HAL open science

Segmentation and Skeleton Methods for Digital Shape Understanding

Franck Hétroy-Wheeler

▶ To cite this version:

Franck Hétroy-Wheeler. Segmentation and Skeleton Methods for Digital Shape Understanding. Graphics [cs.GR]. Université Grenoble Alpes, 2015. tel-01246006v2

HAL Id: tel-01246006

https://inria.hal.science/tel-01246006v2

Submitted on 16 Mar 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial 4.0 International License

UNIVERSITÉ DE GRENOBLE

HABILITATION À DIRIGER DES RECHERCHES

Spécialité : **Informatique**

Présentée par

Franck Hétroy-Wheeler

Segmentation and Skeleton Methods for Digital Shape Understanding

soutenue publiquement le **20 novembre 2015**,
devant le jury composé de :

Marc Daniel
Professeur, Université d'Aix-Marseille, Rapporteur
Adrian Hilton
Professor, University of Surrey, Rapporteur
Michela Spagnuolo
Research Director, Consiglio Nazionale delle Richerche Genova, Rapporteur
Marie-Paule Cani
Professeure, Université Grenoble Alpes, Examinatrice
Raphaëlle Chaine
Professeure, Université de Lyon, Examinatrice
Bruno Lévy
Directeur de Recherche, Inria Nancy Grand-Est, Examinateur

ABSTRACT

Digitised geometric shape models are essentially represented as a collection of primitives without a general coherence. Shape understanding aims at retrieving global information about the shape geometry, topology or functionality, for subsequent uses such as measurement, simulation or modification. In this context, this manuscript presents my main contributions to digital shape understanding, which are mostly based on shape decomposition and skeleton computation. The first part explores the faithfulness of the 3D mesh representation to the real world object, through topological and perceptual analyses, and suggests a conversion to a regular volumetric model. The second part focuses on shapes in motion and details tools to create, modify and analyse temporal mesh sequences. The third part explains through two concrete examples how digital shape understanding can help experts in medicine and forestry. Finally, three open questions for shape understanding of shapes in motion and scanned trees are discussed by way of perspectives.

RÉSUMÉ

Les modèles géométriques de formes numérisées sont pour l'essentiel représentés comme une collection de primitives sans cohérence générale. La compréhension de formes a pour but de retrouver une information globale concernant la géométrie, topologie ou fonction d'une forme, afin de pouvoir la mesurer, la modifier ou l'utiliser en simulation. Dans ce contexte, ce manuscript présente mes principales contributions à la compréhension numérique de formes géométriques, qui sont principalement fondées sur la décomposition de formes et le calcul de squelette. La première partie explore la fidélité de la représentation par maillage 3D à l'objet du monde réel, à travers des analyses topologiques et perceptuelles, et propose une conversion vers un modèle volumique régulier. La deuxième partie se concentre sur les formes en mouvement et détaille des outils permettant de créer, modifier et analyser des séquences temporelles de maillages. La troisième partie explique à travers deux exemples concrets comment la compréhension numérique de formes peut aider les experts dans des domaines comme la médecine et la sylviculture. Enfin, trois questions ouvertes sur la compréhension de formes pour les formes en mouvement et les arbres scannés sont discutées en guise de perspectives.

Acknowledgements

I would first like to thank the three reviewers of this manuscript, Marc Daniel, Adrian Hilton and Michela Spagnuolo for their time and their very relevant comments. Thanks too, to the three other committee members, Marie-Paule Cani, Raphaëlle Chaine and Bruno Lévy, for their questions and encouragement.

I have learned a lot over the past years from several researchers, with whom I worked: many thanks to my PhD thesis advisors Annick Montanvert and Dominique Attali, to Marie-Paule Cani and Edmond Boyer who welcomed me in their teams and gave me the opportunity to conduct my research in almost perfect conditions, to my colleagues from the EVASION and Morpheo teams Georges-Pierre Bonneau, François Faure, Jean-Sébastien Franco, Jean-Claude Léon, Olivier Palombi, Lionel Revéret and others, to my collaborators, especially Eric Casella, Florent Dupont and Kai Wang. I have also learned from my students, especially my PhD students Sahar Hassan, Romain Arcila, Phuong Ho, Benjamin Aupetit, Georges Nader and Li Wang as well as Dobrina Boltcheva and the interns I tutored. Each of them was different and helped me question myself: thank you all.

I also wish to thank my colleagues and students from Grenoble INP - Ensimag, because teaching is the best way to learn something in depth, and teaching skills are also useful for research.

Finally, I wish to acknowledge the support of my parents-in-law Frederick and Francisca Wheeler who provided me with perfect conditions to write this manuscript in a quiet environment in Stockport. Thanks to them also for their encouragement and for their help in proof-reading the manuscript. Thanks also to my parents, my family and in particular my dear Elisa and Emily for their help, patience, support and love. This habilitation is dedicated to them.

Contents

1 Introduction **1**
1.1 Research problem: digital shape understanding . . . 1
1.1.1 Relation to digital geometry processing . . . 2
1.2 Approach: segmentation and skeleton computation . . . 2
1.3 Contributions . . . 3
1.4 Influential encounters . . . 5
1.4.1 Quick contextual elements . . . 5
1.4.2 Advised students . . . 6
1.4.3 Main projects and collaborations . . . 7
1.4.4 Remark . . . 8

2 Geometrical, topological and perceptual analysis of 3D meshes **9**
2.1 Introduction . . . 9
2.1.1 3D meshes . . . 9
2.1.2 Objectives . . . 10
2.2 Mesh repair with topology control . . . 11
2.2.1 Surface singularities . . . 11
2.2.2 Discrete voxel membrane . . . 12
2.2.3 Interactive topology modification . . . 12
2.3 Retrieving the homology of simplicial complexes . . . 13
2.3.1 Background on simplicial homology . . . 14
2.3.2 Constructive homology . . . 15
2.3.3 Manifold-Connected decomposition . . . 16
2.3.4 Mayer-Vietoris algorithm . . . 16
2.4 Just-Noticeable Distortion profile for meshes . . . 17
2.4.1 Local perceptual properties . . . 19
2.4.2 Just Noticeable distortion profile . . . 20
2.4.3 Perceptually optimal vertex coordinates quantization . . . 21

2.4.4 Perceptually optimal mesh simplification 22
2.5 Regular volumetric discretisation 22
2.5.1 Regular Centroidal Voronoi Tessellations 23
2.5.2 A hierarchical framework 24
2.6 Perspectives 25
2.6.1 Perceptual analysis of smooth shaded surfaces 25
2.6.2 Regularly tessellated volumes from point clouds 25

3 Digital geometry processing for shapes in motion 27
3.1 Introduction 27
3.1.1 Objectives 27
3.1.2 3D+t vs. 4D 28
3.1.3 Temporal coherence 28
3.2 Harmonic skeleton for character animation 30
3.2.1 A short survey on skeleton computation methods 30
3.2.2 Reeb graph computation 32
3.2.3 Embedding into an animation skeleton 33
3.2.4 Atlas generation 33
3.2.5 Skinning weights 34
3.3 A discrete Laplace operator for temporally coherent mesh sequences . 35
3.3.1 Definition of a 4D DEC Laplace operator 35
3.3.2 Matrix representation 39
3.3.3 Behaviour for large and small time steps 40
3.3.4 Application to as-rigid-as-possible mesh sequence deformation 41
3.4 Mesh sequence decomposition into rigidly moving components 43
3.4.1 Shape-in-motion segmentation classification 43
3.4.2 Mesh matching 44
3.4.3 Motion-based spectral clustering 45
3.4.4 Shape in motion segmentation evaluation 46
3.5 Perspectives 47
3.5.1 Visual differences between shapes in motion 47
3.5.2 3D+t Laplace operator spectral behaviour 47
3.5.3 Modelling the factors of variability for human shape in motion 48

4 Understanding digital shapes from the life sciences 49
4.1 Introduction 49
4.1.1 Context 49
4.1.2 General approach: skeleton for segmentation and measurement 50
4.2 Cerebral aneurysm characterisation and quantification 51
4.2.1 Centreline extraction 51
4.2.2 Aneurysm detection and quantification 52
4.2.3 Aneurysm localisation 54
4.3 Tree seedling segmentation and measurement 54
4.3.1 Graph computation 56

4.3.2 Spectral embedding 57
4.3.3 Segmentation in spectral space 57
4.4 Perspectives 58
4.4.1 TLS point cloud noise detection and correction 58
4.4.2 Towards an accurate digital tree model 59

5 Conclusion **61**
5.1 Summary 61
5.2 General comments 62
5.3 Open questions for shapes in motion and forest science 62
5.3.1 Appropriate shape representations 63
5.3.2 Mathematical and computational tools 63
5.3.3 Incorporating additional knowledge 64

A Selected papers **67**
A.1 Mesh repair with user-friendly topology control 69
A.2 An iterative algorithm for homology computation on simplicial shapes 101
A.3 Just Noticeable Distortion profile for flat-shaded 3D mesh surfaces . . 115
A.4 A hierarchical approach for regular Centroidal Voronoi Tessellations . 141
A.5 Harmonic skeleton for realistic character animation 159
A.6 Simple flexible skinning based on manifold modeling 171
A.7 Segmentation of temporal mesh sequences into rigidly moving components 179
A.8 Automatic localization and quantification of intracranial aneurysms . 193
A.9 Segmentation of tree seedling point clouds into elementary units . . . 203

B List of publications **233**
B.1 Geometrical, topological and perceptual analysis of 3D meshes 233
B.2 Digital geometry processing for shapes in motion 234
B.3 Understanding digital shapes from the life sciences 235

Bibliography **237**

Index **251**

CHAPTER 1

Introduction

1.1 Research problem: digital shape understanding

Nowadays, shape digitisation is ubiquitous. Virtual geometric models allow for actions that are cumbersome or even impossible in the real world to be realised. For example, surgical training [FLA+05], forest inventory [OVSP13] or creation of special effects for the entertainment industry have been greatly simplified thanks to digitised models. Virtual shapes can be either modelled from scratch, thanks to the skills of the user and the assistance of a computer, or digitised from real objects or scenes using sensors such as cameras or scanners. In general, digitisation can easily create a more complex and faithful representation of the real world than shape modelling.

Unfortunately, digitised shape models are essentially a collection of simple primitives (points, triangles, voxels, ...) with no general coherence. Global information and semantics about the object (e.g. "this is an arm", "this part is cylindrical") and its functionality are usually lost during the digitisation process. Only local information is explicitly available: point coordinates, possibly normals or colours, neighbouring relationships.

Nevertheless, high-level information can often be retrieved, by using powerful mathematical tools to analyse the geometry of the shape together with some prior knowledge. This knowledge depends on the application and can be either injected to the processing algorithms or interactively given by the user. The first solution is sometimes preferable since the process is then automated. However, it is not always possible since it requires the knowledge to be formalised. While it is always possible to ask the user to provide the knowledge, it may be time consuming and the output

may be different for different users. Once higher level information is retrieved, the shape model can subsequently be used with its semantics, for instance for dedicated measurements. Figure 1.1 describes the whole pipeline. The central process is called *digital shape understanding*, which can be defined as *the process of recognising an object or its parts in a specific context* [BLMS14].

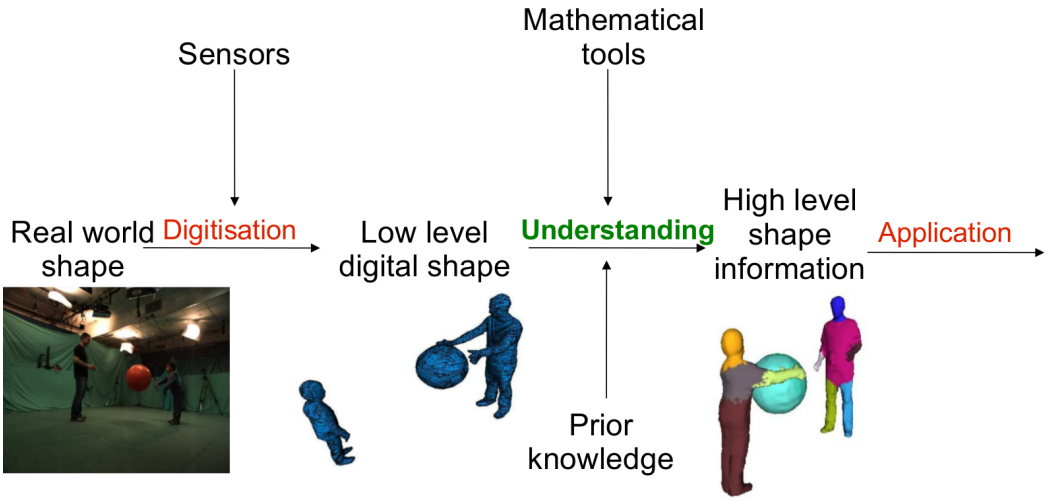


Figure 1.1: *From a real world object or scene to an application in the virtual world: a pipeline proposal.*

1.1.1 Relation to digital geometry processing

Digital shape understanding is connected to *digital geometry processing* through shape analysis [LÓ8, All09]. Alliez includes in digital geometry processing a whole pipeline ranging from the acquisition of a raw digital model to shape analysis to advanced processes such as editing, protection or printing. The term *digital shape understanding* is usually restricted, in this pipeline, to the geometrical and topological analyses of the shape. It focuses on their subsequent use to retrieve the semantics and functionality of the object at a global level [LZ11, BLMS14].

1.2 Approach: Segmentation and Skeleton Computation

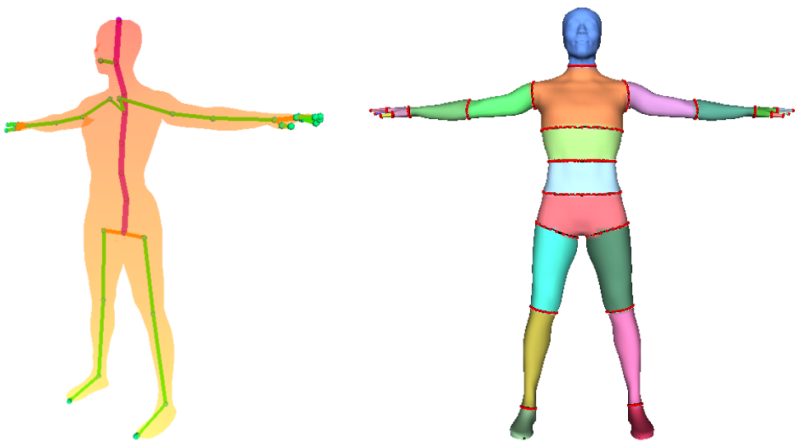
In this manuscript two main approaches for digital shape understanding are explored, segmentation and skeleton computation.

According to Hao Zhang [LZ11], the two most fundamental tasks in digital shape understanding are *the segmentation (partitioning) into and the correspondence between meaningful shape parts*. Indeed, segmentation is key to many different fields

(think "divide and conquer"), and also makes sense for the analysis of a complex shape. The identification of its sub-parts and the understanding of their connections often help to recognise the overall shape. The "meaning" of a shape element is of course application-dependent, as this document will demonstrate. An overview of existing segmentation methods can be found e.g. in [Sha08, TPT15].

Another popular technique for digital shape understanding is the reduction of the input shape to a one-dimensional graph, often called a *skeleton*. While geometrical details are lost with such an approach, the skeleton usually keeps the main topological features of the shape, thus enabling a rough general recognition. A short review of state of the art skeleton computation techniques for 3D meshes will be given in Section 3.2.1. It supplements the one in [Tag13].

Note that both of these approaches may be seen as dual: the edges of a skeleton often represent the coherent sub-parts of the shape under analysis. The nodes are the connection between these sub-parts (see Fig. 1.2 for an example), and give information about the general structure of the shape. They are also complementary for shape understanding, in the sense that a skeleton gives the correspondence between shape parts by connecting each of them to its neighbours.



(a) (b)

Figure 1.2: *Skeleton (a) and segmentation (b) of a human character digital model.*

1.3 CONTRIBUTIONS

This manuscript describes my main contributions to the area of *digital shape understanding*, developed during the past twelve years. These contributions fall into three different categories:

1. three-dimensional mesh analysis (Chapter 2), in which I question the faithfulness of a discrete polygonal mesh to represent the real world object that has been digitised;

2. moving shape understanding (Chapter 3), where I develop tools to retrieve general information on both the geometry and the motion of digitised dynamic models;

3. digital shape understanding for life sciences (Chapter 4), where I explore two practical applications of digital shape understanding for medical imaging and dendrometry (tree dimensions measurement).

In Chapter 2, I introduce topological and geometrical segmentation methods to help recognise the general structure of an object represented by a 3D mesh. This is done in different stages:

- firstly, I present an interactive tool to convert an unorganised set of triangles into a well-behaved (two-manifold) mesh. The user is proposed with a limited set of possible segmentations. Thus he or she provides knowledge about the correct topology of the mesh;

- secondly, I develop an algorithm to compute the complete topological information of simplicial complexes. This allows for a decomposition of such objects into manifold-connected components. The knowledge used in the algorithm comes from the mathematical theory of constructive homology [Ser94];

- thirdly, I investigate the discretisation into regular cells of the volumetric object surrounded by a given two-manifold mesh. Here again, knowledge about what a regular cell should be comes from a mathematical theory of quantisation [CS82].

In Chapter 3, I use both a skeleton and a segmentation process to decompose a shape in motion into meaningful sub-parts, in two different contexts. They are:

- 3D animation creation, for which I introduce a method to compute an accurate animation skeleton from a static meshed character as well as a method to generate a variety of animations with this skeleton thanks to flexible skinning weights. Here, the knowledge is related to the bone anatomy of the character. This has been encoded into the skeleton computation algorithm for both humans and quadruped animals;

- motion analysis from videos, where I study the segmentation of a 3D moving shape, represented as a sequence of meshes without temporal coherence. Since the focus of this work is on human motion, the knowledge that the body is composed of rigidly moving parts is imposed.

In Chapter 4, I combine both skeleton and segmentation tools. A skeleton is first computed to get a general picture of the object under study. It then drives a subsequent segmentation of this object into the elements of interest. This is done for two different applications:

- characterisation and quantification of cerebral aneurysms. Knowledge about the topological structure of the cerebral tree is encoded into the algorithm;
- segmentation of a tree seedling point cloud into elementary elements (branches, petioles, leaves). Basic observations about the structure of a tree enable the algorithm to split the point cloud into meaningful sub-sets. They can then be refined if the user decides. Doing so, he brings additional knowledge to the process.

Note that beside skeleton computation and segmentation methods, I have developed other approaches. In Chapter 2 I detail a perceptual study on vertex displacement visibility on a mesh, and in Chapter 3 I present a generic tool to analyse shapes in motion (a discrete Laplace operator).

1.4 INFLUENTIAL ENCOUNTERS

This manuscript is intended to give an overview of my research work as an assistant professor ("*maître de conférences*") in computer science at Grenoble INP - Ensimag (part of the Université Grenoble Alpes, France) since 2004. Of course, I have not worked alone on the topic of digital shape understanding. Before entering into the details in the next chapters, allow me to pay tribute to my co-workers.

1.4.1 Quick contextual elements

I defended my PhD thesis (advised by Prof. Annick Montanvert and Dr. Dominique Attali, in the GIPSA-Lab in Grenoble) in September 2003. In February 2004, I joined the computer graphics team of Prof. Pere Brunet at the Universitat Politècnica de Catalunya in Barcelona, Spain, thanks to a grant from the French Ministry of Foreign Affairs. The work described in Section 2.2 was started there. Within a few months, I was appointed to my current position at Grenoble INP - Ensimag in September of the same year. As part of the university of Grenoble, I joined Prof. Marie-Paule Cani's team EVASION at Inria and GRAVIR lab (later moved to the current Laboratoire Jean Kuntzmann). While there, I was lucky enough to collaborate with a number of people who are listed below, and to advise many students, including my first two PhD students. In 2011, EVASION came to an end and Marie-Paule decided to focus her new team IMAGINE on the creation of digital content. As I was more interested in analysing and processing real world data, I accepted the offer of Prof. Edmond Boyer to join his new team Morpheo, which we created together with Dr. Lionel Revéret and soon joined by Dr. Jean-Sébastien Franco. In Morpheo our work is at the intersection of computer graphics and computer vision, since our overall goal is to model, recognise and animate moving shapes from multiple camera systems. I am currently advising two PhD students.

1.4.2 Advised students

PhD students

Current PhD students:

- Georges Nader (started in 2013). *Evaluation of the perceptual quality of 3D dynamic meshes and applications*. Co-advised by Prof. Florent Dupont (Université de Lyon) and Dr. Kai Wang (CNRS Grenoble). See Section 2.4;
- Li Wang (started in 2013). *Centroidal Voronoi tessellations for shape reconstruction*. Co-advised by Prof. Edmond Boyer (Inria Grenoble). See Section 2.5.

Past PhD students:

- Sahar Hassan (2007-2011). *Integration of a priori anatomical knowledge into geometrical models*. Co-advised by Prof. Georges-Pierre Bonneau (Université Grenoble Alpes). See Section 4.2;
- Romain Arcila (2008-2011). *Mesh sequences: a classification and segmentation methods*. Co-advised by Prof. Florent Dupont (Université de Lyon). See Section 3.4.

I also advised 2 students who did not finished their PhD:

- Thi Phuong Ho (2011-2012, co-advised by Prof. Bruno Lévy from Inria Nancy), whose PhD was stopped after 16 months;
- Benjamin Aupetit (2011-2014, co-advised by Prof. Edmond Boyer from Inria Grenoble), who resigned after 3 years.

Master students

- Romain Rombourg (Télécom Physique Strasbourg, 2015). *Detection and correction of mixed point noise in laser scans*. Co-advised by Dr. Eric Casella (Forest Research).
- Antoine Fond (Ecole Centrale Nantes, 2014). *Recognition of actions using a shape sequence database*. Co-advised by Dr. Jean-Sébastien Franco (Université Grenoble Alpes).
- Li Wang (Grenoble INP - Ensimag, 2013). *An optimal transport formulation of centroidal Voronoi tessellations*. Co-advised by Prof. Edmond Boyer (Inria Grenoble).
- Benjamin Aupetit (Grenoble INP - Ensimag, 2011). *A morphable model for bird skeleton meshes*. Co-advised by Dr. Lionel Revéret (Inria Grenoble).

- Sara Merino Aceituno (Université Grenoble Alpes, 2010). *Homology computation for unions of simplicial complexes: a constructive Mayer-Vietoris algorithm*. Co-advised by Prof. Jean-Claude Léon and Dr. Dobrina Boltcheva (Université Grenoble Alpes). See Section 2.3.

- Sahar Hassan (Université Grenoble Alpes, 2007). *Characterisation and quantification of aneurysms in volumetric models*. Co-advised by Dr. François Faure and Dr. Olivier Palombi (Université Grenoble Alpes).

- Grégoire Aujay (Université Grenoble Alpes, 2006). *From a geometrical skeleton to an animation skeleton*. Co-advised by Dr. Francis Lazarus (CNRS Grenoble). See Section 3.2.

Postdoc

- Dobrina Boltcheva (2009-2011). *Computation of the homology of simplicial complexes* (2009-2010) and then *virtual plant reconstruction mixing incomplete geometric data and prior knowledge* (2010-2011). See Sections 2.3 and 4.3.

1.4.3 Main projects and collaborations

- ASLAAF (2014-2015), PI. Funded by the Université Grenoble Alpes (AGIR framework). *Analysis of tree laser scans and applications for forestry*. Collaboration with Forest Research (Dr. Eric Casella).

- PADME (2013-2016). Funded by the Rhône-Alpes Région (ARC6 framework). *Evaluation of the perceptual quality of 3D dynamic meshes*. Collaboration with Université de Lyon (Prof. Florent Dupont) and GIPSA-Lab Grenoble (Dr. Kai Wang). See Section 2.4.

- MORPHO (2011-2015), PI. Funded by the National Research Agency (ANR). *Human shape and motion analysis*. Collaboration with Inria Nancy (Prof. Bruno Lévy, Dr. Dobrina Boltcheva) and GIPSA-Lab Grenoble (Dr. Olivier Martin). See Section 2.5.

- IDEAL (2009-2011). Funded by the Université Grenoble Alpes (BQR framework). *Idealised objects modelling*. Collaboration with Università degli Studi di Genova (Prof. Leila de Floriani). See Section 2.3.

- PlantScan3D (2009-2011). Funded by the Agropolis Foundation and Inria (ARC framework). *Reconstruction of geometrical models of plants and trees from laser scans*. This was a large collaborative project between computer scientists and biologists, within which I interacted mostly with CIRAD/Inria Montpellier (Prof. Christophe Godin, Dr. Frédéric Boudon) and Forest Research (Dr. Eric Casella). See Section 4.3.

