (experimentally set at 10%), our algorithm predicts the type of a face as the inverse of the type that is the most represented among the adjacent already traversed faces.

A similar algorithm is used for the prediction of the edge symbols s_{e_k}. If the polygons are well-shaped, then the inserted edges are in general longer than the original edges due to the re-edging process.

To predict the group a face or an edge belongs to, we use average values of face areas and edge lengths computed on the part of the mesh already traversed. We make the assumption that the mesh is uniformly sampled. When the current LOD regularity is bad or the last decimation step removed few vertices, the simple coding scheme is more effective. Therefore, at the beginning of each connectivity entropy encoding step, a bit is written to indicate if the connectivity prediction algorithm is used or not.

4.3.2 Curvature prediction for geometry encoding

As described in Section 4.2.1 the geometry data is composed of the removed vertex residuals $\mathbf{r}_{f_i}$. To improve the geometry data encoding, we use a curvature prediction method. Instead of directly encoding the residual $\mathbf{r}_{f_i}$, we encode:

$$\mathbf{g}_{f_i} = \mathbf{r}_{f_i} - \alpha \mathbf{l}_{f_i}, \tag{4.2}$$

where $\mathbf{l}_{f_i}$ is the average Laplacian of the set V_{f_i} of the vertices of f_i. We define the Laplacian of a vertex v_j as:

$$\mathbf{l}_{v_j} = \frac{1}{|V_{v_j}|} \sum_{v_k \in V_{v_j}} \mathbf{p}_{v_k} - \mathbf{p}_{v_j},$$

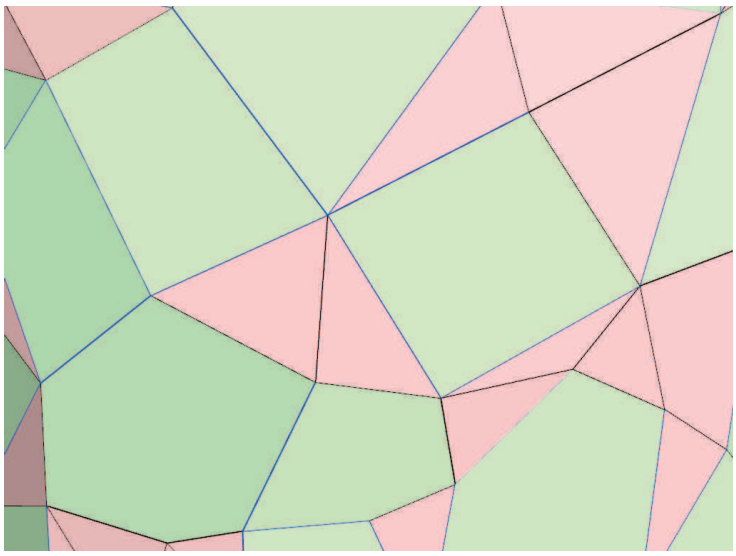


Figure 4.13: Result of a decimation traversal. The average surface of the face with a removed center vertex (green) is larger than the average surface of the other faces (red). The average length of the inserted edges (blue) is also superior to the average length of the others (black).

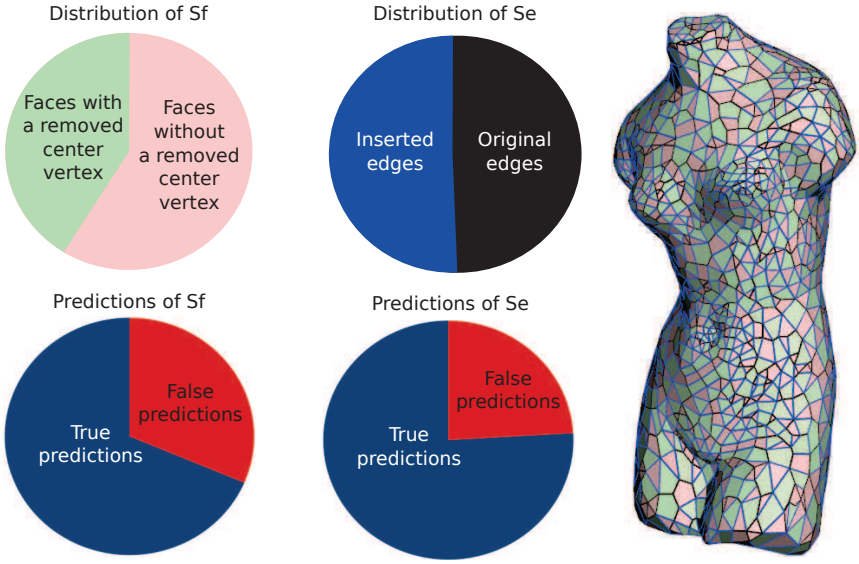


Figure 4.14: Example of connectivity symbol distributions after one decimation step. The first row shows the distribution of S_f and S_e. The second row shows the distributions of the prediction symbols.

with V_{v_j} the set of neighbor vertices of v_j and $\mathbf{p}_{v_k}$ is the position of the vertex v_k. Therefore we have:

$$\mathbf{l}_{f_i} = \frac{1}{|V_{f_i}|} \sum_{v_k \in V_{f_i}} \mathbf{l}_{v_k}.$$

In our experiments, we set $\alpha = 0.5$. For most meshes, $\mathbf{g}_{f_i}$ has a more biased distribution than the one of $\mathbf{r}_{f_i}$ and hence can be more effectively entropy coded.

4.4 Improving the rate-distortion

We now describe the inclusion to our codec of two methods that improve its rate-distortion performance. The first is based on a wavelet decomposition with a *lifting step*. It improves the rate-distortion ratios at low rates by about 25% without impacting the final compression rate. The second is the adaptive global quantization method taken from [Lee et al., 2012]. It improves the rate-distortion ratios at low rates by about 35% but increases the final compression rate by 1 to 2 bpv.

4.4.1 Wavelet formulation of the geometry compression

We formulate here the mesh geometry compression as a wavelet decomposition using the lifting scheme [Sweldens, 1996]. The idea of using a wavelet decomposition for the geometry compression of irregular meshes is not new [Valette and Prost, 2004b]. However, the wavelet decomposition we present in this section is specific to our method as we use different mesh decimation operators and geometry encoding schemes.

When a new level of detail is generated, two types of geometry data are computed during the wavelet decomposition:

$$C^{l-1} = A^l.C^l, \tag{4.3}$$

$$D^{l-1} = B^l.C^l. \tag{4.4}$$

C^{l-1} is the $m \times 3$ global matrix of the coarse coefficients, the m vertex coordinates of M^{l-1}. D^{l-1} is the $p \times 3$ global matrix of the detail coefficients, the p local $\mathbf{r}_{f_i}$ values that are encoded. The analysis filters A^l and B^l are defined by the decimation operations completed to generate M^{l-1}. A^l is the matrix that extracts the vertex positions of M^{l-1} from the vertex positions of M^l. B^l is the matrix that computes all the detail coefficients, as locally defined in (4.1) (see Figure 4.15 a).

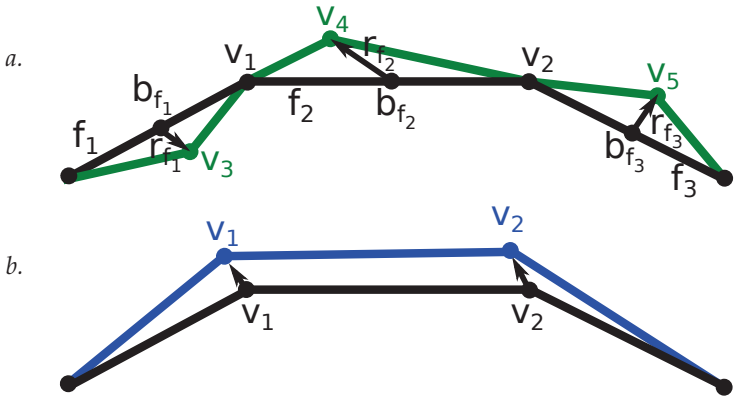


Figure 4.15: Transversal views of a mesh during the compression with the lifting scheme enabled. *a.* The mesh before decimation is depicted in green. The mesh after decimation is depicted in black. The detail coefficients $\mathbf{r}_{f_i}$ are the vectors between the barycenter of the face vertices (b_{f_1}, b_{f_2} and b_{f_3}) and the removed vertices (v_3, v_4 and v_5). *b.* The mesh before the lifting step is in black. The mesh after is in blue. The lifting step moves the position of the remaining vertices v_1 and v_2 according to the neighbor face $\mathbf{r}_{f_i}$ values to improve the rate-distortion distortion performance.

During the decompression, once the connectivity has been decoded, the coarse coefficients C^l can be obtained from the coarse coefficients C^{l-1} because the vertices of M^{l-1} are a subset of the vertices of M^l. The detail coefficients D^{l-1} are decoded from the compressed data. From (4.1), we have:

$$\mathbf{p}_{v_j} = \mathbf{r}_{f_i} + \mathbf{b}_{f_i}.$$

So, thanks to this local formula, it is possible to recover C^l from C^{l-1} and D^{l-1}. This process can be written under the form of the following equation:

$$C^l = P^l.C^{l-1} + Q^l.D^{l-1},$$

where P^l and Q^l are the synthesis filters determined by the previously described process.

This scheme corresponds to the lazy wavelet transform. It just separates the low and high frequency terms during the compression and put them together during the decompression.

Lifting step

To improve the rate-distortion performance of our coder, after a decimation step the position of the mesh remaining vertices (C^{l-1}) are moved in function of the positions of the removed vertices (see Figure 4.15 b). Each vertex position $\mathbf{p}_{v_j}$ is locally modified as follows:

$$\mathbf{p}_{v_j} = \mathbf{p}_{v_j} + \gamma \frac{1}{|F_{v_j}|} \sum_{f_i \in F_{v_j}} \mathbf{r}_{f_i},$$

where F_{v_j} is the set of the neighbor faces of v_j. If a face f_i does not have a removed vertex, then $\mathbf{r}_{f_i} = 0$. In our experiments, we set $\gamma = 0.5$ because it provided the best results with our datasets.

This local process can be formulated as a lifting step in our global wavelet formulation. So, (4.3) and (4.4) become:

$$C^{l-1} = A^l.C^l + \gamma L^l.B^l.C^l,$$
$$D^{l-1} = B^l.C^l,$$

where L^l is the matrix that computes the average residual of the neighbor faces (second term of the formula 4.4.1). During the decompression it is possible to rebuild the matrix L^l and then to restore the vertex positions with the decoded residuals using the following formula:

$$C^l = P^l.(C^{l-1} - \gamma L^l.D^{l-1}) + Q^l.D^{l-1}.$$

Curvature prediction and residual projection

As explained in Section 4.3.2, the entropy coder does not directly encode the residuals $\mathbf{r}_{f_i}$ but instead encodes the $\mathbf{g}_{f_i}$ values projected in the local Frenet frames. Given (4.2), the global matrix of the symbols values S^{l-1} is determined by the following equation:

$$S^{l-1} = U^{l-1}.(D^{l-1} - \alpha G^{l-1}.C^{l-1}),$$

where G^{l-1} is the matrix used to compute the $\mathbf{l}_{f_i}$ values and U^{l-1} is the matrix used to perform the local projections. During the decompression, the wavelet coefficients matrix can be restored with

$$D^{l-1} = V^{l-1}.(S^{l-1} + \alpha G^{l-1}.C^{l-1}),$$

where V^{l-1} is the matrix that performs the reverse local projections, before applying the rest of the lifting scheme. The whole process is summarized by Figure 4.16.

4.4.2 Adaptive quantization

To get the best rate-distortion performance when compressing a mesh, two variables can be played with: the number of vertices V and the number of geometry quantization bits B. For the single rate compression of triangle mesh, King and Rossignac proposed in [King and Rossignac, 1999b] methods to optimize the choice of V and B to minimize either the approximation error or the file size. For the progressive compression of

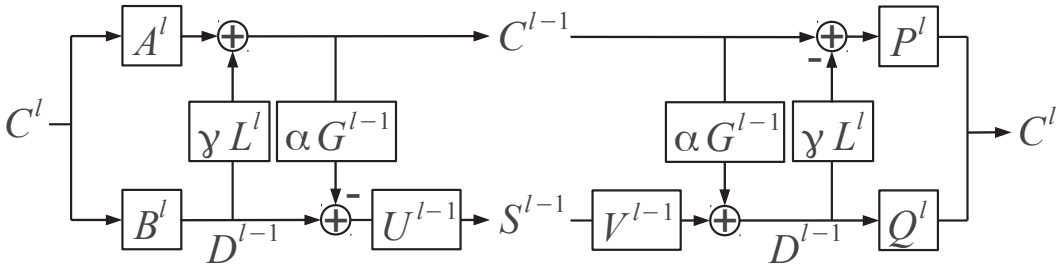


Figure 4.16: One level wavelet analysis and synthesis lifting scheme.

triangle meshes, Lee et al. [Lee et al., 2012] showed that the rate-distortion performance of the original AD coder [Alliez and Desbrun, 2001a] can be significantly improved by interleaving decimation conquests with vertex global quantization operations (see Figure 4.17). The main rationale is that a precise level of the initial quantization is not needed for low LODs which have very few vertices. The decimation contests are encoded with the AD coder and the quantization contests are encoded with the Peng and Kuo geometry coder [Peng and Kuo, 2005].

Lee et al. propose two methods to choose at each iteration whether to decimate the mesh or to quantize its vertex positions. In the first method, denoted by *optimal* the decimated mesh and the quantized mesh are generated with their compressed data. The two rate-distortion ratio are then compared. The chosen operation is the one that yields the lowest ratio. This method provides the best rate-distortion performance for all levels of detail but is computationally intensive. It requires to generate both meshes, to determine the size of the encoded data and to measure the two distortions with the initial mesh. Therefore, the authors recorded the choices made by their optimal coder on a mesh corpus to learn the parameters μ and β of a function $q_G(K_G)$ that provides the best number of quantization bits according to the level of decimation:

$$q_G(K_G) = round(\mu * log(K_G) - \beta)$$

where $round()$ corresponds to the nearest integer rounding function. K_g is defined as:

$$K_G = \frac{volume\ of\ bounding\ box}{area \times\ number\ of\ vertices}.$$

$\mu = -1.248$ and $\beta = -0.954$ are reported as the best parameter values for the selected triangle mesh corpus. We used these values in our experiments as a first attempt to prove that the adaptive quantization is also suitable for polygon mesh compression.

A second method, denoted by *quasi-optimal*, was proposed to choose at each iteration whether to decimate or to quantize. If the current $q_G(K_G)$ value is lower than the current number of quantization bits, then the mesh is quantized. Else, it is decimated. The authors experimentally demonstrated that the *quasi-optimal* and *optimal* methods yield similar results. We implemented Lee's *quasi-optimal* method to improve the rate-distortion performances of our coder at low rates.

4.5 Experimental results

To demonstrate the efficiency of our scheme, we implemented the encoder and the decoder in C++ using the halfedge data structure of the CGAL library[1]. We divided the experiments in two parts. In the first part, we experimented the compression of polygon meshes. In the second part, we studied the compression of triangle meshes.

[1] http://www.cgal.org/

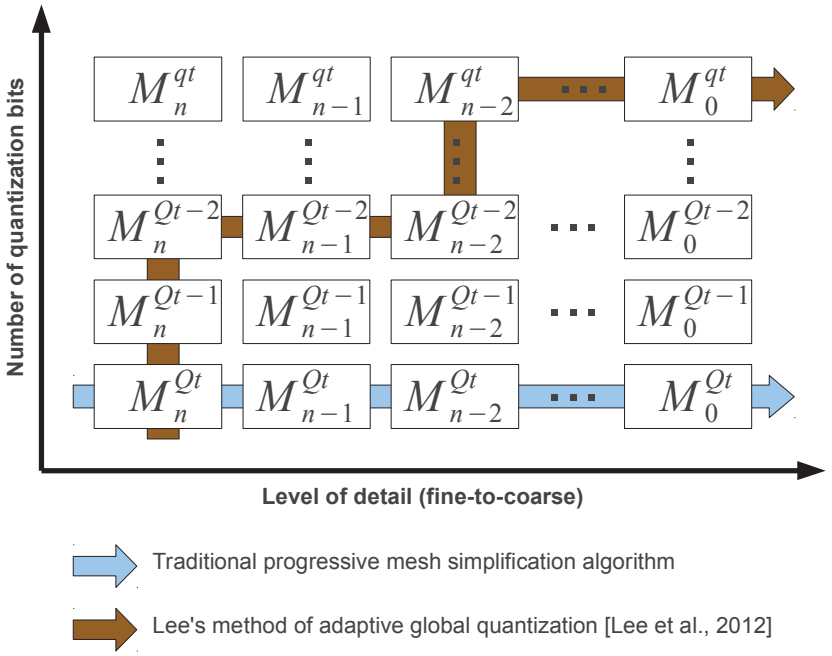


Figure 4.17: Progressive mesh traditional simplification vs. Lee's method for adaptive quantization [Lee et al., 2012]. Figure inspired from [Lee et al., 2012].

For both type of meshes, we measured the performance of the approach by determining final compression rates and drawing rate-distortion curves. All results presented in this section were obtained with the predictions described in Section 4.3. For the rate-distortion curves, we measured the distortion through the METRO software tool [Cignoni et al., 1998], after triangulating each polygon through inserting a vertex at the barycenter of its vertices and connecting it with all the polygon vertices.

We observed in our experiments that the computation times are approximately linear in the number of vertices. A 10 M face mesh is compressed in 1 min 48 s and decompressed in 1 min 22 s on a desktop computer equipped with an Intel Core i7 CPU clocked at 2.80 GHz and 8 GB of RAM.

4.5.1 Progressive compression of polygon meshes

This section presents polygon mesh compression results. The Figure 4.18 shows that the lifting scheme presented in Section 4.4.1 clearly improves the rate distortion curve at low rates without increasing the final compression rate. The rate-distortion optimization algorithm provides even better distortion at low rates but increases the overall compression rate.

The Table 4.1 lists compression rates on polygon models depicted in the Figure 4.19. In order to compare the effectiveness of our method against simple triangulation prior to compression (see Section 3.2.7), we triangulate polygonal models by choosing for each polygon an arbitrary vertex as pivot and adding edges between this vertex and all the others. The triangulated models are then compressed with the state of the art progressive compression method specialized to triangle meshes [Lee et al., 2012]. We add to the obtained compression rates the cost of the edge flag encoding required to restore the mesh initial connectivity after decompression (see Section 3.2.7). This cost is computed for each mesh with the Shannon entropy of the Boolean symbol sequence. The obtained values clearly improve over the trivial one. For our polygon mesh corpus, the trivial method costs on average 4 more bpv than our method.

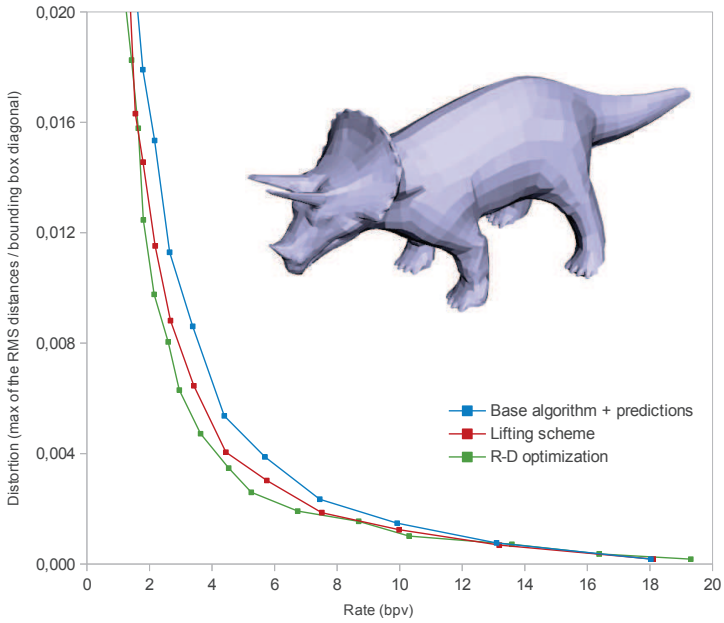


Figure 4.18: Rate-distortion curves for the compression of the Triceratops model with 10 bits quantization.

Our codec results are also compared with the results of the single-rate coder from [Isenburg and Alliez, 2002b] to evidence the cost of the progressiveness. This cost is high for the simple models (Shark, Teapot, Triceratops, Beethoven, Fandisk, Elephant) as their regularity gets rapidly worse during the decimation. Complex, regular models (Neptune, Chinese lion, Gargoyle, Rabbit) are efficiently compressed. Our algorithm performs well with irregular models (Horse, Fertility, Ramesses, Dinosaur) compared to the single-rate approach. It even improves on very irregular models such as the VSA-remeshed [Cohen-Steiner et al., 2004] Lucy or the Hippo models. The Figure 4.21 depicts the decompression of the Bimba quadrangle surface mesh. The Figure 4.20 shows levels of detail generated by PPMC.

4.5.2 Progressive compression of triangle meshes

We now compare our coder with state-of-the-art compression methods specialized to triangle meshes. Figure 4.22 and 4.23 depict rate-distortion curves obtained with methods specialized to triangle meshes, and our algorithm. Some of the results of these coders were already published in [Lee et al., 2012]. For our algorithm, we provide a curve with the lifting scheme and a curve with the adaptive quantization algorithm. We notice that several coders achieve better results in term of compression ratio and rate-distortion curves than our algorithm. In particular, [Mamou et al., 2010] performs very well, at the price of high compression and decompression times (3 minutes for a mesh with 20,000 vertices). The approach of Valette et al. [Valette et al., 2009] also provides very good results: the compression ratio are high and the rate-distortion curve is excellent with the rabbit model, at the price of not guaranteeing the restoration of the initial connectivity. The octree coder [Peng and Kuo, 2005] gives good compression rates but the distortion is high at low rates. The rate-distortion optimized coder from [Ahn et al., 2011] also performs well in terms of compression ratio but is weaker than our coder in terms of distortion at low rates. For triangle meshes, our algorithm provides similar distortion at low rates than the approach described in [Lee et al., 2012]. It also yields better compression rates as shown in the second part of Table 4.1. For the two presented triangle meshes, the lifting scheme yields slightly better final compression rates than the adaptive quantization method. For the irregular horse model, however, the rate-distortion performance is worse at some point.

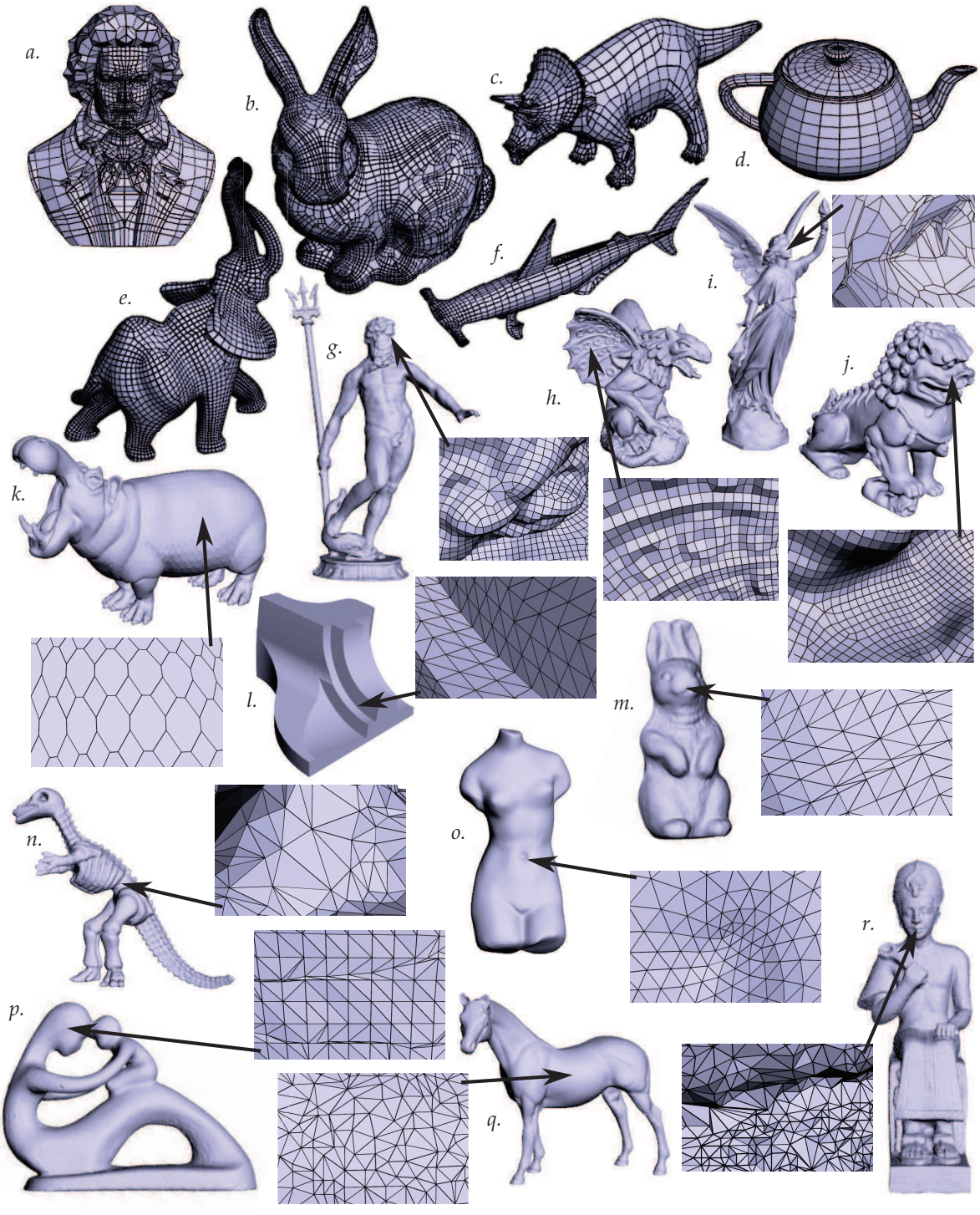


Figure 4.19: Input meshes from Table 4.1 and their tessellations. *a.* Beethoven *b.* Bunny *c.* Triceratops *d.* Teapot *e.* Elephant *f.* Shark *g.* Neptune *h.* Gargoyle *i.* Lucy VSA *j.* Chinese lion *k.* Hippo *l.* Fandisk *m.* Rabbit *n.* Dinosaur *o.* Venusbody *p.* Fertility *q.* Horse *r.* Ramesses.

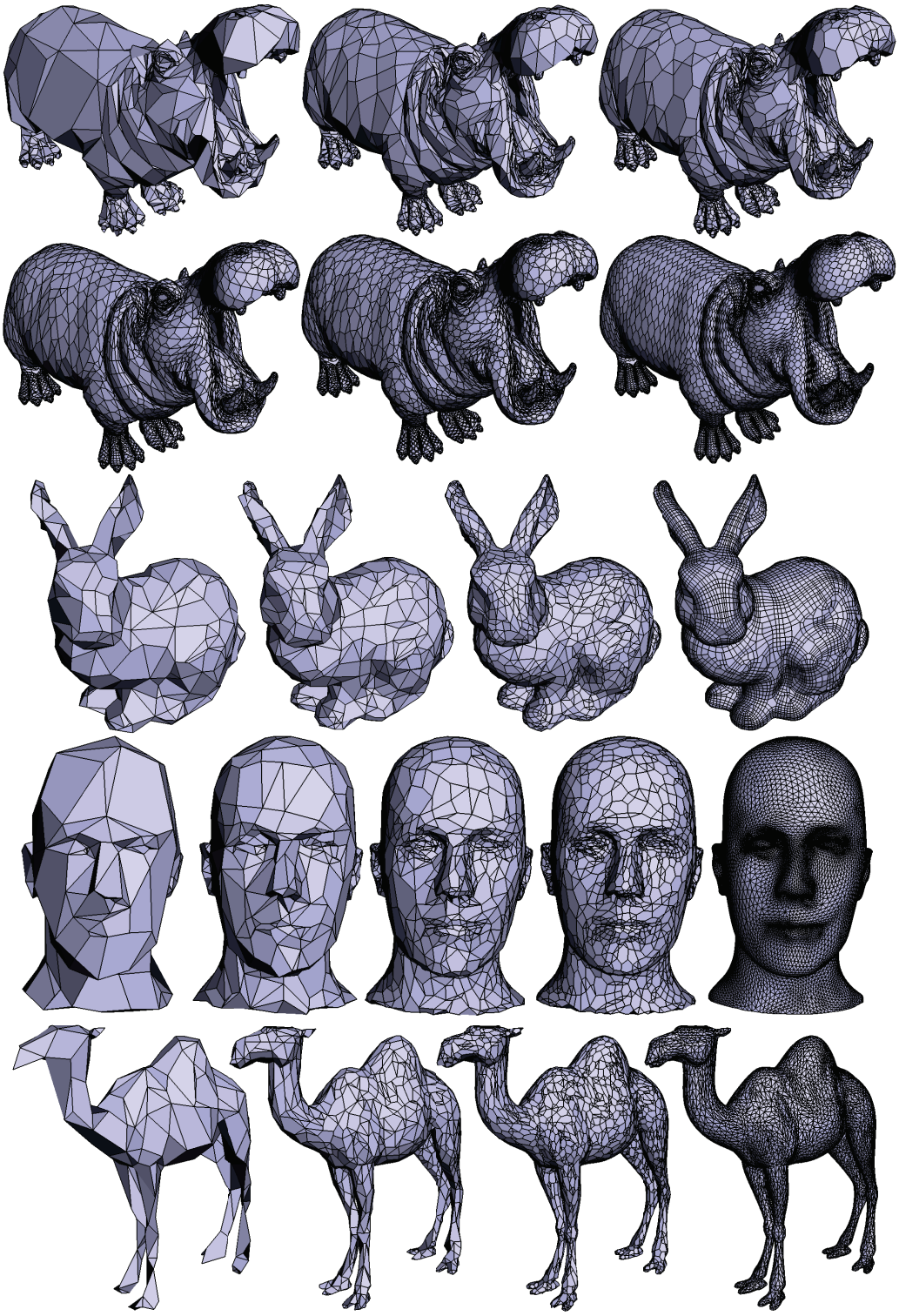


Figure 4.20: Levels of detail generated by PPMC.

Model	# poly.	Quant.	Our scheme			Lee 2012	Isen. 2002
			C.	G.	Tot.		
Beethoven	2812	10	5.7	16.7	22.4	26.0	16.6
Bunny	8814	10	3.8	11.8	15.6	20.3	12.2
Elephant	10895	12	3.4	12.3	15.8	25.4	11.8
Shark	2562	10	5.1	11.4	16.5	21.0	8.1
Teapot	1290	10	4.7	13.9	18.6	25.5	12.2
Triceratops	2834	10	5.4	12.6	18.0	22.1	11.9
Neptune	112658	12	1.5	6.7	8.1	17.8	5.6
Chinese lion	128339	12	1.4	8.0	9.4	19.6	6.9
Gargoyle	32126	12	2.7	12.5	15.3	23.9	12.1
Lucy VSA	76646	12	5.4	13.8	19.2	21.5	20.3
Hippo	32658	12	4.0	11.8	15.8	21,8	16.0
Horse	39698	12	4.1	15.2	19.3	20.4	19.1
Fandisk	12986	10	3.9	11.9	15.8	15.7	9.8
Dinosaur	28136	12	4.6	16.3	21.0	21.2	20.2
Venusbody	22720	12	3.6	12.6	16.1	16.9	13.4
Rabbit	134074	12	3.2	11.4	14.6	16.2	12.1
Fertility	483226	12	3.6	10.2	13.7	14.7	13.4
Ramesses	1652528	12	4.2	7.7	11.9	12.3	11.9

Table 4.1: Compression rates in bits per vertex, without any rate-distortion optimizations. The first part of the table contains meshes with arbitrary face degrees. The second part contains only triangle meshes. C. stands for connectivity. G. stands for geometry. To compress polygon meshes with the method of Lee et al. [Lee et al., 2012], a preliminary triangulation is performed. We add to the obtained compression rates the cost of the edge flags required to restore the original connectivity. The results of our scheme are compared against the results from the algorithms described in [Lee et al., 2012] and [Isenburg and Alliez, 2002b]

While being more general than progressive coders specialized to triangle meshes, our coder achieves competitive results for the compression of triangle meshes. It works with any 2-manifold mesh and exhibits good rate-distortion performances at low rate and average compression rates.

4.6 Conclusion

In this chapter, we introduced PPMC, a new progressive mesh compression algorithm. One distinctive property of our method is that it can handle surface meshes with arbitrary face degrees unlike previous approaches which only implement triangle mesh compression. It therefore answer to our objective of building **generic** approaches. However, the adaptation of this scheme to non-manifold connectivities does not seem to be trivial.

Starting from a simple algorithm based on decimation traversals, we propose solutions to improve the compression of both geometry and connectivity. We then incorporate two methods to optimize the rate-distortion performance: one based on wavelet lifting scheme, and the other based on adaptive global quantization. Experimental results show the effectiveness of our technical choices. Beside being more general than previous approaches, our method is also competitive for the compression of surface triangle meshes.

In our current implementation of the adaptive quantization method from [Lee et al., 2012], parameters computed with a corpus or triangular meshes are used. As our algorithm can compress polygon meshes, parameters suited for these models may yield better performances.

We also think that a better control of the decimation would improve the performance of the approach. This can be easily done since the decimation and encoding steps are independent. The Chapter 5 deals with the decimation of polygon meshes and introduces a simple volume metric. In the Section 5.3, a new

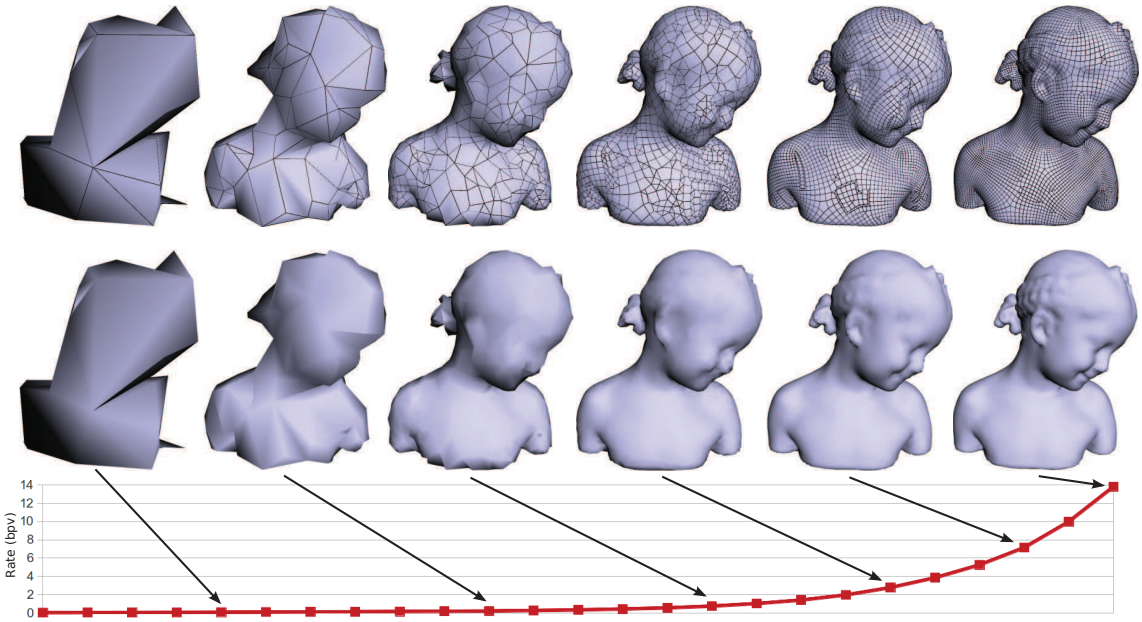


Figure 4.21: Decompression of the Bimba model (15770 quads) with the lifting scheme. The final compression rate is 13.8bpv with 12 bits quantization.

decimation method for PPMC is experimented. It removes first the vertices which removal would introduce the lowest distortion and allows in some cases to achieve better rate-distortion performance.

The multiresolution structure embedded in the compressed data generated by PPMC is an efficient tool for the **adaptation** of 3D data to network, visualization device capabilities and user preferences. In the Chapter 6, we show how an adaptation framework can take benefit of the features offered by a progressive mesh compression algorithm to enable the efficient remote visualization of scientific visualization meshes.

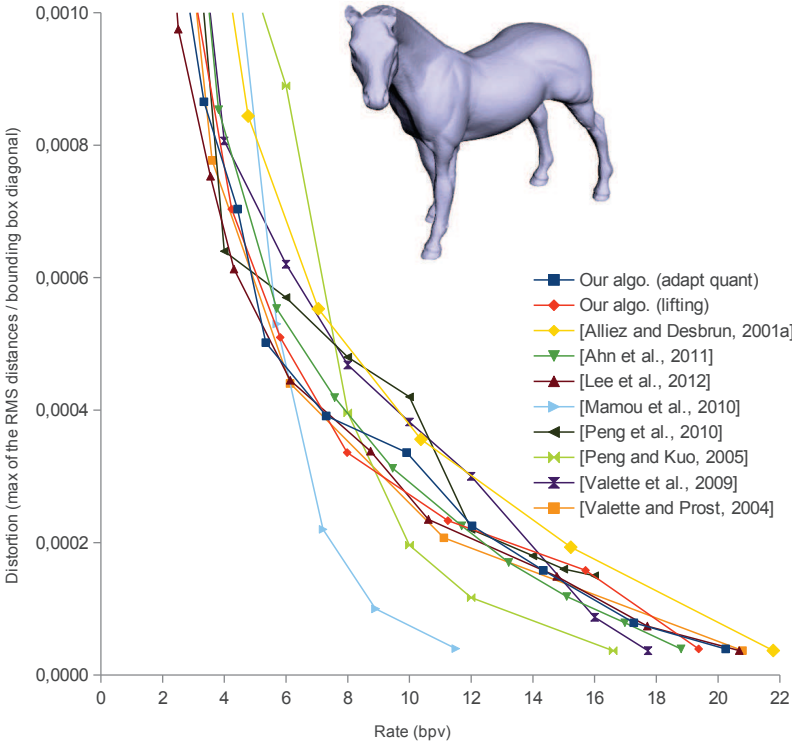


Figure 4.22: Rate-distortion curves for the compression of the Horse model with 12 bits quantization.