- MADRAS (2008-2011). Funded by the ANR. *Representation and segmentation of static and dynamic 3D models*. Collaboration with Université de Lyon (Prof. Florent Dupont, Drs. Florence Denis, Guillaume Lavoué and Christian Wolf) and with Université de Lille (Prof. Mohamed Daoudi and Dr. Jean-Philippe Vandeborre). See Section 3.4.

- MEGA (2006-2007), PI. Funded by the Université Grenoble Alpes and Inria. *Geometrical methods for the decomposition and deformation of 3D surfaces for computer animation*. Collaboration with GIPSA-Lab Grenoble (Dr. Cédric Gérot). See Section 3.2.

1.4.4 Remark

As many researchers, I have worked on very different topics through various collaborations. To keep this document coherent, it is not an exhaustive summary of my research to date. It leaves aside some of my contributions, including classification of non-manifold singularities (SMI 2009 paper [LDFH09]), a method to compute constriction curves on mesh surfaces (Eurographics 2005 short paper [H05]), some non-published work on visually loss-less mesh sequence temporal compression (started with my second PhD student Romain Arcila, through a collaboration with Dr. Ron Rensink from the University of British Columbia), and the second half of Sahar Hassan's PhD thesis on ontology-guided mesh segmentation [HHP10].

Figure 1.3 gives an overview of the PhD students and postdoc I have advised, the projects I have been involved in, and my publications. One colour corresponds to one chapter of this manuscript. The projects for which I have been principal investigator or coordinator are underlined.

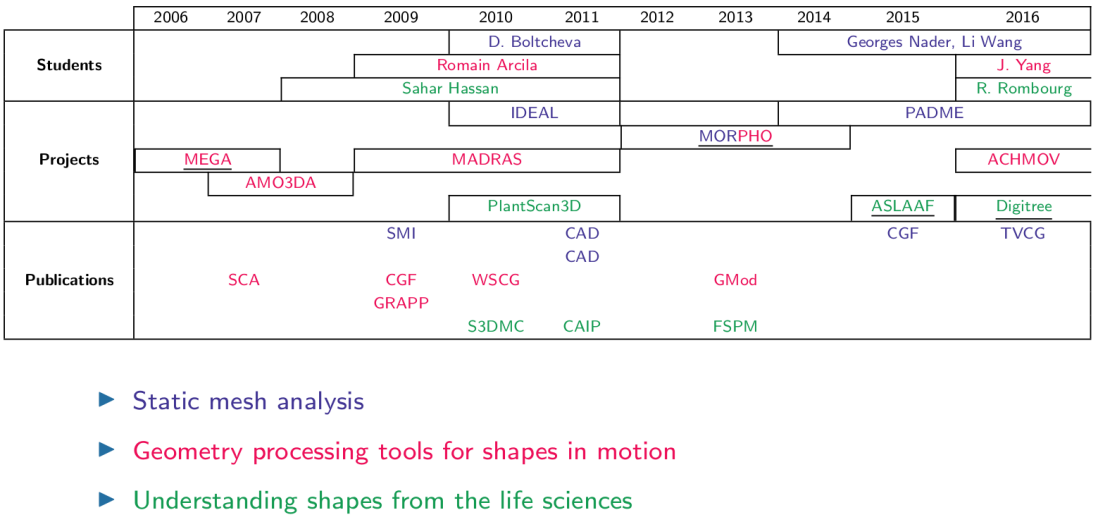


Figure 1.3: *Students, projects and publications throughout the years.*

CHAPTER 2

Geometrical, topological and perceptual analysis of 3D meshes

2.1 INTRODUCTION

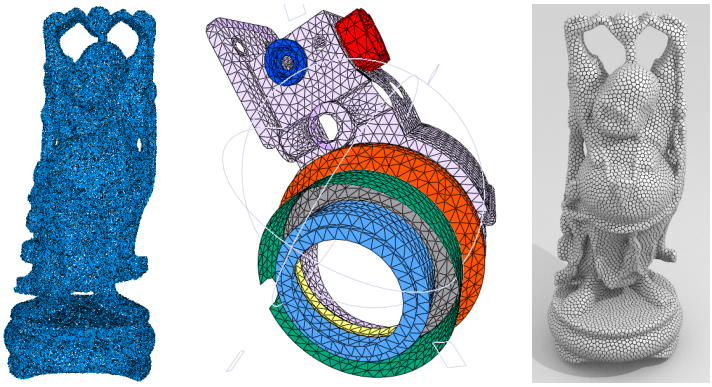
2.1.1 3D meshes

This chapter focuses on static 3D meshes. A mesh is usually defined as a triplet $M = (V, E, F)$ where V is a set of vertices, i.e. 3D points, of M. E is a set of edges, i.e. segments between neighbouring vertices, and F is a set of faces which define a piece of the surface. Faces are usually polygonal and most often triangular since a triangle is the simplest polygon that defines a surface. The 3D mesh is essentially a local representation of the shape's surface because only the neighbourhood of a given point of a mesh is explicitly known.

Although other representations are also widely used, such as voxel sets in medical imaging (see Section 4.2 for an example), it is meshes that are nowadays the most commonly used discrete 3D shape representation in many contexts. This results from the simplicity of creating a mesh from raw data, such as point clouds, even though mesh reconstruction methods may not easily generate a watertight model of the object because of inconsistencies or occlusions in the data. This problem will be addressed in Section 2.2. Meshes are extremely useful since they allow for an easy rendering and detection of collisions/intersections. Further to this, meshes provide a basis to define piecewise linear functions which are crucial to subsequent pipeline processes such as solving partial differential equations using finite element methods.

2.1.2 Objectives

The input, in this chapter, is a triangle mesh which may be completely unstructured (see Fig. 2.1 (a)). Such a mesh is often the output of 3D scanners, for instance. This representation makes it very difficult to recognise the underlying shape at a general level. Our goal is, therefore, to convert it into a more usable digital model.



(a) (b) (c)

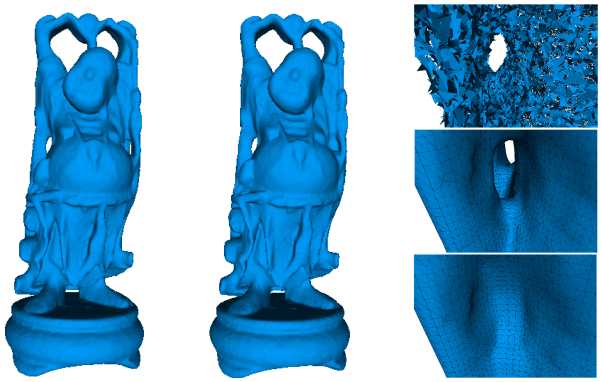
Figure 2.1: *(a,b) Input data: (a) Buddha statue represented as a "soup" of triangular polygons; (b) sump represented as a simplicial complex (colours indicate the manifold connected components). (c) Result: volumetric shape model with regular sampling.*

I first present a method to "repair" any unstructured mesh into a two-manifold, in which each edge is exactly shared by 2 triangles (Section 2.2). The user brings some knowledge about the shape by interactively choosing its topology. This method segments the shape into topologically ambiguous areas and non-ambiguous ones. I then focus on simplicial complexes (Section 2.3). Simplicial complexes are widespread in Computer Aided Design as representations of mechanical parts. They are usually semi-automatically created from spline representations (for instance) using software such as Autodesk's AutoCAD and Dassault Systèmes' CATIA. A mechanical piece may be geometrically very complex (see Fig. 2.1 (b)), with many "T-junctions". I show how the theory of *constructive homology* can be used to retrieve the complete topological information of a simplicial complex. This enables its segmentation into (almost) manifold components. The objective of the work described in Section 2.4 is to investigate the visual perception of a manifold mesh. More precisely, I have investigated with my colleagues how the displacement of a vertex is noticed by a human user. We have been able to define a *Just Noticeable Difference* profile for 3D meshes. This allows us to compute the optimal vertex coordinates quantization level for any mesh. Finally, in Section 2.5 I propose a regular decomposition of the interior of a shape described by its manifold meshed boundary into uniform elements. It enables me to represent the shape with a uniform, anisotropic volumetric sampling (Fig. 2.1 (c)). This enriched digital representation can then be used as input for various processes, such as volumetric shape tracking [AFB15].

2.2 MESH REPAIR WITH TOPOLOGY CONTROL

The digitisation process from a real object often creates inconsistent meshes. This is due to the inherent limitations of the sensors, but also to the inner geometry of the object. For instance, occluded parts cannot be faithfully reconstructed by a laser scanner. Unfortunately, a consistent mesh is often necessary for further processing purposes. By "consistent", it is often required that the neighbourhood of any point is homeomorphic to a disk (or a half disk in case of a mesh representing a surface with boundary). For instance, T-junctions in which three faces are incident to an edge are to be avoided. Such a consistent mesh is called a *two-manifold*. Many methods have been introduced during these last years to transform a given inconsistent mesh into a two-manifold one, see [ACK13] for a review. However, most of the time the user lacks control of the result. In particular, the topology (number of handles and number of connected components of the mesh) may be ambiguous in the input data and the result may not be the one desired.

Here, I describe a method which converts an inconsistent mesh into a two-manifold interactively. Several possible topologies are suggested to the user, who takes the final decision (see Fig. 2.2 for an example). The knowledge about the shape is thus jointly provided by the program and the user.



(a) (b) (c)

Figure 2.2: *Two different repairs of the mesh depicted in Fig. 2.1 (a). (a) Genus 6 two-manifold mesh. (b) Genus 4 two-manifold mesh. (c) Close-up on the left side of the statue (back view): from top to bottom, input mesh, genus 6, genus 4.*

2.2.1 Surface singularities

We take as input a mesh with singularities, that is to say features that prevent the mesh from being a two-manifold. We distinguish three types of singularities:

- *Combinatorial singularities* prevent the mesh, seen as a combinatorial object, from being two-manifold [GTLH01]. This includes edges with not exactly 2

incident faces, isolated vertices, vertices whose link is neither a cycle or a chain, etc.

- *Geometrical singularities* prevent the mesh, embedded in $\mathbb{R}^3$, from being the boundary of a solid 3D object [RC99]. For instance, faces intersecting in their interior create a geometrical singularity.

- *Topological singularities* are unwanted handles or connected components which prevent the surface from having the desired topology.

A mesh with singularities can be as general as a polygon "soup" (Fig. 2.1 (a)), that is to say a set of polygons, together with their incident vertices and edges, without any explicit combinatorial relation.

The mesh repair algorithm presented below is able to remove all geometrical, combinatorial and topological singularities from an arbitrary polygonal mesh. This algorithm converts the mesh into a voxel set, called a *discrete membrane*, and iteratively applies morphological operators to detect areas which are likely to accept topologically different reconstructions. The user then chooses the desired topology, before the model is converted back to a two-manifold mesh.

2.2.2 Discrete voxel membrane

A *discrete membrane*, described in [EBV05], is a set of vertex-connected voxels of the space which divides the remaining voxels into the interior of a shape and its exterior. This means that there is no face-connected voxel path that goes from an interior voxel to an exterior one without intersecting the membrane. A discrete membrane has the advantage of being a coarse approximation of the input triangles while already being almost a one-voxel-thick two-manifold.

A discrete membrane is initialised as the boundary of the voxelisation. It is then contracted using sets of $n \times n$ voxels that form a square parallel to a coordinate plane with decreasing values of n. The voxels containing the input mesh triangles locally terminate the shrinking; the process is stopped when there is nowhere on the membrane to be contracted.

2.2.3 Interactive topology modification

In order to track down areas of the object where topology may be wrong (that is to say, irrelevant handles creating tunnels or connecting different components, or on the contrary missing tunnels or bridges between several parts of a connected component), we use two morphological operators, *opening* $\mathcal{O}^n$ and *closing* $\mathcal{C}^n$. An opening of order n is a sequence of n erosions followed by n dilations, while a closing of order n is a sequence of n dilations followed by n erosions. Openings can expand holes and disconnect parts, while closings can close holes and connect previously disconnected

parts of the volume, as shown in Fig. 2.3.

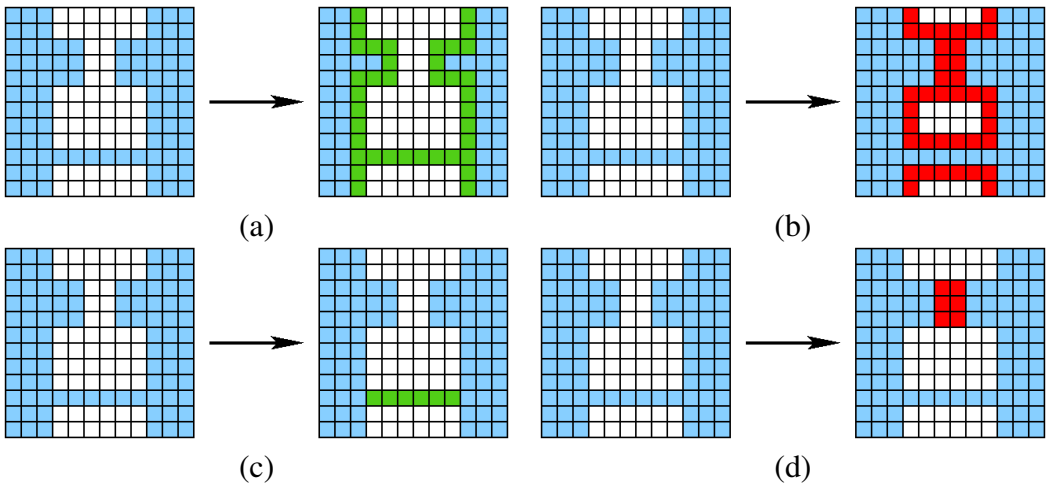


Figure 2.3: *Morphological operators of order* 1 *applied on a 2D set of blue pixels: (a) erosion, (b) dilation, (c) opening, (d) closing. Removed pixels are in green while added pixels are in red.*

We apply these operators on the set $\mathcal{V}$ of voxels which includes both the discrete membrane and the interior voxels, for a user-chosen value of n. Varying n enables one to detect topologically ambiguous areas of various sizes. We cluster voxels of $\mathcal{V}\backslash\mathcal{O}^n$ and $\mathcal{C}\backslash\mathcal{V}$ in 26-connected components. For each component $\mathcal{K}$, we check the topology (i.e., the number of connected components, tunnels and cavities) of the new set of voxels $\mathcal{V}\backslash\mathcal{K}$ or $\mathcal{V}\cup\mathcal{K}$, and compare it to the topology of $\mathcal{V}$. If one of them changes, we have detected a topologically critical area, which is displayed to the user.

Once the voxel set $\mathcal{V}$ is corrected with the desired topology, we compute a two-manifold isosurface using a topologically robust variant of the Marching Cubes algorithm [LC87]. The resulting mesh is subsequently smoothed using the bilateral denoising algorithm of Fleishman et al. [FDCO03]. More details about this method can be found in Appendix A.1.

2.3 Retrieving the homology of simplicial complexes

It is sometimes interesting not only to capture geometric but also topological features of a shape. These features are invariant under continuous deformations and provide global rather than local information about the shape. In the previous section, the topology was to be chosen by the user among several possibilities. In this section, we focus on topologically complex shapes for which the user wants to enrich his knowledge of

the object. This is typically the case in Computer-Aided Design for industrial shapes which have been assembled from different pieces, see Fig. 2.1 (b) for an example. These shapes are usually represented as manifold-by-part meshes, with each piece being a manifold mesh with boundary. Retrieving global information about the shape is necessary to deduce how pieces are assembled. This is often visually impossible, thus computational tools are required.

In order to retrieve such complete topological information, we have suggested in this work to rely on the *constructive homology* theory from Prof. Francis Sergeraert [Ser94], who explained the details of this theory to us in person. Homology is one of the most useful and algorithmically computable topological invariants. It characterises a mesh (technically, a simplicial complex) through the notion of *homological descriptors*. Homological descriptors are defined in any dimension k and are related to the non-trivial k-cycles in the complex which have intuitive geometrical interpretations up to dimension two. In dimension zero, they are related to the connected components of the complex, in dimension one, to the tunnels and the holes, and in dimension two, to the shells surrounding voids or cavities. Constructive homology focuses on homology computation. It provides a tool, the *constructive Mayer-Vietoris sequence*, which offers an elegant way of computing the homology of a simplicial complex from the homology of its sub-complexes and of their intersections. This is precisely what is needed for our purpose.

2.3.1 Background on simplicial homology

I will now lay down the ground work and background of simplicial homology.

A *k-simplex* $\sigma = [v_0, \ldots, v_k]$ is simply the convex hull of a set V of $k+1$ affinely independent points in $\mathbb{R}^n$ (with $n = 2$ or 3 in our case). k is called the *dimension* of the simplex. For example, a 0-simplex is a point, a 1-simplex is an edge connecting two points, a 2-simplex is a triangle, and a 3-simplex is a tetrahedron. For every non-empty subset $T \subset V$, the simplex σ' spanned by T is called a *face* of σ. A *simplicial complex* X is a collection of simplices such that all the faces of any simplex in X are also in X and the intersection of two simplices is either empty or a face of both. The dimension of the simplicial complex is defined as the largest dimension of any simplex in X. A subset Y of a simplicial complex X is called a *sub-complex* of X if Y is itself a simplicial complex. Each k-simplex of a simplicial complex X can be oriented by assigning a linear ordering to its vertices. The *boundary* of an oriented k-simplex is defined as the alternate sum of its incident $(k-1)$-simplices:

$$d_k([v_0, \ldots, v_k]) = \sum_{i=0}^{k} (-1)^i [v_0, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k].$$

Homology is defined on simplicial complexes thanks to an algebraic object called a *chain complex*. Let X be a simplicial complex and n be its dimension. A *k-chain* is defined for each dimension $k \leq n$ as $a_k = \sum_i \lambda_i \sigma_i^k$, where $\lambda_i \in \mathbb{Z}$ are coefficients

assigned to each k-simplex σ_i^k of X. The kth *chain group*, denoted as $C_k(X)$, is formed by the set of k-chains together with the addition operation, defined by adding the coefficients simplex by simplex. The set of oriented k-simplices of X forms a canonical basis for this group. The *chain complex*, denoted as $C_* = (C_k, d_k)_{k\in\mathbb{N}}$, is the sequence of the chain groups $C_k(X)$ connected by the boundary operator d_k:

$$(C_*, d_*) : 0 \xleftarrow{0} C_0 \xleftarrow{d_1} C_1 \xleftarrow{d_2} \dots \xleftarrow{d_{n-1}} C_{n-1} \xleftarrow{d_n} C_n \xleftarrow{0} 0.$$

The chain complex $C_*(X)$ can be encoded as a set of pairs $\{(B_k, D_k)\}_{0\leq k\leq n}$, where B_k is the canonical basis of C_k and D_k is an integer matrix, called the *incidence matrix*, which expresses the boundary operator with respect to B_{k-1} and B_k. Such a matrix is usually expressed in a special basis called the Smith basis, and is then named a *Smith normal form* (SNF) [Mun99]. Given a chain complex $C_*(X)$, homology groups are derived from two specific subgroups of the chain groups defined by the boundary operators:

$$Z_k = Ker\ d_k = \{c \in C_k(X) | d_k(c) = 0\}$$

and

$$B_k = Img\ d_{k+1} = \{c \in C_k(X) | \exists a \in C_{k+1} : c = d_{k+1}(a)\}.$$

For each $k \in [0, n]$, the kth *homology group* of X is defined as the quotient of the cycle group over the boundary group, i.e., $H_k = Z_k/B_k$. Thus, the elements of the homology group are equivalence classes of k-cycles which are not k-boundaries. H_k can be written as a disjoint sum:

$$H_k = \mathbb{Z} \oplus \cdots \oplus \mathbb{Z} \oplus \mathbb{Z}/\lambda_1\mathbb{Z} \oplus \cdots \oplus \mathbb{Z}/\lambda_p\mathbb{Z}. \tag{2.1}$$

The number of occurrences of $\mathbb{Z}$ in the free part $\mathbb{Z} \oplus \cdots \oplus \mathbb{Z}$ is called the kth *Betti number* β_k. It corresponds to the maximal number of independent k-cycles that do not bound the complex. For instance, β_0 is the number of connected components, β_1 is the number of tunnels (i.e., the genus if X is a surface) and β_2 is the number of cavities (voids) inside the object if X is volumetric. The values $\lambda_1, \dots, \lambda_p$ are strictly greater than one and such that λ_i divides λ_{i+1}. They are called *torsion coefficients*. Intuitively, the torsion coefficients characterise the non-orientable aspect of X. An orientable manifold has no torsion coefficient. The decomposition of Eq. 2.1 also shows that there exists a finite number of independent equivalent classes from which we can deduce all elements of H_k. Any set composed of one element of each of these classes is called a *set of generators* for H_k. The generators, Betti numbers and torsion coefficients form the complete homology information of the simplicial complex X.

2.3.2 Constructive homology

Constructive homology has been developed in order to reformulate homology concepts into concepts with a computational nature, thus leading to effective implementable algorithms. It handles homology computations over chain groups of infinite dimension

[Ser94]. One fundamental concept is that of *reduction* which relates, through a morphism, a large chain complex $\hat{C}_*$ to a small one C_*. C_* is constructed so that it contains the same homological information in the most compact way. The morphism can itself be represented as a chain complex, using a notion called the *cone of a morphism*. Further explanations can be found in Appendix A.2.

In our work we introduce a specific kind of reduction which we call the *homological Smith reduction*. Given a simplicial complex X of finite dimension n, this reduction relates its chain complex, $X*$, to a very small chain complex, $EX*$, which contains only the homological information of $X*$. This information is computed through the SNF algorithm, which transforms each incidence matrix into its Smith normal form.

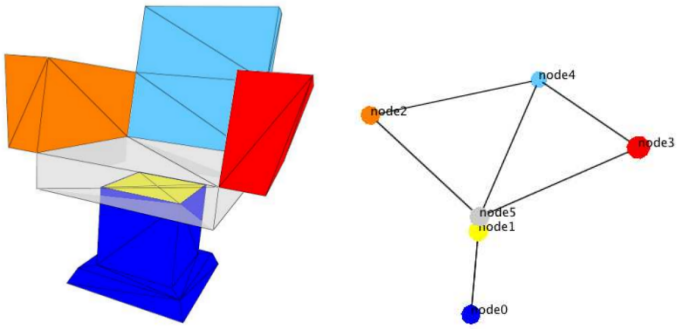
2.3.3 Manifold-Connected decomposition

We use a *Manifold-Connected (MC) decomposition* of X [HF07] to compute the complete homology information of a simplicial complex X through homological Smith reductions. Such decomposition is based on the notion of *manifold connectivity*. Given a regular simplicial complex X in which all simplices with the maximum dimension share the same dimension, two k-simplices (with $k \leq n$) σ and σ' in X are *manifold-connected* if there exists a path P joining them. P is formed of k-simplices such that any two consecutive k-simplices in P are adjacent through a manifold $(k-1)$-simplex. A regular simplicial complex in which every pair of n-simplices are manifold-connected is itself said to be manifold-connected. The unique MC decomposition of a regular simplicial complex X is a collection of all *manifold connected components* in X. They are the equivalence classes of the maximum dimension simplices of X with respect to the manifold-connectivity relation. The MC decomposition of a non-regular complex is the collection of the MC decompositions of its maximal regular sub-complexes of X.

The MC decomposition of a simplicial complex can be encoded as a graph in which each MC component is a node and each intersection between two MC components is an arc. Figure 2.4 shows an example of a MC decomposition and the associated graph.

2.3.4 Mayer-Vietoris algorithm

The algorithm we developed to compute the full homological information of a simplicial complex X begins by computing the MC decomposition of X and its associated graph G. It then iteratively computes the homology of the union between two neighbouring MC components A and B, connected through an arc in G, and then merges the two components. The corresponding nodes are also merged in G. The algorithms terminates when the graph consists of a single node. The final computed homology is that of X.