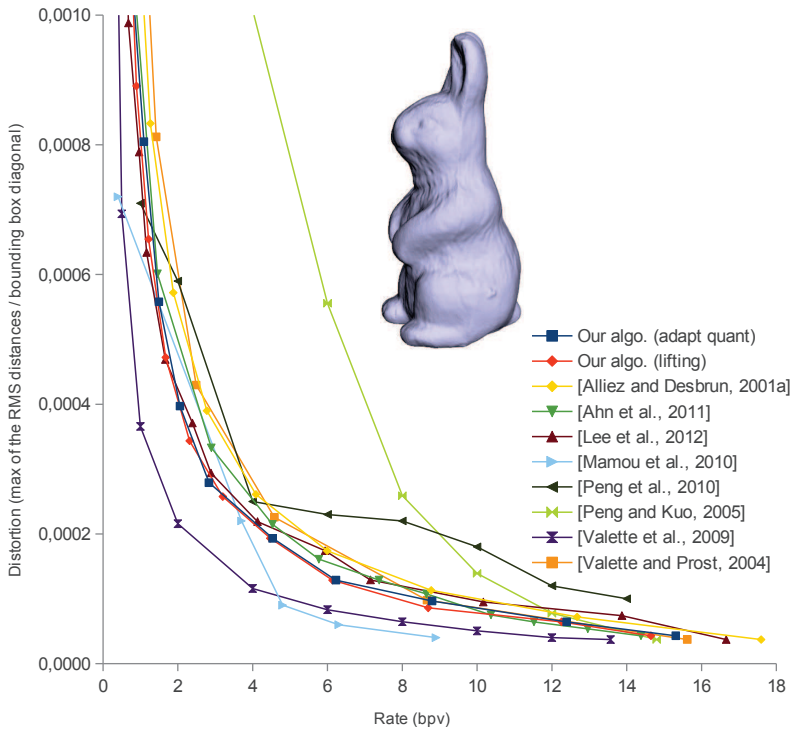


Figure 4.23: Rate-distortion curves for the compression of the Rabbit model with 12 bits quantization.

Chapter 5

A simple volume metric for polygon mesh decimation

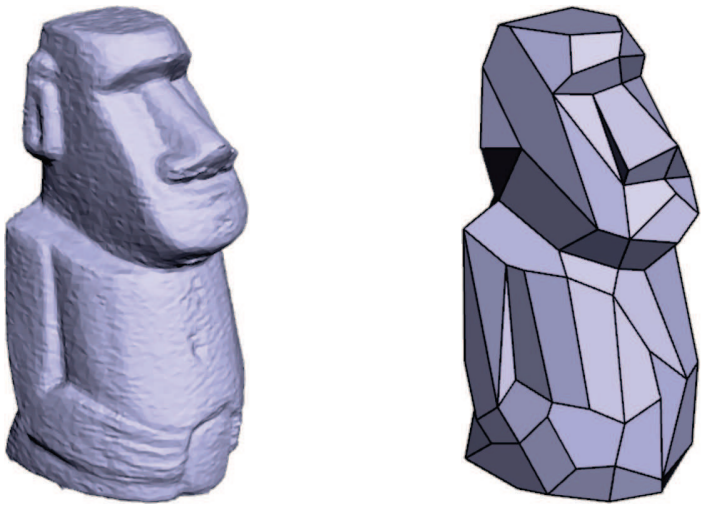


Figure 5.1: Simplification (100 faces) of the Moai model (20000 faces).

5.1 Introduction

Progressive mesh compression algorithms offer interesting features for the **adaptation** of 3D mesh to network, visualization device capabilities and user preferences. As seen in Chapter 3 and 4, progressive mesh compression implies the generation of levels of detail. So, progressive mesh compression is linked to mesh decimation. To well preserve the input mesh features, mesh simplification algorithms must be driven by a metric. For progressive mesh compression algorithms, the quality of the decimation is important to achieve a **high rate-distortion performance**. The choice of metrics for triangle mesh decimation has been widely studied [Matias van Kaick and Pedrini, 2006]. The case of general polygon meshes retained less interest.

Polygonal remeshing has been presented in the literature as an efficient way to represent a surface with few polygons of arbitrary degree. Alliez et al. proposed an Anisotropic Polygonal Remeshing method based on the determination of the mesh curvature directions [Alliez et al., 2003]. The Variational Shape Approximation of Cohen-Steiner et al. [Cohen-Steiner et al., 2004] is based on the clustering of the input mesh faces driven by a normal-based metric. More Recently, Lévy and Liu proposed an anisotropic polygonal remeshing

method based on their L_p-Centroidal Voronoi Tessellation framework [Lévy and Liu, 2010]. Pellenard et al. in [Pellenard et al., 2013] proposed an extension of the Variational Shape Approximation framework to generate rectangle-shaped polygons.

As far as we know, no incremental decimation approaches with a corresponding metric have been proposed for general polygonal mesh simplification. Some incremental algorithms were proposed for quad meshes simplification [Tarini et al., 2010, Daniels et al., 2009]. However, the simplification of meshes with faces of arbitrary degrees seems to have not yet been studied.

The content of this chapter can be summarized as follows.

- We first describe a very **simple polygon mesh decimation** algorithm and its corresponding **volume metric**. Used together, they generate decimated meshes with anisotropic faces of arbitrary degrees.
- We then apply the presented volume metric to the **selection of the patches** to decimate for **PPMC**, the progressive algorithm presented in Chapter 4.
- We demonstrate the efficiency of the both techniques by presenting **experimental results**.

5.2 Polygon mesh decimation

We present in this section an incremental decimation algorithm that generate polygon meshes. At each step, it performs the simplification operation that has the lowest cost until a stopping criterion is met.

5.2.1 Principle

Two basic operators are used for the decimation: the *halfedge collapse* and the *edge removal*. An halfedge collapse operation consists in merging a vertex with one of its neighbors and removing the degenerated faces if needed. It is depicted on Figure 5.2 *a*. An edge removal operation merges two adjacent faces by removing their common edge. It is depicted on Figure 5.2 *b*. We call *patch*, a set of faces that are modified by a decimation operation. To preserve the quality of the mesh, the algorithm does not perform the following operations:

- The operations that generate **non-manifold connectivities**.
- The operations that generate **concave faces**. To determine if a non-planar polygon is concave, we first compute its normal with Newell's method [Tampieri, 1992]. Then, the polygon is said concave if its edges, projected on a plane directed by its normal, form a concave polygon.
- The operations that generate **normal flips**. The algorithm checks for each face modified by an halfedge collapse that its normal before and after the collapse are not opposite. Following a similar idea, two faces that have an opposite normal cannot be merged by an edge removal.

All the authorized operations are ranked according to a volume metric detailed in Section 5.2.2. The operation that has the lowest cost, i.e that modify the least the volume of the mesh, is performed first. If a halfedge collapse operation and an edge removal operation have the same lowest cost, the edge removal is privileged.

When an operation has been performed, the operations related to the modified faces need to be rechecked and their cost updated. Thus, after an halfedge collapse, we check and update all the collapse costs of the halfedges of the faces adjacent to the vertex to split. We also update the removal costs of the edges of these faces. This process is illustrated on Figure 5.3 *a*. After an edge removal, for each vertex of the face that is left, we check and update the collapse cost of its incoming and outgoing halfedges. We also update the removal cost of all the edges incident to these vertices. This process is illustrated on Figure 5.3 *b*.

The decimation ends when a stopping criterion is met. This can either be that the targeted number of vertices or faces has been reached or that the next operation has a cost superior to a set threshold.

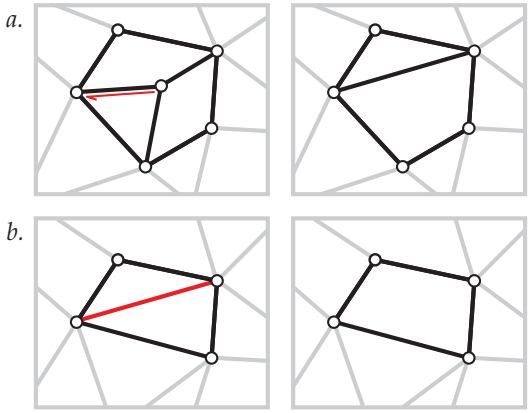


Figure 5.2: Decimation operators. *a.* The halfedge collapse. The halfedge in red is collapsed. *b.* The edge removal. The edge in red in removed.

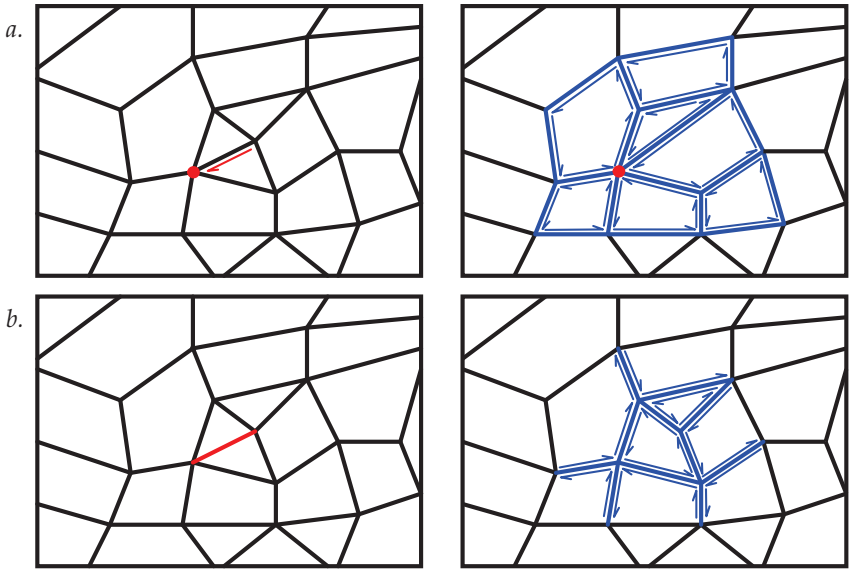


Figure 5.3: Update of the operation costs. In the second column, the halfedges which collapse costs are recomputed are in blue. The edges which removal costs are recomputed are also in blue. *a.* After an halfedge collapse. *b.* After an edge removal.

5.2.2 The simple volume metric

The simple volume metric we used measures the volume difference generated by a decimation operation. This volume is a polyhedron bounded by the local surface before the operation and the local surface after the operation. It is illustrated on Figure 5.4.

We use the divergence theorem to compute the volume of this polyhedron. We consider a polyhedron that bounds a volume Ω with a surface S, a normal $\mathbf{n}$ defined at all the point of S and a vector field $\mathbf{F}$. The divergence theorem can be expressed with the following formula:

$$\iiint_{\Omega} (\nabla \cdot \mathbf{F})\, d\Omega = \oiint_{S} (\mathbf{F} \cdot \mathbf{n})\, dS$$

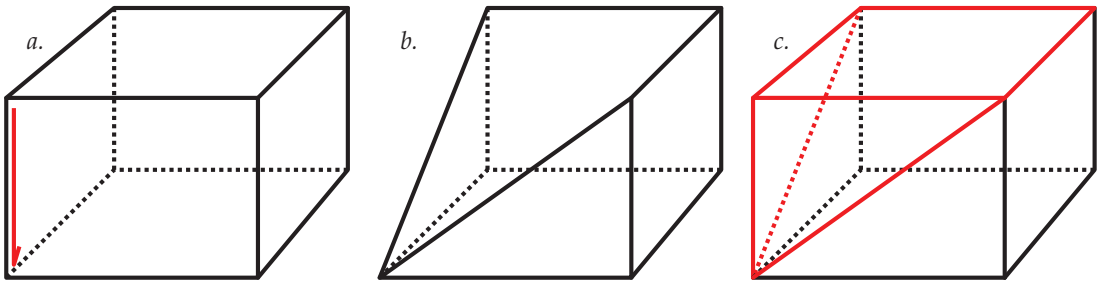


Figure 5.4: The volume metric. *a.* The mesh before the halfedge collapse shown in red. *b.* The mesh after the halfedge collapse. *c.* The value of the volume metric is equal to the volume of the red polyhedron that represent the difference between the volume of the mesh before and after the collapse.

It we consider the vector field $\mathbf{F}(\mathbf{x}) = \frac{1}{3}\mathbf{x}$, its divergence $\nabla \cdot \mathbf{F}$ is equal to 1. Therefore, the divergence theorem becomes:

$$Volume(\Omega) = \oiint_S (\mathbf{F} \cdot \mathbf{n})\, dS$$

We need to define analytically the surface generated by each non-planar polygonal face to compute this integral. This question is not trivial. It can have many answers but we chose a very simple solution that proved to be sufficient in our experiments. Each face of the polyhedron is triangulated by introducing a vertex at its barycenter and creating a triangle fan around it with the other face vertices. Therefore, the formula simplifies to:

$$Volume(\Omega) = \frac{1}{3} \sum_{f_i} \mathbf{x_i} \cdot \mathbf{n_i}\, A_i,$$

where $\mathbf{x_i}$ is the position of any point of the face f_i, $\mathbf{n_i}$ is the normal of f_i and A_i is the area of f_i. As f_i is a triangle, A_i and $\mathbf{n_i}$ are easy to compute.

5.2.3 Results

We implemented this simple decimation algorithm in C++ using the halfedge data structure from OpenMesh [Botsch et al., 2002]. The algorithm simplifies a mesh at about 2000 vertices per second on a desktop computer with an Intel Core i7 processor at 2.80 GHz.

The Figure 5.5 shows some examples of polygonal simplifications. Anisotropic polygons are naturally produced by the method. The shape of the input model is well preserved. Comparing to previous techniques [Alliez et al., 2003, Cohen-Steiner et al., 2004, Pellenard et al., 2013], the shape regularity of the generated polygons seems inferior. We think that our decimation method has, however, some advantages over previous work.

- It can simplify **polygon mesh** as input.
- It is **incremental**. The decimation algorithm outputs a valid mesh at any step. It can stop when the targeted number of faces or vertices is reached or when the metric value exceeds a threshold.
- Its implementation is **simple**. The method is based on elementary decimation operators and a simple volume metric. An implementation based on a library integrating an halfedge data structure is easy to built.

One way to improve the decimation quality could be to relocate the vertex to split after an halfedge collapse to produce nicely shaped polygons. Vertex relocation after an edge collapse has been widely used for progressive triangle meshes [Hoppe, 1996, Garland and Heckbert, 1997]. However, as far as we know, the case of polygonal models has never been studied.

5.3 Progressive compression guided by the volume metric

We presented in the Chapter 4 PPMC, a progressive mesh compression algorithm that can compress polygon meshes. As explained in Section 4.2.1, local patch decimation operations are applied to generate the successive levels of details. The order of the operations is not constrained. Therefore it is possible to prioritize the operations that modify the least the volume of the mesh.

The decimation algorithm of PPMC described in Section 4.2.1 starts from a random seed vertex and progressively conquers the whole mesh by trying to generate new patches with adjacent vertices that do not belong to already marked faces. We propose here a different strategy. To generate a new level of detail, all the allowed decimation operation costs are computed. The volume metric described in Section 5.2.2 computes the volume defined by the local surface before the patch decimation and the local surface after the patch decimation. Then at each step, the algorithm performs the operation that has the lowest cost. After a decimation, some neighbor operations become no longer possible because they would modify the just decimated patch. The cost of other neighbor operations has also to be recomputed because of the last re-edging operation.

This new decimation techniques used together with the lifting scheme proposed in Section 4.4.1 benefits to the rate-distortion performance at low rates for the compression of irregular meshes. However, as this decimation method tends to remove less vertices per step, the final compression rate is increased. This is also why this method does not improve the rate-distortion performance for the compression of regular meshes. The Figure 5.6 compares the rate performance of this new decimation method, the simple lifting scheme and the adaptive quantization method for the compression of irregular models.

5.4 Conclusion

We showed in this chapter that a simple volume metric can efficiently drive the simplification of meshes with arbitrary face degrees. We first described a polygon mesh decimation algorithm based on two operators: the halfedge collapse and the edge removal. We then presented a decimation strategy for the PPMC algorithm (see Chapter 4) that performs first the decimation operations that modify the least the volume of the mesh. This new method improves the **rate-distortion performance** at low rates for the progressive compression of irregular meshes.

Regarding the simplification application, we think that there is much space to improve the quality of the decimated model by allowing vertex relocation after an halfedge collapse. A fast vertex relocation scheme that produces well shaped polygon and minimizes the distortion, such as the approach of Garland [Garland and Heckbert, 1997] for triangle meshes, would be a significant contribution.

The next chapter shows how an **adaptation** framework can take benefit of the multiresolution structure provided by a progressive mesh compression algorithm to enable the efficient remote visualization of scientific visualization meshes.

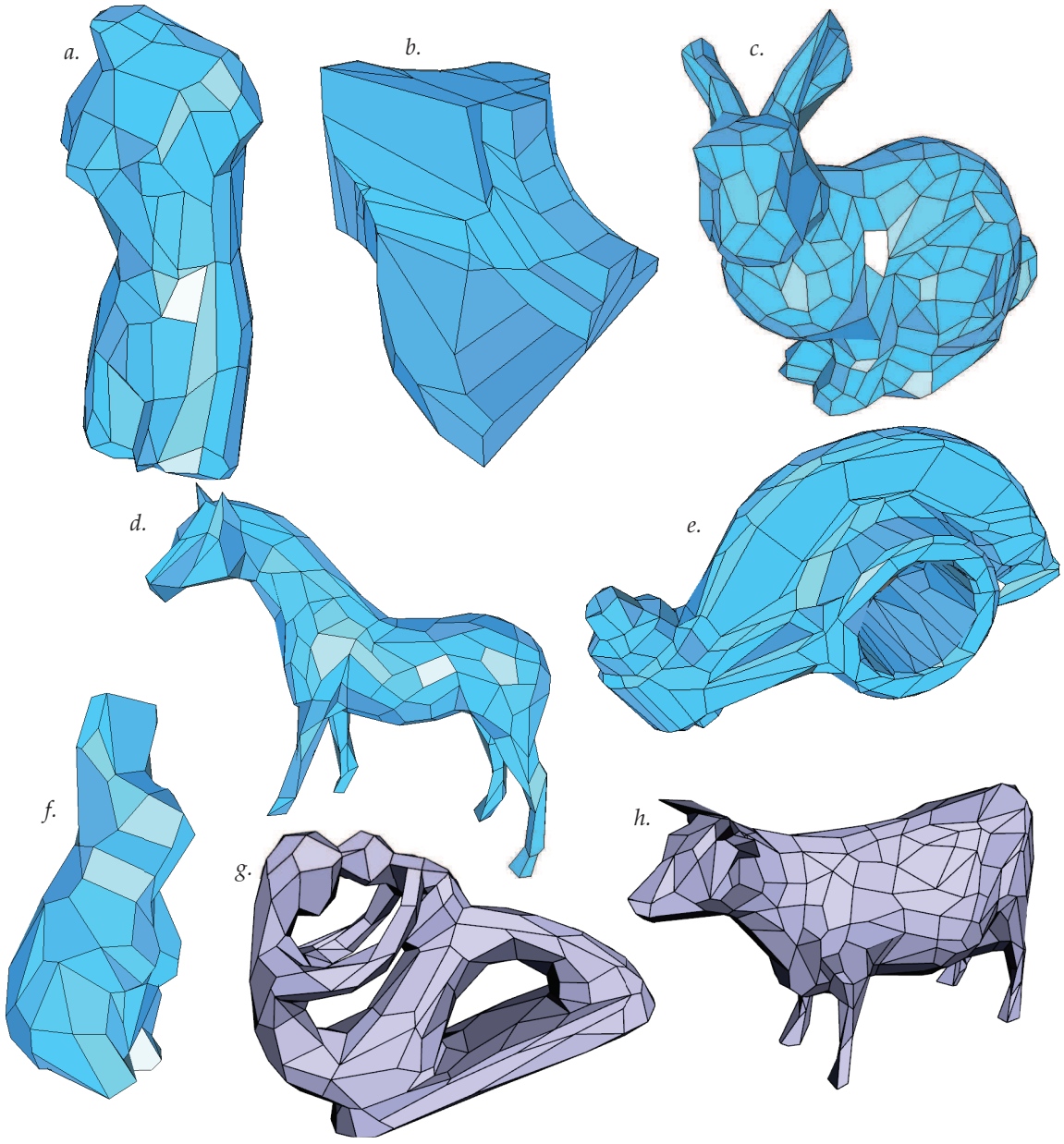


Figure 5.5: *a. venusbody* (200 faces). *b. fandisk* (100 faces). *c. bunny* (500 faces). *d. horse* (300 faces). *e. rocker arm* (500 faces). *f. rabbit* (100 faces). *g. fertility* (300 faces). *h. cow* (600 faces).

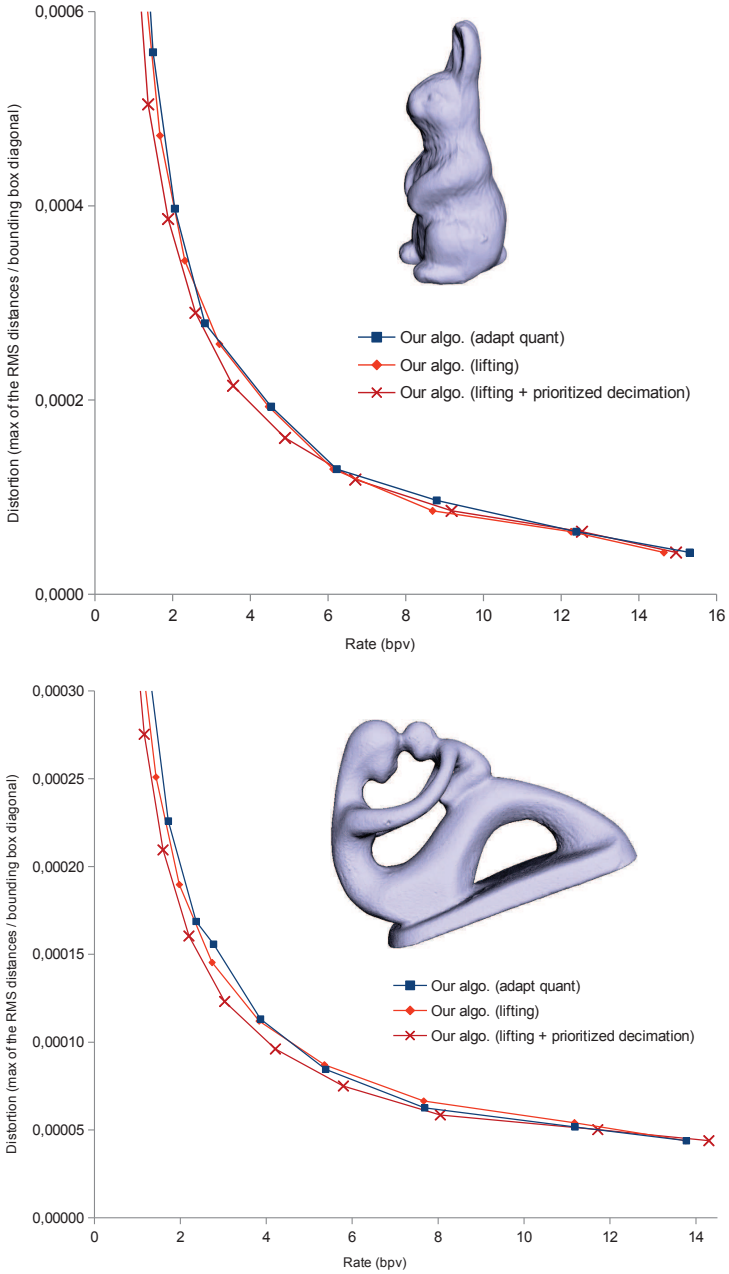


Figure 5.6: Rate-distortion curves for the compression of the Rabbit and Fertility models with 12 bits quantization.

Chapter 6

Remote visualization of progressive Meshes

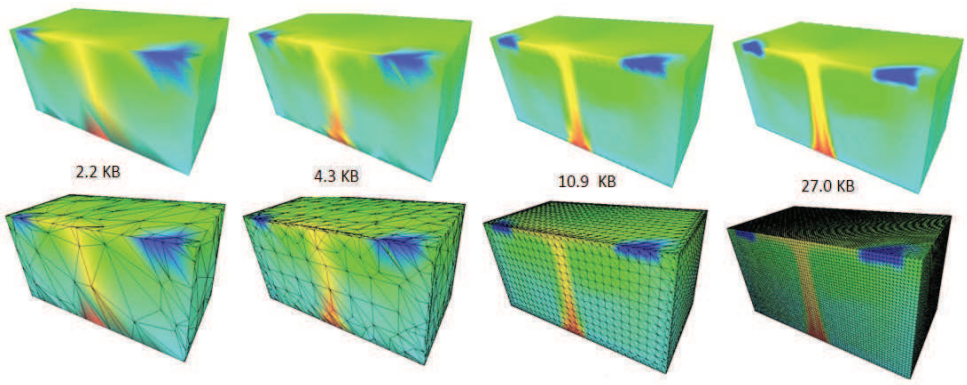


Figure 6.1: Progressive decompression of the *radiator* model. Its original X3D size is 953 KB (16002 vertices).

6.1 Introduction

In the Chapter 3, we presented a state-of-the-art on progressive mesh compression. In the Chapter 4, we described a new progressive mesh compression algorithm that can compress manifold polygon meshes. In these previous chapters, we claimed that the multiresolution structure embedded in data compressed with progressive algorithms is useful tool for 3D data **adaptation**. In this chapter, we demonstrate how to exploit this feature in the context of remote visualization of scientific data. This use case comes from the Collaviz project [Dupont et al., 2010], which aimed at building a remote collaborative platform for simulation applications.

As with other application fields using meshes, increasing needs for precision and quality in scientific simulation lead to an important increase in the complexity of 3D data sets. For instance, the number of mesh elements used in finite element resolution methods can now reach about one billion. Moreover, these data is nowadays produced in high performance computing centers and have to be analyzed by teams of scientists and engineers who are more and more often geographically distant. Nevertheless, even with the increase of available network bandwidth together with the available computational power resources, remote visualization of large 3D scientific datasets still suffers from several shortcomings among which the downloading time, the lack of interactivity and the management of different transmission, visualization and user constraints. Therefore, the idea of providing an efficient and fast remote access to large 3D scientific data

has motivated three related research and engineering areas in these twenty last years: 3D mesh streaming, 3D mesh compression and 3D data adaptation.

We present in this chapter a basic **streaming and adaptation framework** for remote scientific visualization of 3D meshes generated by finite element computation methods. The aim of this framework is to provide high quality intermediate colored meshes and efficient adaptation mechanisms dealing with different constraints (network bandwidth, display capabilities, etc.). We also propose to embed this framework into an extension of the X3D specification since it represents a well-known standardized format for 3D data exchange.

In the studied case, the targeted representations are surfacic. Color values representing scalar fields are linked to the vertices of the meshes. They are of prime importance for the visualization and the analysis. The meshes are encoded with the progressive compression algorithm described in [Lee et al., 2012] (see Section 3.2.2). Based on the valence-driven progressive approach of Alliez and Desbrun [Alliez and Desbrun, 2001a], it achieves a good rate-distortion performance compared with equivalent schemes. We chose to use this scheme rather than the progressive PPMC approach described in the Chapter 4 for two reasons. The first reason is that we designed this adaptation framework before PPMC. The second reason is that the approach of Lee et al. [Lee et al., 2012] was designed to efficiently compress meshes with color attributes at vertices (see Section 3.2.9) while our current implementation of PPMC cannot handle them.

The content of this chapter can be summarized as follows.

- First, we describe previous work on 3D streaming and adaptation, particularly in the context of the X3D file format.
- Then, we present the proposed **streaming and adaptation framework** and its integration to the X3D format.
- Finally, the utility of the approach is illustrated by some **experimental results** of the compression scheme and the full framework.

6.2 Previous work on 3D data streaming and adaptation

3D data streaming can be defined as the continuous and real-time delivery of 3D content over network connections in order to:

- enable user interactions without a full download of the data,
- provide a visual quality as if the data were stored locally.

3D data can be streamed over lossless or lossy network. In the case of a lossy network, various approaches have been proposed to handle data loss during the transmission of 3D mesh [Bischoff and Kobbelt, 2002, Chen et al., 2003, Cheng and Basu, 2007]. Data redundancy techniques and the influence of packet loss over the reconstructed data were studied for the different schemes. The application of remote scientific visualization do not tolerate data loss. Therefore, in the case of a lossy network, error correction schemes must be set up, as for instance in the TCP/IP protocol. The channel can then be considered as lossless.

Two kinds of approaches exist related to adaptive 3D streaming. On the one hand, work has been proposed to stream efficiently progressive meshes with a view-dependent way [Southern et al., 2001, Cheng and Ooi, 2008], by optimizing the rendering perception [Chen and Nishita, 2002] or packetization [Cheng et al., 2007a]. On the other hand, the rendering of progressive 3D models taking into account device capabilities and user perception have also been studied in [Funkhouser and Séquin, 1993, Gobbetti and Bouvier, 1999, Pasman and Jansen, 2002]. The framework of Tack et al. [Tack et al., 2005, Tack et al., 2006] optimizes the levels of detail of each object or part of object inside the 3D scene thanks to Pareto plots (error in function of rendering cost).

However, these two problems have not been much studied together, i.e. for the case when 3D models are rendered over networks. The framework built in [Jessl et al., 2005] delivers progressive meshes over networks

using subdivision-surface wavelets. Fixed thresholds, related to the transmission and the visualization, are used to drive the transmission. In [Schneider and Martin, 1999] the authors propose a client server architecture to deliver 3D model at interactive frame rates over networks. Their approach optimizes trade-offs between all the constraints (networks, device capabilities, user preferences). They use benchmarks to measure the environment characteristics.

Martin [Martin, 2000] describes a framework dedicated to the automatic selection of the best representation modality for objects in a scene. Their performance model takes into account, for each modality, the estimated delivery time, the quality of the representation and the degree of interaction it supports. In [Teler and Lischinski, 2001, Deb and Narayanan, 2004, Fathy et al., 2011], an adaptive framework for remote walkthrough applications is proposed. In [Ngoc et al., 2002], a QoS framework for the delivery of 3D content over network is built. The authors define a benefit, cost optimization problem solved by using heuristics. PSNR is used as the quality metric of rendered models.

X3D is an open software standard that describes a XML-based file format for the interactive 3D data storage and delivery. X3D is the successor to the Virtual Reality Modeling Language (VRML). It can be used for a board range of applications such as computer aided design, computational simulation, geographical information system or medical imaging. Mechanisms have been proposed in X3D to allow the efficient delivery of 3D data by selecting the best modality in function of the user viewpoint.

First, as it is not always needed to render the finest representation of an object, particularly when it is far from the user viewpoint, the X3D specification includes an adaptation mechanism called LOD [1]. This mechanism enables the manual specification, by the scene designer, of multiple alternatives (e.g. for instance the entire mesh in different level of resolutions) for an object together with a set of distance ranges indicating when alternatives have to be rendered. In [Pasman and Jansen, 2002], a similar idea has been proposed based on impostors, accuracy curves describing the minimal amount of resources required to reach an accuracy and a slight extension of the VRML node specification.

Nevertheless, these X3D/VRML specifications do not fit yet to the constraints and challenges of remote scientific visualization:

- The proposed specifications need the complete description of each alternative and thus does not enable progressive representation.
- The specifications do not define strategies related to the transmission and streaming part, i.e. intelligent transmission of the different alternatives.
- The level of detail techniques are more bounded to the 3D scene and not to the interaction of the user.

Related to the first issue, approaches based on a client-server architecture have been proposed [Fogel et al., 2001, Guéziec et al., 1998]. In [Guéziec et al., 1998], the authors propose their own progressive mesh representation where an external file, downloaded on demand, holds the level of detail data. [Fogel et al., 2001] extends the VRML/X3D language with two nodes: *progIndexedTriSet* which extends the node *IndexedFaceSet* and *ProgLOD* which holds the level of detail data (triangle index, coordinates, normal, etc.). An alternative is also the specification of URLs of external files which contain the different binary compressed representations of the data. Both approaches do not specify how to use the progressive representation for progressive download and rendering.

Related to the other issues, the basic transmission strategy consists in downloading the entire scene and in rendering it. In [Arikawa et al., 1996], this issue is tackled by the proposition of a dynamic LOD VRML extension and a client server architecture enabling user-dependent level of detail streaming and rendering. Nevertheless, this approach does not use progressive representation of meshes. Besides, it is much designed for remote walkthrough and not for remote scientific visualization.

As far as we know, no work have been proposed addressing the particular case of adaptive remote 3D scientific visualization.

[1] http://www.web3d.org/x3d/specifications/ISO-IEC-19775-1.2-X3D-AbstractSpecification/index.html

6.3 Adaptation and streaming framework with X3D

This section is dedicated to the description of our adaptation framework for 3D scientific visualization over networks. It uses a basic client-server architecture.

6.3.1 The adaptation parameters

Our adaptation framework takes into account constraints coming from the network, the device graphic capabilities and the user view-point and preferences. The algorithm aims at maintaining the following metrics below their threshold:

- $T_{dl}(M)$, the total time spent to download successively levels of detail of a mesh M. The download of new level is halted when $T_{dl}(M)$ overtakes T_{dlmax}.
- Q, the total quantity of memory taken by all the level of detail data. New level downloading is stopped when Q is above Q_{max}.
- T_r, the rendering time of an image which measures the difficulty for a device to render a 3D scene. If T_r is above the threshold T_{rmax}, then the scene complexity must be reduced.
- $F_{ED}(M_i, R, P)$, the element distinction metric, where i is the current level index of M. As the visualization devices have not the same resolution, our adaptation framework tries to find the best level of a mesh M to render for a given viewpoint P and a screen resolution R. F_{ED} represents the ability for a user to well distinguish the mesh cells. Rendering faces which sizes are below the size of one pixel of the screen is in most cases a waste of computing resources.

  We compute F_{ED} by randomly extracting a set of triangles of M_i (about 10%) and using the following formula:

  $$F_{ED}(M_i, R, P) = \frac{n_1}{n_2}$$

  where n_1 is the number of pixels of the lines when the triangles are rendered in line mode and n_2 is the number of pixels of the triangles when they are rendered in filled mode. These two renderings are done with no lighting and anti-aliasing filters and with uniform colors distinct from the background (see Figure 6.2).

Figure 6.2: Computation of metric F_{ED}. a. The object with its visualization attributes is shown. b. 10% of the triangles are rendered in filled mode. n_2 is obtained by counting gray pixels. c. The same triangles are rendered in line mode. n_1 is obtained by counting black pixels.

If F_{ED} is near 0, it means that the mesh elements can be well distinguished in the screen space. On the contrary, the more it is far from 0 the more the elements are small and cannot be easily distinguished. F_{ED} has to stay below F_{EDmax}.

The size of the randomly extracted triangle set is important. It sets the accuracy of the metric for the current viewpoint. Indeed, if all the faces are rendered with a depth test, only the visible faces are

taken into account in the metric computation. However, this is done at the expense of the computation time.

As the following section describes it, we benefit from the inactivity of the computational resources, when the user has found the viewpoint he is interested in, to compute this metric and ask, if needed, for refinements. In our current implementation, we extract 10% of the triangles to compute F_{ED}. For the highest levels, the computation time is approximately half of the rendering time.

6.3.2 The adaptation algorithm

Our adaptation algorithm is triggered by events. An event can be: an addition of a new mesh into the scene, a first rendering after a mesh refinement, a first rendering after a mesh coarsening, a changing visibility of one object or a changing view-point. Note that the adaptation algorithm is run when the user has found his interest view-point and does not move anymore. The adaptation algorithm computation time does not impact the frame rate during transient moves.

L is the set of the meshes from the scene that the user has selected to be visible. After each event the adaptation algorithm is started. Basically it chooses among these three choices:

- do nothing,
- if $T_r > T_{rmax} * (1 + r)$, **coarse** the mesh M of L with the highest $F_{ED}(M_i, R, P)$,
- if $T_r < T_{rmax}$, **refine** a mesh M of L given by the findMesh() function.

An hysteresis thresholding (introduced by the r parameter experimentally set) avoids jumps between levels because of T_r varying around T_{rmax}. It allows T_r to overtake T_{rmax} without spawning detail reduction.

The findMesh() function returns the mesh with the minimal $F_{ED}(M_i, R, P)$ which satisfies all the following constraints:

- $Q < Q_{max}$,
- $T_{dl}(M) < T_{dlmax}$,
- and $F_{ED}(M_i, R, P) < F_{EDmax}$.