(a) (b)

Figure 2.4: *(a) MC decomposition of a simplicial complex. Each MC component is shown in a different colour. (b) The associated graph.*

The homology of the union $A \cup B$ of two complexes A and B is computed in the following way. First, the homological Smith reductions of the three complexes A, B and $A \cap B$ are computed. As a result, the reduced chain complexes $EA*$, $EB*$ and $E(A \cap B)*$ are homology equivalent to the large chain complexes $A*$, $B*$ and $(A \cap B)*$ but they contain only the homological information. Then, the constructive Mayer-Vietoris sequence is built. The *Mayer-Vietoris sequence* is an algebraic tool which relates the chain complex of the union $(A \cup B)*$ to the chain complexes of the disjoint sum $A * \oplus B*$ and the intersection $(A \cap B)*$ through morphisms, see Appendix A.2 for details. We then use a theorem, called the *Short Exact Sequence (SES) theorem*, which gives an homological equivalence between the cone of the inclusion morphism $i : (A \cap B)* \rightarrow A * \oplus B*$ and the chain complex of the union $(A \cup B)*$. Another theorem, the *cone reduction theorem*, gives another homological equivalence between i and the cone of the inclusion morphism Ei, which relates the reduced chain complexes $EA * \oplus EB*$ and $E(A \cap B)*$. This last chain complex is very small (in terms of number of simplices) and contains only the homological information of the sub-complexes A, B and $A \cap B$. Its homology is then obtained through its homological Smith reduction. Using two last reductions, we are finally able to extract the homology information of $A \cup B$.

This algorithm is summarised in Fig. 2.5. Some one- and two-dimensional generators for the sump model of Fig. 2.1 (b) are also shown.

2.4 JUST-NOTICEABLE DISTORTION PROFILE FOR MESHES

Complementary to works presented above, I have also focused on the perceptual comprehension of shapes described by two-manifold meshes. The goal here is to obtain knowledge of how a human being visually perceives a mesh. In particular, the question tackled so far is: to what extent two meshes are perceived identical? This is a

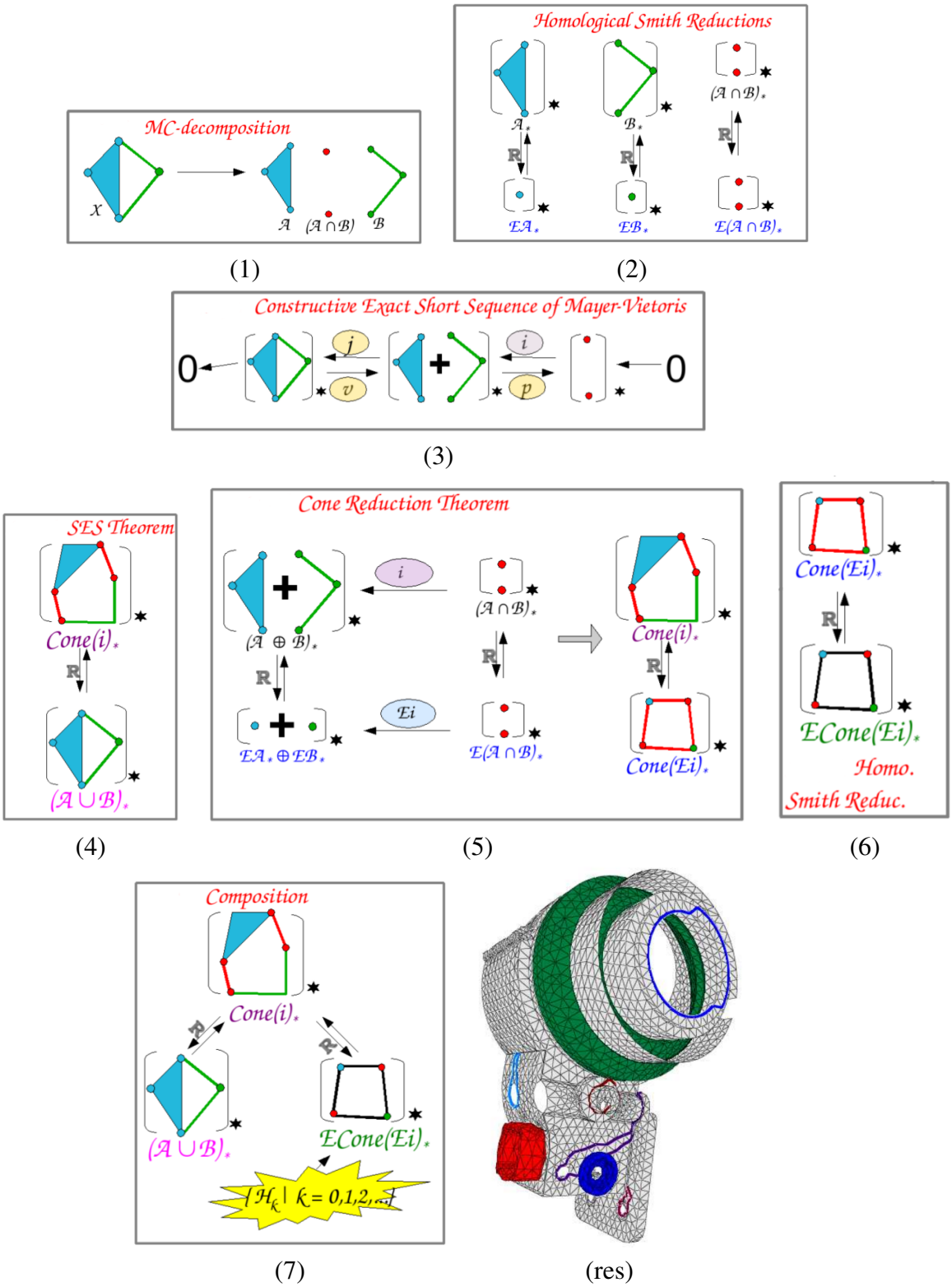


Figure 2.5: *(1) to (7): Successive stages of the homology computation algorithm. (res): Some* $1-$ *and* $2-$*generators of the homology groups* H_1 *and* H_2 *computed on the sump model of Fig. 2.1 (b).*

key question in assessing the quality of mesh processing algorithms, for instance for simplification, compression or watermarking purposes, which try to replace a given mesh by another one with the least visible differences. In this section, I describe a bottom-up approach to the perception of differences between meshes. The method relies on the known properties of the human visual system and is based on a series of user experiments. These experiments define a Just-Noticeable Distortion (JND) profile for two-manifold meshes, which describes how much any vertex of a given mesh can be displaced without the change being noticed by the majority of users (see Fig. 2.8 (c)). Applications to the perceptually optimal quantization and simplification of meshes are shown.

This work forms part of the PhD thesis of Georges Nader.

2.4.1 Local perceptual properties

It is well known that for a human being the visibility of a visual pattern depends both on its *local contrast* and its *spatial frequency* [Wan95]. Local contrast refers to the change of light intensity over the light intensity of its surrounding. The spatial frequency is defined as the size of light patterns on the retina. It is expressed in units of cycles per degree (cpd). To quantify how these properties affect the visibility of a visual stimulus, two main concepts have been introduced in the literature. First, the concept of *contrast sensitivity function* (CSF) describes the visibility threshold with respect to spatial frequency. For any stimulus, CSF usually exhibits a peak at around 2 to 5 cpd then drops off to a point where no detail can be resolved. The exact shape of a CSF curve depends on the nature of the stimulus. The second concept, *contrast masking* is the change in visibility of a stimulus (target) due to the presence of another stimulus (mask). The rate of change of the visibility threshold of the target with respect to the applied mask quantifies the effect of contrast masking. The curve possesses two asymptotic regions. The first region is for mask contrast values below the mask visibility threshold and is constant indicating that there is no masking effect. The second region occurs for higher mask contrast values and has a positive slope of about 0.6 to 1, depending on the stimulus [LF80].

In the case of a 3D mesh, we have defined the contrast between two adjacent faces as:

$$c = \| \cos\alpha \times \tan\theta \times \tan\frac{\phi}{2} \|, \tag{2.2}$$

where α and θ are the spherical coordinates of the light direction in the local coordinate system defined by $\vec{n_1} - \vec{n_2}$, $\vec{n_1} + \vec{n_2}$ and their outer product (see Fig. 2.6). ϕ is the angle between the normals of the two faces. This formula takes into account both the scene illumination and the surface geometry. A direction of light close to the normal direction ($\theta \approx 0°$) will minimise the value of the contrast, as will a smooth surface ($\phi \approx 0°$).

The spatial frequency has been defined as:

$$f = \frac{2d_{obs} \times \tan \frac{1}{2} \frac{\pi}{180}}{n_{px}/ppcm} \approx \frac{d_{obs}}{n_{px}/ppcm} \times \frac{\pi}{180}, \tag{2.3}$$

where d_{obs} is the observer's distance to the screen in cm, n_{px} is the number of pixels occupied by the visual stimulus, and $ppcm$ is the number of pixels in 1 cm of the screen. Our visual stimulus is the perspective projection on the screen of the segment between opposing vertices of adjacent faces.

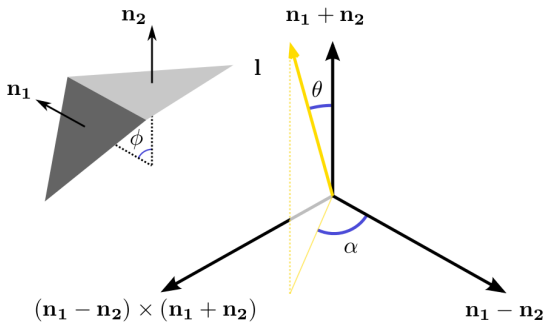


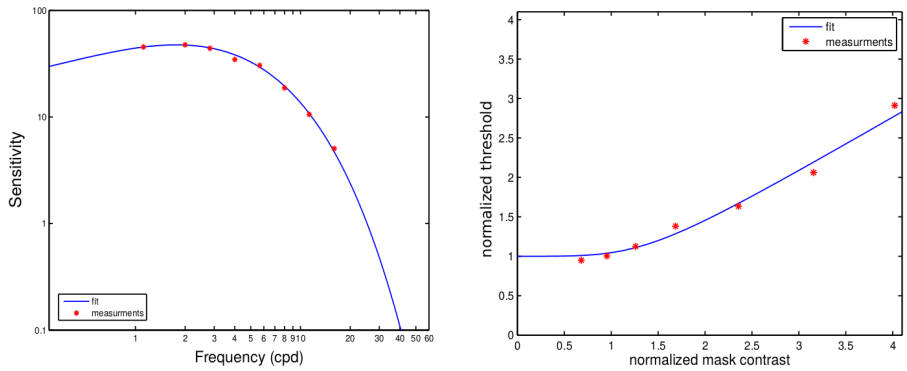
Figure 2.6: *The contrast between adjacent faces is computed using the angle between their normals and the spherical coordinates of the light direction in the local coordinate system defined by the faces normals.*

We have then carried out two experiments to compute the contrast sensitivity function and the contrast masking curve, respectively. In the first experiment a vertex is moved from the surface of a regular plane whose contrast is zero. We display the resulting surface side by side to the original, flat plane and ask the observer if he/she notices any difference between them. In the second experiment a vertex is shifted from a sphere approximated by a three-times subdivided icosahedron. The contrast between two adjacent faces of the sphere (stimulus of about 2 cpd) is visible for an observer and represents the mask signal. Details of these experiments are provided in Appendix A.3. Results are shown in Fig. 2.7. Note that in the second experiment both the mask contrast and the visibility threshold are normalised by the mask CSF.

2.4.2 Just Noticeable distortion profile

We are now able to compute the threshold T necessary to detect a difference between a pair of adjacent faces considered as the mask, and its adjacent faces considered as the target stimulus. Using the previously defined local contrast c and spatial frequency f of a pair of adjacent faces, as well as the computed CSF and contrast masking curve, the threshold T is defined as:

$$T = \frac{masking(c \times csf(f))}{csf(f)}. \tag{2.4}$$



(a) (b)

Figure 2.7: *(a) Computed mean contrast sensitivity function over the observers. (b) Computed mean normalised contrast masking curve over the observers.*

When a vertex of the mesh is moved the associated threshold, T, allows one to compute the probability of detecting the change in local perceptual properties. This probability is defined by the so-called *psychometric function* p [Dal93]:

$$p(\delta c) = 1 - e^{-(\delta c/T)^{\beta}}, \tag{2.5}$$

where δc is the local contrast difference before and after the displacement of the vertex and $\beta = 3.6$.

We are now able to compute the vertex displacement threshold, above which the change in local perceptual properties would be visible. This is done by computing, for each pair of faces affected by the displaced vertex, first the evolution of the local perceptual properties at increasing displacement magnitudes, then the visibility threshold T and the detection probability p as defined in Eq. 2.4 and 2.5, respectively. This results in a set of curves describing the probability of detecting a change for a given pair of faces with respect to the displacement magnitude, see Fig. 2.8 (b). The displacement threshold is then set to the magnitude where the probability of the highest curve reaches 0.95.

2.4.3 Perceptually optimal vertex coordinates quantization

The computed JND profile provides a simple way of determining automatically the optimal quantization level of any two-manifold mesh. That is to say, the optimal quantization level (in bits per coordinate, bpc) has the highest quantization noise energy given that the distorted mesh remains visually indistinguishable from the original mesh. The quantization noise for a given vertex is computed as the norm of the displacement from the vertex on the original mesh to the vertex on the quantized mesh. The mean over all vertices of the ratio between this noise and the JND value for the vertex gives a score s. If $s > 1$, the noise is over the visibility threshold. We define the perceptually optimal

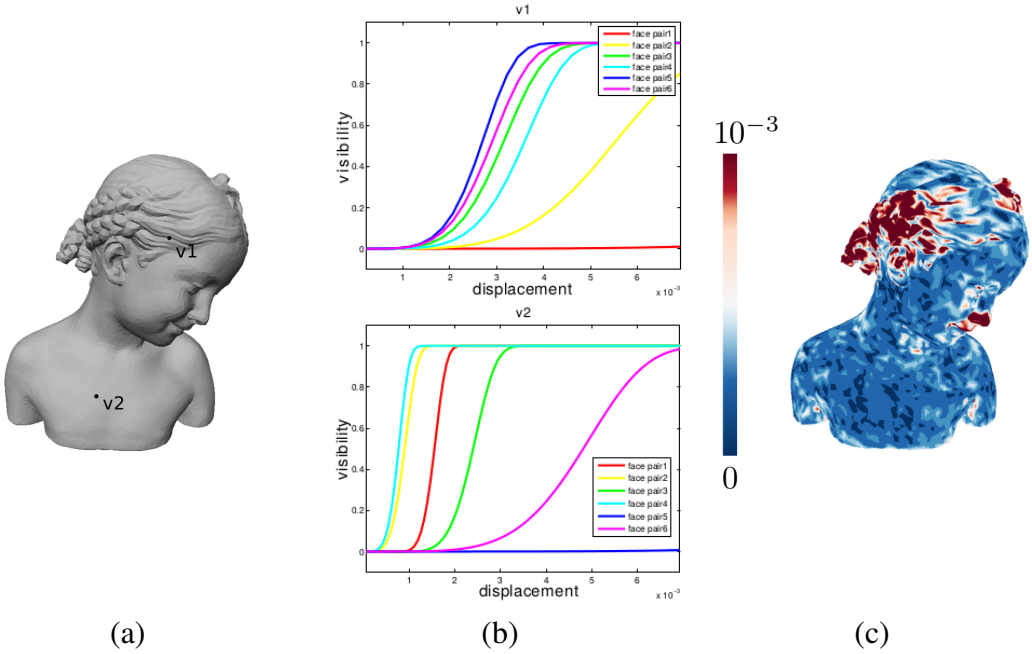


Figure 2.8: *(b) Detection probability curves for two vertices of a mesh (a). (c) JND profile for the same mesh and vertex displacements in the normal direction of each vertex in a light independent mode.*

quantization level as the highest quantization level with a score $s \leq 1$. It allows us to store efficiently the mesh in memory.

2.4.4 Perceptually optimal mesh simplification

The JND profile also enables to simplify a mesh to a perceptually similar mesh, without any additional parameter such as a targeted number of vertices. This can be done using the classical *edge collapse* operation, which merges two neighbouring vertices into one. The choice of the vertices to merge and the position of the resulting vertex are computed using the JND profile. Details are given in Appendix A.3. Edge collapses are iterated until the next collapse would introduce a visible distortion on the mesh.

2.5 Regular volumetric discretisation

In this section, I describe work highly relevant to the aims of the Morpheo team but also to a wider community. We consider a two-manifold mesh which is a watertight boundary digital representation of a 3D object. Our aim is to generate a volumetric discretisation of this object. In other words we want to sample its interior with uniform, anisotropic and regularly spaced elements which partition the whole shape. In the context of the work currently carried out within the Morpheo team, such a volumetric sampling is useful to track shape deformation, since it allows us to define additional

volume constraints on the deformations [AFB15]. Other potential applications include particle-based physical simulation, finite-element modelling, etc.

This work forms part of the PhD thesis of Li Wang. It is described with more details in Appendix A.4.

2.5.1 Regular Centroidal Voronoi Tessellations

Centroidal Voronoi Tessellations (CVTs) are Voronoi tessellations such that each site is located at the centre of mass of its cell. Among existing point-based volumetric shape decomposition schemes, CVTs are good candidates for our purpose thanks to their regularity property. Gersho's conjecture [Ger79] states that all cells of a distortion-minimising CVT are congruent to the same polytope, which only depends on the dimension of the space. In other words, cells are all the same shape. Gersho was working on information quantization and was interested in the following question. Suppose we want to approximate a continuous space by a defined number of discrete points which lie in the space. Where should these points be located if each point is placed such that it approximates the part of the space closer to it than to the other points? His conjecture has been proven in 2D, for which the polytope is a hexagon (Fig. 2.9 (a)): a site approximates all points in a hexagon around it. It has also been proven in 3D for lattice-based CVTs: the optimal lattice is body-centred cubic (BCC) (Fig. 2.9 (b)), for which the polytope is a truncated octahedron (Fig. 2.9 (c)). We discard other solutions such as Delaunay tetrahedrisations, since their cells would be more isotropic.

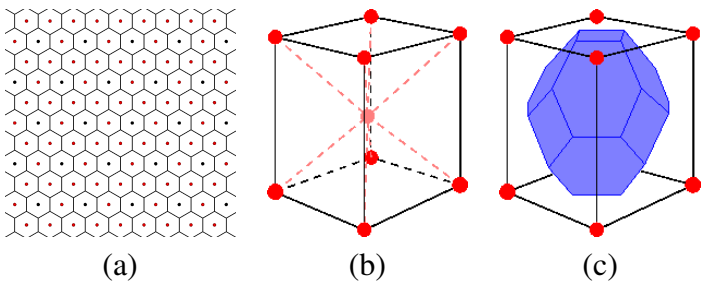


Figure 2.9: *Optimal CVT cells. (a) Optimal 2D cells are hexagons. (b) The optimal 3D lattice is the BCC lattice, for which a site is added at the centre of each cell of a cubic grid. (c) Optimal 3D cells are truncated octahedra.*

The quality of a given CVT is usually evaluated using the so-called CVT energy function which I now define. Let $X = \{\hat{x}_i\}_{i=1}^n$ be a set of n sites defining a CVT, $\{\Omega_i\}_{i=1}^n$ the associated cell decomposition of the shape and ρ a positive density function, then the CVT energy function is defined as

$$E(X) = \sum_{i=1}^{n} F_i(X) = \sum_{i=1}^{n} \int_{\Omega_i} \rho(x) \|x - \hat{x}_i\|^2 \, \mathrm{d}\sigma, \tag{2.6}$$

where $\mathrm{d}\sigma$ is the area differential.

An optimal CVT is obtained when a minimum of this function E is reached. Unfortunately, this function depends on both the shape size and the number of sites, making it difficult to compare across different shapes or discretisations. Inspired by early works from Conway and Sloane [CS82], we rather suggest to evaluate CVTs using a criterion we call a *regularity criterion* which is dimensionless and defined as:

$$\overline{G}(X) = \frac{1}{n}\sum_{i=1}^{n} G(\Omega_i) = \frac{1}{nm}\sum_{i=1}^{n} \frac{\int_{\Omega_i} \|x - \hat{x}_i\|^2 \,\mathrm{d}x}{\left(\int_{\Omega_i} \mathrm{d}x\right)^{(m+2)/m}}, \tag{2.7}$$

with m the dimension of the embedding space.

Both criteria are related since in the case of a uniform tessellation (uniform density function), the optimal value E_m of an infinite CVT with n sites and volume V for each cell is:

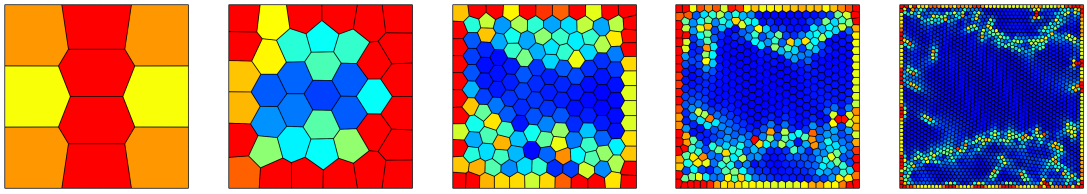
$$E_m = mnV^{(m+2)/m}G_m,$$

where G_m is the optimal m-dimensional cell quantizer, that is to say $G_2 = \frac{5}{36\sqrt{3}} = 0.0801875\ldots$ for a hexagon and $G_3 = \frac{19}{192\sqrt[3]{2}} = 0.0785433\ldots$ for a truncated octahedron. However, our regularity criterion does not depend on the shape size or on the number n of sites. Therefore, it allows for a general evaluation of CVTs, for instance, to check if the sampling is dense enough to get a regular tessellation. Note that a CVT is usually stable since it is computed using an integral and not a derivative.

2.5.2 A hierarchical framework

Existing methods to compute a CVT usually alternate between the construction of the Voronoi tessellation of the sites (the tessellation is clipped to the boundary of the object, i.e. to the input mesh) and the update of the site positions towards the centre of mass of the Voronoi cells. Based on the observation that a CVT with a small number of sites is more likely to be regular than with a large number of sites [LSPW12], here we devise a hierarchical strategy which reveals better regularity than existing approaches (see Appendix A.4 for the details).

We first build a CVT with a small number of sites, automatically computed from the desired final number n of sites. This CVT is then iteratively subdivided by adding new sites initially located at the midpoints of the dual Delaunay edges. Iterating the subdivision-update process tends to increase the area where the CVT is optimal for regularity, as shown in Fig. 2.10.



(a) (b) (c) (d) (e)

Figure 2.10: *Hierarchical CVT computation. From an initial CVT with* 10 *sites (left), successive subdivisions and updates lead to CVTs with* 40, 160, 640 *and* $n = 2560$ *sites (from left to right). The cell regularity measure* $G_2(\Omega_i)$ *is colour-coded from blue (regular) to red (far from regular).*

2.6 PERSPECTIVES

In this chapter I have presented a full pipeline to create regularly tessellated volumes from unstructured meshes of their boundaries. My efforts have first focused on the topological and perceptual comprehension of these meshes, then on the discretisation of their interior into regular anisotropic cells. The short-term perspectives include extensions of the last two works which I now discuss. They are tackled as part of Georges Nader and Kai Wang's PhD theses.

2.6.1 Perceptual analysis of smooth shaded surfaces

The main limitation of the work presented in Section 2.4 is its restriction to flat shaded surfaces. We rely on the visibility difference between adjacent faces to define 3D local perceptual properties. In the case of smooth shaded surfaces, the mask signal is more complicated than a simple local contrast between two adjacent faces. Thus, in order to extend this work to smooth shaded surfaces, we need to take into account other aspects of the human visual system, such as entropy masking [Wat97] or the free-energy principle [Fri10], which model how the visual system adapts to uncertainty. A step further would be to define not only a JND profile but also a perceptual metric to compare meshes. While the JND profile only says if meshes are perceived similar or not, such a metric would provide a quantitative assessment of the perceptual difference between meshes.

2.6.2 Regularly tessellated volumes from point clouds

The framework introduced in Section 2.5 has been applied to shapes defined by their meshed boundary surfaces. Computed Centroidal Voronoi Tessellations are clipped to the mesh (see Appendix A.4 for the details). Theoretically, nothing prevents this framework from being applied to other shape representations. We are currently investigating the case of implicit surfaces. This is of particular interest since it offers an alternative smoother way of defining a surface from a point cloud (see e.g. [CBC+01, OBA+03, KBH06]). It would also be interesting to investigate how this

work can be extended to anisotropic cells, which align to some geometric features. L_p CVTs [LL10] are a good candidate for this. The behaviour of some L_p regularity criterion is still to be explored. Both the criterion and the hierarchical approach also need to be improved in order that the boundary cells of the CVT be more regular and adapt to the geometry of the surface, since the regularity properties are only proven only in an infinite, unbounded case so far. Optimisation of the boundary surface sampling together with the volume sampling is also still to be investigated. Finally, as stated in the previous section, this uniform, anisotropic volumetric tessellation has been used for shape tracking [AFB15]. Another short-term plan is to combine such volumetric decomposition of a moving shape with physical simulation methods, to generate new complex animations from a real motion.