The whole adaptation algorithm is written in pseudo-code in Figure 6.3.

```
if T_r < T_rmax then
  M = findMesh()
  if M exists then
    if M_{i+1} is not in memory then
      start downloading M_{i+1} and store it in memory
    end if
    replace M_i by M_{i+1}
  end if
else if T_r > T_rmax * (1 + r) then
  select the visible mesh M with the highest F_ED(M_i, R, P)
  replace M_i by M_{i-1}
end if
```

Figure 6.3: The adaptation algorithm. The first rendering after a mesh refining or coarsening triggers a new run of the adaptation procedure.

As the user may be interested in visualizing the highest level of detail of a certain mesh M of the scene, he can force the rendering of its highest level M_n even if it can not be achieved at an interactive frame rate.

In this case, all the levels are downloaded and M_n is rendered without taking into account the results of the previous adaptation algorithm.

6.3.3 Streaming levels of detail with X3D

At the client side, the resources in terms of network bandwidth, device capabilities (CPU, RAM, graphic computational power, etc.) are limited while the resources at the server side can be managed. Therefore, we use a client-server architecture to stream the progressive meshes.

The Figure 6.4 presents the compressed stream provided by the Lee et al. algorithm [Lee et al., 2012]. This stream is naturally decomposed into several parts, each standing for a certain level of detail and each containing connectivity (C), geometry (G) and color (Cl) information. The first part is the base mesh (usually several dozens of vertices) which is encoded using a standard mono-resolution compression technique; then each part of the stream, together with the already decompressed level, allows to build the next level of detail. At the end of the stream decoding the original object is retrieved.

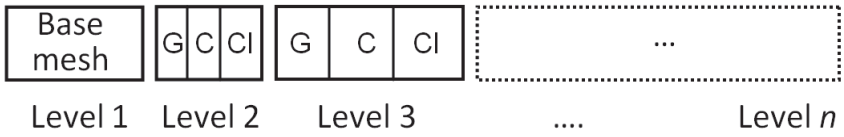


Figure 6.4: Format of the encoded stream. Each part of the stream contains geometry (G), connectivity (C) and color (Cl) data needed for mesh refinement.

We integrate our framework to the X3D language with the same approach as in [Fogel et al., 2001]. The 3D scene is described in a X3D file which contains a *ProgrTriSet* node for each object. *ProgrLOD* nodes receive the URL indicating where to download the level of detail data (see Figure 6.5). When a client requests a mesh, it receives this X3D file. Then, it can start downloading the levels according to the previously described adaptation strategy.

```
<ProgrTriSet>
    <ProgrLOD level="0" url="http://collaviz.org/lod0.ps" />
    <ProgrLOD level="1" url="http://collaviz.org/lod1.ps" />
    <ProgrLOD level="2" url="http://collaviz.org/lod2.ps" />
    <ProgrLOD level="3" url="http://collaviz.org/lod3.ps" />
    ...
</ProgrTriSet>
```

Figure 6.5: Integration of our progressive compressed remote mesh representation in X3D. '.ps' files correspond to the binary level of detail data generated by our progressive mesh encoding scheme.

Our client asks for new levels of detail using HTTP requests. The server responds by delivering the mesh level of detail data that has been already computed off-line by our progressive compression scheme. Once the client has received a new level, it restarts the adaptation procedure in order to maximize the user experience and maintain the metrics below their thresholds. Using the HTTP(S) protocol with its standard TCP/IP port, the proposed traffic can pass through most of the firewalls and proxy servers deployed in companies for their connection to the Internet.

6.4 Experiments

We divided our experiments in two parts. The first part reports compression results obtained by compressing scientific visualization meshes with the progressive coder embeded in our framework [Lee et al., 2012]. The second part relates experiments obtained with the complete streaming and adaptation framework. The experiments ran on a station with 2,8 Ghz processor and a NVIDIA Quadro FX 580.

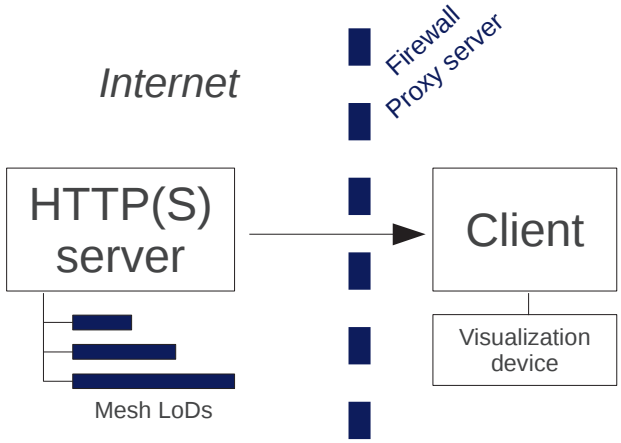


Figure 6.6: Our streaming architecture for progressive 3D meshes. Using the HTTP(S) protocol, we are able to pass through most of proxy servers and firewalls.

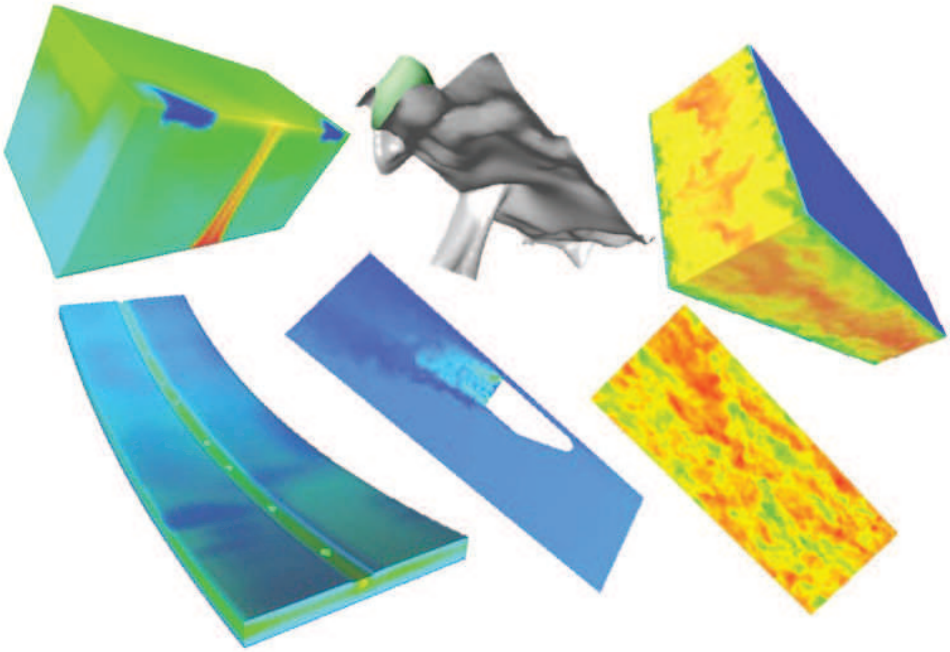


Figure 6.7: Our corpus of 3D models of scientific data. First row: *radiator*, *radiator_Iso* and *velocity*. Second row: *tank*, *vistcurb_CP* and *velocity_CP*.

6.4.1 Progressive encoding scheme

We tested the progressive compression method of Lee et al. [Lee et al., 2012] on several 3D models coming from post-processing (cutting-plane, iso-surfacing) applied on volumetric scientific simulation results (Fluid Dynamics and geophysics). These six models (see Figure 6.7) are X3D ASCII files and all contain color information on vertices except radiator_Iso.

The Table 6.1 presents respectively the original sizes of the models (X3D ASCII file format) and the sizes of the compressed streams corresponding to a lossless compression (all levels of detail) with 11 bits precision. The compression ratios are very good (between 2.75% and 4.51%). For comparison a binary encoding method

like GZIP provides compression ratios between 15% and 20%. The whole decompression times (all levels) are between 0.14 and 6,015 seconds, respectively for *vistcurb* (4.3Kvertices) and *tank* (123Kvertices) on a 2Ghz processor. The number of levels is automatically determined so as to obtain a base mesh of around a hundred vertices. It goes from 10 (*vistcurb*) to 26 (*tank*). This number of levels depends on both the number of vertices and the shape complexity of the model.

Name	Original (KB)	Compressed (KB)	Ratio
velocity_CP	7739	227.9	2.94%
velocity	2979	83.4	2.80%
radiator	953	27.0	2.83%
radiator_Iso	713	19.6	2.75%
vistcurb_CP	354	16.0	4.51%
tank	10065	266.0	2.64%

Table 6.1: Lossless compression results for several objects from scientific simulations with the algorithm from [Lee et al., 2012].

The Figure 6.1 illustrates several levels of detail from the progressive decoding of the *radiator* model; even after decoding only a very small amount of data (eg. 4.3 KB, corresponding to a decompression time of 30 ms) the resulting models are nevertheless visually correct allowing the client to be able to visualize a nice model even through a very low bandwidth channel.

These levels of detail also allow to adapt the resolution of the scene to the processing capacity of the display device. The Figure 6.8 illustrates several levels of detail of the *radiator_Iso* model. If only a small part of the stream is decoded then the resulting model owns only a small number of triangles allowing an efficient rendering even on low capacity devices. As it is shown in the next experiments, these properties combined to our adaptation algorithm will allow a very efficient streaming framework.

Figure 6.8: Progressive decompression of the *radiator_Iso* model.

6.4.2 Complete streaming and adaptation framework

We have tested our whole framework by simulating a remote scientific visualization session with a mesh coming from a real study case [Rupp I., 2008] of 460192 facets. All the level of detail data were stored on an HTTP server. Our adaptation framework successively downloaded and rendered them. We did two experiments which are described below.

Viewpoint adaptation. The first experiment consisted in studying the effects of our adaptation framework when the user changes the viewpoint. After each move, we waited for the data requested by our adaptation algorithm to be completely downloaded before taking snapshots of the new rendering. Then, we compared these snapshots with their equivalent taken with the full model by computing the PSNR. The results of this experiment are presented in the Table 6.2. Snapshots of the viewpoints used

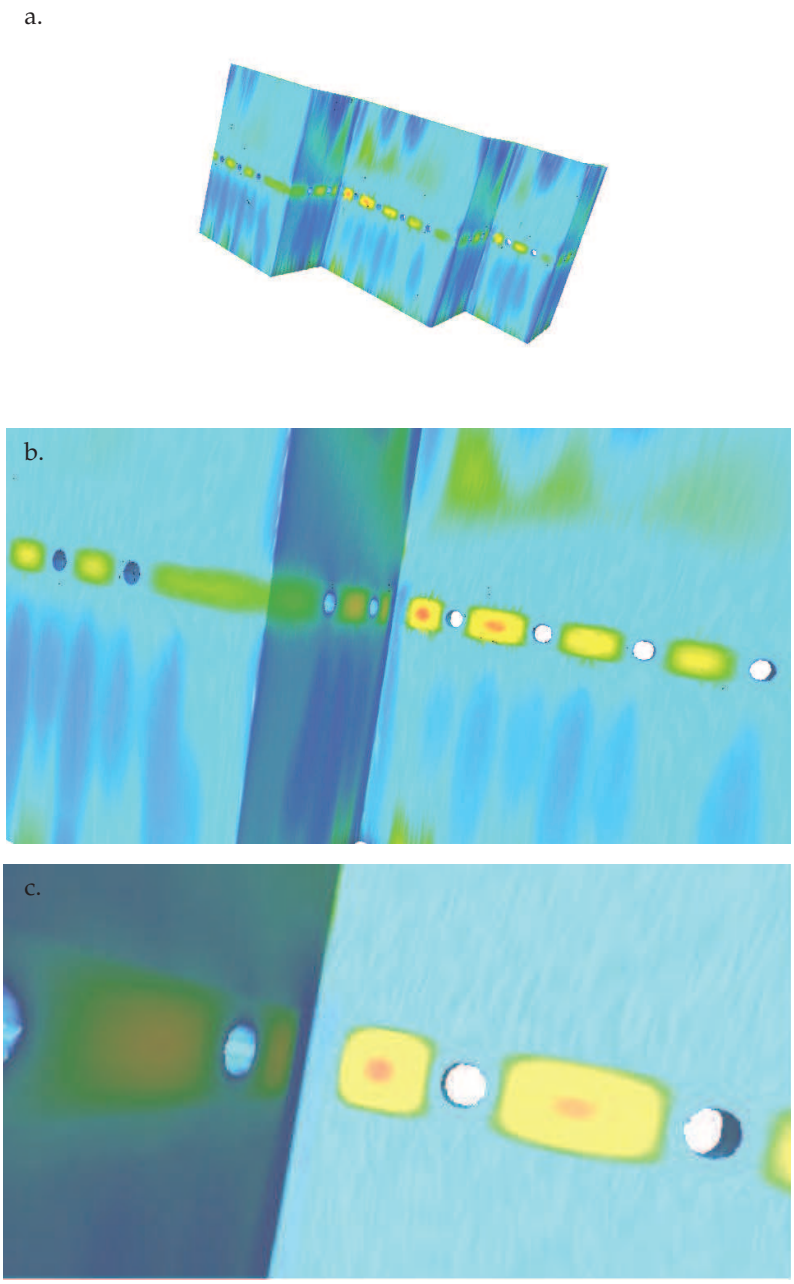


Figure 6.9: Snapshots taken during our test of progressive download and rendering of our model with different viewpoints.

during the experiment can be seen on Figure 6.9. We computed the F_{ED} metrics by extracting 10% of the triangles for each level. It took a maximum time of 110 ms for the finest level of detail.

Results demonstrate that our framework permits to achieve a good visualization quality and interactive frame rates for a global viewpoint by downloading a low amount of data (see Figure 6.9 *a*). Indeed, for this level of detail, only 27% of the data need to be downloaded and only 50000 vertices have to be displayed. Then when the user zooms on a specific part of the model, more data is downloaded and the level of detail increases.

Point of view (Figure 6.9)	*a.*	*b.*	*c.*
Level of detail	29/34	32/34	34/34
Nb of triangles	50412	150650	460192
% of downloaded data	26.6	52.1	100
T_r (ms)	45	230	266
PSNR (dB)	35.83	37.80	-

Table 6.2: Results of our test of progressive download and rendering of our model with different viewpoints. We set $F_{EDmax} = 1.8$, $T_{rmax} = 200$ and $r = 0,5$.

Resolution adaptation. The second experiment consisted in studying the effects of our adaptation framework when considering a single viewpoint (see Figure 6.10) but using three different screen resolutions: one of a mobile device (640x480), one of a standard notebook computer (1280x800) and one (1882x932) which can be reached on a powerful graphic station with a 24 inches monitor. For each screen resolution, our algorithm automatically adapted the level of detail of the 3D model. Like in the previous experiment we compared, using the PSNR, the snapshots obtained with the level of detail selected by the framework with the snapshots taken with the full mesh; the results are given in the Table 6.3 and attest the efficiency of our adaptation algorithm since the PSNR values remain quite high.

Resolution	640x480	1280x800	1882x932
Level of detail	30/34	32/34	33/34
Nb of triangles	70862	150650	238720
% of downloaded data	31.5	52.1	70.1
PSNR (dB)	35.61	40.64	44.13

Table 6.3: Results of our test of progressive download and rendering of our model with a common viewpoint and different screen resolutions. We set $F_{EDmax} = 1.8$, $T_{rmax} = 200$ and $r = 0,5$.

6.5 Conclusion

In this chapter, we have proposed an adaptive and progressive streaming framework dedicated to remote scientific visualization. Targeted data is surfacic representations of 3D meshes computed with finite elements methods. Due to the importance of the scalar fields linked to the vertices of the meshes for the visualization and the analysis, our framework integrates a progressive encoding method that handles colors [Lee et al., 2012]. Efficient adaptation mechanisms have been proposed to tackle different constraints: network bandwidth, device capability and user preferences. Based on previous works on X3D, the proposed framework is also X3D-compliant. Our first results on 3D scientific models show the relevance of the approach. We demonstrated how to take benefit from the multiresolution structure offered by progressive mesh compression algorithms in an applicative context of remote visualization. We therefore experimentally prove that progressive mesh compression is a powerful tool for **3D data adaptation** in a context of remote visualization.

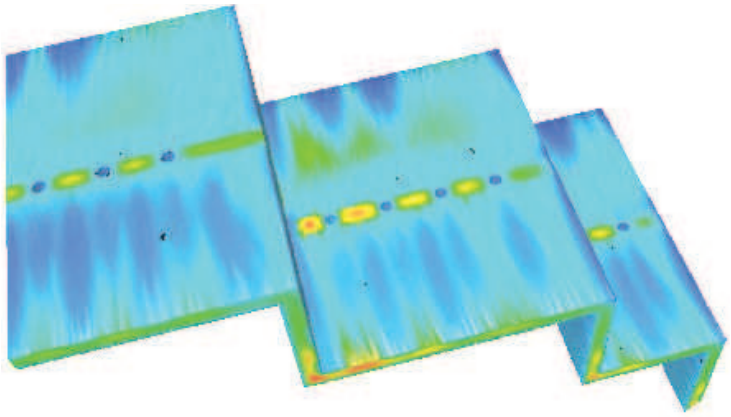


Figure 6.10: Snapshot of the viewpoint used during our test of progressive download and rendering of our model with different screen resolutions.

Nevertheless, the constraint modeling as well as the adaptation algorithm are basic. Thus, our element distinction metric F_{ED} relies on a simple concept. Its main advantage is that it can be quickly implemented with OpenGL. Yet, as it is based on the determination of an average number of pixels per rendered triangle, if the triangles of the current level of detail do not have an uniform size, this metric might lead to wrong adaptation choices. Furthermore this metric requires intense computations: the GPU has to perform the two rendering only useful for the metric. The generated images have to be transfered to main memory in order to count the pixels with the CPU. Better performance may be obtained by computing directly the metric on th GPU.

The last limitation of this approach comes from the nature of progressive mesh compression algorithms. If the user is interested in a precise region of the mesh at the highest level of detail as in Figure 6.9 c, all the levels of detail and therefore the full object must be downloaded and decompressed. Progressive compression algorithms are not very efficient in this case since the majority of the processed data is useless. Random-accessible mesh compression approaches are a solution to this issue: they allow to download to decompress only the requested regions of a mesh. In the following of this thesis, the Chapter 7 review existing work on random accessible mesh compression. The Chapters 8 and 9 propose two new random-accessible and progressive mesh compression algorithms, which are also suited to 3D mesh adaptation.

Chapter 7

State of the art on random accessible mesh compression

7.1 Introduction

The main idea developed in this thesis is that mesh compression is an efficient **adaptation** tool for the storage, the transmission and the visualization of meshes. In the Chapter 6, we demonstrated how the multiresolution nature of data generated by progressive mesh compression algorithms is interesting for adaptation purposes. They allow to display progressively refined models as more data is decoded and to choose the best level of detail in function of the viewpoint or the visualization device capabilities.

Progressive mesh compression techniques have, however, an important limitation. If the user wants only to visualize a particular *region of interest* of the mesh at the highest level of detail, the full model must be downloaded and decompressed. This can represent a significant waste of network and computing resources. Moreover, if the compressed model is too large, it may even not be possible to decompress and visualize it in low resource conditions.

Random accessible mesh compression techniques have been developed to address this issue. During the decompression the user can interactively select which part of the mesh he is interested in and the algorithm then decompresses just this region of interest. Yet, the user will not have any overview of the other regions of the mesh. He will not be able to easily locate an other region of interest.

Progressive random accessible mesh compression algorithms allow to decompress any part of the input mesh at any level of detail. The user can therefore refine more the regions he wants to visualize while seeing what the other parts of the mesh look like.

This chapter presents a state of the art on random-accessible mesh compression and progressive random-accessible mesh compression. The multiple modalities they offer during the decompression allow the adaptation of the 3D data to its efficient transmission and visualization, even in low resource conditions.

7.2 Random accessible compression

As seen in the Chapter 2, single-rate mesh compression algorithms mainly target the reduction of the mesh storage size. This is often achieved at the cost of inserting dependencies between the mesh elements. Consequently, if the user wants to access to a specific part of a mesh, he must wait for the decompression of the whole mesh. When the mesh is big, the compression and the decompression can be time and memory consuming. Some out-of-core approaches such as streaming mesh compression [Isenburg and Lindstrom, 2005,

Isenburg et al., 2005c] have a limited resource usage, but the data dependency is still persistent (see Section 2.4). The aim of random accessible mesh compression is to partially remove the dependencies between the mesh elements. Before the decompression, the user can select which regions of interest of the mesh he is interested in and the algorithm will decompress just these regions. Two paradigms were proposed in the literature: the cluster-based and the hierarchical representations.

Cluster-based random accessible compression

The algorithm of Choe et al. [Choe et al., 2004, Choe et al., 2009] first segments the input mesh using Lloyd's method [Lloyd, 1982]. Generated *charts* are desired to be planar and compact in order to achieve high compression ratios. The important characteristic of this codec is that each chart is is independently compressed using the single rate Angle Analyzer encoder [Lee et al., 2002]. In order to not duplicate the geometry information of the border vertices shared by two charts, their positions are compressed independently in sequences called *wires*. The position of the next wire vertex to encode is predicted as a linear combination of the two previously encoded positions. The connectivity between the clusters is encoded under the form of a polygonal mesh with the encoding scheme proposed by Khodakovsky et al. [Khodakovsky et al., 2002]. The Figure 7.1 illustrates this compression scheme. Experimental results show that this method has a small overhead that increases with the number of clusters.

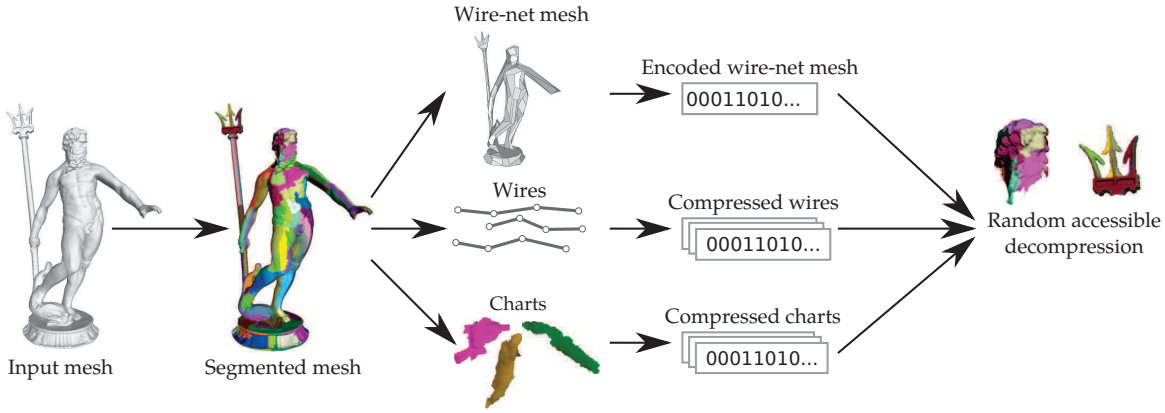


Figure 7.1: Cluster-based random-accessible mesh compression. The approach of Choe et al. [Choe et al., 2004, Choe et al., 2009] is illustrated in this figure.

Chen et al. [Chen and Georganas, 2008] used a segmentation algorithm that generates meaningful regions. Each cluster is then compressed with the Edgebreaker algorithm [Rossignac, 1999]. The difference here is that the boundaries between the clusters are triangle strips and not wires. These strips are also compressed by the Edgebreaker algorithm but no geometry information is embedded in the border data. The vertex positions are encoded inside each cluster.

Yoon and Lindstrom [Yoon and Lindstrom, 2007] also built a cluster-based random accessible compression scheme. They added the random-accessible support to streaming mesh compression [Isenburg et al., 2005c]. The mesh is compressed by sequentially accessing and grouping the mesh triangles and vertices in a cache-oblivious layout. Each cluster is composed of a constant number of triangles (a few thousand) and are compressed independently with the streaming mesh compression scheme. The geometry information of the border vertices is not duplicated: it is encoded one time in a cluster and referenced for the others. This scheme comes with a mesh access programming interface that allows to access any element of the mesh by their identifier at a low computational cost. Experimental results report 45:1 speedup over standard streaming mesh compression. However, the compression overhead is about 40% compared to the approach of Choe et al. [Choe et al., 2004].

The approach of Yoon and Lindstrom [Yoon and Lindstrom, 2007] was recently extended to support geometry random access by compressing bounding volume hierarchies composed of axis-aligned bounding boxes

[Kim et al., 2010]. In this hierarchy, a parent bounding volume is split into two children. The compression aims at preserving the cache coherency of the generated hierarchy. A set of bounding volume clusters that contains about four thousand bounding volumes are generated. These clusters are then compressed independently to enable the random access. The vertex indices of the triangles are encoded for each leaf bounding volume to enable the geometry random access to the mesh surface.

Hierarchical random accessible compression

Courbet and Hudelot proposed in [Courbet and Hudelot, 2009] a alternative hierarchical representation based on sequences of vertices also called wires. The input mesh, which contains an already encoded exterior boundary, is split into two balanced partitions. The start and end vertices of the border wire between the two clusters as well as its size are encoded and form the connectivity information. The wire geometry is encoded with the same linear predictor as in [Choe et al., 2009]. Both partitions are then recursively split in the same way until each partition contains only one polygon. This tree structure allows the decompression of only one of its paths. Therefore, the random accessibility granularity is high (see Figure 7.2). Besides, polygon meshes can be directly compressed. However, the compression efficiency for triangle meshes is inferior to the two previous approaches.

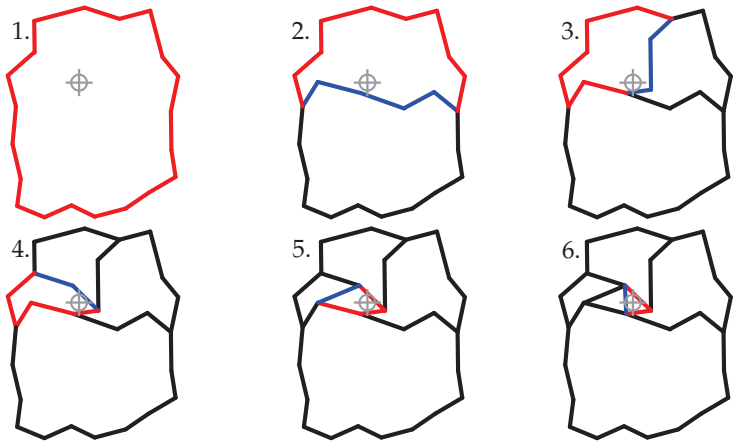


Figure 7.2: Random-accessible hierarchical decompression of a triangle mesh. This figure illustrates the approach of Courbet and Hudelot [Courbet and Hudelot, 2009]. The target points the requested part of the mesh. For each step, the current decompressed wire is in blue. It forms with the red wire, the cluster that is split in the next step.

7.3 Progressive and random accessible compression

As explained in the previous section, random accessible mesh compression schemes allow the decompression of only the required part of the mesh but do not provide any overview of the other parts. Progressive schemes allow to extract levels of detail during the decompression but the full mesh must be decompressed even if only a specific region of interest is required at the finest level of detail. These drawbacks are addressed by progressive and random accessible schemes, which allow the decompression of different parts of a mesh at different levels of details. Therefore, they embed a multiresolution random-accessible data structure.

In this section, we will firstly talk about the progressive random accessible compression approaches that restore the initial connectivity after the decompression. Then, we will present some schemes that resort to semi-regular remeshing.

7.3.1 Connectivity-preserving schemes

We distinguished two types of progressive random accessible connectivity-preserving compression schemes.

Connectivity-based algorithms

Kim et al. [Kim et al., 2006] based their multiresolution random accessible mesh compression algorithm on their previous mesh refinement framework [Kim and Lee, 2001]. During the decompression, a vertex can be split even if its neighbors are not the same than during the decimation. This breaks the symmetry of the operations of standard progressive mesh representation [Hoppe, 1996]. Thus, on the decompressed model finely refined regions can be adjacent to coarse regions. The connectivity and the geometry are compressed through an efficient encoding of the vertex split hierarchy. This approach is therefore connectivity-based. It offers a fine-grained multiresolution random access but the compression performance is limited (11 bits per vertex for the connectivity and 20 bits per vertex for the geometry with a 12-bit quantization).

Cheng et al. [Cheng et al., 2007b] described a progressive random accessible mesh compression algorithm based on an initial *meaningful* segmentation. The clustering is obtained by cutting the mesh into parts along concave feature contours. Each part of the mesh is then encoded with a modified version of the progressive encoder from [Alliez and Desbrun, 2001a] (see Section 3.2.2). The positions of inserted vertices are encoded with polar coordinates and different quantizations for each component. No attention is, however, provided to the part boundaries. The article does not tell if the border vertex information is duplicated. It does not neither describe how the holes between parts decompressed at different levels of detail are handled to output closed meshes.

Geometry-based algorithms

Better compression performances were obtained by geometry-based encoders. Jamin et al. [Jamin et al., 2009] and Du et al. [Du et al., 2009] proposed both to add random access support to the original Gandoin and Devilliers progressive algorithm [Gandoin and Devillers, 2002] (see Section 3.2.3). Both algorithms are out-of-core (see Section 2.4). During the decompression, different levels of detail can be selected for the different cells of the kd-tree. However, a post-processing step must handle the boundaries between cells decompressed at different levels of detail.

The CHuMI viewer from Jamin et al. [Jamin et al., 2009] partitions the mesh bounding box into a hierarchical structure called *nSP-tree*. This structure is composed of *SP-cells* encoded independently. A SP-cells must contain a minimal number of triangles to be split into child cells. The vertices that belong to several SP-cells are duplicated to allow independent decoding of each cell. Each SP-cell is subject to a Gandoin and Devilliers kd-tree decomposition [Gandoin and Devillers, 2002]. This decomposition stops when the minimal precision of the current SP-cell is reached. It encodes the connectivity and geometry information as in the original progressive algorithm. The authors also bring a solution to the problem of blocky effect at low rates due to the low quantization. Their scheme encodes some geometry bits in advance, in relation to the connectivity.

The approach of Du et al. [Du et al., 2009] decomposes the kd-tree of the original Gandoin and Devilliers algorithm [Gandoin and Devillers, 2002] into two layers. The top tree is the first layer while the set of its child subtrees are in the second layer. The top tree and each of the child sub-trees are compressed separately to enable the random access. During the decompression, the border vertices can be decompressed independently from the internal vertices. This feature is useful since there is a *prefix dependency* relationship between the subtrees. They must be compressed and decompressed in a predefined order. Thus, to fully decompress a subtree, the border vertices of the previous subtrees in the dependency list must be decompressed. These vertices can be later collapsed if they are not interesting to the user. As this scheme do not duplicate the border vertices, the compression performance should be higher compared to the CHuMI viewer [Jamin et al., 2009]. Yet, the prefix dependency implies that not desired data is decompressed, which is not the case with the CHuMI viewer.

7.3.2 Connectivity-oblivious schemes

As for single-rate and progressive mesh compression algorithms, resorting to remeshing for progressive and random accessible compression offers a new degree of freedom that allows to achieve better compression rates. These schemes are often based on wavelet decomposition framework for semi-regular meshes as their progressive counterparts (see Section 3.3.1).

In the algorithm of Liu and Zhang [Liu and bin Zhang, 2004], each independently compressed cluster corresponds to a triangle of the base mesh. 3 zero-trees, one for each edge, are associated to this base mesh triangle. Sim et al. [Sim et al., 2005] used the normal mesh representation [Guskov et al., 2000]. They describe their own scheme that assigns new edges to clusters after the butterfly subdivision. This framework also includes a distortion model, a visibility test method and a visibility priority method to view-dependently decompress the model. In [Roudet, 2010], one level of the wavelet decomposition of an input semi-regular mesh is used to segment the model into regions of homogeneous coefficient magnitudes. Each of these regions are later projected on the input semi-regular model to be later encoded separately. The framework presented by Gioia et al. [Gioia et al., 2004] allows to dynamically add and remove wavelet coefficients to refine or coarse the model without having to decode the whole decomposition.

Wavelet schemes have a key advantage for progressive random accessible mesh compression. No complex algorithms have to be developed to access randomly the original connectivity graph. The difficulty comes indeed from the fact that the connectivity must be compressed into independent clusters. For wavelet schemes, the only requirement is that the wavelet decomposition of one cluster must only depend on the vertices of this cluster. The connectivity is only set by the base mesh and the subdivision scheme. It has not to be encoded for each level of detail.

7.3.3 Data structures for view dependent visualization

The two previous sections describe progressive random accessible mesh compression algorithms. But progressive and randomly accessible compact mesh data structures were also studied in the context of visualization. Most of these structures are not compressed but they allow, like progressive random accessible mesh compression schemes, the rendering of huge meshes by dynamically coarsening and refining the mesh regions depending on the visualization view point. This section outlines the various propositions that have been developed.

Based on progressive mesh [Hoppe, 1996], the VDPM data structure [Hoppe, 1997] allows local refinements by encoding the dependency between vertex splits. An extension for huge terrain grid rendering that allows out-of-core processing and geomorph transitions was then proposed in [Hoppe, 1998]. Pajarola later described the FastMesh data structure [Pajarola, 2001, Pajarola and DeCoro, 2004] that requires less memory than VDPM. However, it is restricted to manifold meshes as it is based on an halfedge data structure. The method proposed by Yoon et al. [Yoon et al., 2004] first clusters the input mesh and then builds a progressive mesh representation for each cluster. Guthe et al. [Guthe et al., 2003] used a geometry octree to partition the mesh. Each node of the octree is simplified with a bottom-up approach. The approach of Shaffer and Garland [Shaffer and Garland, 2005] also decomposes the mesh with an octree. It then assigns to each node a representative vertex computed to minimize the approximation error. Cignoni et al. [Cignoni et al., 2004] proposed to decompose the mesh with a tetrahedral structure. Other methods were also proposed to build parallel view dependent progressive meshes [Hu et al., 2009, Derzapf and Guthe, 2012].

7.4 Conclusion

Random accessible mesh compression algorithms can efficiently **adapt** 3D data to transmission and visualization constraints. Contrary to single-rate and progressive mesh compression schemes, they do not require the full data to be decompressed to provide an access to a specific part of the mesh. Compared to single-rate and progressive mesh compression, few approaches have been proposed for random-accessible and progressive random-accessible mesh compression.

For single-rate random accessible mesh compression, two paradigms were proposed. The first relies on an initial segmentation of the input mesh. The generated clusters are then encoded independently. The second relies on a hierarchical decomposition of the input mesh that compresses the geometry and the connectivity through wires.

For progressive random accessible mesh compression, the wavelet-based schemes have shown their high rate-distortion performance. However, as these methods rely on semi-regular remeshing, they do not recover the initial connectivity. Connectivity-preserving schemes guided either by the mesh connectivity or the geometry have also been proposed for the application that do not tolerate remeshing.

We summarized in the Table 7.1 and the Table 7.2 what we judged as the main random accessible and progressive random accessible compression approaches.

Algorithm	Lossless connect. comp.	Total comp. rates (bpv)	Compress non-manifold meshes	Random access granularity	Remarks
Streaming random accessible [Yoon and Lindstrom, 2007]	yes	28 (12 bit)	no	2/5	Preserve the streaming layout
Hierarchical compression [Courbet and Hudelot, 2009]	yes	20 (12 bit)	no	5/5	
Chart-based compression [Choe et al., 2009]	yes	16 (12 bit)	no	3/5	

Table 7.1: Summary of the main random accessible mesh compression algorithms. Approaches are approximately ranked by their compression performance.

Algorithm	Lossless connect. comp.	Total comp. rates (bpv)	Compress non-manifold meshes	Random access granularity	Remarks
Dependency free vertex splits [Kim et al., 2006]	yes	31 (12 bit)	no	5/5	
Kd-tree cell compression [Jamin et al., 2009]	yes	21 (16 bit)	yes	3/5	
Layered Kd-tree compression [Du et al., 2009]	yes	17 (16 bit)	yes	3/5	
Wavelet compression [Liu and bin Zhang, 2004]	no	?	no	2/5	Fits well to smooth and dense meshes
Normal mesh compression [Sim et al., 2005]	no	?	no	2/5	Like above

Table 7.2: Summary of the main progressive random accessible mesh compression algorithms. Approaches are approximately ranked by their compression performance.

The recovery of the input mesh connectivity with a random access is a complex problem. To achieve it at competitive compression rates is still more difficult. When a new random-accessible compression scheme is designed, a trade-off must be made between the offered random-access granularity and the compression performance. Seeing the few work proposed, we think that there is still space for alternative approaches with alternative trade-offs.

Consequently, in the Chapter 8 and Chapter 9, we propose two new progressive random accessible approaches. The first scheme is based on an initial segmentation of the input mesh. Each generated cluster is then compressed independently by a state-of-the-art progressive mesh algorithm. A particular attention is provided to the boundaries to not duplicate information in the compressed data and produce mesh with no hole at decompression time. The second scheme requires no initial segmentation. The random-accessibility is permitted thanks to clustered encoding of refining operation and the requirement of one level of detail of

difference between adjacent clusters. This approach directly generates one piece decompressed models. It does not need any post-processing steps to stitch adjacent clusters decompressed at different levels of detail.