CHAPTER 3

Digital geometry processing for shapes in motion

3.1 INTRODUCTION

3.1.1 Objectives

Our world is dynamic and not static: moving shapes are clearly of interest in many fields. There is a growing concern for the processing and recognition of 3D shapes in motion not only within the computer graphics and computer vision communities but also in other fields, for instance, among the medical imaging community (see [SFR+12] for an example). Of course, the speed of motion differs according to the application of interest. Whereas human characters acting in a scene are obviously moving shapes, the growth of a tree or a tumour can also be thought of as a moving object in biological or medical applications. However, despite much progress, there is still a lot of work to do, and this is in particular why the Morpheo team devotes itself to this topic.

Of particular interest in understanding a moving shape is the fact that its motion can help to identify the shape, since some redundancy can be expected between successive poses of the shape during its motion. [LB12] for instance recovers the correct topology of a model by studying the changes between successive poses during motion. Decomposing a shape into rigidly moving components, as in [FB11] or as will be described in Section 3.4, can also help to understand the functionality of the shape's parts, particularly if contextual knowledge is added to the process (e.g., if the shape is known to be a human shape, it is easy to identify where are the arms, the legs and the torso). It is also expected that, conversely, the shape can lead to a high-level comprehension of the motion, that is to say to deduce the activity performed by the model.

In this chapter, I describe my first contributions to the field, which are mostly focused on animated human or animal characters. In Section 3.2, I introduce a method to create realistic moving shapes from a static shape, with some control by the user. This method first computes an accurate animation skeleton by using a topological tool called the Reeb graph and prior knowledge about the shape's anatomy. Then it generates flexible skinning weights by decomposing the shape into overlapping parts. My method allows the user to create rigid or elastic movement. This work began in response to a request by an infographist. In Section 3.3, I describe a universal mathematical tool I am developing to process moving shapes. This tool is a proper discretisation of the Laplace operator for such objects. As a practical example, I show how this operator can be applied to edit the shape and the motion of a given character. Finally, I explain in Section 3.4 a way to retrieve the rigidly moving parts of a shape that is represented as an evolving mesh without temporal coherence and I show how to validate segmentations of a moving shape into rigidly moving components. Before doing so, let me introduce several general concepts.

3.1.2 3D+t vs. 4D

Although we add one dimension with respect to the previous chapter, I claim that the study of 3D shapes in motion is somehow different to the study of objects embedded in a general four-dimensional Euclidean space. Although it is of interest to devise universal tools (and I will introduce one in Section 3.3), the temporal dimension needs to be processed in a separate manner to the three geometrical dimensions. This is because we are interested in recognising the three-dimensional shape as well as its motion, and not the general four-dimensional behaviour of the moving shape. I elaborate on this in Section 3.3. Therefore, our subjects of study are described as three-dimensional meshes (usually two-manifold, since they describe the boundary of a watertight object) evolving though time.

3.1.3 Temporal coherence

Since we focus on shapes described by their meshed surfaces, a moving shape is usually defined as a temporal sequence of 3D meshes, sometimes called a *dynamic mesh*, a *time-varying surface* or a *3D video*. Two types of mesh sequence can be considered, depending on whether or not temporal coherence is assumed, i.e. there is a one-to-one correspondence between vertices of successive meshes. In computer graphics, a moving shape is usually created by modelling a static shape and then animating it (see Section 3.2). The temporal coherence is thus implicit, since the connectivity of the mesh remains the same at each time frame. The main drawback of such a representation is that the topology cannot change over time. On the contrary, in computer vision, a moving shape is usually defined as a sequence of static shapes captured at each time step. A standard approach is to reconstruct a 3D mesh from the visual hull of the object's silhouettes as seen from a set of cameras [SH07, FB09]. The lack of explicit temporal coherence is a major concern for any application, and many methods have

been proposed to *track* a template (which may or may not be the reconstruction of the first frame of the videos) over the mesh sequence [KBH12, HBNI14, AFB15].

In order to distinguish between mesh sequences with or without temporal coherence, we introduce the following definitions [ACH+13].

Definition 3.1.1 (Temporally coherent/incoherent mesh sequence (TCMS/TIMS))**.** *Let* $MS = \{M^i = (V^i, E^i, F^i), i = 1 \dots f\}$ *be a mesh sequence, where:* V^i *is the set of vertices of the* i^{th} *mesh* M^i *of the sequence,* E^i *its set of edges and* F^i *its set of faces. If the connectivity is constant over the whole sequence, that is to say if there is an isomorphism between any* E^i *and* $E^j, 1 \leq i, j \leq f$*, then* MS *is called a* temporally coherent mesh sequence (TCMS)*. Otherwise,* MS *is called a* temporally incoherent mesh sequence (TIMS)*.*

Note that the definition of TCMS not only implies that the number of vertices remains constant through time, but also that there is a one-to-one correspondence between the faces of any two meshes. This is why topological changes (genus and number of connected components) are not possible in a TCMS.

Figure 3.1 shows an example of a TCMS and an example of a TIMS.

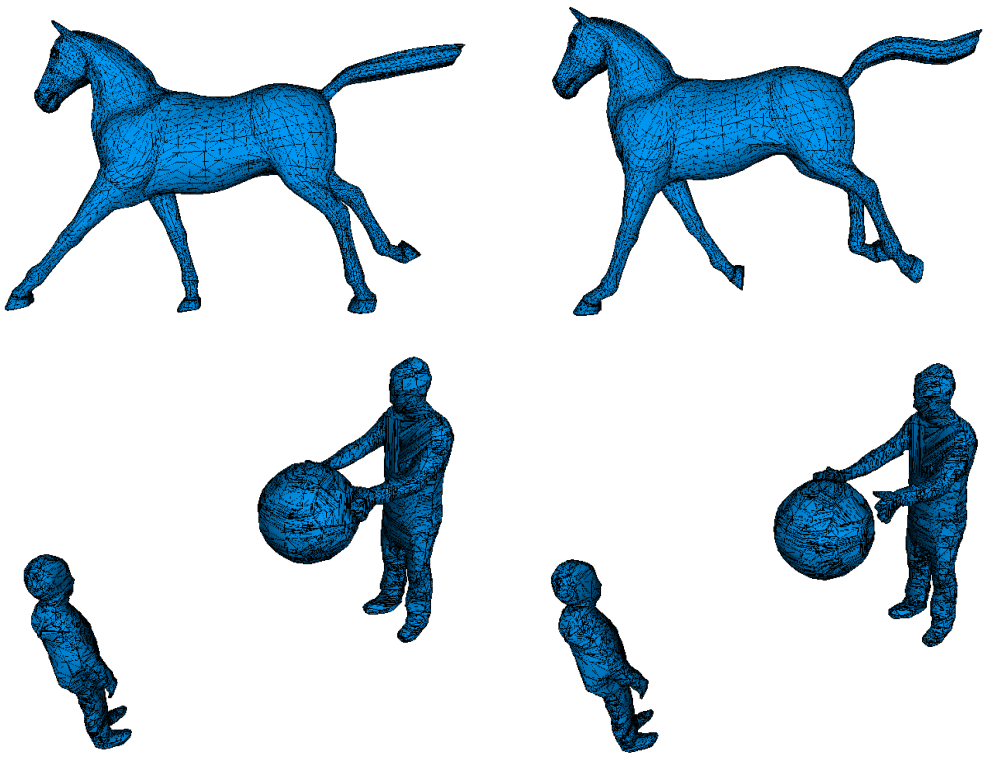


Figure 3.1: *First row: two consecutive frames of a TCMS. Second row: two consecutive frames of a TIMS (in particular, notice the change in topology).*

3.2 Harmonic skeleton for character animation

In this section I describe my contribution to the creation of a TCMS, from a static mesh only, using the standard skeletal animation framework. The TCMS is created by first generating an animation skeleton and then computing skinning weights. The prior knowledge used relates to the character's anatomy, and is encoded in the algorithm. This work is the result of collaboration with an infographist, Christine Depraz. She identified, from her point of view, the anatomical requirements of a skeleton that would generate a realistic animation.

3.2.1 A short survey on skeleton computation methods

Skeleton computation has been a popular topic in computer graphics and shape analysis for a long time, following early work on medial axis computation in image processing [Blu67]. Skeletons are useful for shape analysis [BGSF08], matching [HSKK01], registration [ZST+10], retrieval [BMSF06, TVD09, BB13], or animation and deformation [LCF00], which is the application we focus on hereafter. We restrict our overview to curve skeletons, that is to say skeletons which can be represented as graphs embedded in $\mathbb{R}^3$. This excludes medial representations; see [SP08] for an in-depth description of these. Although skeleton computation methods exist on point clouds [TZCO09, HWCO+13] and voxel sets [CSM07], we restrict ourselves to the case of meshes, which are of interest for the later application. Many different methods exist and they can be classified according to the criteria that they try to fulfil: centredness, homotopy equivalence to the shape, invariance under transformations, robustness against geometric noise, ... For ease of understanding, since a method can fulfil several criteria, I prefer to present them according to their methodological basis.

Segmentation-based methods

As stated in Chapter 1, segmentation and skeleton computation can be seen as dual problems. Consequently, several people have suggested beginning by decomposing a shape into meaningful parts so as to deduce a skeleton. Many methods exist, including [KT03] which is based on a hierarchical decomposition and which gives rather star-shaped skeletons. Lien et al. [LKA06] use centroids and the principal axes of a shape to build simultaneously a segmentation and a skeleton of the given shape. Dellas et al. [DMMT+07] have proposed a specific method to recover an animation skeleton from human scans. This method is based on the segmentation of the shape into semantically meaningful parts and uses prior knowledge about human anatomy. [JXC+13] have iterated a dual graph contraction and a mesh-face clustering process to generate a skeleton.

Force field and thinning-based methods

Following the method of [LcWcM+03], which uses a repulsive force field, several authors have suggested using a force field, which is not necessarily a distance field, to compute a skeleton inside a shape. For instance, [WML+06] combined a medial axis approach with a decomposition and a potential field. These two methods are time consuming, and the behaviour of the algorithm may be difficult to control. The first popular method of this kind was by [ATC+08]. It shrinks the mesh in order to obtain the skeleton. Here, the general idea is to use a volume reduction force field. It has been implemented in [ATC+08] as a constrained mean curvature flow. While this approach is very robust to noise on the surface, there is once again no real guarantee about the reliability of the topology of the resulting skeleton. This approach has been enhanced in [TAOZ12] to obtain a medially centred skeleton. [DS06] chose first to compute an approximation of the medial axis of the surface and then to erode it to get a curve skeleton possessing certain desirable properties, notably centredness and invariance under isometric deformation. [LCLJ10] have proposed a similar approach for cell complexes.

Topology-based methods

The *Reeb graph* is a topological tool defined on a surface for a given function f. It will be properly defined in Section 3.2.2. Once embedded in $\mathbb{R}^3$ and for a suitable function, it gives a useful topological skeleton of the shape. It has been used with several different functions for shape analysis [BGSF08]. The main difficulties in using a Reeb graph are the choices of the function and of the embedding. In our work (see Section 3.2.2), we choose to use a Laplacian function since it guarantees that the endpoints of the graph will be located exactly where they are expected. The same idea has been used independently in a different context (skeletonisation of blood vessels) by Yang et al. [YZH+05]. The embedding of a node of the Reeb graph is often chosen to be the average point on the surface of the corresponding function isoline. Additional features can be used to enhance the geometrical quality of the skeleton [TS05, TVD06].

Example-based methods

Some authors [SY07, HXS09, HTRS10] have suggested constructing a skeleton of a given shape based on previously created skeletons of the same shape under different poses. The main drawbacks are the need to design manually these example skeletons and the choice of the input poses, which may be critical.

Other approaches

The well known Level Set Diagram method of Lazarus and Verroust [LV99] is quite similar to a Reeb graph, except that the graph is not necessarily connected. The computation is fast and generates expected centrelines on tubular shapes. Garro and Giachetti have also suggested computing centrelines on tubular shapes, by choosing a set of

directions and perpendicular sweeping planes [GG13]. Baran and Popović's approach [BP07] is based on a sphere-packing strategy, which works well only for simple shapes since it is not easy to tune. Recently, Livesu et al. have proposed an original approach inspired by the visual hull computation [LGS12]. Their method is very fast and robust, although it may fail if the surface is topologically (high genus) or geometrically detailed.

Skeleton computation for animation

Animation skeletons are the standard control structure for animating 3D character models [LCF00]. Defined as a hierarchy of local reference frames, each frame corresponding to a joint, they consist of a simplified version of the true anatomical skeleton, with only the main joints represented. Pairs (parent, child) of frames are called the "bones" of the skeleton. It is natural to try to compute automatically an animation skeleton from the shape, i.e. the skin, of the character only. However, the bone structure of a human or animal character is not accurately related to its skin. For instance, the backbone does not lie in the middle of the torso but is, rather, close to the back. Hence, purely geometric methods fail to locate accurately the anatomical skeleton of the character to be animated, leading to non-realistic motion. Anatomical knowledge must be added to the process in order to generate a more accurate skeleton. It can either be learnt from a set of examples, or provided as a template. Both approaches have been tackled by various methods: see the dedicated section I wrote for a survey on quadruped animation for some examples [SRH+09].

3.2.2 Reeb graph computation

The first step of our approach is to build a Reeb graph over the input mesh M. Let f be a real-valued function over M. The *Reeb graph R of f* [Ree46] (see Fig. 3.2) is defined as the quotient space $M/\sim$, where $\sim$ is the following equivalence relation on M:

$$x_1 \sim x_2 \iff \begin{cases} f(x_1) = f(x_2) \\ \text{and } x_1 \text{ and } x_2 \text{ belong to the same} \\ \text{connected component of } f^{-1}(f(x_1)). \end{cases}$$

In other words, R is a graph whose nodes correspond to the critical points of f and whose edges encode the connectivity between them. The leaves of the Reeb graph exactly match the local maxima and minima of f.

The choice of the function f is key in revealing geometric information about the shape. In this work we will find a smooth function, whose extrema will be anatomically significant, by solving Laplace's equation $\triangle f = 0$. The main property of such an harmonic function is the absence of extrema except at boundary points. Thus, if we impose these boundary points, we control the leaves of our Reeb graph exactly. Moreover, it is well known that the Laplace equation with non-homogeneous Dirichlet

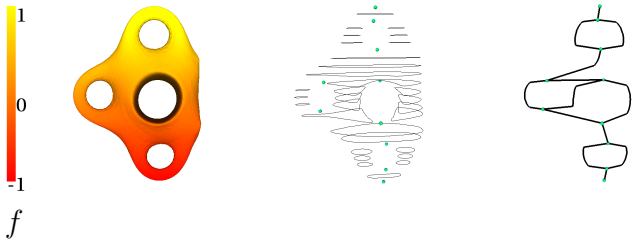


Figure 3.2: *From left to right: a surface, some level sets of f, the Reeb graph of f.*

boundary conditions, that is to say with imposed values $f(x)$ at boundary points x, admits a unique solution. Our pipeline thus includes the following operations:

1. the endpoints of the desired skeleton are chosen by the user or computed (however, at least one of them, called the source node, must be manually chosen on the head of the character);

2. the harmonic function f solving Laplace's equation with non-homogeneous Dirichlet boundary conditions is computed;

3. the Reeb graph of f is computed.

In a post-processing step, the Reeb graph is subsequently filtered to retrieve the symmetry of the character's morphology (i.e., overall structure), the symmetry axis of the graph is detected from the source node, and the graph is refined by inserting regular nodes. Details can be found in Appendix A.5.

3.2.3 Embedding into an animation skeleton

Note that so far we have used one fact that requires prior knowledge: the source node should be chosen by the user on the head of the character. This allows us to retrieve the symmetry axis of the graph, if it exists. This symmetry axis is key if we are to embed the graph in $\mathbb{R}^3$. Firstly, it allows us to distinguish easily between a biped and a quadruped character (except in the case of some amphibians and reptiles); see Fig. 3.3. Secondly, it allows us to divide the edges corresponding to the front and back legs differently, mimicking the anatomical leg bones. This is done automatically using anatomical references, as the embedding of the graph nodes.

20 animation skeletons generated for various quadrupeds using this approach are freely available here: `http://evasion.imag.fr/Membres/Franck.Hetroy/Projects/Skeleton/gallery.html`.

3.2.4 Atlas generation

Next, we propose a simple framework to compute flexible skinning weights that allow quasi-rigid to soft deformations by using the animation skeleton computed previously.

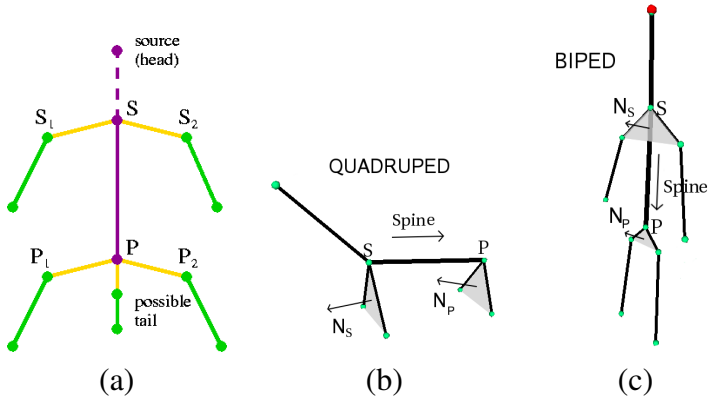


Figure 3.3: *(a) Computed Reeb graph for a biped or a quadruped model. The symmetry axis is coloured in purple. (b,c) The dot products* $Spine.N_S$ *and* $Spine.N_P$ *differentiate between a quadruped and a biped.*

The skeleton is used to guide the decomposition of the input mesh into a set of overlapping areas called an *atlas of charts*. First, the mesh is segmented into non-overlapping regions around the joints (embedded nodes) of the skeleton. Then, we grow the boundary curves between two adjacent regions; see Fig. 3.4.

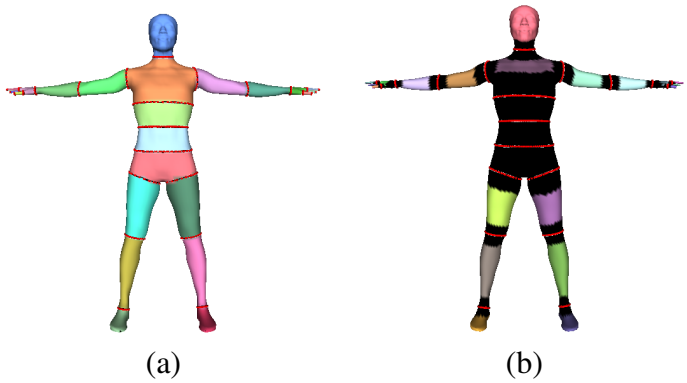


Figure 3.4: *(a) Shape segmentation according to the skeleton. (b) The boundary curves (in red) are grown into overlapping patches (in black).*

3.2.5 Skinning weights

The skinning weights are derived from a normalised geodesic distance map d_j on each extended region R_j, to the boundary of this region. A cubic function is used to smooth this map between 0 and 1. Vertices of the mesh belonging only to the region R_j are attributed the weight 1 for this region, while the weight for vertices on the boundary of R_j is set to 0. Vertices on overlap areas possess several weights, and their sum is equal to 1. Any skin deformation technique can be applied once these skinning weights have

been computed.

The proposed approach (see Appendix A.6 for the details) uses only one parameter per joint, K, which is related to the size of the overlap. Since our skeleton carries information about the shape's anatomy, this parameter could automatically be set for each joint according to its semantics. Manually tuning this parameter also allows for varied deformations; see Fig. 3.5.

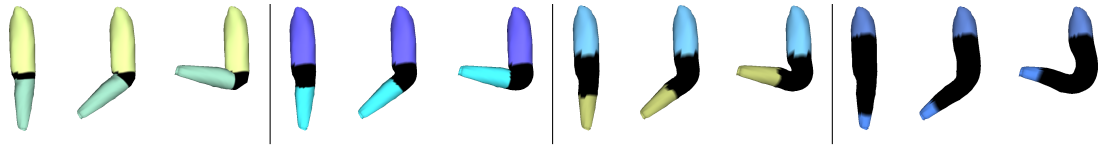


Figure 3.5: *Rest pose, medium deformation and large deformation around an elbow for increasing values of K. Deformations were created using the Dual Quaternion technique [KCZO07].*

3.3 A DISCRETE LAPLACE OPERATOR FOR TEMPORALLY COHERENT MESH SEQUENCES

The Laplace operator, encoding weighted differences in coordinates between a vertex and its neighbours, is a powerful tool for the analysis of static shapes [Sor06, LZ09, ZvKD10]. Several operators have been defined, for discrete representations ranging from point clouds [BSW09, LPG12, PPH+13] to triangular meshes [PP93, Tau95, MDSB03, BSW08, ARAC14] or more general polygonal meshes [BS07, AW11], weighted triangulations [Gli07], simplicial pseudo-manifolds [COT13]. These operators differ in the properties they satisfy [WMKG07, RBG+09].

In this work, I propose to define a discrete Laplace operator for temporally coherent mesh sequences (TCMS), as introduced in Def. 3.1.1. The suggested Laplace operator is based on the theory of Discrete Exterior Calculus (DEC) [DHLM05, CdGDS13], which has been proved to lead to the famous Laplacian *cotangent operator* for static meshes. For our purpose, a TCMS is modelled as a CW-complex (which is a cell complex slightly more general than a simplicial complex) in a four-dimensional non-Euclidean space, modelling spacetime. This 4D space consists of one temporal and three spatial dimensions. A parameter α enables to tune the influence of time with respect to the geometry. To the best of my knowledge, apart from Equation (3.1), Definition 3.3.7 and Properties 3.3.8 and 3.3.9, the remaining of this Section is new.

3.3.1 Definition of a 4D DEC Laplace operator

Let $\mathbb{E}$ be a 4-dimensional Riemannian manifold, equipped with a metric g such that the induced metric tensor $G = Diag(\alpha, 1, 1, 1)$, $\alpha > 0$. In other words, if $X_1 =$

(t_1, x_1, y_1, z_1) and $X_2 = (t_2, x_2, y_2, z_2)$ are two vectors in $\mathbb{E}$, then the *inner product* of X_1 and X_2 is defined as $\langle X_1, X_2 \rangle = \alpha t_1 t_2 + x_1 x_2 + y_1 y_2 + z_1 z_2$. In particular, the *norm* of a vector $X = (t, x, y, z) \in \mathbb{E}$ is defined as $\|X\| = \sqrt{\alpha t^2 + x^2 + y^2 + z^2}$.

$\mathbb{E}$ represents the embedding space of our mesh sequences. The first coordinate t of a vector $X = (t, x, y, z) \in \mathbb{E}$ is called its *time-like coordinate*, while the three others coordinates x, y and z are called its *space-like coordinates*. α is a user-defined parameter that describes the respective influence of space and time in the metric. Now, let us model a temporally coherent mesh sequence as a CW-complex embedded in $\mathbb{E}$.

Definition 3.3.1 (Temporally coherent mesh sequence embedded in $\mathbb{E}$)**.** *Let f be a positive integer, and $t^1 < t^2 < \cdots < t^f$ be f real numbers. Let $MS = \{M^k = (V^k, E^k, F^k), 1 \leq k \leq f\}$ be a TCMS as defined in Def. 3.1.1, such that $\forall k \geq 1, k \leq f$, all vertices in V^k share the same constant time-like coordinate t^k. Let us denote $V^k = \{v_i^k, 1 \leq i \leq n\}, E^k = \{e_i^k, 1 \leq i \leq m\}$ and $F^k = \{f_i^k, 1 \leq i \leq p\}$. Then the union of all $M^k, 1 \leq k \leq f$, together with:*

- *$n(f-1)$ additional edges between all v_i^k and $v_i^{k+1}, 1 \leq i \leq n, 1 \leq k \leq f-1$,*
- *$m(f-1)$ additional 2-cells between all e_i^k and $e_i^{k+1}, 1 \leq i \leq m, 1 \leq k \leq f-1$,*
- *$p(f-1)$ additional 3-cells between all f_i^k and $f_i^{k+1}, 1 \leq i \leq p, 1 \leq k \leq f-1$,*

forms a 3-dimensional CW-complex embedded in $\mathbb{E}$, called an embedded TCMS. *Edges $v_i^k v_i^{k+1}$ are called the* temporal edges *of the embedded TCMS. The other edges are called the* spatial edges *of the embedded TCMS.*

Figure 3.6 (a) depicts part of a temporally coherent mesh sequence. Since in our modelling temporal 2-cells are not triangles but (skew) quadrilaterals and 3-cells are not tetrahedra, our CW-complex is not a simplicial complex. However, its structure is manifold-like by construction: cutting each temporal 2-cell into two triangles generates a 3-manifold tetrahedrisation. Thus, Discrete Exterior Calculus can still be applied [COT13].

I refer to [DHLM05, CdGDS13] for the definition of the discrete exterior derivative $\mathbf{d}$ and the discrete co-differential operator δ. A discrete Laplace-Beltrami operator $\triangle_u$ of a function F defined on a temporally coherent mesh sequence's vertices $\{\sigma^0\}$ is defined as $\triangle_u = \delta \mathbf{d}$ [DHLM05]. This leads to [DHLM05, p. 23-24]:

$$\frac{1}{|\sigma^0|}\langle \triangle_u F, \sigma^0 \rangle = -\frac{1}{|\star\sigma^0|} \sum_{\sigma^1 \succ \sigma^0} \frac{|\star\sigma^1|}{|\sigma^1|}(F(v) - F(\sigma^0)), \tag{3.1}$$

where v is defined as $\partial\sigma^1 = v - \sigma^0$: the (oriented) boundary of edge σ^1 is algebraicly defined as the difference between vertices v and σ^0. $\star\sigma^0$ denotes the dual of vertex σ^0 and $\star\sigma^1$ is the dual of edge σ^1. $|\sigma^k|$ denotes the oriented volume of cell σ^k. This formula shows that the value of the Laplacian of F at σ^0 is computed using the value

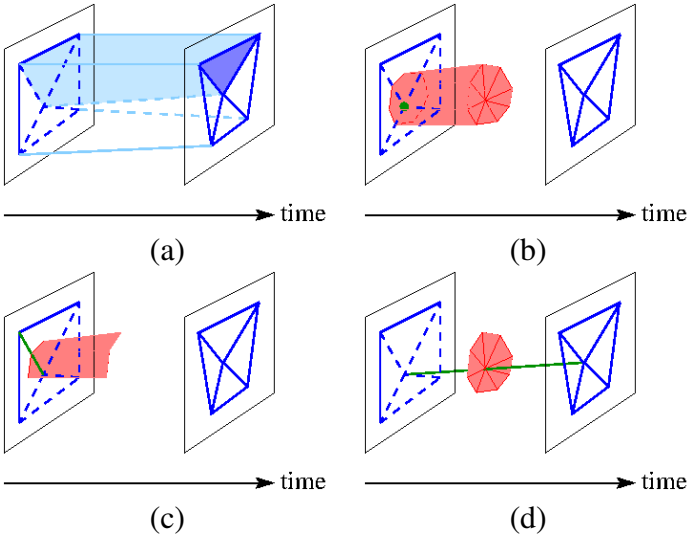


Figure 3.6: *(a) Modelling of a temporally coherent mesh sequence embedded in $\mathbb{E}$. In dark blue are shown spatial edges at two successive time instants t^k and t^{k+1} and a face of M^{k+1}. In light blue are shown corresponding temporal edges and a 3-cell of the CW-complex. (b,c,d) Barycentric dual cells (in red) of a (b) vertex, (c) spatial edge and (d) temporal edge, shown in green. Only parts of the cells with time-like coordinates between t^k and t^{k+1} are shown.*

of F at all neighbouring vertices v, since we sum over the edges σ^1 for which σ^0 is an endpoint.

By definition, the dual $\star\sigma^0$ of a vertex σ^0 is a 3-cell whose vertices are the centres of incident (spatial and temporal) edges, (spatial) triangles, (temporal) quadrilaterals and 3-cells. The dual $\star\sigma^1$ of an edge σ^1 is a 2-cell whose vertices are the centres of incident triangles or quadrilaterals and 3-cells. Figure 3.6 shows the dual cells of a vertex, a spatial edge and a temporal edge. Note that the dual of a spatial edge (Fig. 3.6 (c)) is a set of four temporal quadrilaterals (two only for the first and the last meshes of the sequence), the dual of a temporal edge (Fig. 3.6 (c)) is a set of triangles sharing the same time-like coordinate, and the dual of a vertex (Fig. 3.6 (b)) is a set of 3-cells with 6 vertices, defined by temporal quadrilaterals and spatial triangles.

The centre of a k-cell is chosen to be the isobarycentre (i.e., centroid) of the cell, as in e.g. [GP10, COT13]. Note that in general it is not possible to define circumcentres, thus circumcentric duals as in [Hir03, DHLM05, CdGDS13], because of the quadrangular temporal 2-cells.

The area of a temporal quadrilateral is not properly defined since points this quadrilateral is skew: its four points are not necessarily planar. In our case, we only consider quadrilaterals expressing the motion of an edge $p^t q^t$ from time-like coordinate $t = t^k$ to time-like coordinate $t = t^{k+1}$. As a consequence, we can define the area of the

corresponding 2-cell as the integral of the length of this edge over time, from t^k to t^{k+1}:

Definition 3.3.2 (Area of a temporal 2-cell)**.** *Let p^k, q^k, q^{k+1} and p^{k+1} be the ordered vertices of a temporal quadrilateral $Q^k_{i,j}$. Let t^k be the time-like coordinate of p^k and q^k, and t^{k+1} be the time-like coordinate of p^{k+1} and q^{k+1}. $\forall t \in [t^k, t^{k+1}]$, let $p^t = \frac{t-t^k}{t^{k+1}-t^k}(p^{k+1} - p^k) + p^k$ and $q^t = \frac{t-t^k}{t^{k+1}-t^k}(q^{k+1} - q^k) + q^k$. Then,*

$$|Q^k_{i,j}| = \left\| \int_{p^k}^{p^{k+1}} \|p^t q^t\| dp^t \right\|. \tag{3.2}$$

Similarly, the dual $\star\sigma^0$ of a vertex σ^0 is the union of several 3-cells $F^k_{i,j,l} = p^k q^k r^k p^{k+1} q^{k+1} r^{k+1}$ expressing the displacement of triangles $p^t q^t r^t$ from time-like coordinate $t = t^k$ to time-like coordinate $t = t^{k+1}$. We can thus define the volume of such a 3-cell as the integral of the area of the triangle $p^t q^t r^t$ over time:

Definition 3.3.3 (Volume of a temporal 3-cell)**.** *Let $p^k q^k r^k$ and $p^{k+1} q^{k+1} r^{k+1}$ be the triangles defining a temporal 3-cell $F_{i,j,l}$. Let t^k be the time-like coordinate of p^k, q^k and r^k, and t^{k+1} be the time-like coordinate of p^{k+1}, q^{k+1} and r^{k+1}. $\forall t \in [t^k, t^{k+1}]$, let $p^t = \frac{t-t^k}{t^{k+1}-t^k}(p^{k+1} - p^k) + p^k$, $q^t = \frac{t-t^k}{t^{k+1}-t^k}(q^{k+1} - q^k) + q^k$ and $r^t = \frac{t-t^k}{t^{k+1}-t^k}(r^{k+1} - r^k) + r^k$. Then,*

$$|F^k_{i,j,l}| = \left\| \int_{p^k}^{p^{k+1}} Area(p^t q^t r^t) dp^t \right\|. \tag{3.3}$$

Following [VL08], it can be noticed that the operator $\triangle_u$ is not symmetric but can be symmetrised. The inner product on 0-forms is defined by the diagonal matrix $\star_0$ with elements $\frac{|\star\sigma^0|}{|\sigma^0|}$, that is to say the volumes of the vertex dual cells since for any vertex σ^0, $|\sigma^0| = 1$. The following symmetric Laplace operator can thus be defined.

Definition 3.3.4 (Discrete Laplace operator on mesh sequences)**.** *Let MS be an embedded TCMS as defined in Def. 3.3.1. The operator $\triangle$ on 0-forms on M defined as:*

$$\triangle = \star_0^{1/2} \triangle_u \star_0^{-1/2} \tag{3.4}$$

is called the Laplace operator *on M.*

From Eq. (3.1) and Eq. (3.4) the following expression is derived.

Property 3.3.5. *Let F be a function defined on vertices $\{\sigma^0\}$ of an embedded TCMS MS. Then:*

$$\langle \triangle F, \sigma^0 \rangle = \sum_{\sigma^1 \succ \sigma^0} \frac{1}{\sqrt{|\star\sigma^0||\star v|}} \frac{|\star\sigma^1|}{|\sigma^1|} (F(\sigma^0) - F(v)). \tag{3.5}$$

3.3.2 Matrix representation

The Laplace operator defined in Def. 3.3.4 can be encoded as a $nf \times nf$ matrix, where n is the number of vertices in any mesh of the mesh sequence, and f is the number of frames. Although this matrix is big, it is also very sparse. Being a block tridiagonal matrix, it can also be inverted easily, as I show now.

Property 3.3.6. *Let M be a TCMS composed of f meshes, each having n vertices. The operator $\triangle$ on 0-forms on M as defined in Eq. (3.5) is local, and can be encoded by a sparse $nf \times nf$ symmetric block tridiagonal matrix L which can be written blockwise as:*

$$L = \begin{bmatrix} L^{(1)} & D^{(1)} & & & \\ D^{(1)} & L^{(2)} & D^{(2)} & & \\ & \ddots & \ddots & \ddots & \\ & & D^{(f-2)} & L^{(f-1)} & D^{(f-1)} \\ & & & D^{(f-1)} & L^{(f)} \end{bmatrix}, \tag{3.6}$$

where $\forall k \geq 1, k \leq f-1$, the $n \times n$ matrices $D^{(k)}$ are diagonal, and $\forall k \geq 1, k \leq f$, the $n \times n$ matrices $L^{(k)}$ being symmetric.

Proof. Operator $\triangle$ is obviously local since for any function F and any vertex σ^0, $\langle \triangle F, \sigma^0 \rangle$ only depends on F values on σ^0 and neighbouring vertices v. The matrix expression derives from Eq. (3.5). Diagonal coefficients of matrices $D^{(k)}$ are given by:

$$D^{(k)}_{i,i} = -\frac{1}{\sqrt{|\star v^k_i||\star v^{k+1}_i|}} \frac{|\star v^k_i v^{k+1}_i|}{|v^k_i v^{k+1}_i|},$$

where $\star v^k_i$ is the dual 3-cell of vertex v^k_i and $\star v^k_i v^{k+1}_i$ is the dual 2-cell of temporal edge $v^k_i v^{k+1}_i$. The coefficients $L^{(k)}_{i,j}$ of matrices $L^{(k)}$ are equal to zero if there is no spatial edge in M between vertices v^k_i and v^k_j. Otherwise,

$$L^{(k)}_{i,j} = -\frac{1}{\sqrt{|\star v^k_i||\star v^k_j|}} \frac{|\star v^k_i v^k_j|}{|v^k_i v^k_j|}.$$

Diagonal coefficients are given by:

$$L^{(k)}_{i,i} = -D^{(k)}_{i,i} - \sum_{j \neq i} L^{(k)}_{i,j}.$$

□

Definition 3.3.7 ([Meu92, Sal06])**.** *Let M be a TCMS composed of f meshes with n vertices each. Let L be its associated Laplacian matrix as defined in Prop. 3.3.6. Let $\{\Lambda^{(k)}, 1 \leq k \leq f\}$ be $n \times n$ matrices defined recursively as:*

- $\Lambda^{(1)} = L^{(1)}$;

- $\forall k \geq 2, \Lambda^{(k)} = L^{(k)} - D^{(k-1)}\Lambda^{(k-1)^{-1}}D^{(k-1)}$.

Let $\{\Sigma^{(k)}, 1 \leq k \leq f\}$ *be* $n \times n$ *matrices defined recursively as:*

- $\Sigma^{(f)} = L^{(f)}$;
- $\forall k \leq f-1, \Sigma^{(k)} = L^{(k)} - D^{(k)}\Sigma^{(k+1)^{-1}}D^{(k)}$.

Property 3.3.8 ([Meu92]). *Let* Λ *be the* $nf \times nf$ *block diagonal matrix* $\Lambda = diag(\Lambda^{(1)}, \ldots, \Lambda^{(f)})$ *and* Σ *be the* $nf \times nf$ *block diagonal matrix* $\Sigma = diag(\Sigma^{(1)}, \ldots, \Sigma^{(f)})$. *Let* L_o *be the block lower part of* L*:*

$$L_o = \begin{bmatrix} 0 & & & & \\ D^{(1)} & 0 & & & \\ & \ddots & \ddots & \ddots & \\ & & D^{(f-2)} & 0 & \\ & & & D^{(f-1)} & 0 \end{bmatrix}.$$

Then L *can be decomposed as:*

$$L = (\Lambda + L_o)\Lambda^{-1}(\Lambda + L_o^t) = (\Sigma + L_o^t)\Sigma^{-1}(\Sigma + L_o). \tag{3.7}$$

These two LDLT decompositions of L leads to a simple way to invert this matrix.

Property 3.3.9 ([Meu92]). *Let* $\{U_k, 1 \leq k \leq f\}$ *and* $\{V_k, 1 \leq k \leq f\}$ *be two sequences of* $n \times n$ *matrices such as:*

$$U_1 = I, V_1 = \Sigma^{(1)^{-1}},$$

where I *is the* $n \times n$ *identity matrix and* $\forall k \geq 2$*,*

$$U_k = (-1)^{k-1}D^{(k-1)^{-1}}\Lambda^{(k-1)} \ldots D^{(1)^{-1}}\Lambda^{(1)}, \tag{3.8}$$

$$V_k = (-1)^{k-1}\Sigma^{(1)^{-1}}D^{(1)}\Sigma^{(2)^{-1}} \ldots D^{(k-1)}\Sigma^{(k)^{-1}}. \tag{3.9}$$

Then $\forall j \geq i$*, the* (i, j) *block of* L^{-1} *can be expressed as* U_iV_j*.*

The inverse of L can thus be computed only by computing the inverse of f $n \times n$ matrices, namely the inverse of the $\Sigma^{(k)}$ matrices. Note that in [Meu92] it is requested that L is proper, that is to say that sub-matrices $D^{(k)}$ are non-singular. This is our case since these matrices are diagonal with non zero diagonal elements.

3.3.3 Behaviour for large and small time steps

I now investigate the behaviour of our Laplace operator when α tends to infinity or to zero. Remember from Section 3.3.1 that α is the parameter which scales the temporal dimension of the embedding space $\mathbb{E}$ with respect to the spatial dimensions. A large α decreases the influence of the temporal neighbours v_i^{k-1} and v_i^{k+1} of a given vertex v_i^k on this vertex with respect to its spatial neighbours sharing the same time-like coordinates. Conversely, a small α increases their influence.

Property 3.3.10. *When α tends to infinity, L tends to a block diagonal matrix $Diag(L^{(1)}, \ldots, L^{(f)})$, where each matrix $L^{(k)}$ is the spatial DEC Laplacian matrix with cotangent coordinates [DHLM05, CdGDS13].*

Proof. If $\alpha \gg 1$, the difference between successive areas A_i^{k-1}, A_i^k and A_i^{k+1} is negligible with respect to α. As a consequence, the volume of any vertex dual cell $\star v_i^k$ can be approximated by $\sqrt{\alpha}(t^{k+1} - t^k)A_i^k$, where A_i^k is the area of the spatial dual cell of v_i^k in the mesh M^k.
For the same reason, the area of the dual cell $\star v_i^k v_j^k$ is, for any spatial edge $v_i^k v_j^k$, equivalent to $\sqrt{\alpha}(t^{k+1} - t^k)$ times the length $|\star_{\mathbf{s}} v_i^k v_j^k|$ of the *spatial* dual cell of $v_i^k v_j^k$ in mesh M^k. We thus have $\frac{1}{\sqrt{|\star v_i^k||\star v_j^k|}} \frac{|\star v_i^k v_j^k|}{|v_i^k v_j^k|} \sim \frac{1}{\sqrt{A_i^k A_j^k}} \frac{|\star_{\mathbf{s}} v_i^k v_j^k|}{|v_i^k v_j^k|}$. Thus, the $n \times n$ matrix $L^{(k)}$ is equivalent to the spatial Laplacian matrix for mesh M^k.
The length of any temporal edge $v_i^k v_i^{k+1}$ of the mesh sequence is equivalent to $\sqrt{\alpha}(t^{k+1} - t^k)$, and the area of its dual is small with respect to α. Thus, any coefficient $D_{i,i}^{(k)}$ of the diagonal matrix $D^{(k)}$ is close to zero. □

Property 3.3.11. *Suppose the motion of each vertex is small with respect to the tessellation: $\forall k, \forall i, \forall j$ such that $v_i^k v_j^k \in E^k$, $\|v_i^k v_i^{k+1}\| \ll \|v_i^k v_j^k\|$. Then, when α tends to zero, the motion coefficients $D_{i,i}^{(k)}$ are dominating over the geometry coefficients $L_{i,j}^{(k)}$.*

Proof. If $\forall k, \forall i, \forall j$ such that $v_i^k v_j^k \in E^k$, $\|v_i^k v_i^{k+1}\| \ll \|v_i^k v_j^k\|$, then the difference between successive areas A_i^{k-1}, A_i^k and A_i^{k+1} is negligible with respect to $\|v_i^k v_j^k\|$. As a consequence, the volume of any vertex dual cell $\star v_i^k$ can be approximated by $\|v_i^k v_i^{k+1}\|_s A_i^k$, where A_i^k is the area of the spatial dual cell of v_i^k in the mesh M^k.
For the same reason, the area of the dual cell $\star v_i^k v_j^k$ is, for any spatial edge $v_i^k v_j^k$, equivalent to $\|v_i^k v_i^{k+1}\|_s$ times the length $|\star_{\mathbf{s}} v_i^k v_j^k|$ of the *spatial* dual cell of $v_i^k v_j^k$ in mesh M^k. We thus have $\frac{1}{\sqrt{|\star v_i^k||\star v_j^k|}} \frac{|\star v_i^k v_j^k|}{|v_i^k v_j^k|} \sim \frac{1}{\sqrt{A_i^k A_j^k}} \frac{|\star_{\mathbf{s}} v_i^k v_j^k|}{|v_i^k v_j^k|}$. Thus, the $n \times n$ matrix $L^{(k)}$ is equivalent to the spatial Laplacian matrix for mesh M^k.
The area of the dual of any temporal edge $v_i^k v_i^{k+1}$ of the mesh sequence is equivalent to A_i^k. As a consequence, the coefficient $D_{i,i}^{(k)}$ of the diagonal matrix $D^{(k)}$ is equivalent to $\frac{1}{\|v_i^k v_i^{k+1}\|_s^2}$. Since the motion is small with respect to the tessellation, $D_{i,i}^{(k)} \gg L_{i,j}^{(k)}$ for any spatial edge $v_i^k v_j^k$: the temporal coefficients are dominating. □

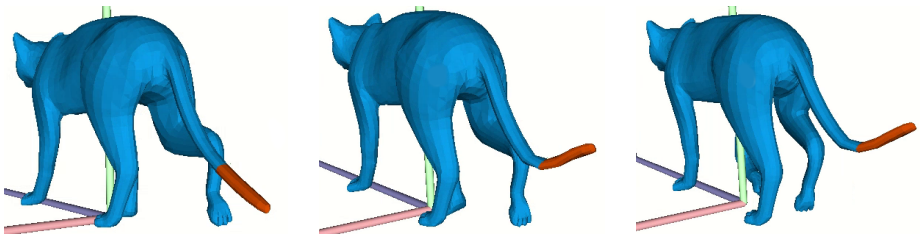
These properties prove that, if α is large, our 4D Laplacian acts as a standard static Laplacian on each frame, and if α is small and the motion is small with respect to the geometric discretisation, our 4D Laplacian enables to recover the motion of each vertex of the mesh independently.

3.3.4 Application to as-rigid-as-possible mesh sequence deformation

I believe the suggested discrete Laplacian operator can be used for various mesh sequence processing operations. In the near future, I plan to investigate several gener-

alisations of static Laplacian mesh processing works [Sor06, LZ09, ZvKD10]. I will detail them in Section 3.5.2. I provide here, as a first example, an extension of the well-known as-rigid-as-possible (ARAP) modelling framework [SA07] to mesh sequences. This work has been initiated in collaboration with four bachelor students from Grenoble INP - Ensimag, Mohammed Azougarh, Mohamed El Bakkali, Lucas Razafindrainijama and Redouane Oubenal, in May and June 2014. It has been revised and extended by two interns, Victoria Fernández Abrevaya and Sandeep Manandhar, during the summer of 2015.

In our application, the user selects a frame k and a vertex v in the mesh M^k of the mesh sequence, as well as an area in M^k around v and a time interval around frame k. Once the user has moved v to its targeted location, the algorithm computes the displacement of all vertices of the selected surrounding area, for all frames in the time interval, so that the overall displacement is as rigid as possible, both in space (between neighbouring vertices in the selected area) and in time (for the same vertex in successive frames). This is modelled as a matrix-vector system $Lp = b$, where L is the previously defined Laplacian matrix, b is a vector defined from the input vertex positions (see [SA07]) and p is the unknown vector of new vertex positions. For any vertex, the rigid transformation constraints are only applied to its spatial neighbours, as in [SA07], not to its temporal ones, for which the rotation matrix is taken as the identity. A result is shown in Figure 3.7.



(a) (b) (c)

Figure 3.7: *(a) Initial mesh M^k to which is applied the deformation. The area to be deformed is in orange. (b) Deformed mesh. (c) Deformation for the next mesh M^{k+1}.*

Note that several mesh sequence editing methods are available in the literature, some of them also using a discrete Laplace operator [XZY+07, TCH13]. The main difference is that they use a static mesh Laplace operator, instead of a general space-time one. Temporal constraints are formulated as a separate term in the energy functional to optimise. One benefit from our approach is thus its single, simple formulation. The parameter α allows tuning the influence of the temporal neighbours of a given vertex with respect to its spatial ones.

3.4 MESH SEQUENCE DECOMPOSITION INTO RIGIDLY MOVING COMPONENTS

In this section we focus on temporal incoherent mesh sequences (TIMS). Our goal is to segment the shape into parts that evolve approximately rigidly. By doing so, we should be able to infer the articulated structure of a human or animal body: arms, legs, torso, ... Possible applications include accurate motion measurement (for instance for validation or limb rehabilitation) and 3D action recognition. The only prior knowledge that we use is that the shape should be decomposable into almost rigidly moving components. As a consequence, this work does not apply to shapes such as rodents [Rev14] or humans wearing loose clothing.

This work forms the main part of the PhD thesis of Romain Arcila [Arc11].

3.4.1 Shape-in-motion segmentation classification

Before going into the details of the proposed algorithm for the segmentation of a TIMS, it is important to clarify what we mean by *shape-in-motion segmentation*, since several definitions can be thought of:

Definition 3.4.1 (Temporal segmentation)**.** *Let $MS = \{M^i, i = 1 \dots f\}$ be a mesh sequence. A* temporal segmentation *Σ_t of MS is a set of sub-sequences $\Sigma_t = \{MS_1, \dots, MS_k\}$ such that $\forall j \in [1, k], MS_j = \{M^{i_j}, \dots, M^{i_{j+1}-1}\}$ where $i_1 = 1 < i_2 < \dots < i_{k+1} = f + 1$.*

Definition 3.4.2 (Coherent segmentation)**.** *Let $MS = \{M^i, i = 1 \dots f\}$ be a mesh sequence. A* coherent segmentation *Σ_c of MS is a set of segmentations $\Sigma^i = \{M^i_1, \dots, M^i_{k_i}\}$ of each mesh M^i of MS, such that:*

- *the number k of sub-meshes is the same for all segmentations: $\forall i, j \in [1, f], k_i = k_j$;*
- *there is a one-to-one correspondence between sub-meshes of any two meshes;*
- *the connectivity of the segmentations, that is to say the neighbourhood relationships between sub-meshes, is preserved over the sequence.*

A coherent segmentation of a mesh sequence can be thought as a segmentation of some mesh of the sequence (for instance, the first one) which is mapped to the other meshes.