Chapter 8

Progressive and random accessible mesh compression based on mesh segmentation

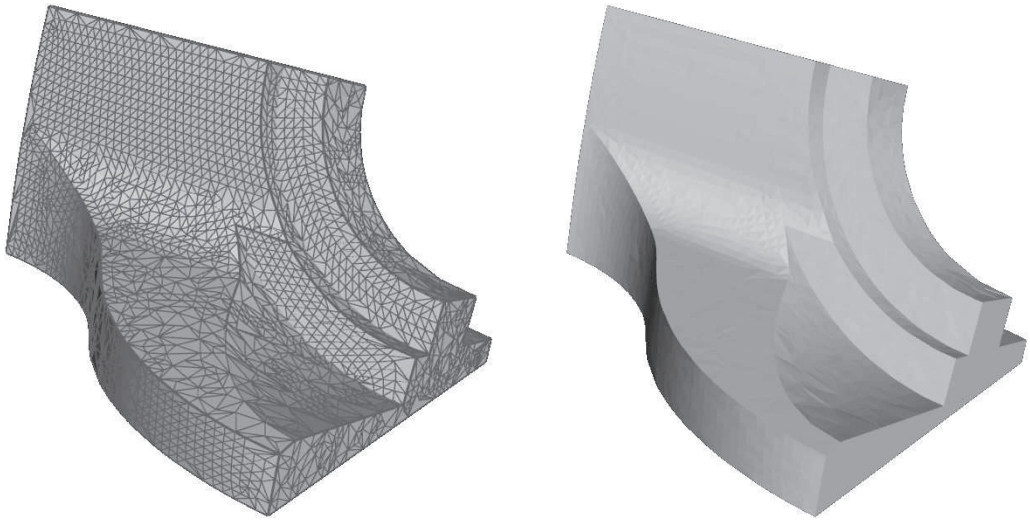


Figure 8.1: Example of a view-dependent decompression of the fandisk model. The side parts of the mesh are decompressed at a higher level of detail compared to the front parts.

8.1 Introduction

We showed in the Chapter 7 that the progressive random accessible mesh compression schemes offer great features for 3D data **adaptation**. In a context of remote visualization, downloading, decompressing and rendering only the required regions of the model while having an overview of the other regions can greatly improve the interactive experience of the user. Less memory and computing resources are needed to provide an access to the data. The network bandwidth is also saved as only the required data is downloaded.

We believe that, among the different types of mesh compression algorithms (single-rate, progressive, random accessible and progressive random accessible), progressive random accessible approaches are the most interesting for 3D data adaptation. Contrary to single-rate and progressive approaches, they do not require the full model to be decompressed to access a specific part of the mesh at the highest level of detail. Moreover, contrary to random accessible approaches, they can refine more or less different parts of the mesh.

The aim of this chapter is to propose a new progressive random accessible algorithm tailored to view-dependent decompression of manifold triangular meshes. If we look at the example of Figure 8.2, it seems obvious that, with this viewpoint, the front face of the model should be more refined that its side faces.

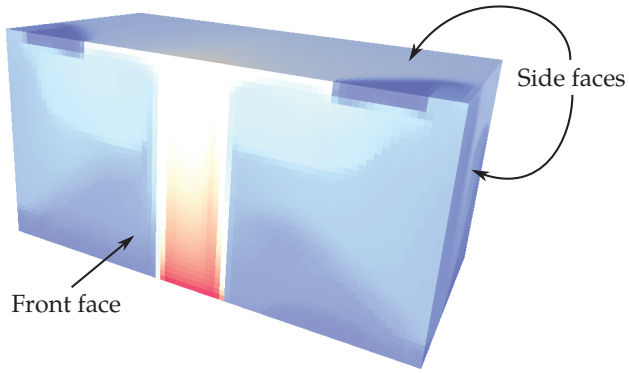


Figure 8.2: Visualization viewpoint of the radiator model. The front face should be more refined than the side faces.

Our new scheme is called **PRAM** for Progressive Random Accessible Mesh. It is based on a preliminary segmentation of the input model. According to the review of Shamir about mesh segmentation [Shamir, 2008], there is two main types of approaches. *Part-type* segmentation aims at partitioning the object defined by the mesh into *semantic* or *meaningful* parts. *Surface-type* segmentation builds on the mesh surface patches based on geometrical properties like normal or curvature. The proposed segmentation algorithms of PRAM belong to the second group. We indeed wanted random-accessibility regions to be suited to the view-dependent visualization. It seems intuitive from Figure 8.2 that a good segmentation should produce clusters of faces that have approximately the same normal.

PRAM is based on the single-rate random-accessible mesh compression scheme from Choe et al. [Choe et al., 2009]. The input mesh is first segmented with one of the two proposed approaches. Then, each of the generated clusters, called *charts*, is compressed independently. We replaced the single-rate chart encoder with the progressive algorithm from [Lee et al., 2012]. As for our adaptation framework for remote scientific visualization presented in the Chapter 6, we chose this algorithm rather than our own progressive polygonal approach PPMC (see Chapter 4) for two reasons. The first reason is that the approach of Lee et al. [Lee et al., 2012] can efficiently compress vertex colors, which is particularly useful for applications such as remote scientific visualization. The second reason is that we designed PRAM before PPMC.

In order to prevent duplication of the geometry and color information of the chart border vertices, they are compressed independently through *wires*. Therefore, we modified the progressive mesh encoder from [Lee et al., 2012] to get the position and color of the chart border vertices from the wire data. The connectivity between the charts is encoded as a polygonal mesh where each face corresponds to one cluster. This mesh is called the *wire-net mesh* as each of its edges has a corresponding wire. It can also be seen as a very coarsely remeshed version of the input model. PRAM allows then to decompress any chart at any level of detail. A post processing algorithm stitches the adjacent charts together to obtain a closed decompressed model. The Figure 8.3 illustrates the global scheme.

The content of this chapter can be summarized as follows.

- We first describe the original **segmentation** approach from [Choe et al., 2009] based on a **variational shape approximation method**. We list additional segmentation topological problems that may appear with some models. We explain how to fix them.
- As an alternative solution, we propose a second **segmentation** approach based on the **polygonal mesh decimation** algorithm described in Chapter 5. This second method produces segmentations with much less topological problems.
- We then detail the **compression** scheme and provide some experimental results.

- Finally, we present our **view-dependent decompression** framework.

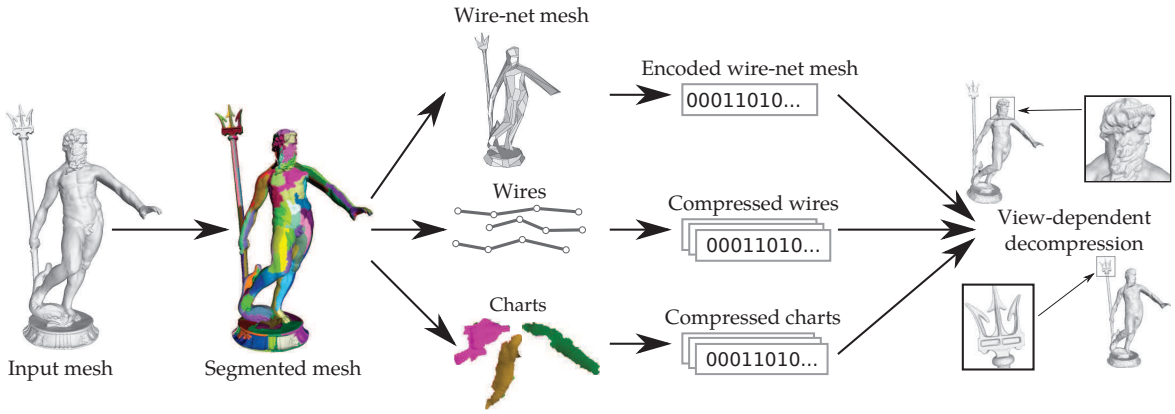


Figure 8.3: Overview of our progressive and randomly accessible compression scheme.

8.2 Variational segmentation

To be able to compress charts independently, the input mesh must first be partitioned. The segmentation approach, originally described in the article of Choe et al. [Choe et al., 2009], is similar to the variational shape approximation of Cohen-Steiner et al. [Cohen-Steiner et al., 2004]. It is based on Lyod's clustering algorithm [Lloyd, 1982]. In this section, we first describe the segmentation algorithm before talking about the potentially generated topological problems and presenting some experimental segmentation results.

8.2.1 Principle

The segmentation is performed by a region-growing algorithm that takes as input parameter the number of charts to generate. It first randomly selects one seed face per chart. At each iteration, it tries to connect to each chart one of its adjacent faces. The chosen face is the one that has the lowest connecting cost presented below.

The aim of the segmentation is to generate charts that are planar, compact and that have approximatively the same number of triangles. The cost of the connection of a face f to a chart c is defined by

$$F(f;c) = \underbrace{\left(\frac{\#f_c}{\#f_{avg}}\right)}_{1} \underbrace{(\lambda - (N_c \cdot N_f))}_{2} \underbrace{(|P_f - P_n|)}_{3},$$

where $\#f_c$ is the current number of faces of c, $\#f_{avg}$ is the current average number of faces per charts, $N_c \cdot N_f$ is the scalar product between the average normal of the faces of c and the normal of f, $|P_f - P_n|$ represents the geodesic distance between the barycenter of f and the barycenter of the face n adjacent to f and belonging to c. The first term of F aims at uniforming the number of faces per chart to guarantee a bounded I/O time for arbitrary random access. The second term ensures that the generated charts are planar and the third term that they are compact. λ is a parameter that regulates the relative influence of the compactness and planarity terms. In our experiments, it is set to 1.

Once all the faces have been assigned to one chart, the chart centroids are selected as the new seed faces. A new iteration of the segmentation algorithm is then started. At the end of this iteration, we obtain the final mesh segmentation. More iterations could be performed as in [Cohen-Steiner et al., 2004] but we want the segmentation to be quick to minimize the total compression time. The segmentation quality is

very important as it directly affects the final compression ratio. The more regular and flat the charts are, the better the compression ratio will be. In practive, two iterations are enough to produce a satisfying segmentation.

To allow an efficient compression of the wire data, the boundaries between the charts are smoothed at the end of the segmentation thanks to the following rule. A triangle face that is adjacent to two faces belonging to the same chart belongs also to this chart. This process is illustrated on the Figure 8.4.

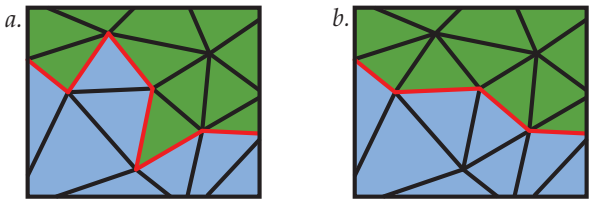


Figure 8.4: Smoothing the border between two charts. a. Before the smoothing. b. After the smoothing. A triangle face that is adjacent to two faces belonging to the same chart belongs also to this chart.

8.2.2 Topological problem handling

The connectivity between the charts is encoded as a polygonal mesh: the wire-net mesh. The previously described segmentation algorithm may generate particular connectivities invalid to be converted into meshes. We automated the detection and the fixing of these problems. We made an other classification of the possible issues than the one of Choe et al. [Choe et al., 2009]. We also discovered a new case. If at the end of the segmentation the following cases are encountered, the original segmentation must be modified:

- A donut shaped chart, a chart that fully surrounds one or several charts (see Figure 8.5 *a*, chart 1).
- Two charts sharing two distinct boundaries (see Figure 8.5 *b*, charts 1 and 5)
- A border chart with only one neighboring chart (see Figure 8.5 *c*, chart 2). This case was not cited in [Choe et al., 2009] but can happen when compressing a mesh with boundaries.

These invalid segmentation topologies are therefore modified as described below and illustrated on the bottom line of the Figure 8.5.

- To fix a donut shaped chart, we divide it into three charts. A region-growing algorithm starts with three faces regularly distributed on one border of the donut shaped chart. At each iteration, an adjacent face is assigned to each of the three sub-charts until no free face is left.
- To fix two charts that share two distinct boundaries, we split one of the chart with the same iterative algorithm.
- To fix a border chart with only one neighboring chart, we split the chart connected to the problematic chart.

In some case, the used iterative conquest algorithm creates other problematic configurations. This is why, the generated sub-charts have also to be checked and fixed if needed. Meshes with a high genus can be problematic as their segmentation can generate many donut shaped charts, which must be split several times to obtain a correct topology. Moreover, in the case of the original mesh containing holes, the segmentation can create non-manifold charts. As the chart compression algorithm of Lee et al. [Lee et al., 2012] is restricted to handling manifold meshes, some faces must be connected to another neighboring chart to create a fan, thus making the chart manifold. More generally, if the input model contains too much of such topologies, the fixing algorithms may fail to generate a proper segmentation.

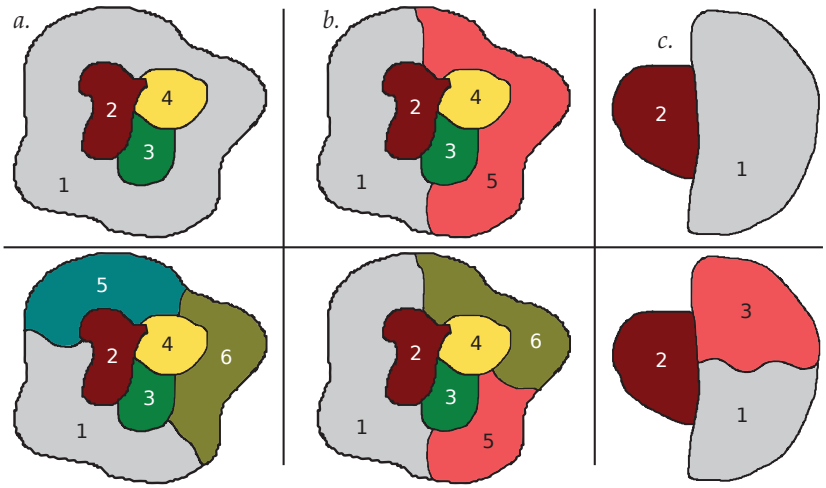


Figure 8.5: Topological problems generated by the segmentation (top row) and how to fix them (bottom row). *a.* The chart 1 fully surrounds the charts 2, 3 and 4. This case is fixed by splitting the chart 1 into 3 charts. *b.* The charts 1 and 5 shares two distinct boundaries. This case is fixed by splitting the chart 5 into 2 charts. *c.* The chart 2 has only one neighbor chart. This case is fixed by splitting the chart 1 into 2 charts.

8.2.3 Results

The Figure 8.6 shows examples of obtained segmentations. The objectives of generating charts that are planar, compact and that have the same number of faces seems to be respected. The Tank model (see Figure 8.6 *d*) illustrates that where the mesh has small handles (here the drilling at the middle of the model), charts need to be split to produce a topologically valid wire-net mesh. This explains why there is a high number of charts with irregular shapes around the drillings.

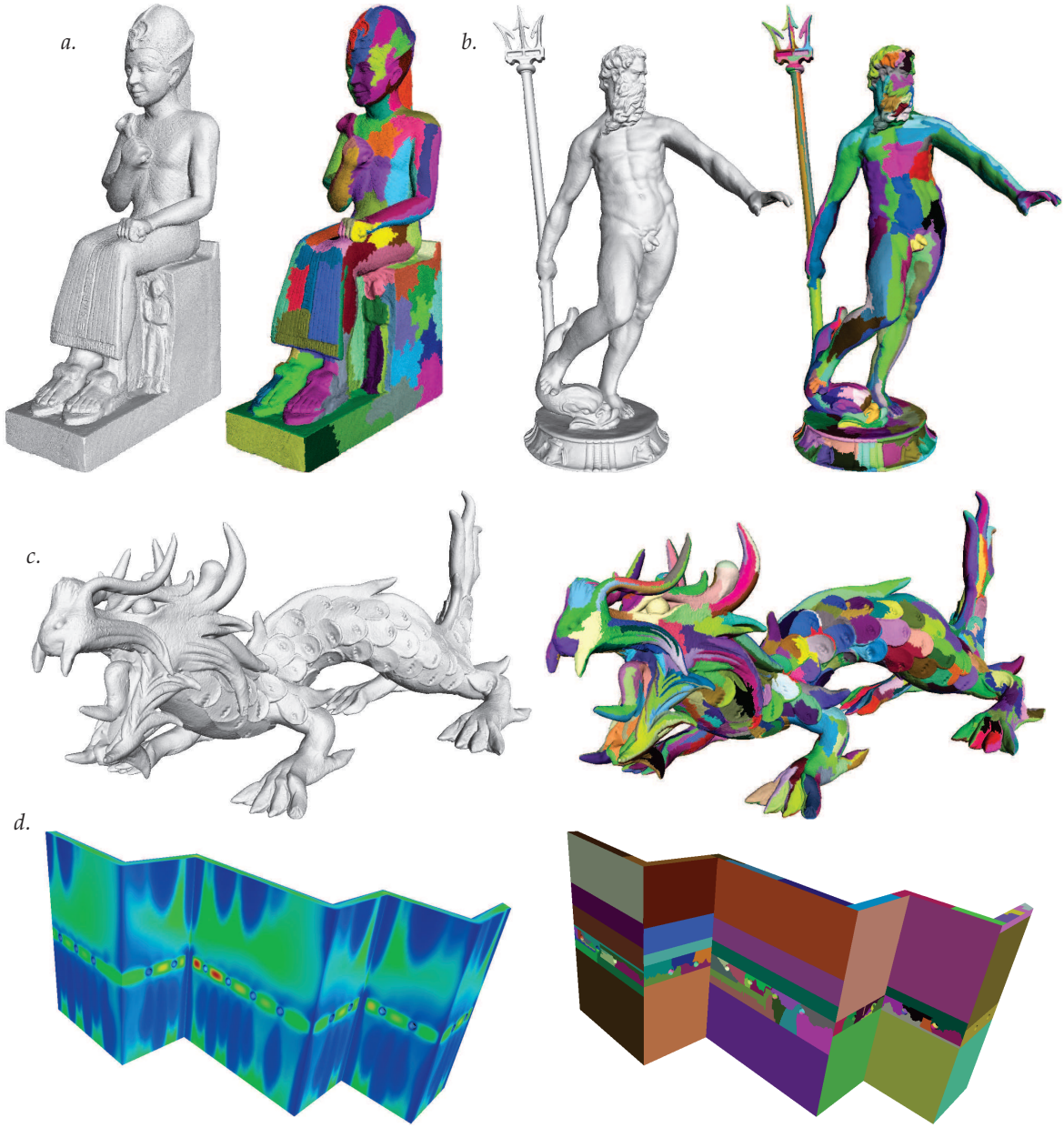


Figure 8.6: Segmentations obtained with the variational shape approximation method. *a.* The Ramesses model with 166 charts. *b.* The Neptune model with 406 charts. *c.* The Dragon model with 725 charts. The Tank model with 142 parts.

8.3 Decimation-based segmentation

The second segmentation method we propose in this section is based on the polygon mesh simplification algorithm described in Chapter 5. This algorithm is based on two incremental decimation operators: the halfedge collapse and the edge removal.

8.3.1 Getting the segmentation from the decimation

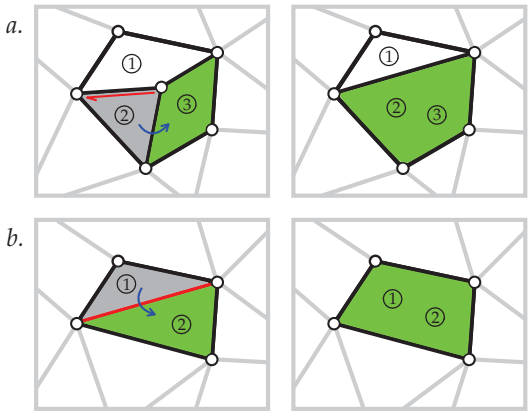


Figure 8.7: Merging faces for the decimation-based segmentation. *a*. After an halfedge collapse. *b*. After an edge removal. The encircled numbers represent the ids of the faces stored inside the list of each face.

At the beginning of the decimation, the algorithm assigns to each face an id. During the decimation, the input mesh faces progressively merge into adjacent faces thanks to the halfedge collapse and edge removal operations. The algorithm keeps in memory, for each face, a list containing its id and all the ids of the faces that are merged into it.

- When an halfedge collapses, the triangle faces adjacent to collapsed edge vanish. The id list of these faces are transferred to the id list of their adjacent faces belonging to the patch. This process is depicted on Figure 8.7 *a*.
- When an edge is removed, the id list of the disappearing face is transfered to the id list of the remaining face of the patch as depicted on Figure 8.7 *b*.

At the end of the decimation, each face of the input model has therefore a corresponding face in the decimated mesh. The segmentation is obtained by creating charts with faces that have a common corresponding face in the decimated model.

It is still possible that the generated segmentation contains the same topological problems as described in Section 8.2.2. In practice this happens much more rarely than with the variational segmentation method. Indeed, the constraint of generating only convex faces can prevent the formation of donut-shaped charts or two charts sharing two distinct boundaries.

8.3.2 Results

On the Figure 8.8, some decimations and corresponding segmentation results are shown. The quality of the obtained segmentation is inferior to the results of the variational method. The number of faces per chart is not constant as the method is not designed to respect this constraint. Charts are also less compact and planar. As reported in Section 8.4.4, this leads to inferior compression rates. Nevertheless, one advantage of this method over the previous one is that it produces less topological problems.

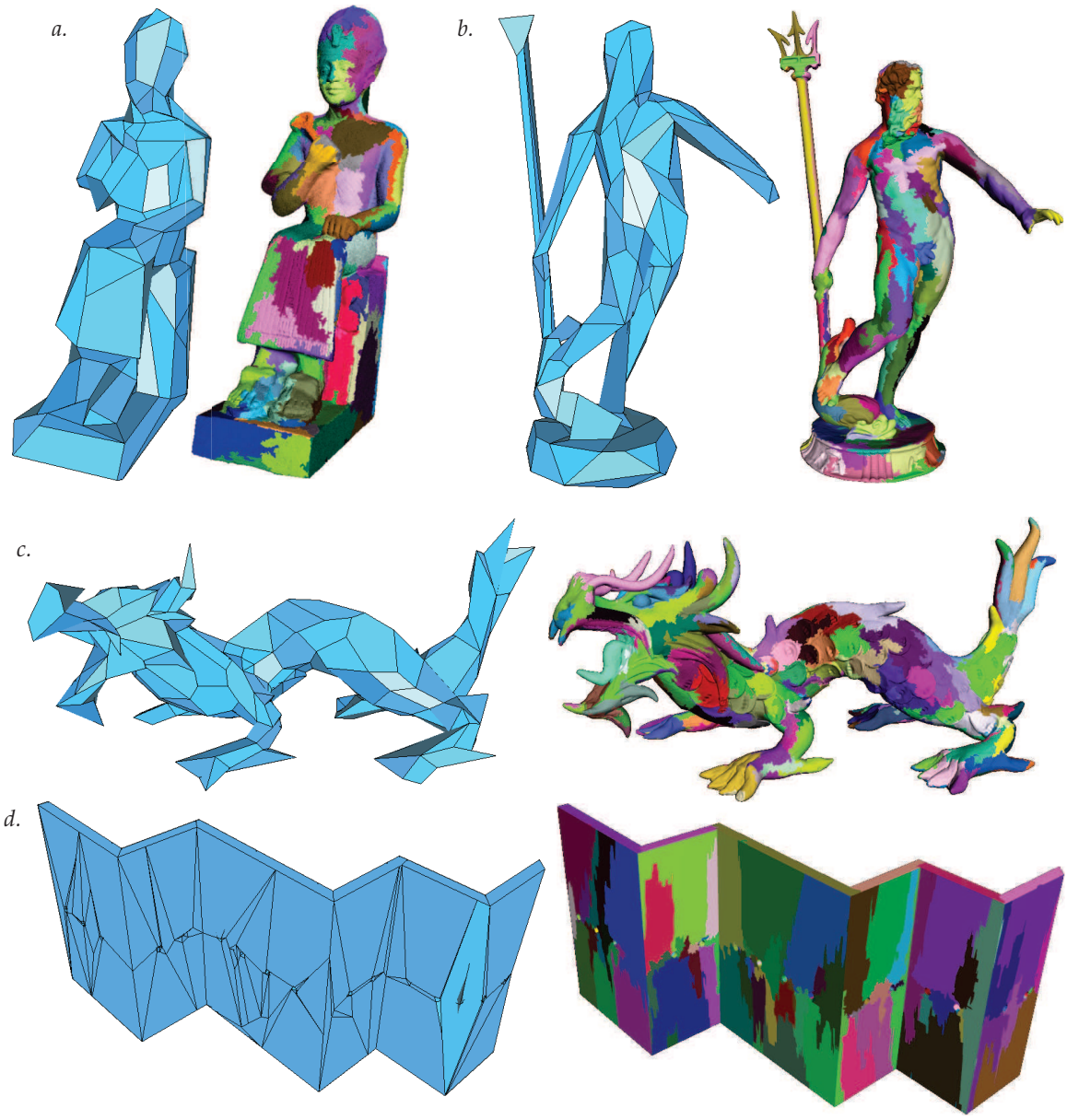


Figure 8.8: Wire-net mesh and segmentations obtained with the incremental polygonal decimation method. *a.* The Ramesses model with 168 charts. *b.* The Neptune model with 204 charts. *c.* The Dragon model with 362 charts. The Tank model with 257 parts.

8.4 Compression

This section now describes how PRAM compresses the different elements of the input model to enable the decompression of any chart at any level of detail.

8.4.1 Wire-net mesh encoding

The original approach of Choe et al [Choe et al., 2009] compresses the wire-net mesh with a single-rate polygon mesh compressor [Khodakovsky et al., 2002]. PRAM simply encodes it in an indexed data structure (see Section 1.3.1) to ease the implementation. In our experiments, the wire-net mesh represents always less than 0.7 percent of the whole compressed data.

8.4.2 Wire compression

Chart border vertices are encoded independently because they are shared between two charts. This avoids the significant duplication of geometry and color information in the compressed data (see Section 8.4.4). Wires are compressed as a sequence of vertices as described in [Choe et al., 2009]. The positions of the first vertex and last vertex are encoded in the wire-net mesh. For the other vertices, a prediction is made and the residual is encoded. The position of the second vertex is predicted as being the same as the position of the first vertex. Else, the position of a vertex i is predicted as a linear combination of the position of the two previous vertex in the wire:

$$PredPos(i) = 2 \times Pos(i-1) - Pos(i-2). \tag{8.1}$$

This process is illustrated on Figure 8.9. The color of the wire vertices is encoded in the same way. All the prediction errors are encoded with a range coder that uses one static model for all the wires. The probability table is generated with the data for all the wires and it is stored with the compressed data.

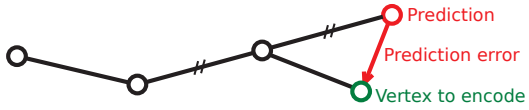


Figure 8.9: Wire compression.

8.4.3 Chart compression

Concerning the chart compression, we reused the progressive mesh compression scheme that can handle vertex colors from [Lee et al., 2012], which is based on [Alliez and Desbrun, 2001a]. This algorithm is described in

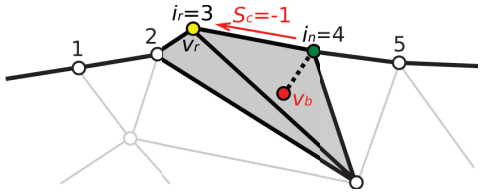


Figure 8.10: Encoding chart border vertices. The numbers are the vertex indices in the wire. The patch border vertex to encode (yellow) is predicted (green) as being the vertex of the wire that is the closest to the patch barycenter v_b (red). The difference s_c between the index of the predicted vertex (green) and the real vertex (yellow) is encoded.

Section 3.2.2. PRAM therefore benefits from the good rate-distortion performance of this algorithm. We modified the compressor and decompressor to integrate it in our randomly-accessible framework.

- We prevent wire-net mes vertices from being removed during the decimation passes in order to attribute easily wires to chart boundaries after the decompression.
- The decimation conquest of the algorithm of Lee et al. [Lee et al., 2012] only removes valence 3 and 4 border vertices. In order to not duplicate the border vertices geometry data for adjacent charts, we created two new connectivity symbols to encode the border vertices belonging to a wire: one for border valence 3 vertices and one for valence 4 vertices. These symbols are followed by another symbol that allows to find the index i_r of v_r in the wire. This symbol is determined by first computing the current patch barycenter v_b. Then, the index i_n of the vertex in the wire that is the closest from v_b is computed. The encoded symbol s_c, corresponds to $i_r - i_n$ (see Figure 8.10). As it can be seen on Figure 8.11, for most cases s_c is null. Therefore, it can be efficiently entropy encoded. The achieved gains of the compression of border vertices in wires can be found in the Table 8.1 (normal vs no wire columns).

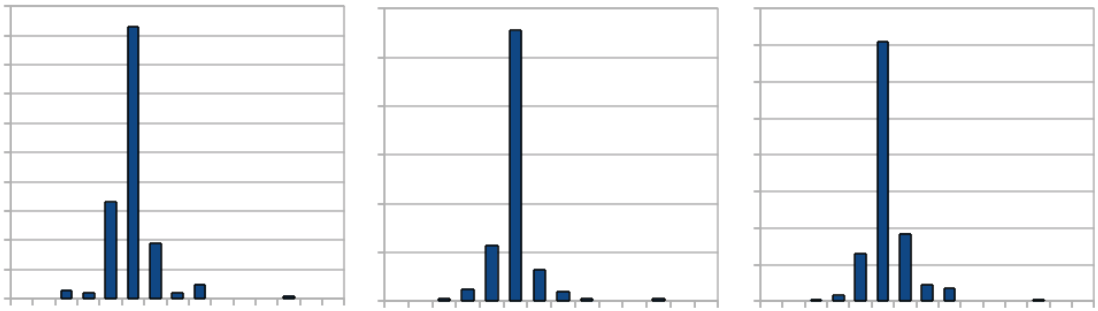


Figure 8.11: Typical distributions of s_c.

8.4.4 Experimental compression results

We implemented PRAM using the halfedge data structures from OpenMesh [Botsch et al., 2002] and CGAL [Flato et al., 1999]. The Table 8.1 and 8.2 report experimental compression results obtained with both segmentation methods: the variational segmentation and the decimation-based segmentation. The experiment ran on a desktop computer with an Intel Core i7 processor at 2.80 GHz with 8 GB of RAM. In this section we comment the results and compare our approach with previous compression schemes.

Compression rates

The obtained compression rates of PRAM highly depend on the number of charts. Generally, the bigger the charts are, the better the compression is, as shown with the Ramesses example in the Table 8.1. To demonstrate the utility of the compression of chart border vertices inside wires, we provide in the Table 8.1 a compression rate with the border vertices duplicated inside each chart. This corresponds to the 'no wire' column. We also provide a compression rate with geometry constraints enabled to facilitate the chart stitching as explained in Section 8.5.2.

In the Table 8.2, compression rates obtained with the two presented segmentation methods are compared. We set up each method to generate approximately the same number of charts. The inferior segmentation quality of the decimation-based method is confirmed here by the inferior compression rates. Consequently, we recommend to use this method only if the input model has a complex topology (high genus, holes). To compare PRAM with other compression schemes, we will always refer to results obtained with the variational segmentation.

Comparing the compression rates of several random accessible mesh compression schemes is not a trivial task as all the methods do not provide the same random accessibility. With about 20.000 faces per chart, the compressed data produced by PRAM are twice smaller than with the approach described in [Kim et al., 2006]. If the number of charts is low (below 500), PRAM achieves similar compression rates than the CHuMI viewer [Jamin et al., 2009]. In order to get a fair comparison, we tuned both algorithms to have approximately the same number of SP-cells (the unit of random accessibility for the CHuMI viewer) than the number of charts. We expect that the approach of Du et al. [Du et al., 2009] would give slightly better compression rates as it does not duplicate any geometry information. No experimental compression results of the method of Cheng et al. are provided in [Cheng et al., 2007b].

The Table 8.1 also includes compression rates obtained with the progressive approach from [Lee et al., 2012] to illustrate the cost introduced by the random-accessibility. An interesting fact is that our method can sometimes achieve better results than the progressive approach. This happens with particular CAO models such as the Tank, which is decomposed by the segmentation into planar and highly regular charts. The generated charts can then be efficiently compressed.

Segmentation quality

The advantage of PRAM over some previous methods is that the random accessible clusters are appropriate to the view dependent decompression. Indeed, they tend to be composed of faces with uniform normals as opposed to schemes based on space subdivision [Jamin et al., 2009, Du et al., 2009]. These geometry-driven approaches provide a random-access based on the decomposition of the space into cubes. Consequently, parts of the mesh that are not visible to the user may be decompressed if they belong to the same cell than visible parts. The meaningful segmentation proposed in [Cheng et al., 2007b] is in the same way not suitable for view-dependent decompression. The clusters generated by this method do not have low varying face normals. Yet, the approach of Kim et al. allows to finely refine the mesh according to the view-point.

Random access granularity

To decompress a cell with the approach of Du et al. [Du et al., 2009], the top tree and all the border vertices of previous cells in the dependency list must be decoded (see Section 7.3). Consequently, if the requested cell is at the end of the dependency list, a significant number of border vertices must be decompressed. The algorithm of Kim et al. [Kim et al., 2006] allows to split a vertex with no neighborhood restriction. However, the hierarchy allowing selective refinements is divided into data blocks, which are the atomic unit for the random accessibility. Consequently, some vertices must be decompressed even if they are not inserted to the decompressed model. The CHuMI viewer [Jamin et al., 2009] can decompress a nSP-cell independently of its neighbors if its parent cells have been decoded. The different charts of PRAM can be decompressed totally independently after the wire-net mesh and the wires have been decoded. The data dependency is therefore low compared to previous techniques.

Compression times

The chart compression can be easily multithreaded as after the segmentation each chart is compressed independently. However, the compression operations that take the most time are the segmentation and topology problem handling steps. These steps can represent about 80% of the total compression time. The PRAM encoder can therefore be faster or slower than the progressive coder of Lee et al. [Lee et al., 2012], depending on the topology of the input mesh as shown in the Table 8.1. It is, however, in the best case about twice slower than the CHuMI encoder.

8.5 Decompression scheme

The main advantage of PRAM is that it can decompress any chart at any requested level of detail. We designed for PRAM a simple selective decompression framework that exploits this feature and tries to produce an output model without holes between charts.

8.5.1 View-dependent decompression

Progressive random accessible mesh compression such as PRAM allows to decompress different parts of the mesh at different level of detail depending on the visualization viewpoint. The regions of the mesh that contribute the most to the rendered images are more refined than the other regions. The number of decompressed and displayed elements of the mesh is therefore optimized in function of the viewpoint. No computational or network resources are wasted.

To select an appropriate level of detail l for each chart, at decompression time we compute the average normal $\mathbf{d}_c$ of each chart thanks to its coarsest level of detail. The algorithm then computes the angle between $\mathbf{d}_c$ and the opposite of the view direction $\mathbf{d}_v$ (see Figure 8.12 *a*). If this angle is inferior to a parameter α set by the user (30° in our experiments), the chart is decompressed at its finest level. Otherwise, the value l is determined by a linear curve for which the highest level of detail corresponds to α and the lowest to 90° (see Figure 8.12 *b*). Indeed, when the view direction is perpendicular to the chart normal, the user cannot see it.

Our selective decompression scheme can also take into account the screen resolution in order to avoid excessively refining a mesh when the new details cannot be observed at the given screen resolution. This is achieved by computing for a given chart level of detail the average surface of a triangle S_t. Given the screen resolution, the view frustum and the distance between the viewpoint and the chart d, we determine how many triangles with a surface S_t on a plane perpendicular to the view direction and distant of d could be displayed. Then, we deduce the number of pixels per triangle. If this number is below a threshold, the mesh refinement is stopped.

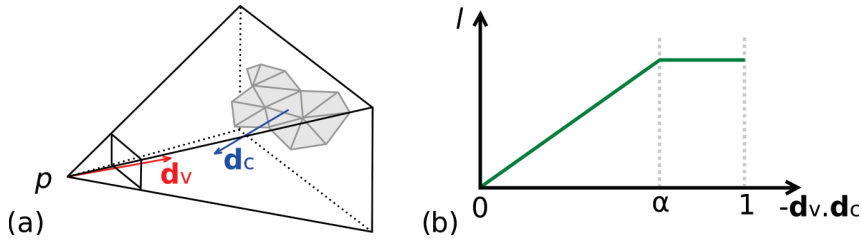


Figure 8.12: The choice of the decompression level of detail (l) for each chart. *a.* A chart with an average normal $\mathbf{d}_c$ inside the view frustum directed by $\mathbf{d}_v$. *b.* The curve giving the decompression level of detail (l) in function of the scalar product $-\mathbf{d}_c \cdot \mathbf{d}_v$.

8.5.2 Dealing with boundaries

One significant drawback of the progressive and randomly-accessible mesh compression schemes based on clusters is that they produce cracks between parts that may be observed as holes. As a first solution to solve this problem, we implemented a quick chart stitching algorithm. For a given boundary between two charts, vertices of the boundary with the highest number of vertices are moved to the positions of their nearest neighbor vertices of the boundary with the lowest number of vertices. The remaining holes are then triangulated; the cracks vanish but some visualization artifacts still remain.

A better stitching can be achieved by checking, for each patch decimation, that v_r is not inside the new border patch after the retriangulation as shown on Figure 8.13. For valence 3 border vertices, we check that the line defined by v_r and the normal of the patch intersects the triangle of the patch after the removal. For

valence 4 border vertices, we first check that the quadrangle of the patch after the removal is convex. If it is convex, we then check that the line defined by v_r and the patch normal intersects this quadrangle. This constraint increases the number of null patches of the progressive chart compression algorithm. Therefore its usage leads to lower compression performance as shown on the Table 8.1 in the 'constrained' column. The Figure 8.14 shows some partial decompression examples.