Definition 3.4.3 (Variable segmentation)**.** *Let $MS = \{M^i, i = 1 \dots f\}$ be a mesh sequence. A* variable segmentation *Σ_v of MS is a set of segmentations $\Sigma^i = \{M^i_1, \dots, M^i_{k_i}\}$ of each mesh M^i of MS which is not a coherent segmentation.*

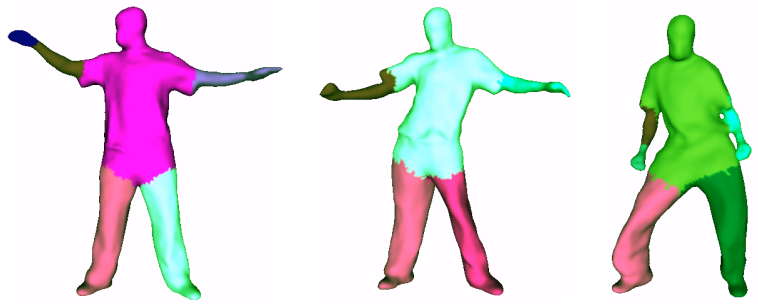


Figure 3.8: *Variable segmentation on the* Dancer *sequence [SH07]. First meshes of the sequence are decomposed into 6 segments, then the right arm and right hand segments merge since they move rigidly. Finally, this segment splits again.*

Possible applications of a temporal segmentation of a TIMS are mesh sequence decompositions into sub-sequences without topological changes [LB12] or motion-based mesh sequence decompositions [TM09]. Coherent segmentations are usually chosen for shape analysis and understanding, when the overall structure of the shape is preserved during the deformation. However, variable segmentations can be helpful in displaying different information at each time step. For instance, they can be used to detect when changes in motion occur. This is useful for animation compression or action recognition, for example. Figure 3.8 shows an example of a variable segmentation.

The work that we describe here is focused on both coherent and variable segmentations. It can be decomposed in two successive steps (see Fig. 3.9), mesh matching and vertex motion spectral clustering, that are iterated over the frames of the sequence. We now briefly describe these steps; see Appendix A.7 for more details.

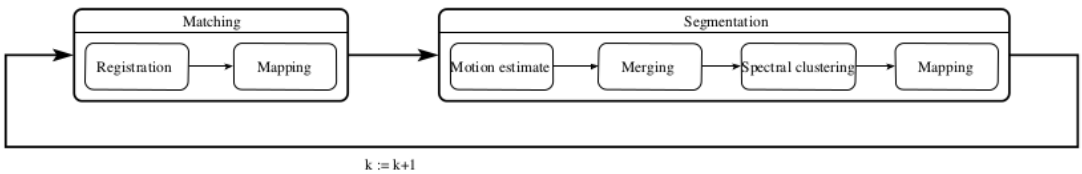


Figure 3.9: *Overall pipeline of our algorithm, at iteration $k, 1 \leq k < f$. As input we have meshes M^k, together with an initial segmentation estimate Σ^k_{est}, and M^{k+1}. As output we get a segmentation Σ^k of M^k and an initial segmentation estimate Σ^{k+1}_{est} of M^{k+1}.*

3.4.2 Mesh matching

The objective of this stage is, given meshes M^k and $M^{k+1}, k \in [1, f-1]$, to provide a mapping from vertices $v_i^{(k)}$ to vertices $v_j^{(k+1)}$, and a possibly different mapping from vertices $v_j^{(k+1)}$ to vertices $v_i^{(k)}$. This mapping is further used to propagate segment

labels over the sequence. We proceed iteratively according to the following successive steps (see Fig. 3.10): first, meshes M^k and M^{k+1} are registered (vertices $v_i^{(k)}$ are moved to new locations $v_i'^{(k)}$ close to M^{k+1}), then displacement vectors and vertex correspondences are estimated.

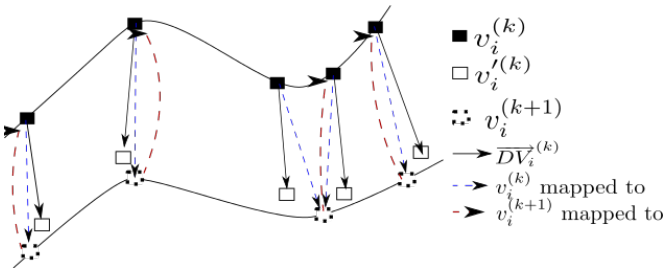


Figure 3.10: *Matching process. Mesh M^k with vertices $v_i^{(k)}$ is first registered to mesh M^{k+1} with vertices $v_i^{(k+1)}$, inducing new vertices $v_i'^{(k)}$. Displacement vectors $DV_i^{(k)}$ are defined by this registration. Finally, mappings from M^k to M^{k+1} and from M^{k+1} to M^k are computed.*

The registration uses the method of [CBI10], which divides the surface into small patches, each of them associated with a rigid frame that encodes for a local deformation with respect to M^k. Point correspondences are iteratively re-estimated using an optimisation procedure that minimises the distance between the two point sets while penalising non-rigid deformations of a patch with respect to its neighbours. The *displacement vector* of each vertex $v_i^{(k)}$ in $M^k, 0 \leqslant i < nv(M^k)$ is then defined as:

$$DV_i^{(k)} = v_i'^{(k)} - v_i^{(k)},$$

where $v_i'^{(k)}$ is the corresponding vertex in M'^k. To create a mapping from M^k to M^{k+1}, the closest vertex in M^{k+1} is found for each vertex $v_i'^{(k)}$ in M'^k using Euclidean distance. A mapping from M^{k+1} to M^k is also created by finding for each vertex in M^{k+1} the closest vertex in M'^k.

3.4.3 Motion-based spectral clustering

The mesh sequence is segmented according to the displacement vectors computed previously. We start with a single segment including all vertices of the first mesh M^1. For each frame k, the displacement vectors are first used to estimate the best rigid transformation matrix $T_i^{(k)}$ that maps each vertex $v_i^{(k)}$ of M^k and its neighbourhood onto M'^k. This is done using Horn's method [Hor87], which represents rotations with unit quaternions; the quaternion for the best rotation is the eigenvector corresponding to the greater eigenvalue of some 4×4 matrix. In practice, in order to be more robust to noise and to detect slow motion, we work in a time window. Then, in case a variable segmentation is required, neighbouring existing segments with similar motions

are merged. A spectral clustering approach is used afterwards to refine the segmentation. The clustering is made on the rigid transformations. On each segment, we build a graph whose nodes are the vertices $v_i^{(k)}$ and whose edges correspond to neighbouring vertices. This graph is weighted with the distances on the special Euclidean group of rigid transformations $SE(3)$ [MLS94]:

$$w_{i,j}^{(k)} = \frac{1}{\|\log(T_i^{(k)^{-1}} T_j^{(k)})\|^2}.$$

We then apply Shi and Malik's normalised spectral clustering algorithm [SM00] to segment the graph. This spectral clustering yields a segmentation of the vertices of M^k, which is mapped onto M^{k+1} to create the initial segmentation of the vertices of M^{k+1}. The final segmentation in the last frame is mapped back to each mesh M^k when a coherent segmentation is required.

3.4.4 Shape in motion segmentation evaluation

Little work has been done so far to evaluate properly existing segmentation methods. In the static case, a few authors have compared their results with manually segmented objects [BVLD09, CGF09, LTBZ13]. Different metrics have also been proposed; see [TPT15] for a survey. Part of our work has been to develop such an evaluation in the dynamic case. We rely on the fact that the *optimal* segmentation of a TCMS or a TIMS into rigid components can be guessed when the kinematic structure is known. This is because we can attach each vertex of a mesh sequence to one joint among its related joints (the joints of the animation skeleton for which the skinning weight is non-zero), namely, the furthest in the hierarchy from the root joint. The joints of the animation skeleton can be clustered into *joint sets*, each joint set representing a different motion. An *optimal segment* of the mesh sequence can then be defined as the set of vertices attached to joints in the same joint set.

We then propose the following three metrics in order to evaluate a given segmentation with respect to the previously defined optimal segmentation:

- *Assignment Error* (AE): for a given mesh, the proportion of vertices which are not assigned to the correct segment. This includes the case of segments which are not created, or which are wrongly created;
- *Global Assignment Error* (GAE): the mean AE among all meshes of the sequence;
- *Vertex Assignment Confidence* (VAC): for a given vertex of a TCMS, the proportion of meshes in which the vertex is assigned to the correct segment.

AE and GAE give a quantitative evaluation of mesh segmentation and mesh sequence segmentation, respectively, with respect to the optimal segmentation. VAC can help to locate wrongly segmented areas.

3.5 PERSPECTIVES

In this chapter I have presented some contributions to the analysis and segmentation of 3D mesh sequences. The overall common objective of all these methods is to contribute to a high-level understanding of shapes in motion, which includes the general recognition of the shape's geometry as well as its motion. In the near term, I plan to develop the three following points.

3.5.1 Visual differences between shapes in motion

Work on extending the perceptive analysis of mesh distortion (Section 2.4) to mesh sequences is planned as part of Georges Nader's PhD thesis. A better comprehension of visual differences between 3D shapes in motion has applications not only in compression and watermarking but also in classification and comparison of dynamic shapes. To the best of my knowledge, little work has been carried so far on this topic [KBAW11]. Defining a Just Noticeable Difference profile would be a first step. Finding a perceptually validated metric to compare shapes in motion is far more ambitious, and would probably need much more effort. I have spent some time investigating this subject between 2011 and 2014, as part of a joint work with Romain Arcila (at the end of his PhD thesis) and Ron Rensink from the University of British Columbia in Vancouver. We have devised an algorithm to segment temporally (see Definition 3.4.1 above) and compress a TCMS. This algorithm takes as input a desired number of key frames and automatically computes them. We have then investigated if the optimal number of key frames, defined as the minimum number for which there is no perceptual difference between the original and the compress sequence, can be automatically determined. This has been done by carrying perceptual experiments on three different sequences. Although the results were inconclusive this method can be used to validate a proposed metric in the future.

3.5.2 3D+t Laplace operator spectral behaviour

There are numerous applications of the discrete 3D+t Laplace operator. It has already been used recently, with other weight definitions, for motion editing and retargeting [NCG13] (using Gaussian weights) and space-time filtering [YXF14] (using a multi-scale combinatorial Laplacian). I plan to investigate the application of the 3D+t Laplace operator defined in Section 3.3 to various problems. Of particular interest are the Laplacian eigenvectors, as they are known in 3D to be related to the geometry of the shape [LÓ6, ZvKD10]. If the 3D+t Laplacian eigenvectors can also be interpreted as vibration modes of either the shape or the motion, by tuning the parameter α, a huge number of applications for understanding shape in motion exist. They range from shape or motion spectral clustering to shape or motion filtering, to the definition of shape or motion descriptors, to classification with respect to shape or motion. This subject will be investigated in 2016 with the help of Stefanie Wuhrer from the Morpheo team and a Master student, Sandeep Manandhar. In the longer term, I wish

to extent this 3D+t Laplace operator to TIMS and to check if a FEM-based definition matches the DEC-based definition, as happens in the case of the 3D cotangent Laplacian [LZ09].

3.5.3 Modelling the factors of variability for human shape in motion

The decomposition of a moving shape into rigidly moving components, as described in Section 3.4, is limited to shapes for which the kinematic structure can be captured. This approach cannot address such situations as the motion of animals with large fat volumes, such as rodents, or humans wearing loose clothing. The latter will be tackled within the ANR-funded project ACHMOV ("Accurate Human Modelling in Videos"), which has just started in October 2015. Together with Stefanie Wuhrer and a PhD student, Jinlong Yang, we plan to develop new statistical and geometric representations for modelling independently body shape, body motion and the motion of clothing. We will use 3D video acquisitions of humans wearing tight and loose clothes while performing the same movements.

CHAPTER 4

Understanding digital shapes from the life sciences

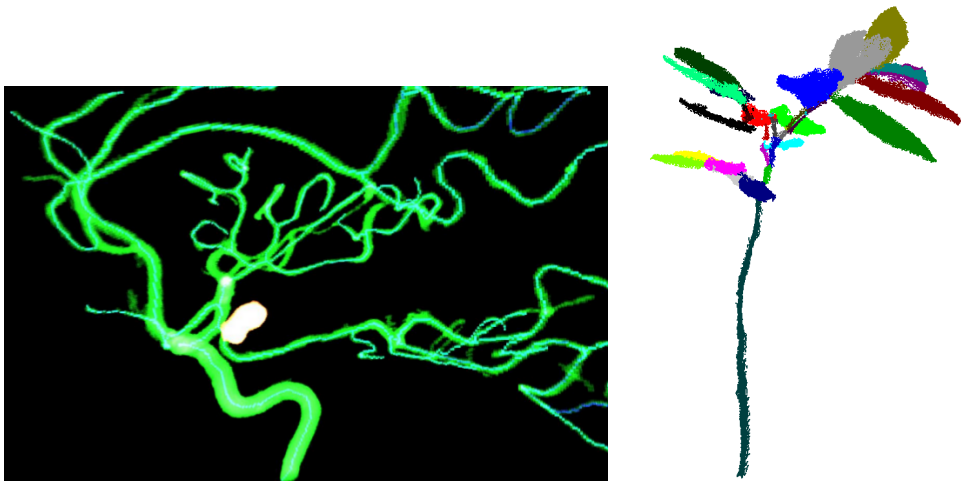
4.1 INTRODUCTION

4.1.1 Context

The previous chapters examined static meshes and temporal sequences of meshes. Although very common, meshes are not the only possible representation of discrete shapes. Depending on the digitisation system, other shape models are widely used for various applications. For instance, medical imaging systems create 3D images as stacks of 2D images. Organs are thus first represented as voxel sets, which are sometimes, but not always, subsequently converted to other shape representations. Another example is the use of laser scanners in remote sensing. These devices generate a cloud of 3D points representing the scanned scene. The conversion of this point cloud into a mesh is often not a trivial task because of non-uniform sampling, missing data due to occlusions, outliers or other types of noise caused by the acquisition process [BTS+14]. As a consequence, it is sometimes simpler to work directly on the point cloud.

In this chapter I describe two shape-understanding methods that directly operate on acquired data, without converting it to a mesh. Both of these methods are focused on particular applications and have been designed for and in cooperation with experts in the relevant fields. These experts, being the end users of the methods, have brought specific requirements, as did the infographist Christine Depraz with whom I worked on the animation skeleton method (Section 3.2). The first application, described in Section 4.2, was initiated by Dr. Olivier Palombi, a neurosurgeon (and also a researcher in computer science and former colleague in the EVASION team). The goal was to locate, to characterise the shape and to quantify the volume of brain aneurysms automatically from voxel sets of the cerebral vascular tree. These voxel sets were segmented from 3D MRI or scan images (see Fig. 4.1 (a)). Such information is crucial in helping neurosur-

geons and neuroradiologists decide on an appropriate treatment (clipping or coiling of the aneurysm). The second application is detailed in Section 4.3. It deals with the accurate segmentation of tree laser scans into their elementary units (leaves, petioles and branches) and its application to dendrometry (tree shape measurements). This problem was introduced to me by Dr. Eric Casella, an ecophysiologist working for the Forestry Commission (UK). A typical result is shown in Fig. 4.1 (b).



(a) (b)

Figure 4.1: *(a) Voxel set representing the cerebral vascular tree, with an aneurysm highlighted in yellow. The computed centreline is in blue. (b) Segmented point cloud of a sweet chestnut seedling.*

4.1.2 General approach: skeleton for segmentation and measurement

Although the representations of the shapes and the application areas are different, each of the problems described in this chapter, namely aneurysm quantification and tree segmentation, can be decomposed in the same way. First, the shape (the cerebral vascular tree for the medical application, a leaf-on tree for the botanical one) must be decomposed into elementary units: the tree's branches and the potential aneurysms in one case, the branches, petioles and leaves in the other. Second, the geometric measurements must be computed: volume and maximum horizontal and vertical diameters of the aneurysms; tree leaf areas and possibly wood volume (although this has not been done in the work described here, this is possible by fitting geometric primitives, see [rRKC15]). This second stage is relatively easy when intended sub-shapes have been accurately isolated. Hence, the main difficulty lies in the segmentation stage. We have chosen the same approach in both cases: a shape set of centrelines is computed using Dijkstra's algorithm (in the tree case, the point cloud is first embedded into a spectral space, leading to a shape made of elongated regions), then voxels or points associated with each centreline are clustered into a segment. In other words, the segmentation is

guided by the skeleton.

The prior knowledge used in the first case is mostly related to the localisation of the aneurysms. The anatomy of the cerebral vascular tree has been encoded into a multiresolution graph which enables the branches with aneurysms to be located. In the second case, the suggested approach for tree segmentation is semi-automatic. That is, the expert introduces knowledge when and where he or she chooses to re-run the segmentation process after inspecting preliminary results through the visualisation interface.

4.2 Cerebral aneurysm characterisation and quantification

Saccular aneurysms are dilatations in the wall of a blood vessel, that are connected to the vessel by a narrowed zone called the *neck*. If not treated, an aneurysm may burst causing a stroke and in most cases the death of the patient. The decision to treat an aneurysm is made according to its risk of rupture. Surveys have shown that the most important factors affecting the risk of rupture are: size, shape, neck, and location of the aneurysm. In this work we devise a tool that detects aneurysms and computes all of these factors from a voxel set representing the cerebral vascular tree. The voxel set is segmented from Magnetic Resonance Angiography (MRA) or Computerized Tomography (CT) scan images. Image segmentation is a difficult problem, but it is not the purpose of our work. We rely on existing techniques, see [LABFL09] for a review.

Our method first computes thin, connected and exactly centred centrelines in the voxel set. These centrelines are then used to study the evolution of the diameters and to detect aneurysms automatically. Blood vessels have a cylindrical shape and thus their diameters so are almost constant, whereas those of aneurysms change considerably. Relevant measures are then computed for the aneurysms that have been found. Their locations are identified by using a partial graph matching technique. Interestingly, another study (which takes as input a meshed surface of the cerebral vascular tree rather than a voxel set) has recently used the same ideas and gone further [BWDJ14] by automatically classifying aneurysms into one of four types by using graph analysis and supervised learning techniques.

This work was the first part of the PhD thesis of Sahar Hassan [Has11].

4.2.1 Centreline extraction

Extracting centrelines from a voxel set is a well-known problem [CSM07]. Since we want to use the centrelines to study the evolution of blood vessel diameters, these centrelines should be:

- *connected*: the centrelines we are looking for should be 26-connected;
- *thin*: a centreline is thin if each voxel of the centreline has only two of its neighbours in the centreline, except for the extremities which have one neighbour in the centreline;
- *centred*: the centrelines should be centred within the vascular tree;
- such that the *connections between branches* are as *orthogonal* as possible.

To get such centrelines, our approach is to adapt Dijkstra's algorithm [Dij59]. Figure 4.2 (a) shows the flowchart of our centreline algorithm. A source voxel S is chosen at an extremity of the set by identifying the furthest voxel from a random voxel [LV99]. A modified Dijkstra's algorithm finds the shortest path from S to the furthest voxel. This path is the first centreline B_0. All the voxels belonging to B_0 have then their distance from the source voxel (DFS) put to 0, and the modified Dijkstra's algorithm is re-run from S. This gives a new centreline B_1, which connects the furthest voxel to B_0. The process is iterated until the new centreline B_i is too short, and at most 50 times, since a cerebral vascular tree usually has fewer than 50 visible branches.

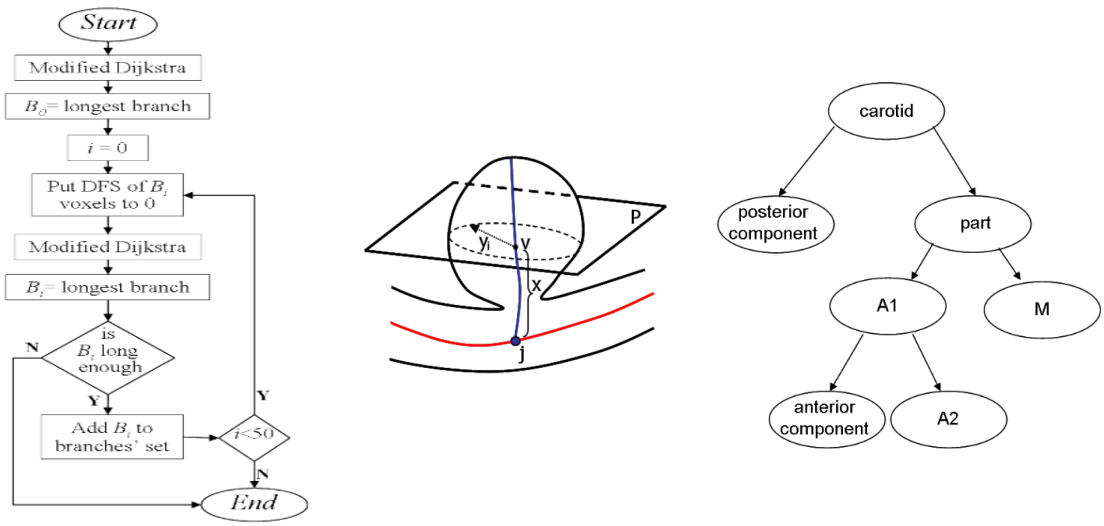
Dijkstra's algorithm is modified by using the following distance instead of the Euclidean distance:

$$d(v_1, v_2) = \frac{d_{eucl}(v_1, v_2)}{1 + DFB(v_1) + DFB(v_2)},$$

where $d_{eucl}(v_1, v_2)$ is the Euclidean distance between the centres of voxels v_1 and v_2, and $DFB(v_i)$ is the Euclidean distance between the centre of voxel v_i and the centre of the closest surface voxel, that is to say the closest voxel with at least one of its 26-neighbours missing in the voxel set. Using d rather than d_{eucl} privileges the voxels that are far from the boundary of the voxel set, and so enforces the centredness of the centreline. Setting the DFS of voxels of extracted centrelines to zero makes the new centreline join orthogonally the set of previous centrelines and avoids cutting corners. Since the centrelines are connected and thin by construction, the four properties listed above hold.

4.2.2 Aneurysm detection and quantification

One key characteristic that differentiates a saccular aneurysm from a normal vessel is that the normal vessel has an almost constant diameter, whereas the aneurysm, which has an irregular shape, has a diameter that changes considerably. In order to model the appearance of a vessel, we define a set of points (x, y) corresponding to the centreline's voxels v. x is the distance between the centre of the voxel and the centre of the origin of the centreline. y represents the approximate diameter of the vessel at v, and is calculated using the real plane P passing through the centre of v and orthogonal to the centreline, see Fig. 4.2 (b). P cuts the vessel or aneurysm surface on voxels v_i, at a distance y_i from v. y is simply defined as the average value of the y_i. y is a reliable



(a) (b) (c)

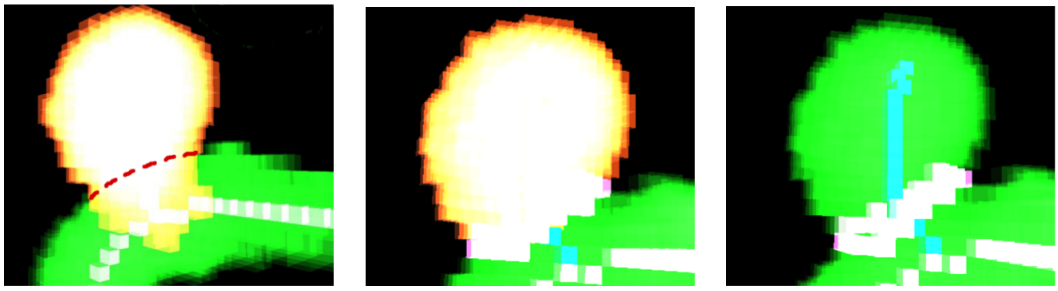
Figure 4.2: *(a) Flowchart of the centreline extraction algorithm. (b) Diameter computation for a centreline. (c) Reference graph used for aneurysm localisation.*

indicator of changes in diameter because of the centredness of centrelines and their orthogonality. A least-square quadratic fit is then applied to the set of points (x, y) of a given centreline. Thresholding on the parameter associated with x^2 enables us to detect aneurysms.

An aneurysm is defined as the set of voxels connected to the associated centreline through the last iteration of the modified Dijkstra's algorithm. However, this includes irrelevant voxels, which are geometrically located between the support vessel and the aneurysm neck, see Fig. 4.3 (a). These voxels are closer to the centreline of the support vessel than its half-diameter and so they are removed, as shown in Fig. 4.3 (b). The neck of the aneurysm is finally computed as the surface voxels of the aneurysm that have at least one neighbour that is not in the aneurysm (Fig. 4.3 (c)).

Once an aneurysm and its neck are computed, several measurements are provided to the user:

- size of the aneurysm: the number of voxels in the aneurysm;
- shape: we compute the maximum vertical and horizontal diameters of the aneurysm (see Appendix A.8 for details);
- neck: the area of the neck (number of voxels) and its perimeter (number of surface voxels).



(a) (b) (c)

Figure 4.3: *(a) Set of voxels associated with an aneurysm centreline. (b) Voxels of the aneurysm. (c) Neck of the aneurysm. The aneurysm centreline is in blue.*

4.2.3 Aneurysm localisation

A reference graph of the vessels of the cerebral vascular tree automatically localises the aneurysm (Fig. 4.2 (c)). This graph is simplified so that multiple graphs can be used, because in practice not all vessels are actually segmented from acquired images.

We use a partial matching method [CFSV04] to match this reference graph with a graph constructed from the set of extracted centrelines. The nodes of the graph correspond to segments of the centrelines, that is to say connected sets of voxels between an end voxel and a junction with another centreline, or between two junctions. The widest segment (aneurysm excluded) is chosen as root, since it corresponds to the carotid. Edges of the graph correspond to centreline junctions. Three attributes are associated with each node of the graph: the length and diameter of the corresponding segment and the number of child nodes. The first two attributes indicate the importance of the segment. Segments with small diameters or short lengths are considered very patient-specific and unimportant and so the corresponding nodes are deleted from the graph. Each time a node is deleted the number of children of its parent is increased by an increment of one. Only this third attribute is then used in the matching step. It helps to differentiate between vessels that are known to have many bifurcations (e.g. vessel M) and those with fewer bifurcations (e.g. vessel A) issuing from the same parent.

4.3 Tree seedling segmentation and measurement

Terrestrial laser scanning (TLS) has become an increasingly popular technique to measure the 3D characteristics of vegetation, from grass to forest plant species [DCF11], since it is fast and non-destructive. Not only are these measurements critical for forest inventory and management but also for carbon-storage estimation, climate-change studies, and so on. They also provide an in situ validation of 3D plant architecture models. These models describe plants as collections of interconnected *elementary units* (inter-nodes, petioles, leaf-blades) which are spatially distributed above and/or

below ground [GCS99].

TLS uses a LiDAR device which generates unstructured clouds of 3D points where the laser beam is incident and reflected. Although it gives a raw sketch of the spatial distribution of plant elements in 3D, it lacks explicit and essential information on their shape or connectivity (which component is related to which). These points need to be subsequently clustered into geometrically meaningful sets for further analysis and dendrometric measurements. The problem of point-cloud segmentation has already been tackled for leaf-off trees; see e.g. [RKr+13]. The problem is more complex for leaf-on trees, not only because of larger occlusions but also because there is substantially more noise. This is the problem we tackle here. We propose an interactive technique that yields very accurate results, as opposed to computer graphics techniques which give visually *plausible* but not *faithful* results [XGC07, LYO+10]. The point cloud is first segmented into a small number of clusters, then the expert selects the clusters that need to be segmented again, and the process is iterated.

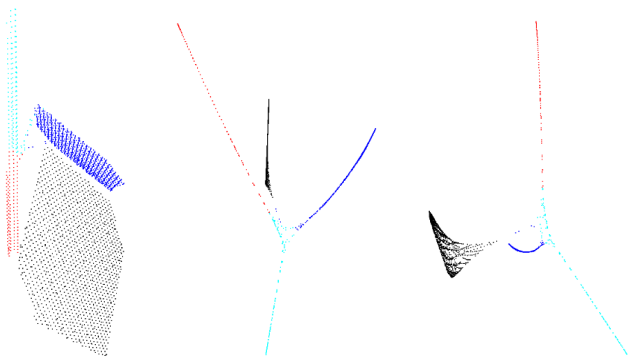
Being interactive, the approach is suitable for plants and trees with a limited number of elementary units, such as the horse chestnut seedling of Fig. 4.4 which is segmented into 38 clusters (for 122022 points in the point cloud). This approach is robust to acquisition noise and occlusions, as shown in the figure. Note that the problem would have been simpler in a controlled environment and for such small plants, since multi-camera systems would have allowed a mesh of the plant's surface to be constructed with only a limited amount of noise. This is the process that has been successfully used for phenotyping by [PSB+12].

(a) (b) (c)

Figure 4.4: *Segmentation of a horse chestnut seedling. (a) Side view. (b) Top view. (c) Close up.*

The main idea underlying our approach is that a tree or a plant is a strongly anisotropic shape and that its natural "intrinsic" directions follow the directions of each stem, branch, petiole and the main directions of each leaf-blade. Thus, our method

transforms the input data into a set of elongated curves, or at least strongly anisotropic shapes, that sample each elementary unit of the plant, as is shown in Fig. 4.5. The segmentation is easier on such transformed data since it does not depend upon the particular shape of the leaves. Moreover, geometrical noise accumulated during the acquisition process is implicitly altered by the transformation. Technically, we use a *dimension-reduction* approach, namely a *spectral embedding* of a graph connecting the TLS points.



(a) (b) (c)

Figure 4.5: *(b,c) Two views of the spectral embedding of the point cloud shown in (a).*

4.3.1 Graph computation

In order to segment the TLS point cloud, our approach first computes a graph G that connects each point to some of its neighbours. Usual methods such as the ε-neighbourhood (which connects a point to all other points within a ball of radius ε) or the k-nearest neighbours (which connects a point to the k closest points) methods are convenient as long as the density of the point cloud is uniform; but this is not the case for our TLS data. We have found that useful connections may be missed if the value of the parameter (ε or k) is too small, and that too many edges may be created otherwise. Moreover, the optimal parameter value depends on the data and is difficult to select in advance.

Therefore we suggest a simple yet more robust method. For a given point p, we compute the k nearest neighbours (with $k = 0.1\%$ of the total number of points), and place them in a priority queue Q ordered by their distance to p. We keep the closest point p_1 to p and for each successive point p_i in Q we check if the angle between the edge pp_i and any previously kept edge pp_j is lower than a threshold a. If so, p_i is discarded. Otherwise p_i is kept as a neighbour of p (i.e. pp_i as an edge of G). The algorithm is summarised in Appendix A.9.

According to our experiments, the segmentation results are insensitive to the value of the parameter a. As a consequence, we choose $a = \pi/2$, since it minimises the

number of edges in G and thus the computation time for the subsequent stages of the algorithm. Note that $a > \pi/2$ may lead to a graph with multiple connected components.

4.3.2 Spectral embedding

Embedding a discrete shape into a low-dimensional spectral space is known to help to capture its intrinsic features (see e.g. [RWP06]). In our work, we build on the *Laplacian Eigenmaps* framework of Belkin and Niyogi [BN03], except that the graph edges are weighted by the commute-time distance, which has proven to be robust against noise for clustering purposes [QH07]. Using this distance is similar to using the Euclidean distance in spectral space, except that each coordinate is divided by the corresponding eigenvalue: the commute-time distance between two nodes i and j of the graph is given by $\sqrt{\sum_k \frac{(i(k)-j(k))^2}{e(k)}}$, where $i(k)$ and $j(k)$ are the kth coordinates in spectral space of i and j, respectively, and $e(k)$ the kth eigenvalue of the Laplacian matrix of the graph.

The *spectral embedding* of G into a d-dimensional space is given by the d eigenvectors $V_1, \ldots, V_d$ of the Laplacian matrix $L = W - A$ of G associated with the first d non-zero eigenvalues, in increasing order (W is the diagonal valency matrix of G and A is its adjacency matrix). The embedding coordinates of node i of G are given by row i of the matrix whose columns are vectors $V_1, \ldots, V_d$; see [vL07] for a detailed explanation of spectral embedding and clustering. It is known that the eigenvectors associated with the lowest non-zero eigenvalues of L give the main "intrinsic" (curved) directions of the graph [LÓ6]. We have shown (see Appendix A.9) that the segmentation is insensitive to parameter d. In practice, we use $d = 5$ or 10.

4.3.3 Segmentation in spectral space

Once the graph G is embedded into the spectral space, it is segmented into its elongated curves. Methods such as K-means are tailored to isotropic data and would not give satisfactory results with such a strongly anisotropic shape. This is why we use a method similar to that described in Section 4.2.1. Dijkstra's shortest path algorithm is applied n times and this produces $2n - 1$ centrelines within the embedded graph because each new centreline splits an existing centreline in two. The parameter $c = 2n - 1$ is chosen by the user and is equal to the final number of clusters. After the final shortest-path computation, each node of G is assigned to its nearest centreline, namely the first centreline met in its predecessor list.

Our experiments have shown that choosing a large number c of clusters may lead to over-segmentation of leaves. To overcome this problem, we suggest choosing a small value for c initially. According to our experiments, $c \sim 25\%$ of the total final num-

ber of clusters is generally a good initial value. Once the TLS data is segmented, the user can choose clusters through a graphical interface and re-run the whole algorithm. Leaf areas are then computed by projecting the points of a leaf into its least-square fitted plane, computing the Delaunay triangulation of the projected points, projecting the points back to their original positions and summing the areas of the Delaunay triangles.

According to our experiments, this method leads to very accurate segmentations (less than 3.4% false positive and negative for the leaf clusters) and leaf area estimates (less than 1.3% error). Two results are shown in Fig. 4.1 (b) and Fig. 4.4.

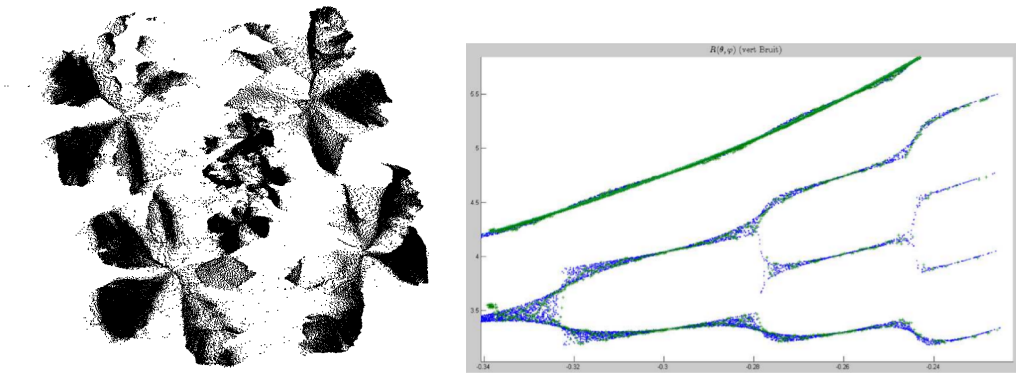
4.4 PERSPECTIVES

Two studies of digital shape understanding for life sciences have been presented in this chapter. Although the shapes are represented by different discrete input data (a voxel set in the first case, a 3D point cloud in the second), they share a common approach based on centreline extraction for shape segmentation.

The first study focused on aneurysm detection, localisation and quantification in cerebral vascular trees. The second targeted tree decomposition into its elementary units (branches, petioles and leaves). Currently, I am not actively working on medical applications. However, I am still working and I plan to continue on applications in forestry and botany. Geometrically speaking, trees are of great interest because of their complexity. Accurately perceiving their shape from a cloud of points is a challenge not only because of the huge number of branches and leaves, but also because both large (the trunk and main branches) and small (petioles, small branches) areas are involved. Moreover, thin, elongated shapes (branches) are mixed with more planar ones (leaves). Short-term perspectives include two projects in collaboration with Dr. Eric Casella from the Forestry Commission (UK), which have recently been funded by the Université Grenoble Alpes.

4.4.1 TLS point cloud noise detection and correction

Any TLS point-cloud processing algorithm on trees is inherently limited by the amount of noise in such geometries. For instance, there will be many outliers between neighbouring leaves and around branches. As an example, Fig. 4.6 (a) shows the initial horse chestnut scan (to be compared with Fig. 4.4 (b)). In our work (Section 4.3), we filter point clouds by removing outliers. We define an outlier as a point whose mean distance to its k nearest neighbours exceeds the mean plus the standard deviation of the mean distances to all points [RC11]. However, removing points alters the information. In particular, estimates of leaf areas will be lower than actual leaf areas and so our idea is to *correct* the outlier positions rather than to suppress the data. We are currently doing this with Romain Rombourg, who studied this problem for his Master thesis in 2015.



(a) (b)

Figure 4.6: *(a) Raw scan of the horse chestnut (top view). (b) Modelling the acquisition noise. In green is displayed the input scan as a function of distance to the LiDAR; in blue is our simulation, taking into account our mixed pixel mathematical model.*

Several physical phenomena have been claimed to be responsible for outliers in the literature [HK92], of which the so-called *mixed pixel effect* can explain the majority of outliers in our tree TLS point clouds (see Fig. 4.6 (b)). The effect occurs when a laser beam partially hits the edge of one surface and partially anotherbehind it. We have developed a mathematical model of this effect and methods to detect and correct for outliers.

4.4.2 Towards an accurate digital tree model

Noise correction opens up new perspectives for our interactive segmentation approach (Section 4.3). Our ambition is to make it fully automated, in order to be able to segment a full leaf-on tree accurately and not only seedlings or plants. This is not currently possible because of the amount of noise still present in the data, even after filtering. The boundary between elementary units (e.g. two spatially close leaves) is not always obvious and the graph we create may contain several edges joining elementary units that are not connected in the real tree.

I have recently been granted a PhD studentship by the Université Grenoble Alpes to investigate further the use of LiDAR data for digital tree modelling. The goal of this project is to develop a full pipeline from TLS point-cloud acquisition to single leaf and branch geometric modelling and measurement, through to scan de-noising and automated segmentation into elementary units. The student, Romain Rombourg, has started his PhD thesis in October 2015.

CHAPTER 5

Conclusion

5.1 SUMMARY

In this manuscript I have reviewed my main contributions to digital shape understanding. I have developed several segmentation and skeleton-based methods to recognise shapes from low-level digital representations, that have no overall coherence. My research has encompassed three different areas. First, I have focused on general purpose 3D meshes. I have contributed to their topological and perceptual comprehension, as well as to the transformation of a unorganised set of triangles into a manifold mesh, and then to a regularly sampled volumetric model. Second, I have introduced methods that help to create, modify and analyse sequences of meshes as digital representations of moving shapes. Third, in the context of shape analysis and measurement for medicine and botany, I have worked on skeleton-based shape segmentation from two different digital representations, namely, voxel sets and point clouds.

In each of these studies, I have developed tools that are either interactive (or experiential) or automatic as the particular context and user need has required. Similarly, some of my work has had specific applications in mind while other work has aimed to be more generally applicable. This is summarised in Table 5.1. The mesh sequence decomposition algorithm is in the specific application category since it focuses on rigid motion retrieval. A possible application is motion measurement for limb rehabilitation or sport gesture analysis. The harmonic skeleton tool is classified as interactive since the skinning weights are chosen by tuning a parameter (Section 3.2.5). As a future research project, it could be automated by relating the skeleton to anatomy semantics.

	Automatic methods	Interactive tools
Specific application	Mesh sequence decomposition Aneurysm characterisation	Harmonic skeleton Tree seedling segmentation
General purpose	Homology retrieval Volumetric discretisation Laplace operator	Mesh repair JND profile

Figure 5.1: *Classification of the studies described in this manuscript.*

5.2 General comments

From my experience, at least three points are essential in digital shape understanding for real life applications. First, it is necessary to collaborate with experts from the application domain. These experts not only provide details on the desired output of the algorithm, but also can give insights on the prior knowledge to be used. Second, choosing the right mathematical and computational tools requires a broad view, rather than an expert understanding, of available theories and methods. Therefore, it is important to be open-minded to techniques used in other domains. Finally, evaluation, and in particular quantitative validation, is not to be neglected. This is the way to convince most users that the method is worth being used. Nice images are not sufficient.

Doing so is time consuming. Understanding people from another domain, especially their exact needs, is time consuming. Knowing which tools exist is time consuming. Acquiring data and devising validation experiments is *very* time consuming. Tools used are not always the same, adding another challenge. However, it is rewarding to feel useful and it is a nice way to learn a lot.

5.3 Open questions for shapes in motion and forest science

In the future I plan to continue working on shape understanding for two main applications, namely 3D shapes in motion and forest science, although I am open to new collaborations. The first topic is the main concern of the Morpheo team and so I benefit from the expertise of my colleagues. Trees are of great interest because of their challenging complex structure. I am currently collaborating on this topic not only with Dr. Eric Casella, but also more informally with other experts.

For each of these applications the same questions arise, which are connected to the key points I mentioned above:

- What is the most appropriate shape representation?
- Which mathematical and computational tools may be useful to process the shape?

- Which additional knowledge should be used, and how should it be integrated?

I will consider each of these questions in turn.

5.3.1 Appropriate shape representations

In computer vision, 3D moving shapes are currently represented by temporally incoherent mesh sequences (TIMS, see Definition 3.1.1). We have seen in Section 3.4 that it is not easy to recognise the shape and its motion from such a representation. Therefore it may be relevant to try to find a more appropriate geometric representation. We have recently started focusing on regular volumetric discretisation, through the PhD project of Li Wang (Section 2.5). A recent publication using our approach has shown that there is interest in such volumetric discretisation for shape tracking [AFB15]. This topic will also be investigated within the context of the ANR ACHMOV project (see Section 3.5.3), for clothed human body shapes. In the case of clothes, a volumetric model may not be appropriate since clothes are largely two-dimensional.

In forestry, the need for accurate measurements also requires appropriate shape representations. A recent study has shown that cylinders are a simple yet efficient geometric primitive to represent tree branches [rRKC15]. A similar study is planned for leaves in the context of the AGIR DigiTree project (see Section 4.4.2). However it may not be possible to avoid having many parameters since the geometric variability of leaves is much greater than that of branches. To overcome this problem, a collaboration with Florence Bertails-Descoubes from the BIPOP team of Inria Grenoble is envisaged. Florence's project "From Geometry to Motion: inverse modelling of complex mechanical structures", recently granted by the ERC, aims at estimating mechanical parameters from the geometric shape, for thin elastic rods, plates and shells. Tree leaves would be physically modelled as shells.

5.3.2 Mathematical and computational tools

The question of appropriate mathematical and computational tools is closely related to the issue of representation. In the case of shapes in motion, I believe that the 3D+t Laplace operator, developed on TCMS (see Section 3.3), may be as useful for shape *and* motion processing as discrete 3D Laplace operators are for static mesh processing [LZ09, ZvKD10]. The main benefit of such an operator lies in its ability to process both shapes and motions, by simply varying a parameter. Its computational cost is its main drawback, since it requires a matrix whose size is the square of the total number of vertices in the sequence, although this matrix is very sparse. Methods for sparse matrix manipulations need to be investigated further for this operator to be practically usable. A discrete Laplace operator on TIMS or volumetric space-time representations may be derived later, if such representations prove to be useful for the understanding of shapes in motion.

In the case of tree laser scans, two main options can be envisaged. If the input point cloud can be efficiently de-noised (see Section 4.4.1), it may be converted to a two-manifold mesh. Then, standard mesh processing techniques may be used as was recently done for phenotyping [PSB+12]. However, I believe that, due to the particular geometry of a tree, with many close yet non-intersecting branches and leaves, in most cases there will be too many occlusions and topological ambiguities for a correct mesh reconstruction. Therefore instead, I plan to investigate point cloud processing tools. Since LiDAR have been used for a long time in many fields such as robotics, archaeology or building modelling, such tools are already well developed, see e.g. [RC11, NWH13].

In addition to these deterministic tools, statistical methods will be required to deal with inherent uncertainty in the data. The development of statistical methods is planned for the PhD project that I will supervise with Stefanie Wuhrer on modelling clothed humans in motion (see Section 3.5.3). In the case of scanned leaf-on trees, this is something already quite popular in the literature: because of the large number of occlusions between leaves, an exact reconstruction of each single leaf is impractical.

5.3.3 Incorporating additional knowledge

Prior knowledge is crucial for the efficiency of shape understanding tools. As we have seen in this manuscript, depending upon the application, an expert can provide knowledge about the shape. However, this is not the only possibility, and I think at least two directions need to be explored.

First, this knowledge may be implicitly learnt from examples or from the environment, as has been suggested recently by Hu et al. [HZvK+15]. This is an area of considerable interest nowadays, as is exemplified in the recent work on 3D shape recognition from depth sensors using deep learning techniques by Wu et al. [WSK+15]. Another way to incorporate knowledge without formalising it is through a set of user experiments, as we did with George Nader for 3D mesh visual perception (Section 2.4). One of the main questions in all these cases is to what extent does the example/user database or the environment fully represents the set of possible situations.

Second, new sensors may be used to provide additional information about the shape. For shapes in motion, optical systems and depth cameras are widely used to complement video cameras. Within the Morpheo team, our acquisition platform Kinovis benefits from a 20 Vicon motion capture system combined with 68 video cameras. Cameras can also be thought of as complementing laser scanners in the forestry case. Unfortunately, their resolution is currently too low to help with accurate measurement [CDMM13]. Handheld laser scanners are currently being investigated but again their accuracy remains insufficient at the moment [BBP+14]. Hyper-spectral imaging is a promising field, and several teams have recently started to investigate this technology for remote sensing; see e.g. [HSKC12]. The main challenges when various sensors are

combined are calibration and data fusion. The discrete representation of shapes may also need to be adapted.

APPENDIX A

Selected papers

- Mesh repair with user-friendly topology control, *F. Hétroy, S. Rey, C. Andújar, P. Brunet, À. Vinacua*, Computer-Aided Design 43 (1), Elsevier, 2011.

- An iterative algorithm for homology computation on simplicial shapes, *D. Boltcheva, D. Canino, S. Merino Aceituno, J.C. Léon, L. De Floriani, F. Hétroy*, Computer-Aided Design 43 (11), Elsevier, 2011. Presented at the SIAM Conference on Geometric & Physical Modeling (GD/SPM) 2011.

- Just Noticeable Distortion profile for flat-shaded 3D mesh surfaces, *G. Nader, K. Wang, F. Hétroy-Wheeler, F. Dupont*, IEEE Transactions on Visualization and Computer Graphics, 2016.

- A hierarchical approach for regular centroidal Voronoi tessellations, *L. Wang, F. Hétroy-Wheeler, E. Boyer*, Computer Graphics Forum 35 (1), Wiley, 2016.

- Harmonic skeleton for realistic character animation, *G. Aujay, F. Hétroy, F. Lazarus, C. Depraz*, ACM-SIGGRAPH/Eurographics Symposium on Computer Animation, 2007.

- Simple flexible skinning based on manifold modeling, *F. Hétroy, C. Gérot, L. Lu, B. Thibert*, International Conference on Computer Graphics Theory and Applications (GRAPP), 2009.

- Segmentation of temporal mesh sequences into rigidly moving components, *R. Arcila, C. Cagniart, F. Hétroy, E. Boyer, F. Dupont*, Graphical Models 75 (1), Elsevier, 2013.

- Automatic localization and quantification of intracranial aneurysms, *S. Hassan, F. Hétroy, F. Faure, O. Palombi*, Lecture Notes in Computer Science 6854, Springer, 2011. Presented at the International Conference on Computer Analysis of Images and Patterns (CAIP), 2011.

- Segmentation of tree seedling point clouds into elementary units, *F. Hétroy-Wheeler, E. Casella, D. Boltcheva*, International Journal of Remote Sensing, Taylor & Francis, 2016.

$\therefore$

A.1 MESH REPAIR WITH USER-FRIENDLY TOPOLOGY CONTROL

Franck Hétroy, Stéphanie Rey, Carlos Andújar, Pere Brunet, Àlvar Vinacua
Computer-Aided Design 43 (1), Elsevier, 2011.

Mesh repair with user-friendly topology control

Franck Hétroy [a,*,1,2]

[a] *INRIA, 655 avenue de l'Europe, F-38334 Saint Ismier, France*
Phone: +33 476 615 504 – Fax: + 33 476 615 466

Stéphanie Rey [1] Carlos Andújar [3] Pere Brunet [3] Àlvar Vinacua [3]

Abstract

Limitations of current 3D acquisition technology often lead to polygonal meshes exhibiting a number of geometrical and topological defects which prevent them from widespread use. In this paper we present a new method for model repair which takes as input an arbitrary polygonal mesh and outputs a valid two-manifold triangle mesh. Unlike previous work, our method allows users to quickly identify areas with potential topological errors and to choose how to fix them in a user-friendly manner. Key steps of our algorithm include the conversion of the input model into a set of voxels, the use of morphological operators to allow the user to modify the topology of the discrete model, and the conversion of the corrected voxel set back into a two-manifold triangle mesh. Our experiments demonstrate that the proposed algorithm is suitable for repairing meshes of a large class of shapes.

Key words: topology, morphology, opening, closing, 2-manifold

1 Introduction

Mesh repair refers to the transformation of a mesh with singularities into an "acceptable" one. "Acceptable" often means a *two-manifold* – a surface S

* Corresponding author
Email address: `Franck.Hetroy@imag.fr` (Franck Hétroy).
URL: `http://evasion.imag.fr/Membres/Franck.Hetroy/` (Franck Hétroy).
[1] Université de Grenoble & CNRS, Laboratoire Jean Kuntzmann
[2] INRIA Grenoble Rhône-Alpes
[3] Universitat Politècnica de Catalunya, Barcelona

such that each point on S has a neighbourhood on S homeomorphic to $\mathbb{R}^2$ (in particular, a two-manifold is a closed surface). Other conditions are sometimes required. A singularity can refer to very different things. We distinguish here three different types of surface singularities.