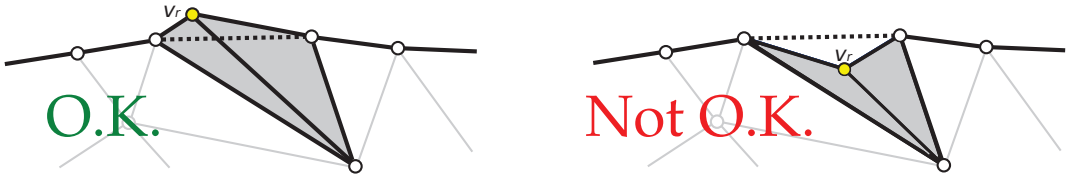


Figure 8.13: Geometry constraint for the decimation of border vertices. If v_r is not inside the new border patch after the retriangulation, the removal is not performed.

8.5.3 Decompression time

An advantage of PRAM is that the decompression, as the compression, can be easily multi-threaded because each chart is processed independently. Therefore, compared to progressive method such as [Lee et al., 2012], a faster full decompression is achieved in our experiments (see Table 8.1). In our experiments, the decompression takes about 5 μs per vertex.

8.6 Conclusion

We demonstrated in this chapter that it is possible to build a progressive extension of the cluster-based random accessible mesh compression scheme described in [Choe et al., 2009]. With PRAM, the input mesh is segmented with one of the two described methods: the original variational segmentation or the decimation-based segmentation. The segmentation has then to be checked and sometimes modified because the chart adjacency relations are coded under the form of a polygon mesh. Each chart is encoded with the progressive algorithm from Lee et al. [Lee et al., 2012]. The border vertices information is, however, stored separately to avoid duplications.

PRAM combines multiresolution and random accessibility with an acceptable cost in term of compression rates. It allows to decompress any generated chart at any level of detail. Only the required data is decompressed. PRAM is therefore an efficient tool for 3D mesh **adaptation** to storage, network and visualization constraints.

PRAM benefits from the **good rate-distortion performance** of the progressive chart compression algorithm [Lee et al., 2012]. Nevertheless cracks can appear between adjacent charts not totally decompressed. A chart stitching algorithm that moves vertices and adds new faces is proposed to generate closed intermediate meshes.

The multi-threaded decompression makes PRAM fits well with the multi-core architectures of recent computers. It allows a **fast decompression** of the mesh. To compress huge meshes, an out-of-core implementation of the compression algorithm could be simply made by using an out-of-core mesh segmentation algorithm.

Nevertheless, the random accessibility offered by this approach is limited as charts must be composed of a significant number of faces to be efficiently compressed. Besides, the proposed chart stitching algorithm sometimes fails to remove all the artifacts. The new progressive and random accessible compression scheme described in the Chapter 9 does not have these issues as it is does not rely on an initial segmentation of the input mesh. It offers a fine random-access granularity. Without any post-processing step, it generates a closed decompressed mesh with smooth transitions between coarse and refined regions of the mesh.

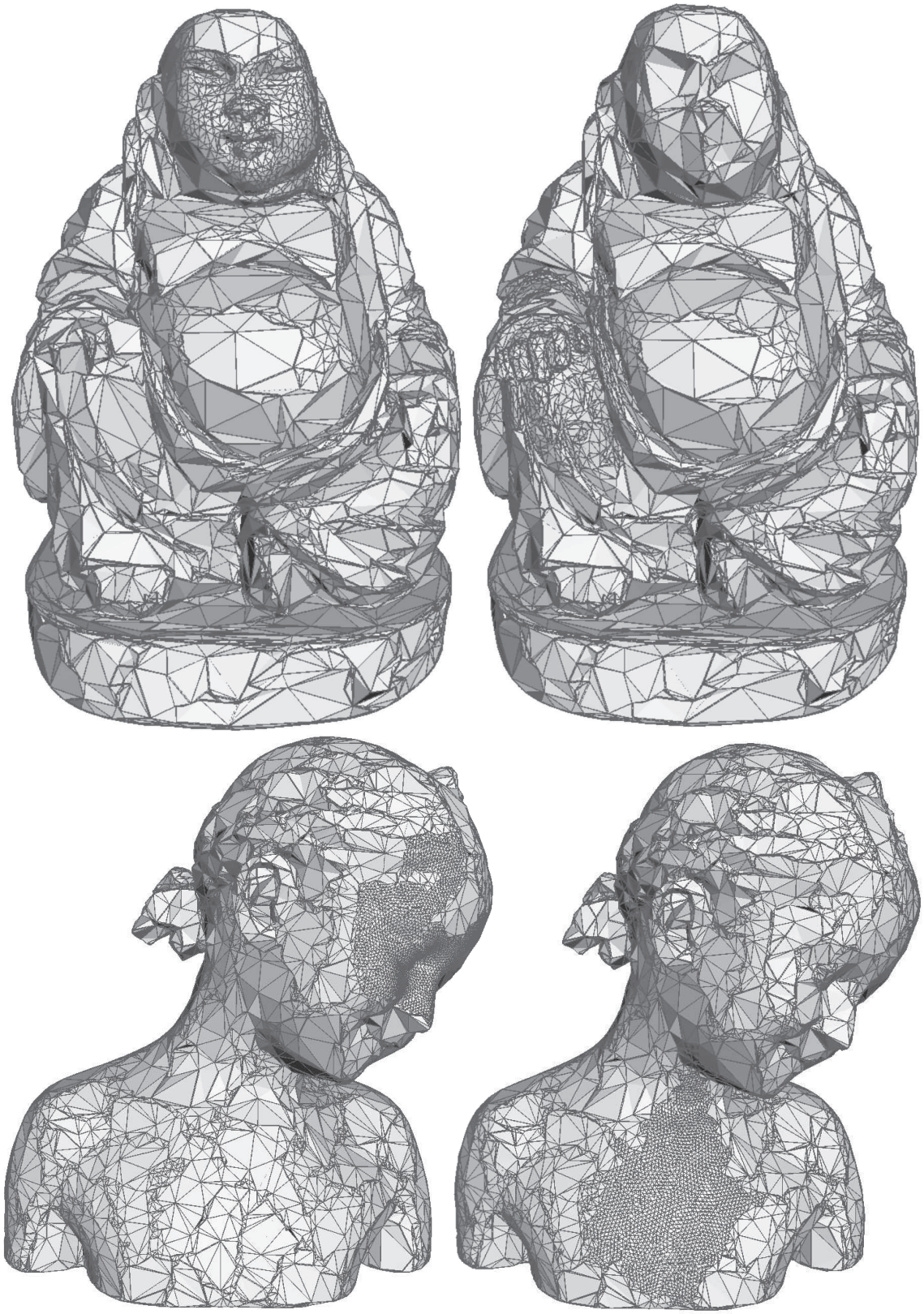


Figure 8.14: Partial decompression obtained with PRAM.

Model	# vertices	# charts	Our scheme					[Lee et al., 2012]			Cost
			Comp. ratio (bpv)			Times (s)		Comp. ratio (bpv)	Times (s)		
			Normal	No wire	Constrained	Comp.	Decomp.		Comp.	Decomp.	
Ramesses	826,266	333	15.5	16.5	16.7	24	3	12.4	36	25	25.3%
		166	14.3	15.0	15.2						15.5%
		84	13.5	14.1	14.3						9.6%
		37	13.0	13.4	13.5						5.3%
		21	12.6	12.9	13.0						2.1%
Neptune	2,003,932	204	10.4	11.0	11.5	59	10	9	73	61	14.7 %
Dragon	3,609,600	362	9.4	10.0	10.4	117	17	-	-	-	-
Statuette	4,999,996	526	11.3	11.8	12.5	179	25	-	-	-	-
Tank (colored)	230,068	114	19.8	22.5	24.5	18	2	20.4	7	7	-2.8%

Table 8.1: Experimental compression results of PRAM with a 12 bit quantization. The last column gives the cost of the normal compression with our scheme compared to [Lee et al., 2012].

Model	# vertices	Variational seg.		Decimation seg.	
		# charts	Comp. ratio (bpv)	# charts	Comp. ratio (bpv)
Ramesses	826,266	166	15.2	168	17.5
Neptune	2,003,932	204	11,5	204	12.2
Dragon	3,609,600	362	10.4	362	11.1
Thai Statue	4,999,996	526	12.5	526	15.5

Table 8.2: Comparison of the experimental compression rates of PRAM obtained with the two segmentation methods, 12 bit quantization and constrained chart decimation.

Chapter 9

Progressive and random accessible mesh compression based on hierarchical vertex clustering

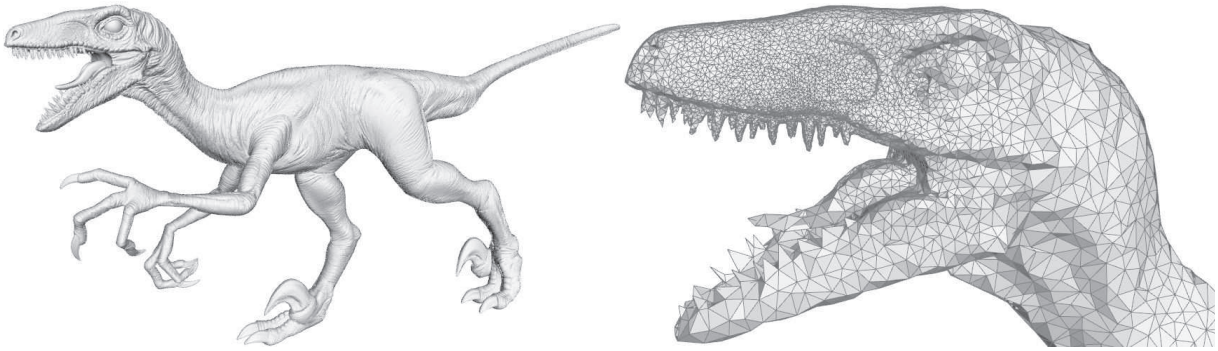


Figure 9.1: Selective decompression of the original raptor model. It was compressed at 15.0 bits per vertex with a 12 bit quantization.

9.1 Introduction

We know from the Chapters 7 and 8 that progressive random accessible mesh compression schemes are useful for 3D mesh **adaptation** to storage, network and visualization constraints. They allow to not waste resources by downloading and decompressing only the data required by the user.

In the Chapter 8, we showed that PRAM, our first progressive random accessible approach based on a preliminary partition of the input model is particularly interesting for view-dependent decompression. It allows to well adapt the decompression level of detail of each cluster depending on the viewing angle. Indeed, the generated clusters have an approximately uniform normal. There is, however, two major problems with this approach:

- The compression performance is poor with a high number of clusters. The random access granularity must be limited to achieve good compression rates.

- The problem of filling the holes between clusters not totally decompressed is difficult to solve perfectly. Even after a post-processing stitching, there is often remaining artefacts appearing in the decompressed models.

In this chapter, we describe a new progressive random accessible algorithm, called **POMAR** for Progressive Oriented Mesh Accessible Randomly. It can compress manifold oriented triangle meshes. During the decompression, each region of the mesh can be decompressed at the level of detail requested by the user. The restriction is that two adjacent regions must have at most one level of detail of difference.

POMAR has been designed for efficient mesh storage and transmission but it also offers interesting features for interactive visualization. It generates decompressed models with visually good triangles and no artefacts. Unlike PRAM (see Chapter 8) or the CHuMI viewer [Jamin et al., 2009], no post-processing operations have to be performed on the boundaries between the regions decompressed at different levels of detail. The decompression algorithm requires a low delay and produces smooth transitions between the fine and coarse regions of the mesh.

The content of this chapter can be summarized as follows:

- We first give an **overview** of the algorithm.
- The **decimation step** is then described. It generates discrete levels of detail with halfedge collapse operations. The hierarchy of collapse operations allows then to define the random accessible clusters.
- The **compression** and the **decompression** both reconstruct the successive levels of detail by performing the reverse operations of the decimation, the vertex splits. During the compression, vertex splits are encoded with four connectivity symbols and a barycentric error prediction for the geometry. The coarse levels of detail are compressed globally as with a standard progressive compression algorithm. However, the finest levels of detail are compressed independently for each random accessible cluster.
- The influence of the **number of clusters** for the random access granularity is then studied.
- We also provide **experimental results** and compare POMAR with PRAM and other previous schemes.

9.2 Overview

The POMAR compression algorithm performs three main tasks.

Mesh decimation. The input model is first simplified with *halfedge collapses* to generate discrete levels of detail. The decimation stops when a coarse version of the input model, called the *base mesh*, is obtained. All the performed operations are kept in memory to allow the later mesh reconstruction by the encoding steps.

Global level of detail compression. Starting from the base mesh, the successive levels of detail generated by the decimation are reconstructed by performing the reverse operations of the halfedge collapses, the *vertex splits*. The symbols needed to rebuild the levels are encoded for the whole mesh, in the same way as a progressive compression algorithm. We call *cluster* a set of vertices of the input mesh that hierarchically collapsed into a common vertex during the decimation. The global level of detail compression stops when, in the current level of detail, there are as many vertices as desired random accessible clusters. This level is called the *base clustered mesh*.

Clustered level of detail compression. Starting from the base clustered mesh, the levels of detail reconstruction assigns inserted vertices to clusters in function of the vertex splits hierarchy. Each vertex of the base clustered mesh has a corresponding random-accessible cluster. The vertices that collapsed into a common vertex v_c of the base clustered mesh belong to the cluster of v_c. The independent encoding of the vertex splits for each cluster enables the random-accessible decompression.

The mesh decompression happens in the same order as the two compression steps. However, only the desired regions of the mesh are decoded and reconstructed with the constraint that there must be at maximum one level of detail of difference between adjacent clusters.

9.3 Decimation

The aim of the decimation step is to generate discrete levels of detail. The decimation operator used is the halfedge collapse. It consists of merging a vertex with one of its neighbors and removing the degenerated faces. A patch is the set of faces modified by an halfedge collapse. We call v_{from} the vertex that is going to merge with the vertex v_{to}, which does not move. After the collapse, v_{ref} is the vertex of the patch that makes with v_{to} the longest edge generated by the collapse. After the collapse of a border vertex, v_ref is the other border vertex of the patch that becomes connected to v_{to}.

v_l and v_r are the two vertices that are connected to v_{from} and v_{to} before the collapse. v_l is on the left of the collapsed halfedge and v_r on the right. All these vertices are represented on Figure 9.2.

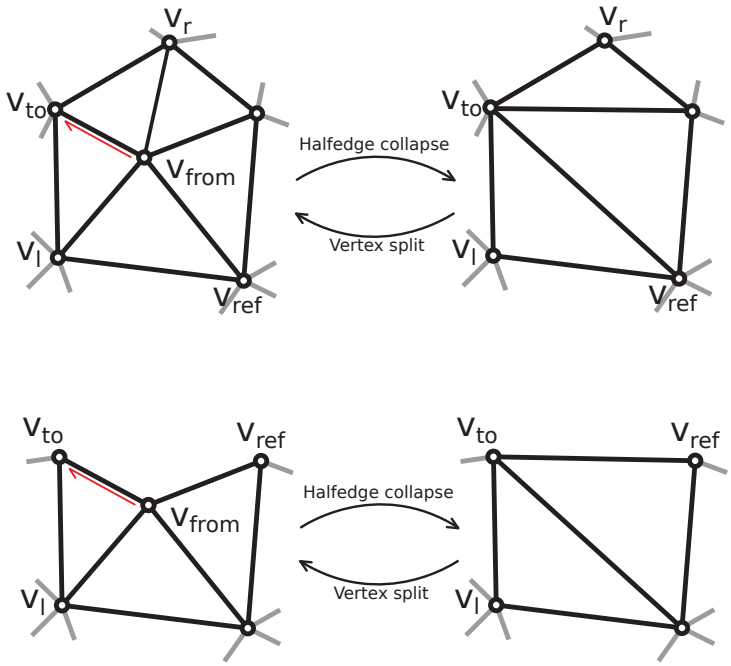


Figure 9.2: Patches modified by halfedge collapse and vertex split operations. The collapsed halfedge is in red. The top case illustrates the collapse of an inner vertex. The bottom case illustrates the collapse of a border vertex.

To generate a new level of detail l, all the halfedges that can be collapsed without violating the manifold property of the mesh are ranked according to an edge length metric. The value of this metric corresponds to the length of the edge v_{to} v_{ref}. The halfedges with the smallest metric values are collapsed first. An halfedge cannot be collapsed if its collapse would generate face normal flips. We ensure that the normals are not altered beyond a certain threshold. We verify for each face of the patch that the cosine of the angle between the normal before and after the collapse does not exceed a threshold (set to 0.7 in our experiments). After a collapse, no more halfedges that would modify the patch can be collapsed for generation of the current level of detail. So these halfedges are marked with a flag.

The generation of the new level of detail l is finished when there are no more halfedge candidates to be collapsed. The generation of the level $l-1$ then begins after having reset all the halfedge flags. In this way,

successive levels of detail are generated until the targeted number of vertices for the base mesh ($l = 0$) is reached.

Other existing decimation metrics, such as the Hausdorff distance in [Pajarola and Rossignac, 2000], the volume metric from [Alliez and Desbrun, 2001a] or the well-known Quadric Error Metric [Garland and Heckbert, 1997], could be used. Nevertheless, our simple edge length metric goes together with the predictions involved in the compression scheme (see Section 9.4). Therefore, the generated levels of detail can be efficiently compressed. Our decimation scheme produces levels of detail with uniform triangle sizes as illustrated by the Figure 9.3.

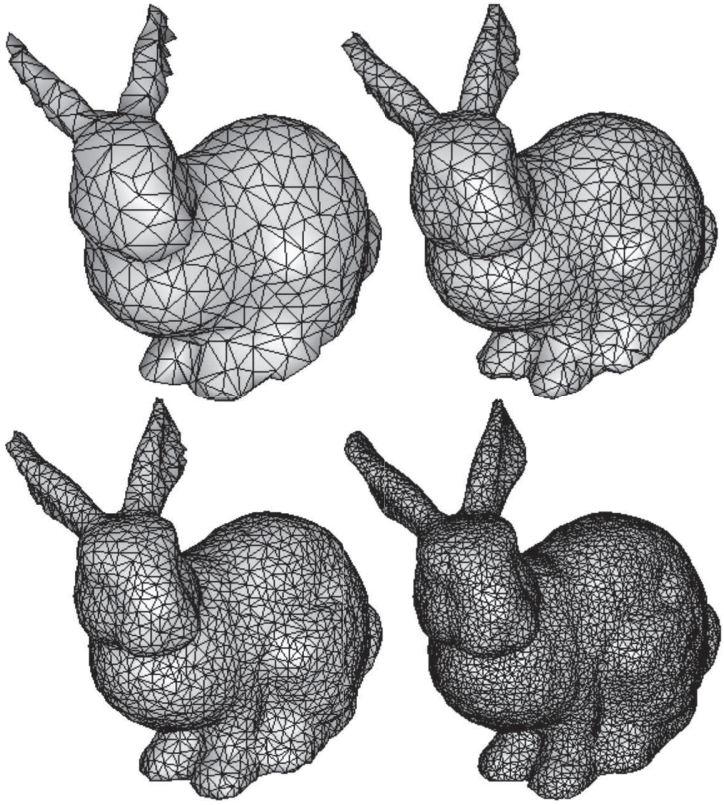


Figure 9.3: Examples of levels of detail generated by our decimation algorithm.

All the performed halfedge collapses are kept in memory because the reverse vertex split operations will have to be performed during the mesh compression. Vertex positions are quantized before the reconstruction and compression steps.

9.4 Compression and reconstruction

Starting from the base mesh ($l = 0$), the successive levels of detail l of the mesh are reconstructed and encoded. The vertex splits, the reverse operations of the halfedge collapses performed during the decimation, are applied. A vertex must be split if it was the v_{to} vertex of an halfedge collapse operation performed during the decimation of the current level of detail.

The reconstructed levels of detail are split into two layers. The coarse levels are not clustered. During the reconstruction, the mesh is uniformly refined as it would be with a progressive mesh compression algorithm. However, the finest levels of detail are clustered. A cluster can be refined more or less than its neighbors. This enables the random access during the decompression.

9.4.1 Global levels of detail encoding and reconstruction

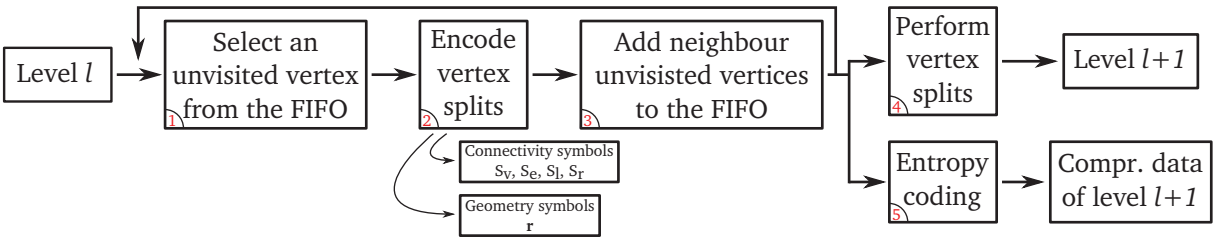


Figure 9.4: Encoding process of a level of detail.

The encoding process of a level of detail is illustrated on Figure 9.4. For each level of detail, vertex splits are first encoded before being performed. Two types of information need to be compressed to perform the same operation during the decompression: the connectivity and the geometry.

The *connectivity* information of a vertex split operation on a triangle mesh can be defined by three vertices: v_{to}, v_l and v_r. We made the choice to encode a vertex split with one additional vertex: v_{ref}. The reason is that v_{ref} can be easily predicted with the length of the edge $v_{to}\ v_{ref}$. When v_{ref} is known, v_l and v_r can be efficiently encoded with offsets.

A deterministic traversal of the mesh vertices is performed. It consists of a breadth-first traversal also carried out by the decoder in the same order. In the base mesh, the first halfedge of the first face is pushed into a FIFO queue. This halfedge is selected since it can be retrieved by the decoder with the base mesh data. It will always be the first processed halfedge for all the global levels of detail. To process a new vertex, an halfedge is popped out the FIFO queue (Figure 9.4, step 1). The data of the vertex it points to is encoded if this vertex has not already been visited (Figure 9.4, step 2). Then, all the outgoing halfedges of the current vertex are added to the FIFO queue in clockwise order (Figure 9.4, step 3). The traversal is continued by popping out another halfedge from the FIFO queue.

For each vertex, we encode its number of splits, the s_v symbols. Thus, the v_{to} vertices will be known by the decoder. If the current vertex has a number of splits different from zero, then the corresponding split(s) must be encoded. The surrounding edges of v_{to} are ranked from the longest to the shortest. The second vertex of these edges are the possible v_{ref} vertices. This ranking forms a stack, the longest edge being on the top of the stack. The algorithm counts the number of edges it has to unstack before getting the correct v_{ref}. This count is the s_e symbol and thus encodes v_{ref}.

To encode the v_l vertex, the algorithm simply counts the number of vertices belonging to the patch between v_{ref} and v_l and so obtains the s_l symbols. The v_r vertex is encoded in the same way with the s_r symbol. For vertices inserted on the boundary, only one of these two symbols are needed.

The *geometry* information of a vertex split is the position of v_{from}. It is encoded with the residual vector $\mathbf{r}$ between the barycenter of the patch b and the position of v_{from}. This residual is projected in a local Frenet frame. This frame is constructed with the direction of the edge $v_{to}\ v_{ref}$ and the average normal of the two faces adjacent to the same edge as illustrated in Figure 9.5. To avoid a post-quantization step and slightly reduce the entropy, we use the bijection proposed in [Lee et al., 2009] to project $\mathbf{r}$ from the global $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ frame to the local Frenet frame $(\mathbf{t_1}, \mathbf{t_2}, \mathbf{n})$.

Once all the split operations of the current level of detail have been encoded, these operations are performed (Figure 9.4, step 4). So, the level of detail l is obtained. The encoding and then the reconstruction of the next level $l+1$ is performed in the same manner. An encoding traversal is illustrated on Figure 9.6.

9.4.2 Clustered levels of detail encoding and reconstruction

The base clustered mesh is obtained once the mesh contains the number of desired clusters for the random accessibility. This corresponds to the level of detail l_c. In the base clustered mesh, each vertex is the parent

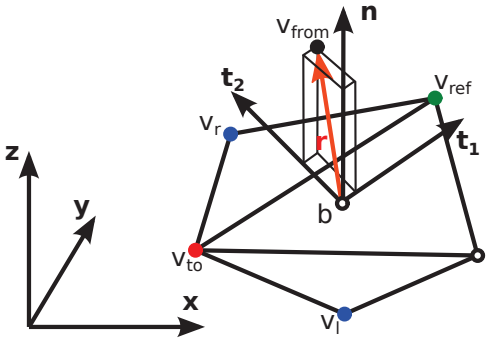


Figure 9.5: Local Fresnet frame $(\mathbf{t_1}, \mathbf{t_2}, \mathbf{n})$ used to encode the geometry residual $\mathbf{r}$.

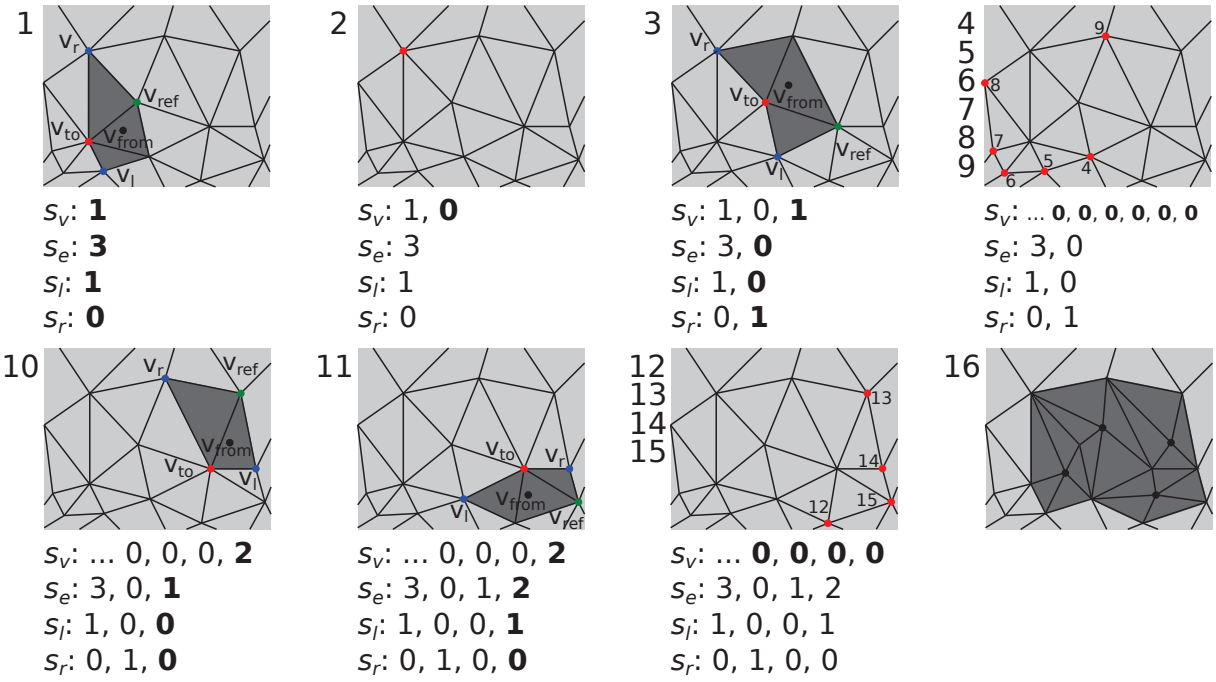


Figure 9.6: Encoding traversal. The vertices are iteratively processed by a deterministic traversal. The current vertex is in red. 1-15: Vertex splits are encoded with the s_v, s_e, s_l and s_r symbols. The symbols related to the current vertex are in bold. 16: Vertices are finally split once the encoding traversal is finished.

of one cluster. In the next level of detail, if a vertex v_c of the base clustered mesh is split, then the inserted vertices will belong to the cluster of v_c. In the following levels, these vertices may also be split, thus adding new vertices to the cluster of v_c. In the end, the cluster corresponding to v_c will be the set of vertices of the input mesh that are the descendant of v_c by vertex splits. The adjacency relations between the base clustered mesh vertices are the same as the adjacency relations between their respective clusters.

To enable the random access during the decompression, the clusters are compressed independently. Separate deterministic traversals that encode the splits are performed for each cluster instead of being performed once for the whole mesh. In this way, only the vertex splits related to the current cluster are encoded in its data block.

For each cluster, the first halfedge to process needs to be selected with a method also available to the decoder. The same breadth-first traversal as during the global levels of detail encoding is performed on the

base clustered mesh. When the current halfedge points to a unvisited vertex, it is set as the first halfedge for the encoding of the vertex cluster.

The encoding traversal of each cluster follows the same principle as the encoding traversal of the global levels of detail. The encoded data is also the same. However, if a current halfedge points to a vertex belonging to another cluster, it is simply omitted.

When the encoding traversal has been performed for all the clusters, the vertices are split to reconstruct the new level. The inserted vertices inherit their cluster id from their parents. The encoding of the next clustered level of detail then starts.

To split a vertex v_{to} and obtain v_{from}, the patch must be in the same configuration as during the simplification. The vertices v_l, v_r and v_{ref} among others must be present. v_{to}, v_l, v_r and v_{ref} belong to the level $l-1$, while v_{from} belongs to the level l. If some patch vertices belong to neighboring clusters, the decoder will simply need to reconstruct these neighboring clusters at the level of detail $l-1$. The data dependency does not include the same level of detail on the whole mesh as with standard progressive mesh compression algorithms. To decompress a cluster C_k at a set level l, C_k and all its adjacent clusters must be reconstructed at the level $l-1$. The V-partition multiresolution model described in [Cignoni et al., 2005] also uses this principle to generate smooth transitions between mesh partitions belonging to several levels of detail.

9.4.3 Entropy coding and compressed file

The connectivity symbols s_v, s_e, s_r and s_l are compressed by a range coder [Schindler, 1998] with a quasi-static probability model for each symbol (Figure 9.4, step 5). The tangential components of $\mathbf{r}$ and its normal components are also encoded by a range coder. One model is used for the tangential components and a different one for the normal.

The compressed file starts with header information such as the mesh bounding box and the number of quantization bits. The base mesh is then stored and not compressed in an indexed data structure (see Section 1.3.1). The compressed data of all the global levels of detail is saved just after, as it allows the restoration of the base clustered mesh. To perform random accessible decompression, the decoder needs to know the position in the file of each data block of compressed cluster data. Hence the size of each cluster compressed data has to be stored to generate a location index. The main bulk of the file is stored at end, containing the compressed data of all the clusters. The Figure 9.7 illustrates the compressed file structure. In a transmission context, each block of the file can be streamed on-demand. Thus, only the data required to decompress the requested parts of the mesh are transmitted to the client.

9.5 Decompression

The decompression starts by reconstructing the base mesh. The global levels of detail are then decompressed until the base clustered mesh is obtained. A map of the required level of detail M for each cluster C_k must then be generated. The user selects the set R_c of clusters he is interested in by picking their corresponding vertex in the base clustered mesh. He then chooses at which clustered level of detail l_k he wants to decompress the selected clusters:

$$\forall C_k \in R_c, M(C_k) = l_k.$$

This map must be completed with the following constraint: there must be at maximum one level of detail of difference between a cluster and its neighbors. This completion can be expressed with the formula:

$$M(C_k) = \max(\left\{ \max_{\forall C_l \in R_c} (M(C_l) - d_t(C_k, C_l)), 0 \right\})$$

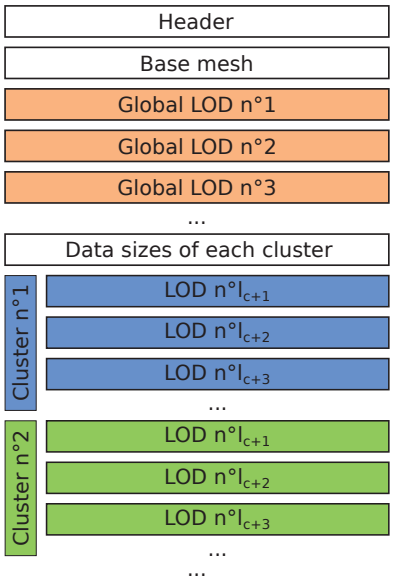


Figure 9.7: Structure of compressed files generated by POMAR.

where $d_t(C_k, C_l)$ is the minimum topological distance in number of edges between the parent vertex of the cluster C_k and the parent vertex of the cluster C_l in the base clustered mesh. In practice, this map is generated with an algorithm that, starting from the parent vertex of each cluster of R_c, traverses all the base clustered mesh vertices and assigns them a required level. Figure 9.8 shows an example of a required levels of detail map and the obtained decompressed mesh.

Once the map M is complete, the decompression of the clustered levels of detail begins. The random access to each cluster data block in the compressed file is permitted thanks to the location table generated with the stored size of each cluster data block. When the required clustered level of detail of one cluster is null, then its data block is completely omitted.

For each clustered level of detail, only the required levels for all the refined clusters are decompressed. The levels are then reconstructed as during the compression. The difference is that only the decompressed parts of the mesh are refined.

The main point of the POMAR decompression is that it is possible to have up to one level of detail of difference between one cluster and its neighbors. So, the decompression can generate a smooth transition between the clusters that the user wants to see at the highest level of detail and the clusters not of interest. The shape of the triangles in the partially decompressed meshes is good as shown in Figure 9.9 because they have been generated by the decimation step guided by the metric.

9.6 Choice of the number of clusters

The choice of the number of clusters is important as it directly influences the random accessibility, the locality of the refinements and the compression performance of the algorithm. The user chooses the number of clusters by setting which generated level of detail l_c will correspond to the base clustered mesh. The base clustered mesh is the coarsest level of detail the user has access before selecting the clusters to refine more. We will now study the impact of the value of l_c on the random accessibility with two examples. Whatever the value of l_c is, the total number of levels of detail is always constant.

A low value of l_c means a low number of clusters. There will be a high number of clustered levels of detail, but during the decompression there can be only one level of detail of difference between two neighbor

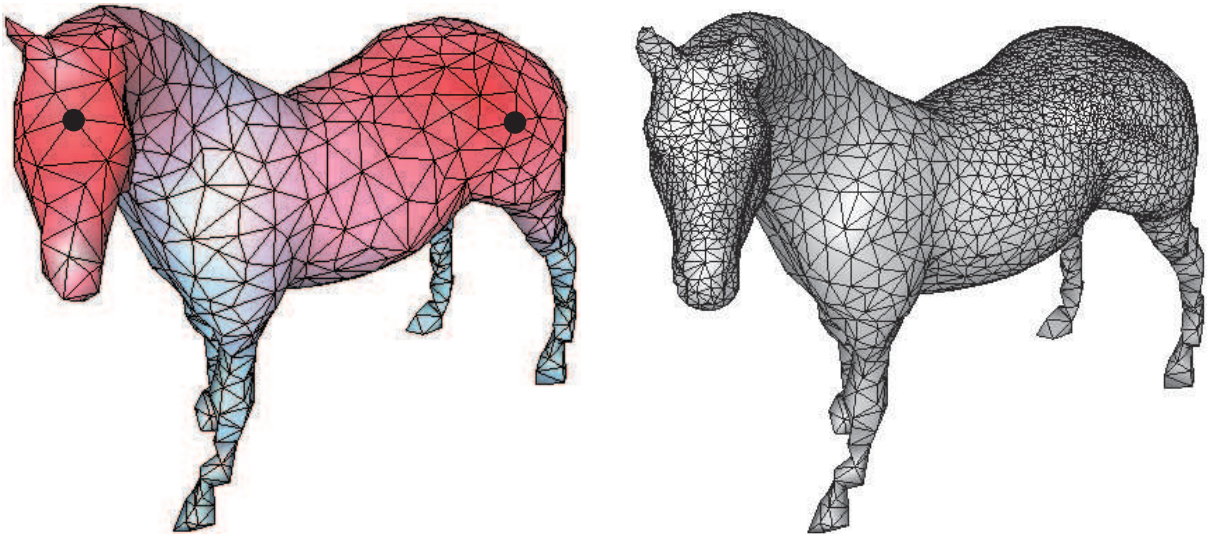


Figure 9.8: Required levels of detail map M displayed on the base clustered mesh and the obtained decompressed mesh. The two black vertices have been chosen to decompress their clusters at the finest levels of detail. The red areas of the base clustered mesh are decompressed at a high clustered level of detail while the blue areas are decompressed at a low clustered level of detail.

clusters. Therefore, if a cluster of the mesh is selected to be decompressed at the finest level of detail, a significant part of the mesh will have to be decompressed to respect this constraint.

If a high value of l_c is chosen, the base clustered mesh will be very refined and there will be a low number of clustered levels of detail. The full decompression of one cluster will only impact a small region of the mesh and there will be a weak gradation between the coarse and refined clusters.

The consequence of this behavior is that, in order to guarantee a correct random accessibility and good compression rates, we advise to choose l_c so that the number of clusters is between 1% and 5% of the input mesh number of vertices. Figure 9.10 illustrates the influence of the number of clusters on the random accessibility and the locality of the refinements. We also study the impact of the number of clusters on the compression rates in Section 9.7.

9.7 Experimental results

We implemented our algorithm using the halfedge data structure from OpenMesh [Botsch et al., 2002] and the range coder from Michael Schindler [Schindler, 1998]. We provide in Table 9.1 experimental results obtained with our first implementation of the POMAR codec, the progressive and random accessible algorithm of the CHuMI viewer [Gandoin and Devillers, 2002] and the random accessible hierarchical approach [Courbet and Hudelot, 2009]. Experiments ran on a desktop computer with an Intel Core i7 CPU at 2.80 Ghz with 8 GB of RAM. Comparing the compression performance of the POMAR codec with the performance of the CHuMI viewer is difficult, as both algorithms do not provide the same type of random accessibility. Nevertheless, each cluster generated by our compression algorithm can be decompressed independently. This is also the case of each nSP-cell of the CHuMI viewer. Therefore we tried, for each tested mesh, to have approximatively the same number of clusters as the number of nSP-cells. Some models could not be compressed with the hierarchical approach as it does not handle meshes with a genus superior to zero or the compression took too much time to complete.

We implemented a selective decompression application to demonstrate the features of our scheme. The user selects, by drawing a rectangle, the clusters he wants to decompress at the highest level of detail on the

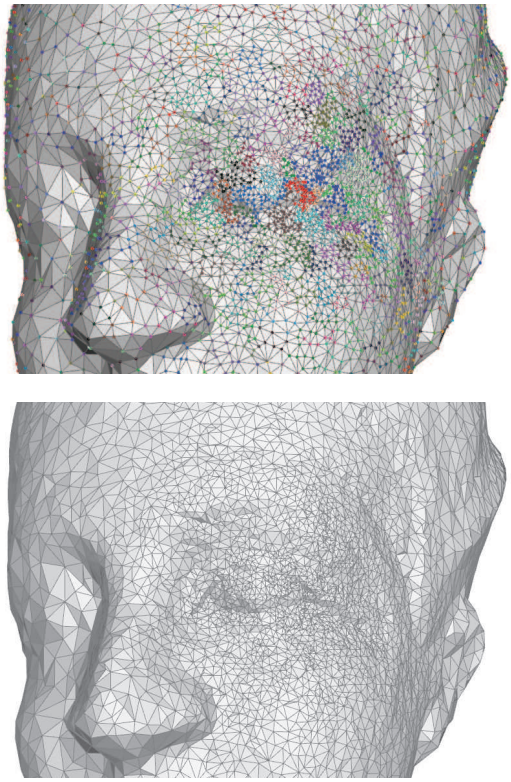


Figure 9.9: Decompression example of the Ramesses model (826266 vertices). On the top image, the vertices belonging to the same cluster have the same colour. The red cluster on the eye was selected to be decompressed at the highest level of detail.

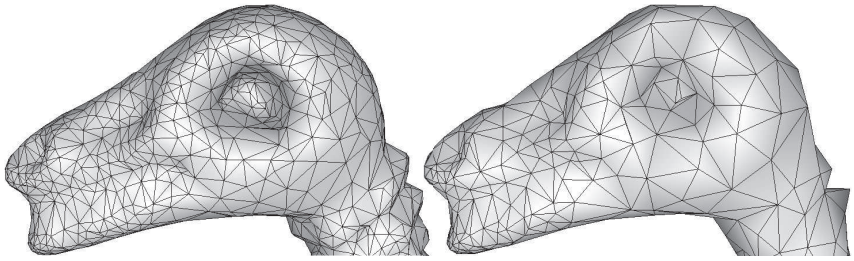


Figure 9.10: Decompression examples of the dinosaur model (14070 vertices). On the left, the model was compressed with a high number of clusters. On the right, it was compressed with a low number of clusters. For both cases, a cluster on the end of the nose was selected to be decompressed at the highest level of detail.

base clustered mesh by their parent vertices. The map M of the required levels of detail for each cluster is computed as described in Section 9.5. The Figure 9.13 shows some examples of partial mesh decompression.

We studied for one mesh the influence of the number of clusters on the compression rate. Figure 9.11 shows the obtained curve. The compression performance decreases when the number of clusters increases. This relation is asymptotically linear with a significant slope. We indeed use a different quasi-static probability model to independently build the probability tables of the entropy coder for each level of detail and each cluster. The more clusters there are, the less the models can well adapt to the data. As a consequence, the choice of the number of clusters must be taken with care as it also influences the random-accessibility (see Section 9.6).

Model	# vertices	POMAR					CHuMI			Hierarchical	
		# clust.	c.	g.	tot.	time	# cells	tot.	time	tot.	time
Igea	134,345	5192	8.4	15.0	23.4	7s	5249	27.4	3s	27.19	21s
Armadillo	172,974	9268	9.1	15.0	24.1	9s	9217	26.0	3s	24.7	46s
Fertility	241,607	2551	8.1	11.8	19.8	14s	2561	22.7	5s	-	-
Raptor	1,000,080	4869	7.6	7.0	14.6	47s	4801	16.44	10s	-	-
Ramesses	826,266	4480	7.6	8.4	16.0	50s	4481	17.4	11s	22.4	2m 44s
Neptune	2,003,932	10498	7.7	5.7	13.5	2m 7s	10497	14.7	24s	-	-
Dragon	3,609,600	14383	7.4	5.7	13.0	3m 40s	14337	15.0	27s	20.0	53m 13s
Statuette	4,999,996	19094	7.9	5.9	13.8	4m 51s	19073	15.0	35s	-	-

Table 9.1: Experimental compression results of the POMAR codec, the CHuMI viewer [Jamin et al., 2009] and the hierarchical approach [Courbet and Hudelot, 2009]. Geometry is quantized to 12 bits. The compression rates are in bits per vertex. c. stands for connectivity and g. stands for geometry.

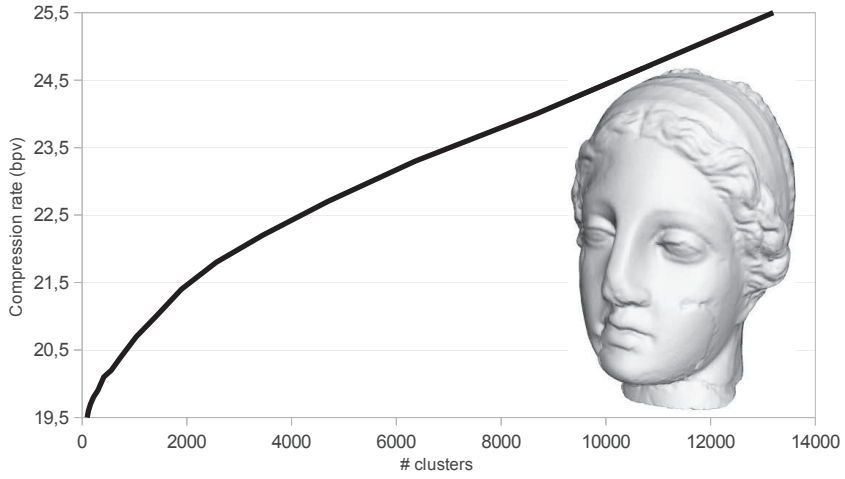


Figure 9.11: Compression rate in function of the number of clusters for the Igea model (134345 vertices). Geometry in quantized to 12 bits.

The POMAR scheme can also be used in pure progressive compression mode. In this case, only global levels of detail are encoded as described in Section 9.4.1. We measured the rate-distortion performance of our algorithm and previous approaches in progressive mode. The results are shown on Figure 9.12.

9.8 Comparisons and discussion

9.8.1 Progressive and random accessible compression

Compression rates

In the conditions described in Section 9.7, the results of Table 9.1 show that, for the tested models, POMAR always provides better compression rates than the CHuMI viewer. The duplication of the vertices belonging to several nSP-cells by the CHuMI encoder may explain why our algorithm provides better compression rates. POMAR does not duplicate any geometry or connectivity information. We expect that the scheme of Du et al. [Du et al., 2009] would provide slightly better results than the CHuMI viewer with the same random access granularity. Both algorithms are a random accessible extension of the original Gandoin and Devilliers algorithm [Gandoin and Devillers, 2002], but the algorithm of Du et al. does not duplicate border vertices. The compression rates provided by POMAR are about 10 bits per vertex lower than the rates

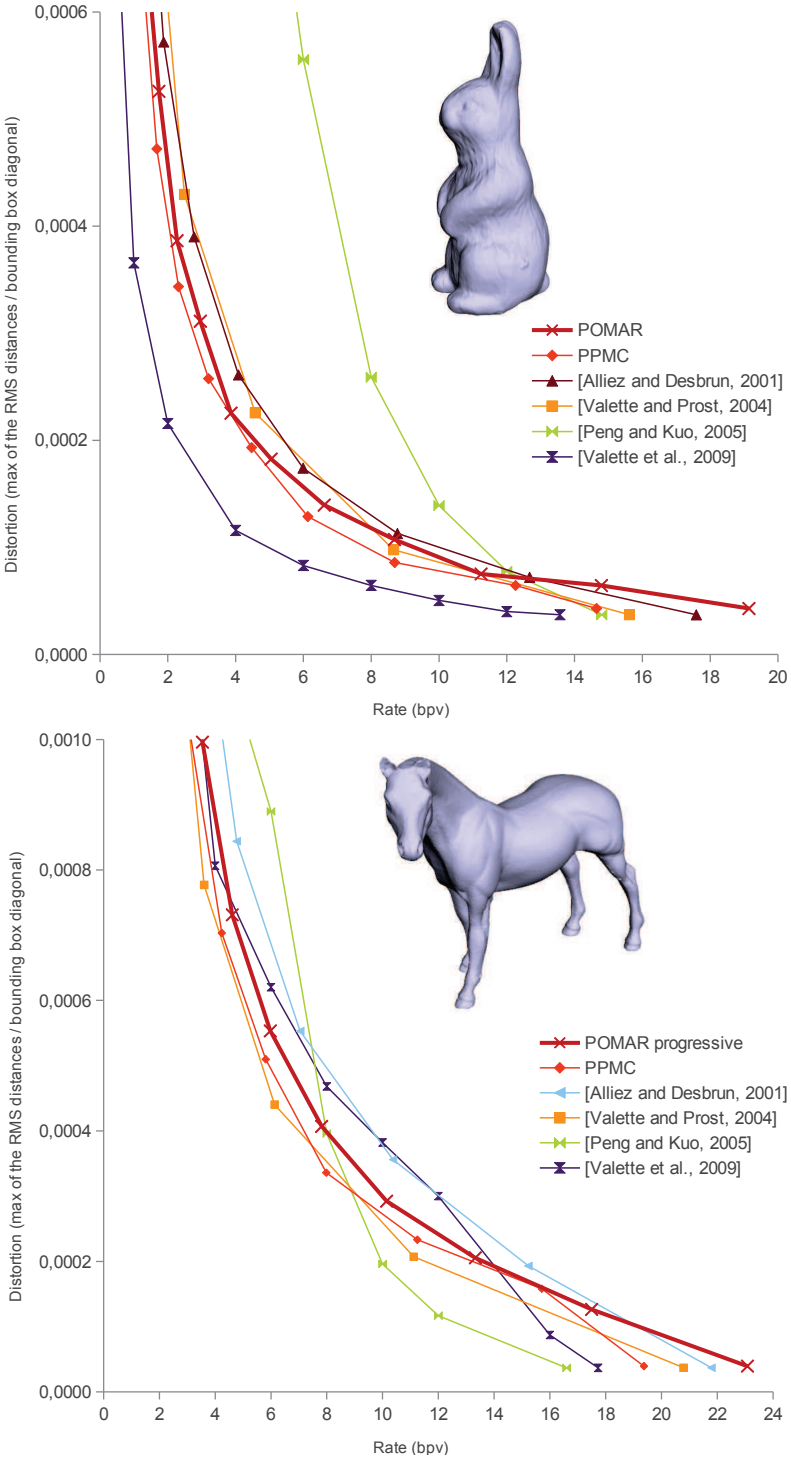


Figure 9.12: Rate-distortion curves for the progressive compression of the rabbit model (67039 vertices) and the horse model (19851 vertices) with a 12 bit quantization.

given by the approach of Kim et al [Kim et al., 2006] and PRAM (see Chapter 8) does not support the high granularity random access of our experiments.

Random-accessibility granularity

With POMAR, to decompress a cluster at a given level of detail l, the global levels of detail must be fully decompressed and the neighbor clusters must be decompressed at the level $l-1$. This constraint spreading on the mesh affects the locality of the refinements and the random-access. To decompress a cell with the approach of Du et al. [Du et al., 2009], the top tree and all the border vertices of previous cells in the dependency list must be decoded. Consequently, if the requested cell is at the end of the dependency list, a significant number of border vertices must be decompressed. The algorithm of Kim et al. [Kim et al., 2006] allows to split a vertex with no neighborhood restriction. However, the hierarchy allowing selective refinements is divided into data blocks, which are the atomic unit for the random accessibility. Consequently, some vertices must be decompressed even if they are not inserted to the decompressed model. The CHuMI viewer [Jamin et al., 2009] can decompress a nSP-cell independently of its neighbors if its parent cells have been decoded. The charts of PRAM (see Chapter 8) can also be decompressed independently after their border vertices have been decoded.

Mesh quality

POMAR has the advantage of reconstructing without any post-processing one piece decompressed models with good triangle shapes. All the decompressed triangles come directly from the decimation and therefore respect its metric. The CHuMI viewer [Jamin et al., 2009] and PRAM (see Chapter 8), however, require a post-processing algorithm to stitch adjacent nSP-cells and clusters together. As for both approaches there is no strong constraint on the boundaries, this step may produce anisotropic triangles and artefacts. The approach of Du et al. [Du et al., 2009] does not need any post-processing step to stitch adjacent cells but if they are decompressed at different levels of detail, anisotropic transition triangles are generated by the decompression.

Progressive compression algorithms based on space subdivision are known to generate decompressed models that have a high distortion at low rates because of the low quantization. This fact is illustrated by the rate-distortion curve of the octree coder on Figure 9.12. To remove this well-known block effect, Jamin et al. [Jamin et al., 2009] proposed for the CHuMI viewer to encode some connectivity information before the geometry, thus increasing the complexity of the compression algorithm. Du et al. [Du et al., 2009] did not address this issue.

Compression times

The POMAR decimation algorithm is based on a ranking of the halfedge candidates to be collapsed. The mesh simplification needs more time than the CHuMI encoder that uses a spatial kd-tree decomposition to simplify the mesh. As expected, the approach of Du et al. [Du et al., 2009] is reported as having similar compression times than the CHuMI encoder.

Decompression times

In our experiments, the decompression takes approximately 4 μs per vertex. Our selective decompression experiments show that the decompression is fast enough to allow interactive decompression like the CHuMI viewer [Jamin et al., 2009] and PRAM (see Section 8.5). The schemes of Du et al. [Du et al., 2009] and Kim et al. [Kim et al., 2006], however, did not demonstrate their ability to perform interactive decompression.

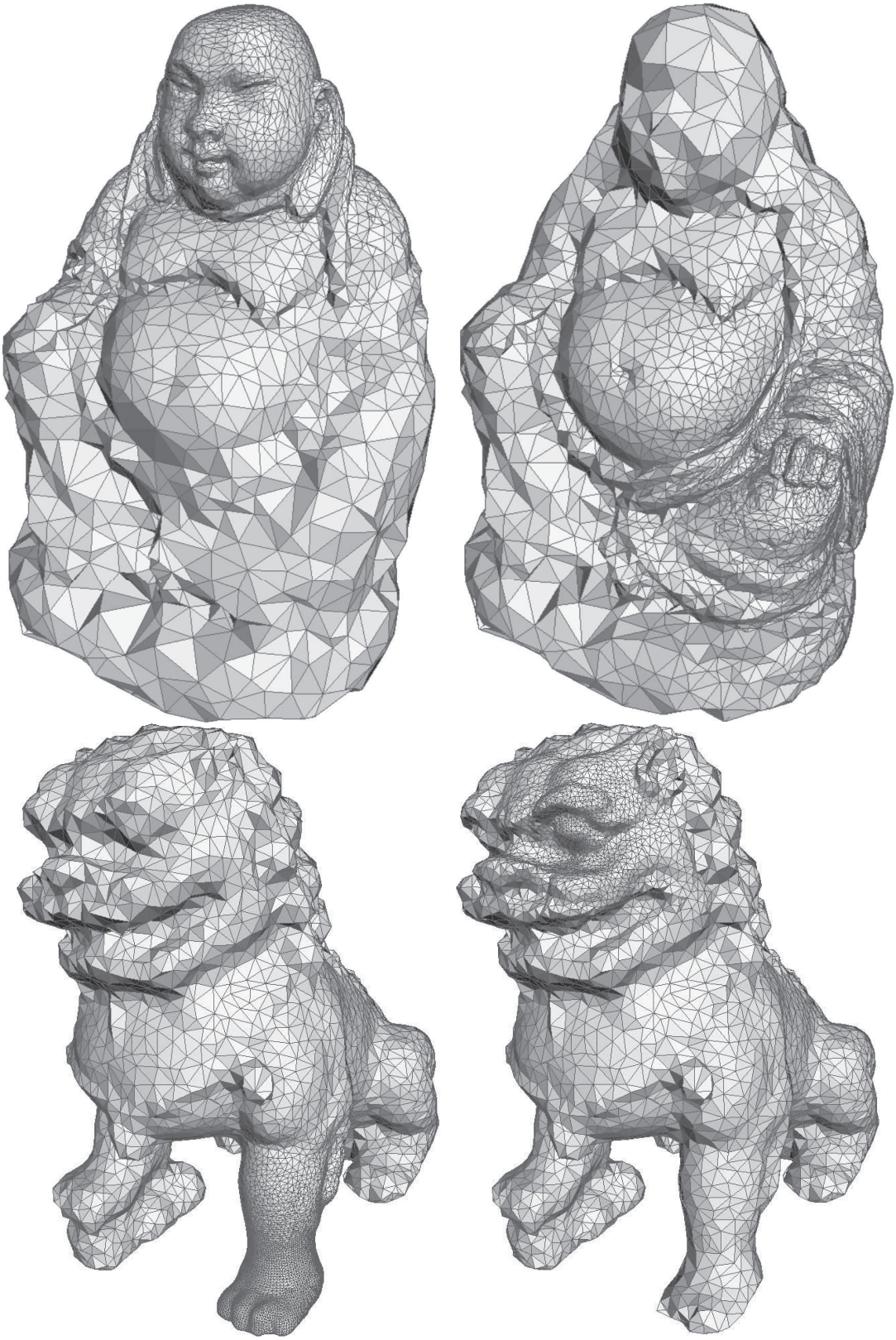


Figure 9.13: Partial decompressions obtained with POMAR.

9.8.2 Progressive compression

As shown on Figure 9.12, compared to previous progressive compression approaches, POMAR achieves competitive rate-distortion performance at low rate. However, the final compression rate, which is illustrated by the right most point of the curve, is superior to the rates obtained by progressive coders. The reason is that our connectivity compression scheme, needed for the progressive random access to clustered levels, is costly. It uses 4 different symbols (s_v, s_e, s_l and s_r) while approaches like the progressive valence encoder [Alliez and Desbrun, 2001a] use only one symbol. Nevertheless, we would like to point out that another quality of the POMAR encoder is its simplicity of implementation.

9.8.3 Compressing large meshes

The approaches described in [Jamin et al., 2009, Du et al., 2009] can compress large meshes that do not fit in main memory. An adaptation of our encoder would also enable out-of-core compression without impacting the final compression rate.

If the input mesh cannot fit in memory, the mesh decimation must be performed out-of-core. One solution is to resort to a block-based simplification as described in [Hoppe, 1998]. Each block is simplified in-core independently without modifying its boundary. Adjacent blocks are then merged into bigger blocks to continue the decimation.

The reconstruction and the compression of global levels of detail would be performed in-core as described in Section 9.4.1. It indeed seems fair that the base clustered mesh can fit in main memory since it is the basis data structure of the random access. The reconstruction and the compression of clustered levels of detail would follow a different strategy. Starting from the base clustered mesh, each cluster would be reconstructed and then compressed separately. As there must be at most one level of detail of difference between adjacent clusters, the compression of one cluster would involve the partial reconstruction of its neighbors. Full levels of detail would not be reconstructed as described in Section 9.4.2.

9.8.4 Data structures for interactive visualization

As seen in Section 7.3.3, progressive and random-accessible mesh data structures have been proposed in the literature for the interactive visualization of large meshes. Most of them were designed to be compact since they must allow the storage of huge meshes either in main memory, in GPU memory or on disk. Their key concept is to enable fast mesh adaptation mechanisms by a quick access to a multiresolution data structure. Consequently, most of these data structures are not compressed. In the recent work of Derzapf and Guthe [Derzapf and Guthe, 2012], which integrates a compressed data structure, the connectivity of the mesh is stored at about 4 bytes per vertex, while our scheme requires about 1 byte. Even if the scheme described in [Derzapf and Guthe, 2012] stores the normal of the mesh vertices while POMAR does not, it saves a mesh with around 11 to 14 bytes per vertex while POMAR requires around 3 bytes per vertex.

The current implementation of POMAR does not support automatic refinement and coarsening of the mesh depending on the viewpoint. This feature is left for future work. The mesh access is slower than with the previously described GPU data structure. It also allows less flexible mesh adaptation. Yet, we think that the much higher compression performance of our approach with its relative progressive and random access to the data is particularly useful for efficient mesh storage and transmission.

9.9 Conclusion

We presented in this chapter POMAR, a new progressive random accessible manifold triangle mesh compression algorithm based on hierarchical vertex clustering. Unlike PRAM (see Chapter 8) or the CHuMI viewer [Jamin et al., 2009], it is not based on a space-subdivision data structure or a initial segmentation of the input model. The levels of detail generated by the decimation are decomposed into two layers. The

levels of the first layers are compressed globally as with a standard progressive mesh compression algorithm. The levels of the second layer are, however, compressed independently for each cluster to allow the random access. During the decompression, the global levels of detail are first decoded. Then each cluster can be decompressed at different levels of detail with the constraint that there must be at most one level of detail of difference between two adjacent cluster.

We believe that POMAR is an interesting tool for the 3D mesh **adaptation** to storage and transmission but it has also nice features for visualization. It enables a **fine-grained random access** to the compressed data. It directly generates one-piece decompressed meshes with smooth transitions between the fine and coarse decompressed regions of the mesh. All the triangles of the decompressed models come from the decimation and therefore respect its metric.

Experimental results show that POMAR compares favourably with previous progressive and random accessible approaches in terms of compression rates with similar random access. Its **fast decompression** algorithm is suitable for iterative selective decompression. Used as a standard progressive algorithm, it achieves good rate-distortion performance at low rates.

POMAR could be improved in several ways. The generalisation of the algorithm to non-manifold meshes could be made by replacing the vertex split operator by generalized vertex splits. A decimation algorithm that better preserves the mesh shape and its adapted compression algorithm could also be investigated. Finally, we also wish to work on a mixed CPU/GPU implementation of our framework to enable automatic refinement and coarsening of the model for interactive visualization. The decompression of the data would be performed by the CPU, while a compact representation of the refinement data would then be transferred to GPU for the rendering.

We also believe that the constraint of having one level of difference between adjacent clusters to allow the random access is a strong idea. It can still be exploited to build new schemes with better performance. Thus, proposing an alternative decimation algorithm that is able to remove more vertices per level of detail than the halfedge collapse operator could increase the locality of the refinements. We think that the wavelet formulation for irregular meshes of Valette and Prost [Valette and Prost, 2004a] (see Section 3.2.4) should be investigated for this purpose.

General conclusion

Contributions

The contributions presented in this manuscript can be summarized as follows:

1. We proposed a new **state-of-the-art** on mesh compression. In particular, we identified four types of mesh compression algorithms: single-rate, progressive, random-accessible and progressive random-accessible. Previous reviews have only described single-rate and progressive mesh compression approaches. Starting from the pioneering work, we tried to exhaustively describe and compare the relevant approaches.

2. The previous state-of-the-art revealed that, as far as we know, progressive compression of meshes with arbitrary face degrees was never studied as a standalone problem. Therefore, we proposed in the Chapter 4, PPMC, a new **progressive mesh compression algorithm for polygon meshes**. The input surface is decimated by several traversals that generate successive levels of detail through a specific patch decimation operator which combines vertex removal and local remeshing. The mesh connectivity is encoded by two lists of Boolean symbols: one for the inserted edges and the other for the faces with a removed center vertex. The connectivity encoding performance is improved with predictions based on the mesh geometry. The mesh geometry is encoded with a barycentric error prediction of the removed vertex coordinates and is improved by integrating a local curvature prediction. We also included two methods that improve the rate-distortion performance: a wavelet formulation with a lifting scheme and an adaptive quantization technique. Experimental results demonstrate the effectiveness of our approach in terms of compression rates and rate-distortion performance, even for the compression of triangle meshes.

3. Progressive mesh compression is linked to mesh simplification as both applications require the generation of levels of detail. So, in the Chapter 5, we proposed a **simple volume metric to drive the simplification of polygon meshes**. To demonstrate the efficiency of this metric, we proposed a simple polygon mesh decimation algorithm based on two operators: the halfedge collapse and the edge removal. At each iteration of the algorithm, the operation that has the lowest metric value is performed. Experimental results show that this scheme can generate coarse polygon meshes with anisotropic polygon faces. We also used this metric to propose an other decimation strategy for PPMC, our progressive polygon mesh compression algorithm. At each decimation step, it prioritizes the patch decimation operations that have the lowest costs. This new strategy leads to better rate-distortion performance at low rates for irregular models.

4. Progressive mesh compression is useful for the remote visualization of 3D meshes. Indeed, it allows to quickly display a coarse model to the user and progressively refine it as more data is received. The multiresolution structure of compressed models is also interesting to adapt the mesh to the computational capabilities of the rendering device. We proposed in the Chapter 6 an **applicative adaptation framework for remote scientific visualization**. Targeted data was 3D meshes with color attributes on their vertices. This framework is based on the progressive mesh compression algorithm from [Lee et al., 2012] that can handle vertex colors. The adaptation algorithm chooses the best level of detail to download and display, taking into account constraints coming from the network, the device

graphic capabilities and the user view-point and preferences. We also propose an extension of the X3D file format to support progressive meshes.

5. The problem of progressive mesh compression algorithms is that when a specific part of the input model needs to be decompressed at the maximum resolution, the full data must be decompressed. Resources are therefore spent to decompress irrelevant parts of the mesh. Progressive random-accessible mesh compression schemes are a solution to this problem since they allow to decompress different parts of the input model at different levels of detail. We described in the Chapter 8, PRAM, a new **progressive random accessible compression** algorithm based on the random-accessible mesh compression algorithm from Choe et al. [Choe et al., 2009]. We described two methods to segment the input model. The first method is the face clustering algorithm of Choe et al. [Choe et al., 2009]. The second method is based on the polygon mesh simplification algorithm described in the Chapter 5, which has the advantage of generating less invalid clustering topologies. To progressively compress the clusters, we replaced the single-rate compression algorithm of the original scheme by the progressive algorithm from [Lee et al., 2012]. We modified this algorithm to compress independently in wires the geometry of the cluster border vertices because they belong to two clusters. At the end of the decompression, adjacent charts are stitched to produce a closed decompressed model. By adding some constraints on the removed chart border vertices, models with few artifacts are obtained. Since the decompression can be easily multithreaded, compared to progressive approaches, the model can be decompressed faster on multi-core architectures. Experimental compression results show that the cost of the random-accessibility is moderate with a low number of charts.

6. Progressive random-accessible mesh compression schemes based on an initial segmentation of the input model need, at decompression time, a post-processing step to stitch together adjacent parts. Besides, to be efficiently compressed, parts must have a significant number of faces. POMAR, our **second progressive and random accessible mesh compression** algorithm, presented in the Chapter 9, does not require the input model to be segmented. The compression algorithm is composed of three main steps.

   - The first step generates levels of detail by performing halfedge collapses. It keeps in memory the connectivity of the performed operations as well as the positions of the removed vertices.
   - The second step, starting from the base mesh, successively reconstructs the levels of detail by performing the vertex splits. The connectivity is encoded with four integer symbols. The geometry is encoded with a barycentric error prediction of the removed vertex coordinates projected in a local Fresnet frame. The first encoded levels of detail are global, as with a standard progressive algorithm, until the base clustered mesh is obtained.
   - The third step of the compression generates clustered levels. A cluster is a set of adjacent vertices that hierarchically collapses into a common vertex of the base clustered mesh. Each cluster is encoded independently to allow the decompression random-access.

   By imposing a maximum of one level of detail of difference between adjacent clusters, the decompression algorithm always generates a closed mesh with smooth transitions without artifact. Experimental results also show that this approach compares favorably with previous techniques.

Our aim in this thesis was to use mesh compression as an efficient tool for the **adaptation** of 3D mesh to storage, transmission and visualization constraints. By reducing the size of the data, all the proposed compression approaches presented in this thesis enable the efficient storage of 3D data. They also allow to reduce the transmission times. The progressive nature of the data generated by PPMC allows to quickly display a model to the user, even if the whole compressed data has not yet been transmitted and decompressed. It also allows to select the best model level of detail in function of the terminal capabilities and the viewpoint chosen by the user. Our progressive random accessible mesh compression approaches PRAM and POMAR allow to download, decompress and display to the user the mesh with the minimum number of polygons according to the region of interest he has defined. The decompressed models can therefore be finely adapted to the various conditions.

Some of the contributions presented in this thesis have been published or will shortly appear in international conferences and journals [Maglo et al., 2010, Maglo et al., 2011, Maglo et al., 2012, Maglo et al., 2013].

Future work

Besides all the possible improvements of the proposed techniques we included in the conclusion of each chapter, we have identified four main trails for future work.

Genericity and industrialization. We think that many techniques in the mesh compression field are now mature enough to be used by the industry. However, tools that can handle all the different types of meshes used by the industry and that are easy to use are missing. As a consequence, we think that, for example, the adaptation of the POMAR algorithm presented in Chapter 9 to the compression of non-manifold meshes may be interesting.

Mesh compression for the web. With the WebGL technology integrated in recent web browsers, it is today possible to interactively visualize 3D meshes inside web pages. As stated in the introduction, 3D meshes can represent a significant amount of data. In such a case, mesh compression can have a positive impact to save bandwidth and possibly reduce the page loading time. Today, there is no standard mesh compression format integrated inside browser, as image compression formats (JPEG, PNG, GIF ...) are integrated. One way to bring mesh compression to the web is to program a decoder in Javascript and embed its code inside the web page. Javascript is indeed the scripting language included in the HTML 5 standard. Implementing a fast compression algorithm in a scripting language might be challenging. However currently, there is a war between the web browser vendors to propose the fastest Javascript interpreting engine. We know that a team of the LIRIS laboratory has already begun to work on this topic. They successfully ported in Javascript their progressive decompression algorithm [Lee et al., 2012]. Nevertheless, there is still work to be done in order to choose the techniques that are the most convenient for 3D inside the web.

Parallel mesh compression and decompression. The modern computer architectures are evolving towards more and more computing cores. The desire to perform the decompression on the GPU in order to preserve a concise representation of the mesh all over the pipeline is also emerging. GPU are already composed of a high number of computing cores. Traditional mesh compression algorithms are monothreaded because they aim at transforming a 3D mesh into a bit string. The algorithm we presented in the Chapter 8 is multithreaded but it suffers from two drawbacks. The computation of the initial segmentation takes times. There is a significant overhead in term of compression rate. We believe that the design of a parallel algorithm with a low overhead is a challenging problem. As seen in the Chapter 2, some approaches have already been proposed for the parallel single-rate encoding and decoding of the connectivity [Meyer et al., 2012, Zhao et al., 2012]. Geometry encoding as well as the progressive and random accessible parallel compression have still to be studied.

Distribution of geometry prediction residuals. In mesh compression, the encoding of the mesh vertex coordinates often relies on the prediction paradigm. The position of a new vertex is predicted in function of the already encoded vertices. The difference between the predicted and real positions of the vertex is then entropy coded. Now, the entropy coder needs a symbol probability table to efficiently encode the residuals. There are two solutions to build this model detailed in Section 1.4.2. We believe that if we could well model the probability distribution of residuals with a low number of parameters, the entropy coder could use, since the beginning of the encoding, an efficient model with a very low amount of overhead data transmitted.

Progressive random accessible mesh compression. An important part of the future research in mesh compression may be related to progressive and random accessible approaches. Indeed, these techniques offer important and useful features for 3D mesh adaptation. Regarding interactive visualization scenarios, the convergence between compression approaches and efficient data structures (or view-dependent

progressive meshes) should be explored. Indeed, performing the decompression directly on the device that renders the mesh can allow to save time. Compression approaches have the advantage of producing very compact data. However, they are much less flexible than view-dependent progressive meshes for adaptation purposes. Besides, contrary to view-dependent progressive meshes, very few GPU implementations of compression algorithms have been proposed in the literature. The data structure described in [Derzapf and Guthe, 2012] is a first step towards this convergence. The proposed data structure is compressed and highly flexible. Yet, its compression rates are far from being competitive with compression approaches.

Appendix: Introduction in French

Contexte

Les ordinateurs ont toujours été utilisés pour stocker et traiter les données de mondes réels et virtuels. Parmi les premiers cas d'utilisation des ordinateurs, on trouve la simulation de processus naturels ou artificiels (calculs de trajectoires, calculs de contraintes à l'intérieur d'une structure...). Aujourd'hui les ordinateurs sont toujours utilisés pour la simulation, mais beaucoup d'autres applications ont émergé comme la bureautique ou le divertissement. Avec le développement du réseau mondial que constitue l'Internet, un autre usage important des ordinateurs est aujourd'hui la communication, le partage de données et d'informations.

Comme la vision et l'ouïe nous permettent de capturer et d'interpréter notre environnement, des modes de représentation des données ont été développés pour permettre aux ordinateurs de traiter du son, des images et des images dynamiques à travers les vidéos. Ce domaine est appelé le traitement multimédia. Les quantités physiques analogiques du monde réel sont numérisées pour se plier à la contrainte que les ordinateurs ne peuvent traiter que des données binaires. Le développement d'applications multimédia a permis aux utilisateurs de visionner des images et des vidéos et d'écouter des sons sur leur ordinateur. Cependant le contenu était diffusé à l'utilisateur sous la même forme que durant sa création. Rapidement, les utilisateurs ont exprimé le désir de pouvoir interagir avec les données. Par exemple, ils voulaient pouvoir changer le point de vue d'une image ou d'une vidéo afin de pouvoir se déplacer virtuellement dans la scène. Un nouveau domaine a alors émergé: la réalité virtuelle. Les ordinateurs étaient ici utilisés pour produire des images successives d'une scène enregistrée selon le point de vue défini par les consignes interactives de l'utilisateur. Ce dernier pouvait ainsi naviguer à l'intérieur d'environnements virtuels.

Une représentation des données tridimensionnelles (3D) était nécessaire pour permettre le rendu d'images d'une scène à partir de différents points de vue. Les maillages 3D surfaciques furent rapidement adoptés pour satisfaire ce besoin. Un maillage 3D surfacique est constitué de polygones élémentaires adjacents appelés faces qui ensemble forment une approximation par parties de la surface représentée. Les maillages, comme les images, peuvent être statiques ou dynamiques, si ils varient ou non au cours du temps. Dans cette thèse, nous nous intéressons seulement aux maillages statiques. Plusieurs méthodes permettent de générer des maillages 3D surfaciques. Ils peuvent être créés directement par un opérateur sur un ordinateur (maillages synthétiques) ou reconstruits à partir de données provenant de scanners 3D (maillages reconstruits). Ces périphériques de capture reconstruisent un maillage 3D grâce à l'acquisition dans l'espace tridimensionnel de multiples points appartenant à l'objet à représenter. La Figure 1 illustre un exemple de maillage 3D synthétique et un exemple de maillage 3D dynamique.

Les maillages 3D sont rapidement devenus populaires car, étant simples, ils peuvent être facilement traités par des ordinateurs. Des composants électroniques dédiés au rendu d'images à partir de maillages 3D furent développés. Appelés unités de traitement graphique (GPU), ils sont aujourd'hui présents dans la plupart des ordinateurs personnels et des smartphones. Des interfaces de programmation d'applications (APIs) telles qu'OpenGL[1] et Microsoft Direct3D[2] ont été développées pour permettre aux logiciels d'utiliser le GPU pour calculer des images de rendu 3D.

[1] http://www.khronos.org/opengl
[2] http://msdn.microsoft.com

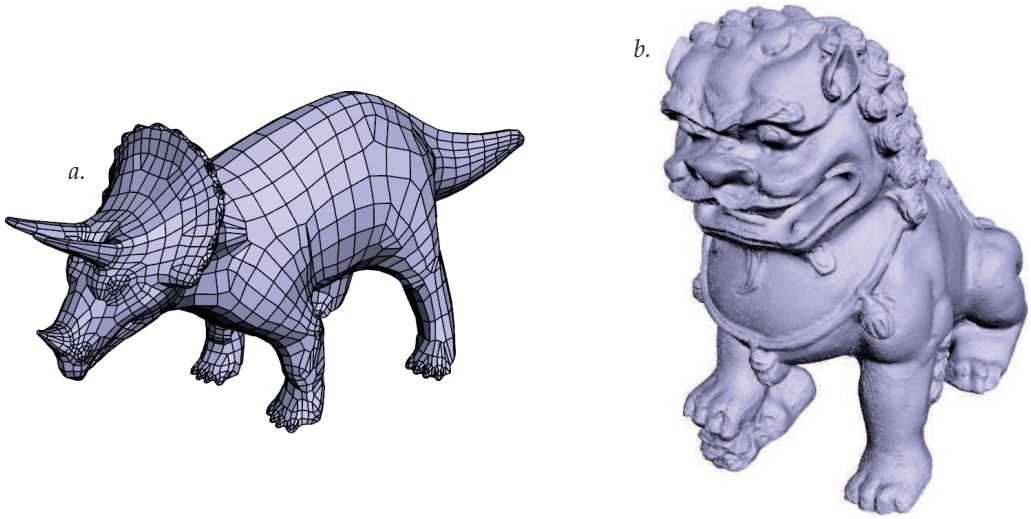


Figure 14: Exemples de maillages 3D. *a.* Un maillage synthétique *b.* Un maillage reconstruit.

Durant cette même période, les applications de simulation numériques ont accru l'usage d'importants maillages volumiques. Les maillages volumiques sont constitués de polyèdres élémentaires adjacents appelés cellules qui ensemble forment une approximation par partie du volume représenté. En effet, lorsque l'on simule un phénomène physique, certaines équations ne peuvent pas être résolues analytiquement. Pour parvenir à une solution approximative, il faut recourir à une discrétisation de l'espace à travers des maillages. Les équations sont alors résolues numériquement sur les cellules jusqu'à obtenir une convergence. Si le phénomène étudié a une nature tridimensionnelle, alors il faut utiliser un maillage volumique.

Dans cette thèse, nous nous intéressons principalement aux maillages 3D surfaciques. Ils sont aujourd'hui utilisés dans de nombreux domaines d'application pour la représentation et la visualisation d'objets 3D.

- En **Conception Assistée par Ordinateur** (CAO), les ingénieurs peuvent visualiser leurs modèles en 3D et interagir avec eux à travers leur représentation sous forme de maillage surfacique (voir Figure 15). Par exemple, Catia[3] de Dassault Systèmes est une célèbre suite logicielle qui permet de concevoir des produits industriels complexes à l'aide de vues 3D.

  Il est aussi possible de retrouver les primitives de conception d'un objet industriel en faisant de la rétro-ingénierie à partir d'acquisitions 3D. Injectées dans un logiciel de CAO, les représentations virtuelles générées peuvent ensuite être modifiées.

  Les utilisateurs ont communément accès à la réalité virtuelle à travers leur écran d'ordinateur. Mais les environnements d'affichage immersifs, tels que les caves et les murs d'images, mettent l'utilisateur au centre d'un monde virtuel. Certaines installations sont équipées d'un système de vision stéréographique, ce qui donne l'impression à l'utilisateur que la scène a une profondeur. Ces systèmes présentent de nombreux avantages pour la conception d'un nouveau produit. Par exemple, ils permettent de tester partiellement une voiture sans aucun prototypage physique coûteux.

- Certaines méthodes d'**imagerie médicale 3D** permettent de construire des représentations 3D sous forme de maillage des organes d'un patient pour faciliter le diagnostique. Ils permettent aussi la simulation d'examens médicaux ou de gestes chirurgicaux. Par exemple, dans [Kühnapfel et al., 2000], les auteurs présentent un système de réalité virtuelle pour l'entraînement à la chirurgie endoscopique.

- La **capture numérique du patrimoine** favorise l'accès aux œuvres d'art à partir de n'importe où à n'importe quelle heure. Elle permet l'accès et l'étude d'importantes pièces historiques. Koller et al. dans [Koller et al., 2010] pointent les challenges présents derrière la création d'archives 3D de patrimoines numériques.

[3] http://www.3ds.com/products/catia/welcome/

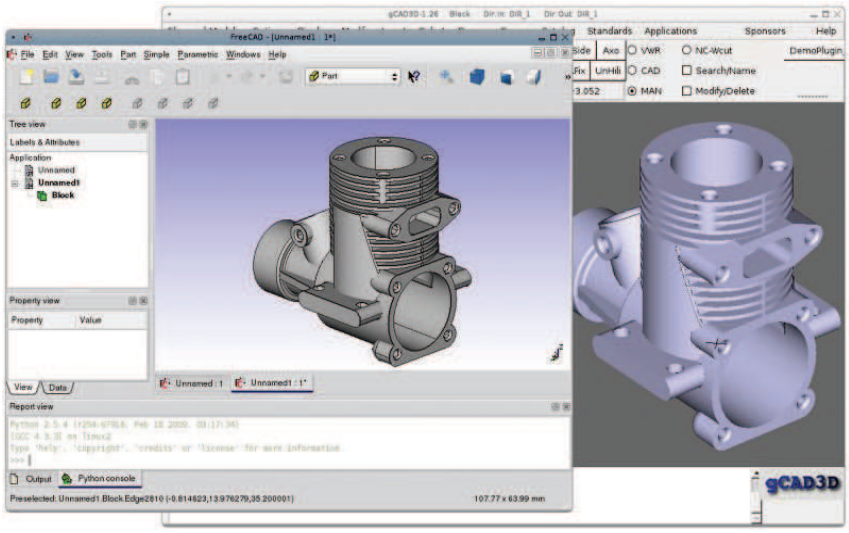


Figure 15: Modeler une pièce en 3D avec un logiciel de CAO. FreeCAD - www.freecadweb.org

- Les applications de **commerce électronique** trouvent aussi des bénéfices à fournir à l'utilisateur des répliques virtuelles interactives de produits vendus. Des entreprises telles que PackshotCreator [4] proposent déjà aujourd'hui des systèmes automatiques qui prennent des photos d'objets à partir de points de vue multiples. Sur la page web du produit, le client peut ainsi changer interactivement le point de vue de l'image affichée afin de pouvoir le visualiser sous tous ses angles.

  Avec le développement de la technologie WebGL [5], qui permet l'inclusion de contenus interactifs 3D à l'intérieur de pages web, les sites web de commerce électronique pourront inclure des représentations de leur produits plus complètes grâce aux maillages. En effet, les maillages permettent des modes de rendu et d'interactions plus variés.

  Des systèmes de réalité augmentée ont été aussi développés pour permettre aux clients d'essayer virtuellement des vêtements [Hilsmann and Eisert, 2009] ou des chaussures [Eisert et al., 2007]. Ils affichent le modèle 3D du produit testé sur la vidéo capturée du client.

- L'**industrie du divertissement** est aussi une grande productrice de maillages 3D. En effet, ces derniers sont souvent utilisés pour représenter des environnements et des personnages dans les jeux vidéos ou les films d'animation (voir Figure 3). Dans ce domaine, beaucoup de recherche sont menées pour générer des jeux et des films d'animation aussi physiquement réalistes que possible.

- Concernant la **simulation numérique**, les résultats des calculs sont souvent visualisés sous la forme de maillages surfaciques avec, par exemple, des couleurs aux nœuds ou aux cellules représentant les valeurs obtenues. On parle alors de visualisation scientifique. Un exemple de maillage de visualisation scientifique est illustré sur la Figure 4.

Le travail présenté dans cette thèse fût mené dans le contexte du projet Collaviz [Dupont et al., 2010]. Le but de ce projet était de concevoir une plate-forme collaborative pour la visualisation scientifique distante. Cette plate-forme devait permettre à des scientifiques travaillant sur des sites distants de partager, discuter et interagir avec leurs résultats de simulations effectuées sur un grappe de calcul à haute performance. Les résultats étaient sauvegardés sur un serveur central commun puis chacun des membres de la session de visualisation pouvaient demander à ce qu'une opération de post-traitement produisant un maillage surfacique soit effectuée. Ce maillage était alors transmis à chacun des clients afin qu'ils puissent le visualiser. Les utilisateurs pouvaient collaborer en partageant des annotations sur les modèles 3D ou des points de vue communs. Ils pouvaient aussi échanger à travers une messagerie instantanée. Le projet Collaviz était décomposé

[4] http://www.packshot-creator.com/
[5] http://www.khronos.org/webgl/

Figure 16: Une image du film d'animation Big Buck Bunny - Blender Foundation | www.blender.org. Les personnages du film ont été modélisés avec des maillages 3D.

en 8 sous-projets. Le but du sous-projet englobant cette thèse était de développer des solutions pour la **visualisation scientifique distante de maillages**. Nous avons donc conçu des outils pour transmettre et visualiser de manière efficace ces modèles. Les utilisateurs devaient pouvoir accéder à leurs données avec différentes conditions de réseau et de terminal. C'est pourquoi, comme expliqué dans la suite, nous avons dû recourir à des techniques d'**adaptation de données 3D**.

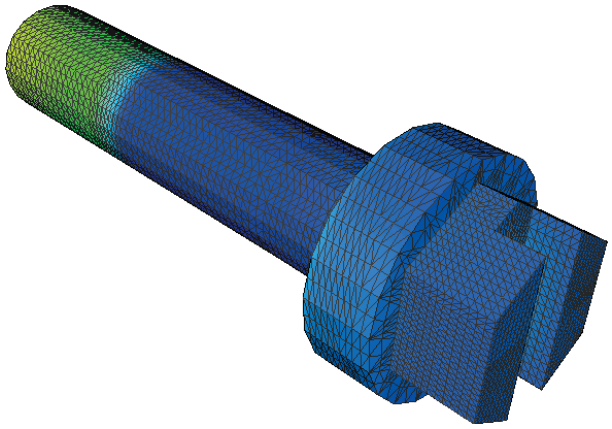


Figure 17: Un exemple de maillage de visualisation scientifique. Les couleurs aux sommets permettent de visualiser la température sur la surface de l'objet.

Dans tous les domaines d'application précédemment décrits, les besoins en terme de précision ne cessent de croître. Par exemple, les maillages générés pour la simulation possèdent un nombre croissant de cellules afin d'obtenir des résultats aussi proche que possible de la réalité. Pour d'autres domaines tels que la sauvegarde numérique du patrimoine, la précision des périphériques de capture s'améliore sans arrêt; ce qui génère des données de plus en plus importantes.

Comme un maillage 3D surfacique est une approximation discrète d'une surface composée de polygones, augmenter la précision signifie augmenter le nombre de polygones. Plus un maillage contient de polygones, plus les données pour le représenter sont importantes. Les domaines d'applications, tels que la visualisation scientifique pour le projet Collaviz, nécessitent que ces données soient stockées et parfois transmises sur un réseau. Or le stockage et la transmission de données ont un coût économique. Certains maillages

sont aujourd'hui si gros que leur stockage et leur transmission représentent un coût non négligeable. La compression de maillage peut aider à réduire ces coûts.

La diversité des périphériques utilisés pour visualiser, ou plus généralement traiter d'importants maillages, est aussi problématique. Un gros maillage de visualisation scientifique qui nécessite l'utilisation d'une grappe de calcul graphique pour être rendu ne peut être directement visualisé sur un smartphone. Entre ces deux systèmes, il y a une énorme différence de stockage, de mémoire centrale, de ressources CPU et GPU disponibles. Le même problème se pose aussi avec le réseau. La quantité de bande passante disponible sur une connexion à fibre optique et une connexion ADSL ne sont pas du même ordre. Cependant, l'utilisateur veut pouvoir accéder à ses données dans ces différentes conditions. Pour permettre l'accès universel, il est nécessaire de recourir à de l'**adaptation** de données.

L'adaptation des données 3D consiste à transformer un ensemble d'objets 3D en entrée en des modalités alternatives afin d'offrir à l'utilisateur la meilleure expérience de visualisation en fonction des caractéristiques du réseau, des capacités de son terminal ainsi que ses préférences. Les mécanismes d'adaptation permettent de transmettre de manière optimale la scène 3D. Dans la section suivante, nous expliquons pourquoi nous pensons que la compression de maillages peut être une solution efficace pour l'adaptation des données 3D.

La compression de maillages

La compression de maillages 3D a pour but d'encoder un maillage d'entrée avec moins d'octets que sa représentation initiale. L'opération de décompression complète restaure un maillage identique au maillage d'entrée (compression sans perte) ou proche de celui-ci (compression avec pertes). En réduisant leur taille, la compression de maillages permet de stocker plus facilement et transmettre plus rapidement les données. Elle peut donc s'avérer utile dans des conditions de faibles ressources. La compression adapte donc les maillages 3D aux contraintes de stockage et de transmission. Certains types d'algorithmes permettent d'accéder durant la décompression à différentes versions du maillage. Ces versions peuvent être soit différents niveaux de détail, soit différentes parties du maillage d'entrée. Celles-ci contiennent moins de polygones que le modèle d'entrée et peuvent être plus facilement visualisées sur des périphériques à faibles ressources. Dans ce cas, la compression adapte les maillages 3D aux contraintes de visualisation. Néanmoins, la compression doit être utilisée en gardant à l'esprit que la compression et la décompression sont des opérations complexes qui consomment du temps de calcul. Parfois, les effets positifs produits par la réduction de la taille de données peuvent avoir des conséquences négatives en terme de temps de calcul.

Nous classons les algorithmes de compression de maillages selon les caractéristiques qu'ils offrent durant la décompression. Nous distinguons quatre types.

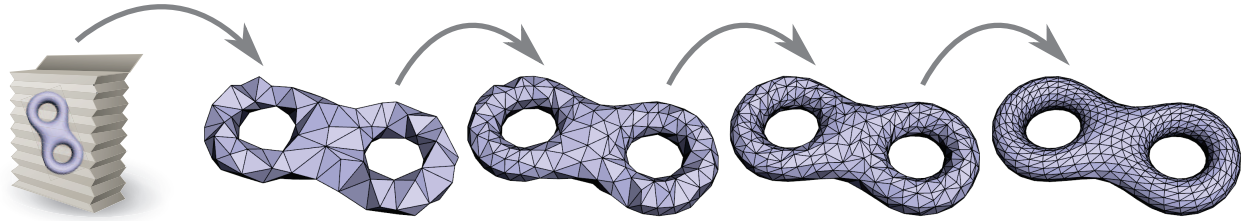
À taux unique. Les algorithmes à taux unique (voir Figure 5 *a*) sont les plus simples. L'algorithme de compression génère une représentation compacte du modèle d'entrée. L'algorithme de décompression génère un maillage qui est identique au maillage d'entré ou qui diffère légèrement. La motivation principale de ces approches est le stockage et la transmission simple.

Progressif. Les algorithmes progressifs (voir Figure 5 *b*) incorporent dans les données compressées une représentation multi-résolution du modèle d'entrée. L'algorithme de décompression reconstruit les niveaux de détail successifs au fur et à mesure que des données supplémentaires sont décodées. Ces algorithmes présentent un intérêt pour la visualisation distante car l'utilisateur n'a pas besoin d'attendre que l'ensemble des données soit téléchargées et décompressées pour pouvoir visualiser le modèle. Un autre avantage de ce type d'algorithme est qu'il est possible de sélectionner le meilleur niveau de détail selon les capacités de rendu du terminal, les contraintes réseau ou le point de vue de visualisation.

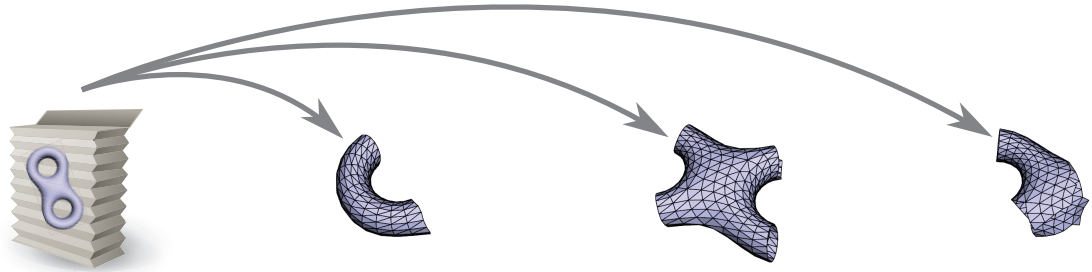
Avec les deux précédents types d'algorithmes, l'ensemble des données doit être téléchargé et décompressé pour accéder à une région d'intérêt spécifique du maillage d'entrée à la résolution originale. Dans un tel cas, l'efficacité de ces approches est fortement limitée. En téléchargeant et décompressant le modèle en entier alors que l'utilisateur a juste besoin d'une petite partie, des ressources de calcul et de réseau sont gâchées.

a. Décompression à taux unique

b. Décompression progressive



c. Décompression avec accès aléatoire



d. Décompression progressive avec accès aléatoire

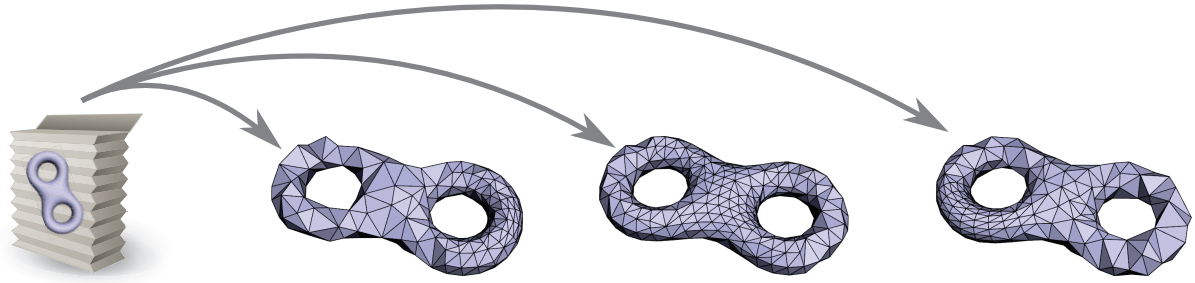


Figure 18: Les quatre types d'algorithmes de compression de maillage et les modalités qu'ils offrent durant la décompression.

De plus, l'utilisateur doit attendre des données dont il n'a pas besoin. Les deux types d'algorithmes de compression de maillages suivants permettent de répondre en partie à ces problèmes.

Avec accès aléatoire. Les algorithmes avec accès aléatoire (voir Figure 5 *c*) permettent de décompresser seulement les parties requises du maillage d'entrée. Les données qui codent les autres parties du maillage sont ignorées. Ces méthodes permettent d'accéder à des modèles qui ne tiennent pas dans la mémoire centrale du terminal. Par contre, l'utilisateur n'a aucune vue d'ensemble sur les parties du maillage non sélectionnées.

Progressif avec accès aléatoire. Les algorithmes progressifs avec accès aléatoire (voir Figure 5 *d*) combinent les caractéristiques des algorithmes progressifs et des algorithmes avec accès aléatoire. Des parties différentes du modèle d'entrée peuvent être décompressées à des niveaux de détail différents. Cela permet ainsi de décompresser un maillage en fonction du point de vue choisi par l'utilisateur. Le maillage ainsi obtenu offre une bonne expérience de visualisation à un point de vue donné avec un coût raisonnable en terme de quantité de données décompressées et de nombre de polygones.

Contributions

Le travail décrit dans cette thèse présente la compression de maillage comme une méthode pour adapter les données 3D. Par conséquent, nous avons mis l'accent sur le développement de techniques progressives et avec accès aléatoire. Après avoir effectué une étude bibliographique, nous avons développé un premier algorithme de compression progressive pour les maillages polygonaux, une métrique pour la décimation des maillages polygonaux, un cadre applicatif pour la visualisation distante ainsi que deux algorithmes de compression progressive avec accès aléatoire. Nous avons conçu chacune de ces approches en gardant en tête les qualités suivantes:

Forte performance de compression. Les données générées par l'algorithme de compression devraient être les plus petites possible pour permettre le stockage et la transmission efficaces des modèles compressés.

Bon compromis débit-distorsion. Avec une quantité minimale de données, l'algorithme de compression devrait sortir un maillage qui est le plus proche possible du maillage original.

Forte granularité d'accès aléatoire. Les approches de compression avec accès aléatoire développées devraient être capable de décompresser les parties demandées les plus petites possibles, avec le moins de surcoût possible.

Faibles temps de traitement. Les algorithmes de compression et de décompression devraient être les plus rapides possible pour ne pas rallonger le temps nécessaire pour accéder au maillage.

Efficace en mémoire. Les algorithmes de compression et de décompression devraient pouvoir fonctionner en consommant une faible quantité de mémoire. L'algorithme de décompression devrait, plus particulièrement, pouvoir être lancé sur des périphériques mobiles à faibles ressources.

Générique. Les approches proposées devraient pouvoir traiter tous les différents types de topologies de maillages, ou être facilement adaptables pour pouvoir le faire.

Parallèle. Comme l'architecture des ordinateurs modernes évolue vers de plus en plus de cœurs de calcul, les algorithmes de compression et de décompression devraient être capable d'effectuer autant d'étapes de calcul que possible en parallèle.

Facile à implémenter. Une implémentation des algorithmes proposés se reposant sur de célèbres bibliothèques tierces devrait être aussi simple que possible.

Les procédés existants de compression progressive de maillages ont été conçu pour compresser des maillages surfaciques seulement composés de faces triangulaires. Néanmoins, d'autres types de polygones sont aussi couramment utilisés dans les maillages surfaciques. Nous avons par conséquent conçu un nouvel algorithme de compression progressive de maillages que nous avons appelé **PPMC** (Progressive Polygon Mesh Compression). Cet algorithme compresse les maillages polygonaux manifolds. Partant d'une approche simple basée sur un nouvel opérateur de décimation, nous avons proposé des méthodes de prédiction de la connectivité, une formulation ondelette et une méthode de quantification adaptative pour améliorer le compromis débit-distorsion. Comme la compression progressive est liée à la décimation de maillage, nous avons aussi proposé une nouvelle métrique volumique simple pour guider la simplification des maillages polygonaux.

Les algorithmes de compression progressive tels que PPMC permettent d'adapter les données 3D aux contraintes de stockage, transmission et visualisation. Cependant, ils doivent être utilisés en conjonction avec un cadre applicatif d'adaptation qui choisit quel niveau de détail doit être téléchargé, décompressé et affiché à l'utilisateur selon ses contraintes. Nous avons donc proposé, dans le contexte du projet Collaviz, un cadre applicatif d'adaptation pour la visualisation scientifique distante de maillages. Ce dernier a été conçu pour optimiser l'expérience interactive de visualisation distante en prenant en compte les contraintes provenant du réseau (bande passante), du terminal de visualisation (mémoire, capacités de calcul et résolution de l'écran) ainsi que de l'utilisateur (distance au modèle).

Néanmoins, avec les algorithmes de compression progressive, pour accéder à une partie spécifique du maillage à la résolution maximale, le modèle entier doit être téléchargé et décompressé. Une meilleure efficacité est possible si la décompression peut s'adapter à un paramètre additionnel: la zone d'intérêt de l'utilisateur. Les algorithmes progressifs avec accès aléatoire permettent de télécharger et de décompresser différentes parties du maillage d'entrée à différent niveaux de détail. L'algorithme de décompression peut raffiner à la demande les régions d'intérêt et laisser les autres régions à un niveau de détail grossier. Cela rend possible la décompression en fonction du point de vue. Ces approches adaptent le nombre et la position des polygones décompressés en fonction de ce que l'utilisateur souhaite voir.

Dans un premier temps, nous avons conçu un algorithme de compression progressive avec accès aléatoire de maillages triangulaires basé sur une segmentation initiale du modèle d'entrée. Nous l'avons nommé **PRAM** (Progressive Random Accessible Mesh). Les groupes de faces générés sont compressés de manière indépendante par un algorithme progressif. Avec une segmentation adéquate, cette approche permet de décompresser un maillage en fonction du point de vue choisi par l'utilisateur. Par contre, elle ne permet pas de combiner une forte granularité d'accès aléatoire avec une forte performance de compression. De plus, le remplissage des trous entre des groupes partiellement décompressés reste aussi problématique.

Nous avons par conséquent conçu un second algorithme de compression progressive avec accès aléatoire de maillages triangulaires nommé **POMAR** (Progressive Oriented Mesh Accessible Randomly). Ce dernier permet un accès aléatoire beaucoup plus fin. Avec cette approche, aucune segmentation initiale n'est requise. L'algorithme de décompression génère un modèle décompressé avec des transitions lisses entre les régions décompressés à un haut et un faible niveau de détail. Tout les triangles décompressés proviennent directement de l'étape de décimation et par conséquent respectent sa métrique.

Contenu de la thèse

Dans les deux premiers chapitres, nous rappelons les préliminaires et les travaux précédents sur les maillages et les algorithmes de compression de maillages à taux unique.

- Le chapitre 1 rappelle des connaissances préliminaires relatives aux maillages 3D et à la compression de données. Il définit les maillages et leurs propriétés. Après une description générale des structures de données qui permettent de les stocker, il décrit les principes généraux de la compression de données.
- Le chapitre 2 est un état de l'art sur la compression de maillages à taux unique. En partant des travaux pionniers, nous évoquons toutes les approches pertinentes proposées pour la compression des maillages statiques.

Les quatre chapitres suivants sont dédiés aux travaux précédents et à nos contributions sur la compression progressive, la décimation et l'adaptation des maillages 3D.

- Le chapitre 3 est un état de l'art sur la compression progressive de maillages.
- Le chapitre 4 décrit PPMC, notre algorithme de compression progressive de maillages. Des résultats expérimentaux illustrent la performance de l'approche.
- Le chapitre 5 traite de la simplification de maillages polygonaux. Il présente notre nouvelle métrique simple qui peut contrôler la décimation de maillages polygonaux. Il montre aussi comment cette métrique peut dans certains cas améliorer le compromis débit-distorsion de PPMC.
- Le chapitre 6 démontre l'utilité des algorithmes de compression progressive de maillages dans un contexte de visualisation scientifique distante. Il décrit un nouveau cadre applicatif simple qui gère les contraintes multiples de ce cas d'utilisation.

Les trois derniers chapitres traitent des travaux précédents et de nos contributions concernant la compression progressive et avec accès aléatoire de maillages.

- Le chapitre 7 est un état de l'art sur la compression progressive et avec accès aléatoire de maillages.
- Le chapitre 8 décrit PRAM, notre premier algorithme de compression progressive avec accès aléatoire de maillages basé sur la segmentation du modèle initial. Des résultats expérimentaux illustrent la performance de l'approche.
- Le chapitre 9 porte sur POMAR, notre second algorithme de compression progressive avec accès aléatoire. Des résultats expérimentaux sont aussi inclus dans ce chapitre.

Enfin, cette thèse se termine par une conclusion générale qui esquisse des perspectives générales pour de futurs travaux.

Bibliography

[Ahn et al., 2006] Ahn, J.-H., Kim, C.-S., and Ho, Y.-S. (2006). Predictive compression of geometry, color and normal data of 3-d mesh models. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(2):291–299.

[Ahn et al., 2011] Ahn, J.-K., Lee, D.-Y., Ahn, M., and Kim, C.-S. (2011). R-d optimized progressive compression of 3d meshes using prioritized gate selection and curvature prediction. *The Visual Computer*, 27:769–779.

[Alliez et al., 2003] Alliez, P., Cohen-Steiner, D., Devillers, O., Lévy, B., and Desbrun, M. (2003). Anisotropic polygonal remeshing. In *Proceedings of SIGGRAPH*, pages 485–493.

[Alliez and Desbrun, 2001a] Alliez, P. and Desbrun, M. (2001a). Progressive compression for lossless transmission of triangle meshes. In *Proceedings of SIGGRAPH*, pages 195–202.

[Alliez and Desbrun, 2001b] Alliez, P. and Desbrun, M. (2001b). Valence-driven connectivity encoding for 3d meshes. *Computer Graphics Forum*, 20(3):480–489.

[Alliez and Gotsman, 2005] Alliez, P. and Gotsman, C. (2005). Recent advances in compression of 3d meshes. In *Advances in Multiresolution for Geometric Modelling*, Mathematics and Visualization, pages 3–26.

[Arikawa et al., 1996] Arikawa, M., Amano, A., Maeda, K., Aibara, R., Shimojo, S., Nakamura, Y., Hiraki, K., Nishimura, K., and Terauchi, M. (1996). Dynamic lod for qos management in the next generation vrml. In *Proceedings of the Third IEEE International Conference on Multimedia Computing and Systems*, pages 24 –27.

[Attene et al., 2003] Attene, M., Falcidieno, B., Spagnuolo, M., and Rossignac, J. (2003). Swingwrapper: Retiling triangle meshes for better edgebreaker compression. *ACM Transactions on Graphics*, 22(4):982–996.

[Aviles and Moran, 2008] Aviles, M. and Moran, F. (2008). Static 3d triangle mesh compression overview. In *Proceedings of the IEEE International Conference on Image Processing*, pages 2684–2687.

[Bajaj et al., 1999] Bajaj, C., Pascucci, V., and Zhuang, G. (1999). Single-resolution compression of arbitrary triangular meshes with properties. In *Proceedings of the Data Compression Conference*, pages 247–256.

[Baumgart, 1975] Baumgart, B. G. (1975). A polyhedron representation for computer vision. In *Proceedings of the May 19-22, 1975, national computer conference and exposition*, AFIPS '75, pages 589–596.

[Bayazit et al., 2010] Bayazit, U., Konur, U., and Ates, H. (2010). 3-d mesh geometry compression with set partitioning in the spectral domain. *Circuits and Systems for Video Technology, IEEE Transactions on*, 20(2):179–188.

[Bayazit et al., 2007] Bayazit, U., Orcay, O., Konur, U., and Gurgen, F. S. (2007). Predictive vector quantization of 3-d mesh geometry by representation of vertices in local coordinate systems. *Journal of Visual Communication and Image Representation*, 18(4):341–353.

[Ben-Chen and Gotsman, 2005] Ben-Chen, M. and Gotsman, C. (2005). On the optimality of spectral compression of mesh data. *ACM Transactions on Graphics*, 24(1):60–80.

[Berjón et al., 2013] Berjón, D., Morán, F., and Manjunatha, S. (2013). Objective and subjective evaluation of static 3d mesh compression. *Signal Processing: Image Communication*, 28(2):181–195.

[Bischoff and Kobbelt, 2002] Bischoff, S. and Kobbelt, L. (2002). Towards robust broadcasting of geometry data. *Computers & Graphics*, 26(5):665–675.

[Botsch et al., 2002] Botsch, M., Steinberg, S., Bischoff, S., and Kobbelt, L. (2002). Open-mesh a generic and efficient polygon mesh data structure. In *OpenSG Symposium*. http://www.openmesh.org/.

[Cai et al., 2009] Cai, K., Jin, Y., Wang, W., Chen, Q., Chen, Z., and Teng, J. (2009). Compression of massive models by efficiently exploiting repeated patterns. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 229–230.

[Cai et al., 2007] Cai, S., Qi, Y., and Shen, X. (2007). 3d data codec and transmission over the internet. In *Web3D '07: Proceedings of the twelfth international conference on 3D web technology*, pages 53–56.

[Cayre et al., 2003] Cayre, F., Rondao-Alface, P., Schmitt, F., Macq, B., and Maître, H. (2003). Application of spectral decomposition to compression and watermarking of 3d triangle mesh geometry. *Signal Processing: Image Communication*, 18(4):309 – 319.

[Cellier et al., 2012] Cellier, F., Gandoin, P.-M., Chaine, R., Barbier-Accary, A., and Akkouche, S. (2012). Simplification and streaming of gis terrain for web clients. In *Web3D '12: Proceedings of the 17th International Conference on 3D Web Technology*, pages 73–81.

[Chaine et al., 2007] Chaine, R., Gandoin, P.-M., and Roudet, C. (2007). Mesh connectivity compression using convection reconstruction. In *Proceedings of the ACM Symposium on Solid and physical modeling*, pages 41–49.

[Chaine et al., 2009] Chaine, R., Gandoin, P.-M., and Roudet, C. (2009). Reconstruction algorithms as a suitable basis for mesh connectivity compression. *Automation Science and Engineering, IEEE Transactions on*, 6(3):443–453.

[Chen and Nishita, 2002] Chen, B.-Y. and Nishita, T. (2002). Multiresolution streaming mesh with shape preserving and qos-like controlling. In *Web3D '02: Proceedings of the seventh international conference on 3D Web technology*, pages 35–42.

[Chen et al., 2005] Chen, D., Chiang, Y.-J., Memon, N., and Wu, X. (2005). Geometry compression of tetrahedral meshes using optimized prediction. In *Proceedings of the European Conference on Signal Processing*.

[Chen and Georganas, 2008] Chen, L. and Georganas, N. D. (2008). Region-based 3d mesh compression using an efficient neighborhood-based segmentation. *Simulation*, 84(5):185–195.

[Chen et al., 2008] Chen, R., Luo, X., and Xu, H. (2008). Geometric compression of a quadrilateral mesh. *Computers & Mathematics with Applications*, 56(6):1597 – 1603.

[Chen et al., 2010] Chen, Z., Bao, F., Fang, Z., and Li, Z. (2010). Compression of 3d triangle meshes with a generalized parallelogram prediction scheme based on vector quantization. In *Proceedings of the International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 184–187.

[Chen et al., 2003] Chen, Z., Bodenheimer, B., and Barnes, J. F. (2003). Robust transmission of 3d geometry over lossy networks. In *Web3D '03: Proceedings of the eighth international conference on 3D Web technology*, page 161.

[Cheng and Basu, 2007] Cheng, I. and Basu, A. (2007). Perceptually optimized 3-d transmission over wireless networks. *IEEE Transactions on Multimedia*, 9(2):386–396.

[Cheng and Ooi, 2008] Cheng, W. and Ooi, W. T. (2008). Receiver-driven view-dependent streaming of progressive mesh. In *NOSSDAV '08: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 9–14.

[Cheng et al., 2007a] Cheng, W., Ooi, W. T., Mondet, S., Grigoras, R., and Morin, G. (2007a). An analytical model for progressive mesh streaming. In *MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia*, pages 737–746.

[Cheng et al., 2006] Cheng, Z., Jin, S., and Liu, H. (2006). Anchors-based lossless compression of progressive triangle meshes. In *Proceedings of Pacific Graphics*, pages 45–50.

[Cheng et al., 2007b] Cheng, Z.-Q., Liu, H.-F., and Jin, S.-Y. (2007b). The progressive mesh compression based on meaningful segmentation. *The Visual Computer*, 23(9-11):651–660.

[Choe et al., 2009] Choe, S., Kim, J., Lee, H., and Lee, S. (2009). Random accessible mesh compression using mesh chartification. *IEEE Transactions on Visualization and Computer Graphics*, 15(1):160–173.

[Choe et al., 2004] Choe, S., Kim, J., Lee, H., Lee, S., and Seidel, H.-P. (2004). Mesh compression with random accessibility. In *Proceedings of the 5th Korea-Israel Bi-National Conference on Geometric Modeling and Computer Graphics*, pages 81–86.

[Chou and Meng, 2002] Chou, P. and Meng, T. (2002). Vertex data compression through vector quantization. *Visualization and Computer Graphics, IEEE Transactions on*, 8(4):373–382.

[Chourou et al., 2008] Chourou, A., Antonini, M., and Benazza-Benyahia, A. (2008). 3d mesh coding through region based segmentation. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1381 –1384.

[Chow, 1997] Chow, M. (1997). Optimized geometry compression for real-time rendering. In *Proceedings of Visualization*, pages 347–354.

[Chuang et al., 1998] Chuang, R.-N., Garg, A., He, X., Kao, M.-Y., and Lu, H.-I. (1998). Compact encodings of planar graphs via canonical orderings and multiple parentheses. In *Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 118–129.

[Cignoni et al., 2004] Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. (2004). Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. In *Proceedings of SIGGRAPH*, pages 796–803.

[Cignoni et al., 2005] Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., and Scopigno, R. (2005). Batched multi triangulation. In *Proceedings of Visualization*, pages 207–214.

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

[Cirio et al., 2010] Cirio, G., Lavoue, G., and Dupont, F. (2010). A framework for data-driven progressive mesh compression. In *Proceedings of the International Conference on Computer Graphics Theory and Applications*, Lecture Notes on Computer Science.

[Cohen-Or et al., 1999] Cohen-Or, D., Levin, D., and Remez, O. (1999). Progressive compression of arbitrary triangular meshes. In *Proceedings of the IEEE Visualization Conference*.

[Cohen-Or et al., 2002] Cohen-Or, D., Rami, C., and Irony, R. (2002). Multi-way geometry encoding. Technical report, The School of Computer Science, Tel-Aviv University.

[Cohen-Steiner et al., 2004] Cohen-Steiner, D., Alliez, P., and Desbrun, M. (2004). Variational shape approximation. In *Proceedings of SIGGRAPH*, pages 905–914.

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

[Corsini et al., 2012] Corsini, M., Larabi, M. C., Lavoué, G., Petřík, O., Váša, L., and Wang, K. (2012). Perceptual metrics for static and dynamic triangle meshes. *Computer Graphics Forum*, pages no–no.

[Courbet and Hudelot, 2009] Courbet, C. and Hudelot, C. (2009). Random accessible hierarchical mesh compression for interactive visualization. In *Proceedings of the Symposium on Geometry Processing*, pages 1311–1318.

[Courbet and Hudelot, 2011] Courbet, C. and Hudelot, C. (2011). Taylor prediction for mesh geometry compression. *Computer Graphics Forum*, 30(1):139–151.

[Courbet and Isenburg, 2010] Courbet, C. and Isenburg, M. (2010). Streaming compression of hexahedral meshes. *The Visual Computer*, 26:1113–1122.

[Daniels et al., 2009] Daniels, J., Silva, C. T., and Cohen, E. (2009). Localized quadrilateral coarsening. *Computer Graphics Forum*, 28(5):1437–1444.

[D'Azevedo, 2000] D'Azevedo, E. (2000). Are bilinear quadrilaterals better than linear triangles? *SIAM Journal on Scientific Computing*, 22(1):198–217.

[Deb and Narayanan, 2004] Deb, S. and Narayanan, P. (2004). Design of a geometry streaming system. In *Proceedings of ICVGIP*, pages 296–301.

[Deering, 1995] Deering, M. (1995). Geometry compression. In *Proceedings of SIGGRAPH*, pages 13–20.

[Denis et al., 2010] Denis, L., Satti, S., Munteanu, A., Cornelis, J., and Schelkens, P. (2010). Scalable intraband and composite wavelet-based coding of semiregular meshes. *IEEE Transactions on Multimedia*, 12(8):773–789.

[Derzapf and Guthe, 2012] Derzapf, E. and Guthe, M. (2012). Dependency-free parallel progressive meshes. *Computer Graphics Forum*, pages 2288–2302.

[Diaz-Gutierrez et al., 2005] Diaz-Gutierrez, P., Gopi, M., and Pajarola, R. (2005). Hierarchyless simplification, stripification and compression of triangulated two-manifolds. *Computer Graphics Forum*, 24(3):457–467.

[Du et al., 2009] Du, Z., Jaromersky, P., Chiang, Y.-J., and Memon, N. (2009). Out-of-core progressive lossless compression and selective decompression of large triangle meshes. In *Proceedings of the Data Compression Conference*, pages 420 –429.

[Dupont et al., 2010] Dupont, F., Duval, T., Fleury, C., Forest, J., Gouranton, V., Lando, P., Laurent, T., Lavoué, G., and Schmutz, A. (2010). Collaborative scientific visualization: The collaviz framework. In *Proceedings of the Joint Virtual Reality Conference of EuroVR - EGVE - VEC*.

[Dyn et al., 1990] Dyn, N., Levine, D., and Gregory, J. A. (1990). A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169.

[Edelsbrunner, 2001] Edelsbrunner, H. (2001). *Geometry and Topology for Mesh Generation*. Cambridge University Press.

[Edelsbrunner and Harer, 2010] Edelsbrunner, H. and Harer, J. (2010). *Computational Topology: An Introduction*. Applied mathematics. American Mathematical Society.

[Eisert et al., 2007] Eisert, P., Rurainsky, J., and Fechteler, P. (2007). Virtual mirror: Real-time tracking of shoes in augmented reality environments. In *Proceedings of the IEEE International Conference on Image Processing*, volume 2, pages 557–560.

[Evans et al., 1996a] Evans, F., Skiena, S., and Varshney, A. (1996a). Completing sequential triangulations is hard. Technical report, Department of Computer Science, State University of New York at Stony Brook.

[Evans et al., 1996b] Evans, F., Skiena, S., and Varshney, A. (1996b). Optimizing triangle strips for fast rendering. In *Proceedings of Visualization*, pages 319–326.

[Fathy et al., 2011] Fathy, G., Hassen, H., Gamal, R., and Sheta, W. (2011). Dynamic transmission of 3d mesh in wireless walkthrough applications. In *Proceedings of the IEEE International Symposium on Signal Processing and Information Technology*, pages 71–79.

[Flato et al., 1999] Flato, E., Halperin, D., Hanniel, I., and Nechushtan, O. (1999). The design and implementation of planar maps in cgal. In *Algorithm Engineering*, volume 1668 of *Lecture Notes in Computer Science*, pages 154–168. http://www.cgal.org/.

[Fogel et al., 2001] Fogel, E., Cohen-Or, D., Ironi, R., and Zvi, T. (2001). A web architecture for progressive delivery of 3d content. In *Web3D '01: Proceedings of the sixth international conference on 3D Web technology*, pages 35–41.

[Funkhouser and Séquin, 1993] Funkhouser, T. A. and Séquin, C. H. (1993). Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of SIGGRAPH*, pages 247–254.

[Gandoin and Devillers, 2002] Gandoin, P.-M. and Devillers, O. (2002). Progressive lossless compression of arbitrary simplicial complexes. In *Proceedings of SIGGRAPH*, pages 372–379.

[Gang et al., 2010] Gang, D., Zhi-quan, C., Jingwen, Z., Liang, L., and Shiyao, J. (2010). An improved progressive lossless compression algorithm. In *International Conference on Audio Language and Image Processing*, pages 249–253.

[Garland and Heckbert, 1997] Garland, M. and Heckbert, P. S. (1997). Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH*, pages 209–216.

[Gioia et al., 2004] Gioia, P., Aubault, O., and Bouville, C. (2004). Real-time reconstruction of wavelet-encoded meshes for view-dependent transmission and visualization. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(7):1009–1020.

[Gobbetti and Bouvier, 1999] Gobbetti, E. and Bouvier, E. (1999). Time-critical multiresolution scene rendering. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 123–130.

[Gu et al., 2002] Gu, X., Gortler, S. J., and Hoppe, H. (2002). Geometry images. In *Proceedings of SIGGRAPH*, pages 355–361.

[Gumhold, 2000] Gumhold, S. (2000). New bounds on the encoding of planar triangulations. Technical Report WSI-2000-1, University of Tübingen.

[Gumhold, 2005] Gumhold, S. (2005). Optimizing markov models with applications to triangular connectivity coding. In *Proceedings of the annual ACM-SIAM symposium on Discrete algorithms*, pages 331–338.

[Gumhold and Amjoun, 2003] Gumhold, S. and Amjoun, R. (2003). Higher order prediction for geometry compression. In *Proceedings of Shape Modeling International, 2003*, pages 59–66.

[Gumhold et al., 1999] Gumhold, S., Guthe, S., and Straßer, W. (1999). Tetrahedral mesh compression with the cut-border machine. In *Proceedings of the conference on Visualization '99: celebrating ten years*, VIS '99, pages 51–58.

[Gumhold and Straßer, 1998] Gumhold, S. and Straßer, W. (1998). Real time compression of triangle mesh connectivity. In *Proceedings of SIGGRAPH*, pages 133–140.

[Guskov et al., 2000] Guskov, I., Vidimče, K., Sweldens, W., and Schröder, P. (2000). Normal meshes. In *Proceedings of SIGGRAPH*, pages 95–102.

[Guthe et al., 2003] Guthe, M., Borodin, P., and Klein, R. (2003). Efficient view-dependent out-of-core visualization. In *Proceedings of the 4th International Conference on Virtual Reality and its Application in Industry*, pages 428–438.

[Guéziec et al., 1998] Guéziec, A., Taubin, G., Lazarus, F., and Horn, W. (1998). Simplicial maps for progressive transmission of polygonal surfaces. In *Proceedings of the third symposium on Virtual reality modeling language*, pages 25–31.

[Hilsmann and Eisert, 2009] Hilsmann, A. and Eisert, P. (2009). Tracking and retexturing cloth for real-time virtual clothing applications. In *Computer Vision/Computer Graphics Collaboration Techniques*, volume 5496 of *Lecture Notes in Computer Science*, pages 94–105.

[Ho et al., 2001] Ho, J., Lee, K. C., and Kriegman, D. (2001). Compressing large polygonal models. In *Proceedings of the conference on Visualization '01*, VIS '01, pages 357–362.

[Hongnian et al., 2009] Hongnian, L., Bo, L., and Hongbin, Z. (2009). Progressive geometry-driven compression for triangle mesh based on binary tree. In *Proceedings of International Conference in Visualisation*, pages 229–234.

[Hoppe, 1996] Hoppe, H. (1996). Progressive meshes. In *Proceedings of SIGGRAPH*, pages 99–108.

[Hoppe, 1997] Hoppe, H. (1997). View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH*, pages 189–198.

[Hoppe, 1998] Hoppe, H. (1998). Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings of Visualization*, pages 35–42.

[Hoppe and Praun, 2005] Hoppe, H. and Praun, E. (2005). Shape compression using spherical geometry images. In *Advances in Multiresolution for Geometric Modelling*, Mathematics and Visualization, pages 27–46.

[Hu et al., 2009] Hu, L., Sander, P. V., and Hoppe, H. (2009). Parallel view-dependent refinement of progressive meshes. In *Proceedings of the symposium on Interactive 3D graphics and games*, pages 169–176.

[Huffman, 1952] Huffman, D. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

[Isenburg, 2002] Isenburg, M. (2002). Compressing polygon mesh connectivity with degree duality prediction. In *Graphics Interface Conference Proceedings*.

[Isenburg and Alliez, 2002a] Isenburg, M. and Alliez, P. (2002a). Compressing hexahedral volume meshes. In *Proceedings of Pacific Graphics*, pages 284–293.

[Isenburg and Alliez, 2002b] Isenburg, M. and Alliez, P. (2002b). Compressing polygon mesh geometry with parallelogram prediction. In *Proceedings of the conference on Visualization '02*, VIS '02, pages 141–146.

[Isenburg and Gumhold, 2003] Isenburg, M. and Gumhold, S. (2003). Out-of-core compression for gigantic polygon meshes. In *Proceedings of SIGGRAPH*, pages 935–942.

[Isenburg et al., 2005a] Isenburg, M., Ivrissimtzis, I., Gumhold, S., and Seidel, H.-P. (2005a). Geometry prediction for high degree polygons. In *Proceedings of the Spring Conference on Computer Graphics*, pages 147–152.

[Isenburg and Lindstrom, 2005] Isenburg, M. and Lindstrom, P. (2005). Streaming meshes. *Proceedings of Visualization*, pages 231–238.

[Isenburg et al., 2006] Isenburg, M., Lindstrom, P., Gumhold, S., and Shewchuk, J. (2006). Streaming compression of tetrahedral volume meshes. In *Proceedings of Graphics Interface*, pages 115–121.

[Isenburg et al., 2005b] Isenburg, M., Lindstrom, P., and Snoeyink, J. (2005b). Lossless compression of predicted floating-point geometry. *Computer-Aided Design*, 37(8):869–877.

[Isenburg et al., 2005c] Isenburg, M., Lindstrom, P., and Snoeyink, J. (2005c). Streaming compression of triangle meshes. In *Proceedings of the Eurographics Symposium on Geometry Processing*, pages 111–118.

[Isenburg and Snoeyink, 2000a] Isenburg, M. and Snoeyink, J. (2000a). Face fixer: compressing polygon meshes with properties. In *Proceedings of SIGGRAPH*, pages 263–270.

[Isenburg and Snoeyink, 2000b] Isenburg, M. and Snoeyink, J. (2000b). Spirale reversi: Reverse decoding of edgebreaker encoding. In *Proceedings of the 12th Canadian Conference on Computational Geometry*.

[Jamin et al., 2009] Jamin, C., Gandoin, P.-M., and Akkouche, S. (2009). Chumi viewer: Compressive huge mesh interactive viewer. *Computers & Graphics*, 33(4):542 – 553.

[Jessl et al., 2005] Jessl, J., Bertram, M., and Hagen, H. (2005). Web-based progressive geometry transmission using subdivision-surface wavelets. In *Web3D '05: Proceedings of the tenth international conference on 3D Web technology*, pages 29–35.

[Jong et al., 2005] Jong, B.-S., Yang, W.-H., Tseng, J.-L., and Lin, T.-W. (2005). An efficient connectivity compression for triangular meshes. In *Proceedings of the International Conference on Computer and Information Science*, pages 583–588.

[Kammoun et al., 2011] Kammoun, A., Payan, F., and Antonini, M. (2011). Optimized butterfly-based lifting scheme for semi-regular meshes. In *IEEE International Conference on Image Processing*, pages 1269–1272.

[Kammoun et al., 2012] Kammoun, A., Payan, F., and Antonini, M. (2012). Sparsity-based optimization of two lifting-based wavelet transforms for semi-regular mesh compression. *Computers & Graphics*, 36(4):272–282.

[Karni et al., 2002] Karni, Z., Bogomjakov, A., and Gotsman, C. (2002). Efficient compression and rendering of multi-resolution meshes. In *Proceedings of the conference on Visualization*, pages 347–354.

[Karni and Gotsman, 2000] Karni, Z. and Gotsman, C. (2000). Spectral compression of mesh geometry. In *Proceedings of SIGGRAPH*, pages 279–286.

[Karni and Gotsman, 2001] Karni, Z. and Gotsman, C. (2001). 3d mesh compression using fixed spectral bases. In *Proceedings of Graphics interface*, pages 1–8.

[Keeler and Westbrook, 1995] Keeler, K. and Westbrook, J. (1995). Short encodings of planar graphs and maps. *Discrete Applied Mathematics*, 58(3):239 – 252.

[Khodakovsky et al., 2002] Khodakovsky, A., Alliez, P., Desbrun, M., and Schröder, P. (2002). Near-optimal connectivity encoding of 2-manifold polygon meshes. *Graphical Models*, 64:147–168.

[Khodakovsky and Guskov, 2003] Khodakovsky, A. and Guskov, I. (2003). Compression of normal meshes. In *Geometric Modeling For Scientific Visualization*, pages 189–206.

[Khodakovsky et al., 2000] Khodakovsky, A., Schröder, P., and Sweldens, W. (2000). Progressive geometry compression. In *Proceedings of SIGGRAPH*, pages 271–278.

[Kim et al., 2006] Kim, J., Choe, S., and Lee, S. (2006). Multiresolution random accessible mesh compression. *Computer Graphics Forum*, 25(3):323–331.

[Kim and Lee, 2001] Kim, J. and Lee, S. (2001). Truly selective refinement of progressive meshes. In *Proceedings of Graphics interface*, pages 101–110.

[Kim et al., 2011] Kim, J., Nam, C., and Choe, S. (2011). Bayesian ad coder: Mesh-aware valence coding for multiresolution meshes. *Computers & Graphics*, 35(3):713–718.

[Kim et al., 2010] Kim, T.-J., Moon, B., Kim, D., and Yoon, S.-E. (2010). Racbvhs: Random-accessible compressed bounding volume hierarchies. *Visualization and Computer Graphics, IEEE Transactions on*, 16(2):273–286.

[King and Rossignac, 1999a] King, D. and Rossignac, J. (1999a). Guaranteed 3.67 v bit encoding of planar triangle graphs. In *Proceedings of the 11th Canadian Conference on Computational Geometry*.

[King and Rossignac, 1999b] King, D. and Rossignac, J. (1999b). Optimal bit allocation in compressed 3d models. *Computational Geometry: Theory and Applications*, 14:91–118.

[King et al., 1999] King, D., Szymczak, A., and Rossignac, J. R. (1999). Connectivity compression for irregular quadrilateral meshes. *GVU Technical Report GIT-GVU-99-36*.

[Koller et al., 2010] Koller, D., Frischer, B., and Humphreys, G. (2010). Research challenges for digital archives of 3d cultural heritage models. *Journal on Computing and Cultural Heritage*, 2(3):7:1–7:17.

[Konur et al., 2008] Konur, U., Bayazit, U., Ates, H. F., and Gürgen, F. S. (2008). Spectral coding of mesh geometry with a hierarchical set partitioning algorithm. pages 682227–682227–8.

[Kovacevic and Sweldens, 2000] Kovacevic, J. and Sweldens, W. (2000). Wavelet families of increasing order in arbitrary dimensions. *IEEE Transactions on Image Processing*, 9(3):480–496.

[Krivograd et al., 2006] Krivograd, S., Trlep, M., and Žalik, B. (2006). A compression method for fem quadrilateral data. In *Proceedings of the 6th WSEAS international conference on Applied computer science*, pages 402–407.

[Krivograd et al., 2008] Krivograd, S., Trlep, M., and Žalik, B. (2008). A hexahedral mesh connectivity compression with vertex degrees. *Computer-Aided Design*, 40(12):1105–1112.

[Kronrod and Gotsman, 2000] Kronrod, B. and Gotsman, C. (2000). Efficient coding of non-triangular mesh connectivity. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, page 235.

[Kronrod and Gotsman, 2002] Kronrod, B. and Gotsman, C. (2002). Optimized compression of triangle mesh geometry using prediction trees. In *Proceedings of the International Symposium on 3D Data Processing Visualization and Transmission*, pages 602–608.

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

[Kühnapfel et al., 2000] Kühnapfel, U., Çakmak, H., and Maaß, H. (2000). Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers & Graphics*, 24(5):671–682.

[Lavoué, 2011] Lavoué, G. (2011). A multiscale metric for 3d mesh visual quality assessment. *Computer Graphics Forum*, 30(5):1427–1437.

[Lavu et al., 2003] Lavu, S., Choi, H., and Baraniuk, R. (2003). Geometry compression of normal meshes using rate-distortion algorithms. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 52–61.

[Ledoux and Shepherd, 2010] Ledoux, F. and Shepherd, J. (2010). Topological and geometrical properties of hexahedral meshes. *Engineering with Computers*, 26:419–432.

[Lee et al., 1998] Lee, A. W. F., Sweldens, W., Schröder, P., Cowsar, L., and Dobkin, D. (1998). Maps: multiresolution adaptive parameterization of surfaces. In *Proceedings of SIGGRAPH*, pages 95–104.

[Lee et al., 2011] Lee, D.-Y., Ahn, J.-K., Ahn, M., Kim, J., Kim, C., and Kim, C.-S. (2011). 3d mesh compression based on dual-ring prediction and mmse prediction. In *Proceedings of the IEEE International Conference on Image Processing*, pages 905–908.

[Lee et al., 2013] Lee, D.-Y., Sull, S., and Kim, C.-S. (2013). Progressive 3d mesh compression using mog-based bayesian entropy coding and gradual prediction. *The Visual Computer*, pages 1–15.

[Lee and Ko, 2000] Lee, E.-S. and Ko, H.-S. (2000). Vertex data compression for triangular meshes. In *Proceedings of Pacific Graphics*, pages 225 –234.

[Lee et al., 2002] Lee, H., Alliez, P., and Desbrun, M. (2002). Angle-analyzer: A triangle-quad mesh codec. *Computer Graphics Forum*, 21(3):383–392.

[Lee et al., 2009] Lee, H., Lavoué, G., and Dupont, F. (2009). Adaptive coarse-to-fine quantization for optimizing rate-distortion of progressive mesh compression. In *Vision, Modeling, and Visualization Workshop*.

[Lee et al., 2012] Lee, H., Lavoué, G., and Dupont, F. (2012). Rate-distortion optimization for progressive compression of 3d mesh with color attributes. *The Visual Computer*, 28:137–153.

[Lee and Park, 2005] Lee, H. and Park, S. (2005). Adaptive vertex chasing for the lossless geometry coding of 3d meshes. In *Advances in Multimedia Information Processing - PCM 2005*, volume 3767 of *Lecture Notes in Computer Science*, pages 108–119.

[Lewiner et al., 2005] Lewiner, T., Craizer, M., Lopes, H., Pesco, S., Velho, L., and Medeiros, E. (2005). Gencode: Geometry-driven compression in arbitrary dimension and co-dimension. In *Proceedings of the Brazilian Symposium on Computer Graphics and Image Processing*, pages 249–256.

[Lewiner et al., 2006] Lewiner, T., Craizer, M., Lopes, H., Pesco, S., Velho, L., and Medeiros, E. (2006). Gencode: Geometry-driven compression for general meshes. *Computer Graphics Forum*, 25(4):685–695.

[Li and Fan, 2010] Li, J. and Fan, H. (2010). Progressive 3d model compression based on surfacelet. In *Entertainment for Education. Digital Techniques and Systems*, volume 6249 of *Lecture Notes in Computer Science*, pages 582–591.

[Li and Kuo, 1998a] Li, J. and Kuo, C.-C. (1998a). A dual graph approach to 3d triangular mesh compression. In *Proceedings of the International Conference on Image Processing*, volume 2, pages 891–894.

[Li and Kuo, 1998b] Li, J. and Kuo, C.-C. J. (1998b). Progressive coding of 3-d graphic models. In *Proceedings of the IEEE*, volume 86.

[Li et al., 2006] Li, J., Tian, D., and AlRegib, G. (2006). Vector quantization in multiresolution mesh compression. *IEEE Signal Processing Letters*, 13(10):616–619.

[Li et al., 2007] Li, Z., Lu, Z.-M., and Sun, L. (2007). Dynamic extended codebook based vector quantization scheme for mesh geometry compression. In *Proceedgins of the International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, volume 1, pages 178–181.

[Lindstrom and Isenburg, 2008] Lindstrom, P. and Isenburg, M. (2008). Lossless compression of hexahedral meshes. In *Proceedings of the Data Compression Conference*, pages 192–201.

[Liu and bin Zhang, 2004] Liu, B. and bin Zhang, H. (2004). Wavelet based progressive mesh compression with random accessibility. In *Proceedings of 2004 International Symposium on Intelligent Multimedia, Video and Speech Processing*, pages 615–618.

[Liu et al., 2007] Liu, Y., Liu, X., and Wu, E. (2007). Variable code-mode based connectivity compression for triangular meshes. In *Proceedings of the IEEE International Conference on Computer-Aided Design and Computer Graphics*, pages 276–281.

[Liu and Wu, 2006] Liu, Y. and Wu, E. (2006). Connectivity compression for non-triangular meshes by context-based arithmetic coding. In *Proceedings of the International conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 417–424.

[Lloyd, 1982] Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137.

[Loop, 1987] Loop, C. (1987). Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, Department of Mathematics.

[Lu and Li, 2008] Lu, Z.-M. and Li, Z. (2008). Dynamically restricted codebook-based vector quantisation scheme for mesh geometry compression. *Signal, Image and Video Processing*, 2:251–260.

[Luo and Zheng, 2008] Luo, X. and Zheng, G. (2008). Progressive meshes transmission over a wired-to-wireless network. *Wireless Networks*, 14:47–53.

[Lévy and Liu, 2010] Lévy, B. and Liu, Y. (2010). Lp centroidal voronoi tessellation and its applications. In *Proceedings of SIGGRAPH*, pages 119:1–119:11.

[Ma et al., 2009] Ma, J., Chen, Q., Chen, B., and Wang, H. (2009). Mobile 3d graphics compression for progressive transmission over wireless network. In *Proceedings of the IEEE International Conference on Computer-Aided Design and Computer Graphics*, pages 357–362.

[Maglo et al., 2012] Maglo, A., Courbet, C., Alliez, P., and Hudelot, C. (2012). Progressive compression of manifold polygon meshes. *Computers & Graphics*, 36(5):349–359. Shape Modeling International (SMI) Conference 2012.

[Maglo et al., 2011] Maglo, A., Grimstead, I., and Hudelot, C. (2011). Cluster-based random accessible and progressive lossless compression of colored triangular meshes for interactive visualization. In *Proceedings of Computer Graphics International*.

[Maglo et al., 2013] Maglo, A., Grimstead, I., and Hudelot, C. (2013). Pomar: Compression of progressive oriented meshes accessible randomly. *Computers & Graphics*. Shape Modeling International (SMI) Conference 2013 - Accepted with minor revisions.

[Maglo et al., 2010] Maglo, A., Lee, H., Lavoué, G., Mouton, C., Hudelot, C., and Dupont, F. (2010). Remote scientific visualization of progressive 3d meshes with x3d. In *Web3D '10: Proceedings of the 15th International Conference on Web 3D Technology*, pages 109–116.

[Mahadevan, 2007] Mahadevan, S. (2007). Adaptive mesh compression in 3d computer graphics using multiscale manifold learning. In *Proceedings of the international conference on Machine learning*, pages 585–592.

[Mamou et al., 2010] Mamou, K., Dehais, C., Chaieb, F., and Ghorbel, F. (2010). Shape approximation for efficient progressive mesh compression. In *Proceedings of the IEEE International Conference on Image Processing*.

[Mamou et al., 2009] Mamou, K., Zaharia, T., and Prêteux, F. (2009). Tfan: A low complexity 3d mesh compression algorithm. volume 20, pages 343–354.

[Martin, 1979] Martin, G. N. N. (1979). Range encoding: An algorithm for removing redundancy from a digitised message. In *Proceedings of the Institution of Electronic and Radio Engineers International Conference on Video and Data Recording*, volume 43.

[Martin, 2000] Martin, I. (2000). Adaptive rendering of 3d models over networks using multiple modalities. *IBM research report*.

[Matias van Kaick and Pedrini, 2006] Matias van Kaick, O. and Pedrini, H. (2006). A comparative evaluation of metrics for fast mesh simplification. *Computer Graphics Forum*, 25(2):197–210.

[Meng et al., 2010] Meng, S., Wang, A., and Li, S. (2010). Compression of 3d triangle meshes based on predictive vector quantization. In *International Symposium on Systems and Control in Aeronautics and Astronautics*, pages 1403–1406.

[Meyer et al., 2012] Meyer, Q., Keinert, B., Sußner, G., and Stamminger, M. (2012). Data-parallel decompression of triangle mesh topology. *Computer Graphics Forum*, 31(8):2541–2553.

[Muller and Preparata, 1978] Muller, D. and Preparata, F. (1978). Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7(2):217 – 236.

[Munteanu et al., 2010] Munteanu, A., Cernea, D., Alecu, A., Cornelis, J., and Schelkens, P. (2010). Scalable l-infinite coding of meshes. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):513–528.

[Ngoc et al., 2002] Ngoc, N., Van Raemdonck, W., Lafruit, G., Deconinck, G., and Lauwereins, R. (2002). A qos framework for interactive 3d applications. In *The 10-th International Conference on Computer Graphics and Visualization '2002*, pages 317–324. Citeseer.

[Ochotta and Saupe, 2008] Ochotta, T. and Saupe, D. (2008). Image-based surface compression. *Computer Graphics Forum*, 27(6):1647–1663.

[Pajarola, 2001] Pajarola, R. (2001). Fastmesh: efficient view-dependent meshing. In *Proceedings of Pacific Conference on Computer Graphics and Applications*, pages 22–30.

[Pajarola and DeCoro, 2004] Pajarola, R. and DeCoro, C. (2004). Efficient implementation of real-time view-dependent multiresolution meshing. *Visualization and Computer Graphics, IEEE Transactions on*, 10(3):353–368.

[Pajarola and Rossignac, 2000] Pajarola, R. and Rossignac, J. (2000). Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics*, 6:79–93.

[Pajarola et al., 1999] Pajarola, R., Rossignac, J., and Szymczak, A. (1999). Implant sprays: compression of progressive tetrahedral mesh connectivity. In *Proceedings of the conference on Visualization*, pages 299–305.

[Pasman and Jansen, 2002] Pasman, W. and Jansen, F. W. (2002). Scheduling level of detail with guaranteed quality and cost. In *Web3D '02: Proceedings of the seventh international conference on 3D Web technology*, pages 43–51.

[Payan and Antonini, 2002] Payan, F. and Antonini, M. (2002). 3d mesh wavelet coding using efficient model-based bit allocation. In *Proceedings of the First International Symposium on 3D Data Processing Visualization and Transmission*, pages 391–394.

[Payan and Antonini, 2005] Payan, F. and Antonini, M. (2005). An efficient bit allocation for compressing normal meshes with an error-driven quantization. *Computer Aided Geometric Design*, 22:466–486.

[Payan and Antonini, 2006] Payan, F. and Antonini, M. (2006). Mean square error approximation for wavelet-based semiregular mesh compression. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):649–657.

[Pellenard et al., 2013] Pellenard, B., Morvan, J.-M., and Alliez, P. (2013). Anisotropic rectangular metric for polygonal surface remeshing. In *Proceedings of the International Meshing Roundtable*, pages 367–384.

[Peng et al., 2010] Peng, J., Huang, Y., Kuo, C.-C. J., Eckstein, I., and Gopi, M. (2010). Feature oriented progressive lossless mesh coding. *Computer Graphics Forum*, 29(7):2029–2038.

[Peng et al., 2005] Peng, J., Kim, C.-S., and Kuo, C.-C. J. (2005). Technologies for 3d mesh compression: A survey. *Journal of Visual Communication and Image Representation*, 16(6):688–733.

[Peng and Kuo, 2005] Peng, J. and Kuo, C.-C. J. (2005). Geometry-guided progressive lossless 3d mesh coding with octree (ot) decomposition. In *Proceedings of SIGGRAPH*, pages 609–616.

[Peyré and Mallat, 2005] Peyré, G. and Mallat, S. (2005). Surface compression with geometric bandelets. In *Proceedings of SIGGRAPH*, pages 601–608.

[Popović and Hoppe, 1997] Popović, J. and Hoppe, H. (1997). Progressive simplicial complexes. In *Proceedings of SIGGRAPH*, pages 217–224.

[Prat et al., 2005] Prat, S., Gioia, P., Bertrand, Y., and Meneveaux, D. (2005). Connectivity compression in an arbitrary dimension. *The Visual Computer*, 21:876–885.

[Rissanen and Langdon, 1979] Rissanen, J. and Langdon, G. G. (1979). Arithmetic coding. *IBM Journal of Research and Development*, 23(2):149–162.

[Rissanen, 1976] Rissanen, J. J. (1976). Generalized kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20(3):198–203.

[Rondao-Alface et al., 2003] Rondao-Alface, P., Macq, B., Cayre, F., Schmitt, F., and Maltre, H. (2003). Lapped spectral decomposition for 3d triangle mesh compression. In *Proceedings of the International Conference on Image Processing*, volume 1, page 781.

[Rossignac, 1999] Rossignac, J. (1999). Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5:47–61.

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

[Roudet, 2010] Roudet, C. (2010). A region-based progressive coding of semi-regular 3-d meshes for view-dependent transmission. *Proceedings of the International Conference on Signal Image Technology and Internet Based Systems*, pages 51–59.

[Rupp I., 2008] Rupp I., Peniguel C., T.-M. M. (2008). Large scale finite element thermal analysis of bolts of a french pwr core internal baffle structure. In *Proceedings of the 7th International Topical Meeting on Nuclear Reactor Thermal Hydraulics, Operation and Safety NUTHOS-7*.

[Salomie et al., 2004] Salomie, I., Munteanu, A., Gavrilescu, A., Lafruit, G., Schelkens, P., Deklerck, R., and Cornelis, J. (2004). Meshgrid-a compact, multiscalable and animation-friendly surface representation. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(7):950–966.

[Sander et al., 2003] Sander, P. V., Wood, Z. J., Gortler, S. J., Snyder, J., and Hoppe, H. (2003). Multi-chart geometry images. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 146–155.

[Schindler, 1998] Schindler, M. (1998). A fast renormalisation for arithmetic coding. In *Proceedings of the Data Compression Conference, 1998*, page 572. http://www.compressconsult.com/rangecoder/.

[Schläfli, 1901] Schläfli, L. (1901). *Theorie der vielfachen kontinuität*. Cornell University Library historical math monographs. George & Company.

[Schneider and Martin, 1999] Schneider, B. and Martin, I. (1999). An adaptive framework for 3d graphics over networks. *Computers & Graphics*, 23(6):867–874.

[Shaffer and Garland, 2005] Shaffer, E. and Garland, M. (2005). A multiresolution representation for massive meshes. *IEEE Transactions on Visualization and Computer Graphics*, 11(2):139–148.

[Shamir, 2008] Shamir, A. (2008). A survey on mesh segmentation techniques. *Computer Graphics Forum*, 27(6):1539–1556.

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423.

[Shi et al., 2012] Shi, Y., Wen, B., Ding, W., Qi, N., and Yin, B. (2012). Realistic mesh compression based on geometry image. In *Picture Coding Symposium (PCS), 2012*, pages 133 –136.

[Shikhare et al., 2001] Shikhare, D., Bhakar, S., and Mudur, S. P. (2001). Compression of large 3d engineering models using automatic discovery of repeating geometric features. In *Proceedings of the Vision Modeling and Visualization Conference*, pages 233–240.

[Sim et al., 2005] Sim, J.-Y., Kim, C.-S., Kuo, C.-C., and Lee, S.-U. (2005). Rate-distortion optimized compression and view-dependent transmission of 3-d normal meshes. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(7):854–868.

[Sim et al., 2002] Sim, J.-Y., Kim, C.-S., Kuo, J.-C., and Lee, S.-U. (2002). Normal mesh compression based on rate-distortion optimization. In *Proceedings of the IEEE Workshop on Multimedia Signal Processing*, pages 13–16.

[Sim et al., 2003] Sim, J.-Y., Kim, C.-S., and Lee, S.-U. (2003). An efficient 3d mesh compression technique based on triangle fan structure. *Signal Processing: Image Communication*, 18(1):17–32.

[Southern et al., 2001] Southern, R., Perkins, S., Steyn, B., Muller, A., Marais, P., and Blake, E. (2001). A stateless client for progressive view-dependent transmission. In *Web3D '01: Proceedings of the sixth international conference on 3D Web technology*, pages 43–50, New York, NY, USA. ACM.

[Sweldens, 1996] Sweldens, W. (1996). The lifting scheme: A custom-design construction of biorthogonal wavelets. *Applied and Computational Harmonic Analysis*, 3(2):186–200.

[Szymczak et al., 2001] Szymczak, A., King, D., and Rossignac, J. (2001). An edgebreaker-based efficient compression scheme for regular meshes. *Computational Geometry*, 20(12):53–68.

[Szymczak and Rossignac, 1999] Szymczak, A. and Rossignac, J. (1999). Grow & fold: compression of tetrahedral meshes. In *Proceedings of the fifth ACM symposium on Solid modeling and applications*, pages 54–64.

[Szymczak et al., 2002] Szymczak, A., Rossignac, J., and King, D. (2002). Piecewise regular meshes: Construction and compression. *Graphical Models*, 64(3–4):183–198.

[Tack et al., 2006] Tack, K., Lafruit, G., Catthoor, F., and Lauwereins, R. (2006). Platform independent optimisation of multi-resolution 3d content to enable universal media access. *The Visual Computer*, 22:577–590.

[Tack et al., 2005] Tack, N., Lafruit, G., Catthoor, F., and Lauwereins, R. (2005). Pareto based optimization of multi-resolution geometry for real time rendering. In *Web3D '05: Proceedings of the tenth international conference on 3D Web technology*, pages 19–27.

[Tampieri, 1992] Tampieri, F. (1992). *Newell's method for computing the plane equation of a polygon*, pages 231–232.

[Tarini et al., 2010] Tarini, M., Pietroni, N., Cignoni, P., Panozzo, D., and Puppo, E. (2010). Practical quad mesh simplification. *Computer Graphics Forum*, 29(2):407–418.

[Taubin et al., 1998] Taubin, G., Guéziec, A., Horn, W., and Lazarus, F. (1998). Progressive forest split compression. In *Proceedings of SIGGRAPH*, pages 123–132.

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

[Teler and Lischinski, 2001] Teler, E. and Lischinski, D. (2001). Streaming of complex 3d scenes for remote walkthroughs. *Computer Graphics Forum*, 20(3):17–25.

[Tian et al., 2012] Tian, J., Jiang, W., Luo, T., Cai, K., Peng, J., and Wang, W. (2012). Adaptive coding of generic 3d triangular meshes based on octree decomposition. *The Visual Computer*, 28:819–827.

[Touma and Gotsman, 1998] Touma, C. and Gotsman, C. (1998). Triangle mesh compression. In *Graphics Interface 98 Conference Proceedings*, pages 26–34.

[Turán, 1984] Turán, G. (1984). On the succinct representation of graphs. *Discrete Applied Mathematics*, 8(3):289–294.

[Tutte, 1962] Tutte, W. (1962). A census of planar triangulations. *Canadian Journal of Mathematics*, 14:21–38.

[Tutte, 1963] Tutte, W. (1963). A census of planar maps. *Canadian Journal of Mathematics*, 15:249–271.

[Ueng, 2003] Ueng, S.-K. (2003). Out-of-core encoding of large tetrahedral meshes. In *Proceedings of the Eurographics/IEEE TVCG Workshop on Volume graphics*, pages 95–102.

[Valette et al., 2009] Valette, S., Chaine, R., and Prost, R. (2009). Progressive lossless mesh compression via incremental parametric refinement. In *Proceedings of the Symposium on Geometry Processing*, pages 1301–1310.

[Valette and Prost, 2004a] Valette, S. and Prost, P. (2004a). Wavelet-based multiresolution analysis of irregular surface meshes. *Visualization and Computer Graphics, IEEE Transactions on*, 10(2):113–122.

[Valette and Prost, 2004b] Valette, S. and Prost, R. (2004b). Wavelet-based progressive compression scheme for triangle meshes: Wavemesh. *IEEE Transactions on Visualization and Computer Graphics*, 10:123–129.

[Váša and Brunnett, 2013] Váša, L. and Brunnett, G. (2013). Exploiting connectivity to improve the tangential part of geometry prediction. *IEEE Transactions on Visualization and Computer Graphics*.

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

[Xiang et al., 1999] Xiang, X., Held, M., and Mitchell, J. S. B. (1999). Fast and effective stripification of polygonal surface models. In *Proceedings of the Symposium on Interactive 3D graphics*, pages 71–78.

[Ying et al., 2010] Ying, L., Mingli, D., Zhongming, H., and Dagao, D. (2010). An edgebreaker & code-mode based connectivity compression for triangular meshes. In *Proceedings of the International Conference on Advanced Computer Control*, volume 2, pages 96–101.

[Yoon and Lindstrom, 2007] Yoon, S.-E. and Lindstrom, P. (2007). Random-accessible compressed triangle meshes. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1536–1543.

[Yoon et al., 2004] Yoon, S.-E., Salomon, B., Gayle, R., and Manocha, D. (2004). Quick-vdr: interactive view-dependent rendering of massive models. In *IEEE Visualization*, pages 131 – 138.

[Zhao et al., 2011] Zhao, C., Sun, H., and Qin, K. (2011). Efficient wavelet-based geometry compression. *Computer Animation and Virtual Worlds*, 22(2-3):307–315.

[Zhao et al., 2012] Zhao, J., Tang, M., and Tong, R. (2012). Connectivity-based segmentation for gpu-accelerated mesh decompression. *Journal of Computer Science and Technology*, 27(6):1110–1118.

[Zheng et al., 2004] Zheng, H., Liu, B., and Zhang, H. (2004). Region-of-interest coding of 3d mesh based on wavelet transform. In *Multi-Agent Security and Survivability, 2004 IEEE First Symposium on*, pages 438–441.

[Zorin et al., 1996] Zorin, D., Schröder, P., and Sweldens, W. (1996). Interpolating subdivision for meshes with arbitrary topology. In *Proceedings of SIGGRAPH*, pages 189–192.