- *Combinatorial singularities* prevent the mesh, seen as a combinatorial object, from being a two-manifold. We refer to Guéziec et al. [22] for standard definitions related to manifold meshes. Among combinatorial singularities, we find:
  · singular edges: edges with at least three incident faces;
  · singular vertices: vertices whose link is neither a chain nor a cycle.
  See Figure 1 for an example.
  There also are conditions which are singularities only with respect to manifolds without boundaries:
  · boundary edges: edges with only one incident face;
  · boundary vertices: regular (i.e. not singular) endpoints of boundary edges.
  Isolated elements may also be considered singularities:
  · isolated edges: edges with no incident face;
  · isolated vertices: vertices which are not endpoints of any edge.

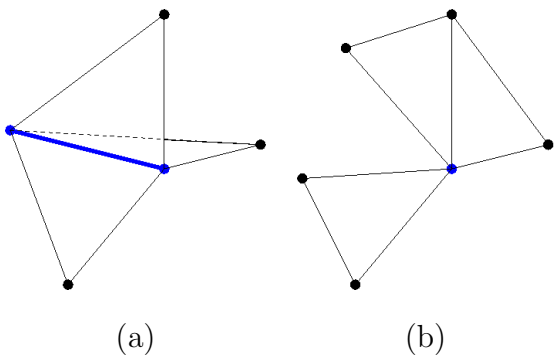


Fig. 1. (a) Singular edge (its endpoints are singular vertices), (b) singular vertex.

- *Geometrical singularities* prevent the mesh, seen as the *embedding* of a surface in $\mathbb{R}^3$, from being the boundary of a three-dimensional object. For example, two triangles whose interiors intersect each other create a geometrical singularity. See [40] for a list of possible embedding inconsistencies.
- *Topological singularities* prevent the surface from having the desired genus or the desired number of connected components. As an example, complex meshes often contain small undesired handles, creating multiple small tunnels in the object they represent.

Another common singularity created by modern acquisition processes is a complex hole with (possibly) tiny islands within (see e.g. [15], or Figures 9, 10 and 11). This singularity can be seen either as a set of combinatorial singularities (boundary edges and boundary vertices), or as a geometrical singularity, since it prevents the surface from being the boundary of a volumetric object.

Mesh repair is important mainly for two reasons. First, meshes are widely used to represent (the surface of) 3D objects in computer graphics, because of their flexibility in visualization, manipulation and computation tasks. Second, the acquisition process from a real object (for example, using a scanner) often creates inconsistent meshes. These are due to the inherent limitations of 3D acquisition devices, but also sometimes to the inner geometry of the object. For instance, hidden parts of an object cannot be reconstructed correctly by a scanner. This might not be important, but unfortunately, many applications require as input mesh a valid two-manifold. Consequently, there is much work on tackling this problem of removing singularities from meshes.

In this paper we present a mesh repair algorithm able to remove all geometrical, combinatorial and topological singularities from an arbitrary polygonal model. The main idea is to discretize the input model into a voxel-set representation and to iteratively apply morphological operators to detect areas which are likely to accept topologically-different reconstructions. We allow the user to quickly identify these parts and to choose the desired topology in a user-friendly manner. Once the topology has been fixed, the algorithm extracts a valid two-manifold surface from the voxel set. The use of an intermediate discrete representation allows our algorithm to guarantee a valid two-manifold output for any input model, including polygon soups. To the best of our knowledge, this is the first mesh repair method where the user is assisted by an explicit indication of topologically-ambiguous areas in a discrete representation.

2 Related work

A number of surface reconstruction methods have been proposed to create manifold meshes from various types of input data, including point clouds (see [31] for a recent paper with a comprehensive discussion of related work). These algorithms can be applied also to mesh repair (taking as input e.g. the set of mesh vertices) but they offer no topological control, which is particularly important in presence of noisy or unproperly sampled data. In this section we focus on mesh repair methods for removing combinatorial and geometrical singularities (Section 2.1) and topology modification techniques (Section 2.2).

2.1 Combinatorial and geometrical singularity removal

Mesh repair methods can be split into two categories: surface-based methods and volume-based methods. The former operate directly on the input mesh,

while the latter use an intermediate voxel representation. Note that comprehensive overviews of existing works can be found in [11] and [26].

2.1.1 Surface-based methods

Guéziec et al. propose a method to remove combinatorial singularities [22]. Their method converts a set of polygons into a 2-manifold by applying local operators. It has several advantages: since it does not handle the geometry, coordinates are not important and there is no approximation error; moreover, attributes such as colours, normals and textures can be preserved, and the algorithm works in linear time. Unfortunately, it removes only combinatorial singularities, and much user intervention is often required. Borodin et al. remove several combinatorial and geometrical artefacts such as unwanted gaps and cracks using a vertex-pair contraction operator and an iterative decimation algorithm [10]. Unfortunately, this method does not handle the very complex holes (with possible tiny "islands" inside) produced by modern acquisition hardware. Creating a closed mesh that fills these holes is sometimes known as the *surface completion* problem. Davis et al.'s method [15] was one of the first to tackle this problem. Unfortunately, in some cases it can produce excessively curved regions. Other surface completion methods include [32,42,41,4]. Surface-based methods are often automatic, but fail to repair geometrical singularities such as self-intersecting polygons. For instance, in order to fill holes, these methods only consider the neigbourhood of theses holes, and do not prevent the patches they create from intersecting the surface away from them. An interactive surface-based method is described in [6]. In this work, both the input model and the currently (partly) corrected one are displayed in a visualization interface. Several types of geometrical or combinatorial singularities are highlighted, and the user can select the ones he wants to repair. Attene's automatic method [5] supposes that the sampling of the model is regular, thus practically avoids the previous problem since it modifies the mesh locally and as less as possible. Recently, Pauly and colleagues have proposed to repair a model by replicating discovered regular features [37]. Contrary to previous methods, this technique does not act locally but globally.

2.1.2 Volume-based methods

Volume-based methods generate consistent surfaces, since their output will be the boundary of a volume. Moreover, they provide accurate error bounds between original and final models. One of the first volume-based method is, to our knowledge, Murali and Funkhouser's [35]. This method uses a BSP tree to represent the original surface, but is quite expensive. Recently, Ju proposed a new volume-based algorithm to convert a "polygon soup" into a 2-manifold [25], using an octree to guarantee the creation of a closed surface. This method

is robust in the sense that it preserves detailed geometry and sharp features of the original model. However, it does not handle correctly thin structures and also does not remove topological singularities. Podolak and Rusinkiewicz recently presented a volume-based method for mesh completion [38]. The volume is represented by a graph, which is subsequently separated into two sub-graphs representing the interior and the exterior of the model. This method allows different ways of filling some holes, depending on the object's desired topology. Bischoff et al. [8,9] presented a method to remove combinatorial, geometrical and topological singularities from a CAD model, using an octree. As far as we know, this is the first work which solves all three types of singularities; unfortunately, it is designed mostly for CAD models, since it generates an approximation of the original model (the model is resampled), where sharp features are preserved. Moreover, holes in the mesh are closed only if greater than a user-defined threshold, whereas the value of a relevant threshold may differ from one part of the model to another, depending on the geometry.

2.2 Topology simplification

Removing topological singularities from a mesh is often seen as a different problem. Besides methods simplifying both topology and geometry [18,1,3], a few methods try to simplify topology while preserving the geometry of a model. Guskov and Wood use a local wave front traversal to cut small handles [23], but they cannot detect long thin handles. Moreover, they need a 2-manifold as input. Similarly, the recent method proposed by Wood et al. [43] operates only on 2-manifolds. It finds handles using a Reeb graph, and then measures their size in order to select those to be removed. The final task is quite slow, leading to a relatively high computation time. In the context of medical imaging (the aim is to correctly segment a genus-0 cortex), Kriegeskorte and Goebel propose to use a heuristic estimate of the misclassification damage caused by inverting a voxel in the segmentation, in order to choose between cutting a handle and filling a hole [28]. Nooruddin and Turk proposed a method based on a volumetric representation and on morphological operators to repair and simplify the topology of a mesh before simplifying its geometry [36]. They first voxelize the model, using several scanning directions, apply open and close operators to simplify the topology, extract an isosurface, and then simplify it. Whether they can completely control the topology of the final object or not remains unclear. Recently Zhou et al. proposed a fast and robust method to break the smallest handles of a model [45]. This is done using a volumetric representation of the model, which is thinned to a topological skeleton. Smallest handles are removed by breaking skeleton cycles and then growing the modified skeleton accordingly. The same year, the authors proposed in another paper an original approach in which the user can control the location of handle removal or hole filling by sketching lines [27]. Another user-

assisted program has been proposed in [4]. This method can not only repair some geometrical singularities, but also automatically detect tiny handles. Finally, Campen and Kobbelt propose an approach to modify the topology of a polygonal model, which combines an adaptive octree and nested binary space partitions [12]. Their approach can be used to remove geometrical singularities in a mesh.

3 Method overview

We propose a new method to convert a triangular mesh with geometrical, combinatorial and topological singularities into a 2-manifold whose topology is supervised by the user. It combines volume-based 2-manifold creation and adapted topology modification. The algorithm proceeds through the following steps:

(1) the input surface is converted into a set of voxels, called a discrete membrane;
(2) morphological operators (openings and closings) are applied to the discrete membrane to detect areas which can change topology (hole creation or filling, shell connection or disconnection);
(3) the user selects the voxels to be added or deleted from the discrete membrane
(4) the modified voxel set is converted into a 2-manifold with guaranteed topology.

The pipeline of our algorithm is depicted on Figure 2. Note that stages (2) and (3) can be iterated several times.

We have chosen to use a volumetric intermediate model to be sure to remove all combinatorial and geometrical singularities. The output model is guaranteed to be a 2-manifold. Our method can be related to Nooruddin and Turk's [36] since we also use morphological operators to control the topology. However our classification between interior and exterior voxels is more robust due to the discrete membrane, and we allow the user to monitor the topology modification step. As in [27], topology modification is interactive: as the user often knows the topology of the object (including the location of handles and holes), this provides a better repair than a fully automatic method. But, unlike [27], we have chosen to assist the user during the process, by indicating topologically ambiguous areas. Another interactive program to repair geometrical singularities and remove tiny handles is described in [4]. This program is perfectly suited for meshes with a relatively low number of defects; however, since it uses a surface-based approach, it cannot handle completely degenerate meshes such as polygon soups (see Fig. 15 (a)).

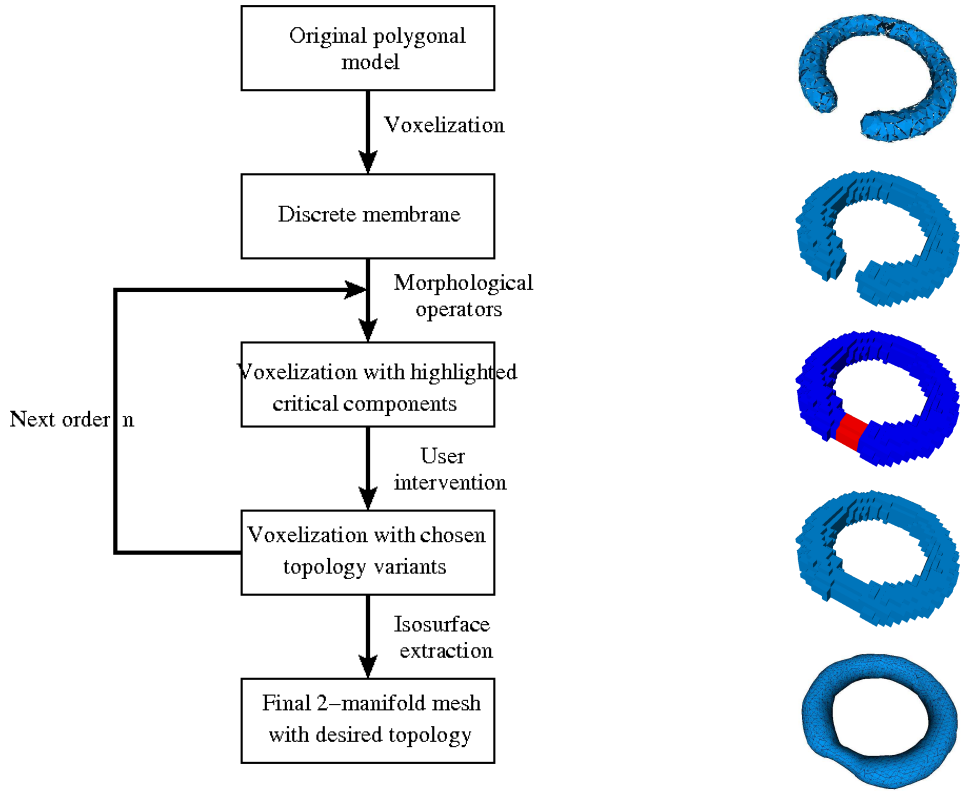


Fig. 2. Algorithm overview.

In our approach the input mesh can be as degenerate as a "soup" of triangles. It is supposed to be a potentially extremely noisy approximation of the boundary of a real, smooth, closed 3D object. The properties of the output mesh are the following:

- all types of singularities listed in Section 1 are solved, i.e. the final mesh is the 2-manifold boundary of a 3D object;
- its topology is controlled by the user;
- its geometry is an approximation of that of the input mesh.

We have chosen not to rely on the input mesh geometry because it can be extremely noisy. However, in case the user wants to repair only a small part of the input mesh, our algorithm can be applied locally. This is discussed in sections 4.4 and 6.3.

Our main contribution is the interactive correction of the mesh topology. We make the following assumptions:

- the mesh represents the boundary of one or several solid objects $O_1, \ldots, O_k$, and the set $\mathbb{R}^3 \backslash \{O_1, \ldots, O_k\}$ is connected; in other words there is no cavity inside any object at the beginning of the process;
- the user knows the correct topology of these objects;

- this topology is relatively simple with respect to the geometry – the number of connected components and holes in the objects is much lower than the number of triangles in the mesh;
- this topology is not necessarily trivial (0 genus), and the (geometrical) location of holes or handles needs user assistance.

Because there is no threshold on the feature size in our method (the user can choose to fill some holes while not filling smaller ones), the algorithm can automatically repair very tiny topological imperfections, in case greater ones exist. See Figure 12 for an example.

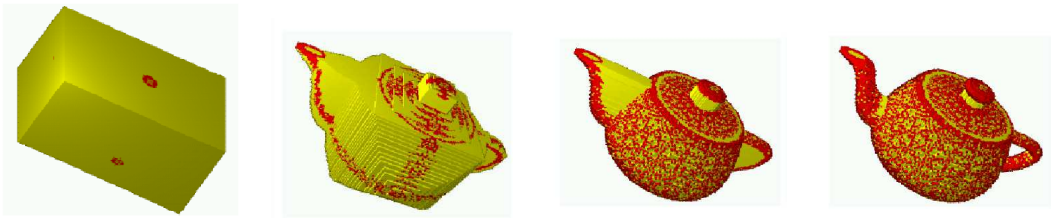
4 Voxelization

4.1 The Discrete Membrane algorithm

To construct a voxel set representing the input model, we use an adapted version of the algorithm described in [19]. This algorithm takes as input a cloud of points, voxelizes the space containing the point set and computes a *discrete membrane* of voxels containing these points (see Figure 3). The discrete membrane is a set of 6- (face-)connected voxels which divides the remaining voxels into *interior* and *exterior*, which means there is no path made of 26- (vertex-)connected voxels disjoint with the membrane that goes from a voxel labeled interior to a voxel labeled exterior. However, to be consistent with the subsequent stages of our method (see sections 5.1 and 5.2), we consider instead the "dual" case where the membrane is required to be 26-connected, while the path-connectedness uses 6-connected paths.

The discrete membrane is initialized as the boundary of the voxelization. It is then contracted using plates, which are sets of $n \times n$ voxels that form a square parallel to a coordinate plane, for decreasing values of n. Each plate is given an orientation, perpendicular to the plate. This orientation allows to distinguish between its front and back sides [19]. Front voxels are the voxels located in front of the plate according to its orientation. Lateral voxels are the voxels located around the plate. Lateral front voxels are the voxels located around the front side of the plate. A plate contraction converts discrete membrane voxels belonging to the plate to outside voxels, while the front, lateral and lateral front voxels of the plate are converted to discrete membrane voxels. The contraction operation is applied recursively at the front, up, down, left and right directions in relation to the plate orientation. If an incursion inside the model is detected, the contraction is undone and corresponding voxels are *frozen*. See [19] for details. The voxels containing the input points locally terminate the shrinking; the process is stopped when the membrane cannot be

contracted anywhere. The number of frozen voxels is at most the number of voxels of the discrete membrane minus the number of voxels containing input points. Note that the membrane is not necessarily simply connected; it can also have a non-0 genus, see for instance Figure 3 (d). Finally, the discrete membrane is relaxed to obtain a smoother surface afterwards.



(a) (b) (c) (d)

Fig. 3. From a cloud of points to a discrete membrane [19]: (a) voxelization of the 3D space (voxels containing input points are shown in red), (b,c) silhouette shrinking with reducing plate size, (d) final discrete membrane.

The main advantage of this algorithm is its robustness – it can handle point clouds with non-uniform density. Although the construction algorithm is not hierarchical and might be relatively slow for huge voxelizations, we have adopted it because because it guarantees us to obtain, as a starting point for subsequent topological and geometrical processing, a coarse approximation of the input soup of triangles which already is almost a 2-manifold: the set of external faces of the discrete membrane. Moreover, the topology of the computed discrete membrane is explicitly related to the distance between the input points (see [19], prop. 7 and 8). In other words, the correct genus and number of connected components are recovered if the point cloud is sufficiently dense (w.r.t. the size of the tunnels or the distance between connected components).

4.2 Voxelization size

The resolution of the voxelization is crucial, since it has a high influence on the results of morphological operators that would be applied to the voxel set. It is a user-defined parameter, but can also be automatically estimated (see [19], section 4.3). On one hand, to be able to represent a topological feature (handle, hole), the size of the voxels at the chosen resolution should be smaller than the feature. This gives a lower bound on the required resolution see Figure 12. However, the higher the resolution, the slower the computation: Table 2 presents timings for three different example voxelizations of the same object (a Sierpinski complex). This algorithm supposes that no voxel of the external "layer" of the initial voxelization belongs to the final discrete membrane. In other words, the voxelization size in each direction is at least equal to the

discrete membrane size in the same direction + 2.

4.3 Extension to meshes

Since our input is not a cloud of points but a triangulated mesh, we have modified the algorithm of [19]. We not only compute the voxels containing the input points, but also the voxels intersecting the faces. This is done using a small additional function, described below (Algorithm 1). Computation time depends on the voxelization size but is bounded by the number of voxels intersecting the triangle bounding boxes, thus the computation is usually done very fast: about 2 minutes for a Buddha model (see Figure 15) with about 300K faces and a voxelization size of more than 2M voxels on a low-end computer (see Section 7.5 for details on timings).

Algorithm 1 Computation of the intersection between the input mesh and the voxelization

```
function ComputeIntersection(Voxelization W, Mesh M)
  for each triangle T of the mesh M do
    Compute the bounding box BB of T;
    for each voxel V of W that intersects BB do
      if V intersects T then
        Label V as red;
      end if
    end for
  end for
  Return all voxels labeled as red;
end function
```

4.4 Local voxelization

To speed up the process, the user can choose to select only a part of the mesh and to apply this voxelization stage locally. This does not change the input mesh in distant areas from the selected polygons. For the subsequent topology correction to be efficient, the selected zone must be significantly greater than the size of the topological features to be repaired. Also, since the original discrete membrane algorithm aims at recovering a closed object, additional voxels must be added at the boundary of the computed voxelization to fill unwanted tunnels and get a closed set of voxels (see Figure 14 for an example). To do this, we add planes of voxels, lying on the boundary voxels. Areas of the final surface corresponding to these voxels will be removed later, once the topology has been modified.

4.5 Comparison with other voxelization techniques

As stated previously, we use an enhanced version of the algorithm presented in [19] because it can handle difficult cases, including non-uniform polygon soups. Some other techniques produce fast results but cannot be applied in all cases, such as [17] which requires as input a watertight mesh. We compared our method with the `binvox` program [34], which is based on the algorithm proposed by Nooruddin and Turk [36] (discussed in Section 2.2), and with the algorithm proposed by Haumont and Warzée [24]. On the noisy Buddha triangle soup (see Fig. 15 (a)), our method took 129s to create a $100 \times 234 \times 100$ voxelization, while `binvox` took 112s to create a $132 \times 132 \times 132$ voxelization. Results are similar voxelizations with a similar number of voxels. In the meantime, the method by Haumont and Warzée only took 56s on the same low-end computer to voxelize the input polygon soup with an octree of depth 7. However, some voxels outside the model are wrongly included in the voxel set.

5 Interactive topology modification using morphological operators

Before converting the discrete membrane into a two-manifold mesh, it is important to let the user decide the final topology, including the number of surface shells and their genus (number of holes or, similarly, handles). To control the topology of the output of our algorithm, we apply morphological operators on a volume. The volume is not the discrete membrane itself, but the discrete membrane plus the interior of the object it bounds, which is automatically known from the discrete membrane construction. Since each voxel of the discrete membrane is 6-connected to at least one interior voxel, this voxel set is a 3-manifold with boundary – the neighbourhood of each point is homeomorphic to either a sphere or a hemisphere.

5.1 Topology of discrete volumes: notations and definitions

Let $\mathcal{V}$ be the voxel set. Its numbers of vertices, edges, faces and voxels are respectively noted $k_0(\mathcal{V})$, $k_1(\mathcal{V})$, $k_2(\mathcal{V})$ and $k_3(\mathcal{V})$. The Euler characteristic χ of $\mathcal{V}$ is defined as $\chi = k_0(\mathcal{V}) - k_1(\mathcal{V}) + k_2(\mathcal{V}) - k_3(\mathcal{V})$.

The topology of a 3-manifold can be characterized by three numbers, named *Betti numbers*. j^{th} Betti number β_j is defined as the rank of the jth homology group H_j (an introduction to homology groups, with accurate definitions of Betti numbers, can be found in [16]). What is more interesting for our study is

that Betti numbers correspond to numbers of connected components ($\beta_0(\mathcal{V})$), tunnels ($\beta_1(\mathcal{V})$) and voids (or cavities) ($\beta_2(\mathcal{V})$) of the volume $\mathcal{V}$. Betti numbers are also related to χ, because $\mathcal{V}$ is a cell complex: $\chi = \beta_0(\mathcal{V}) - \beta_1(\mathcal{V}) + \beta_2(\mathcal{V})$.

The topology of the final surface is linked to the topology of our voxel set, since this surface corresponds to its boundary. The number of connected components of the surface equals $\beta_0(\mathcal{V})$, and its genus (sum of the numbers of holes of all connected components) equals $\beta_1(\mathcal{V})$, provided that we use consistent neighbourhood definitions. In the following, we use the 26-neighbourhood relationship for the volume $\mathcal{V}$, and the 6-neighbourhood relationship for $\mathcal{V}^C$ the complementary set of $\mathcal{V}$: two voxels sharing a vertex, but not an edge, are said to be neighbours if they both belong to $\mathcal{V}$, but not if they belong to $\mathcal{V}^C$. This prevents topological inconsistencies. We use the 26- instead of the 6-neighbourhood relationship for $\mathcal{V}$ to be consistent with the computation of χ as $k_0(\mathcal{V}) - k_1(\mathcal{V}) + k_2(\mathcal{V}) - k_3(\mathcal{V})$.

Computing Betti numbers is not a trivial task [21]. The number of connected components $\beta_0(\mathcal{V})$ can be computed in various ways, for instance using disjoint-set data structures [14]. The number of cavities equals the number of connected components of $\mathcal{V}^C$ minus one (representing the "exterior" of $\mathcal{V}$), so it can also be computed. However $\beta_1(\mathcal{V})$ is not easily found. Lee et al. compute $\beta_1(\mathcal{V})$ counting the number of non-separating cuts [30] ($\beta_1(\mathcal{V})$ is the maximal number of non-separating cuts not increasing $\beta_0(\mathcal{V})$). Another algorithm to detect non-separating cuts has been proposed by Guskov and Wood [23]. In our approach, we prefer to compute $\beta_1(\mathcal{V})$ locally, following the approach of Bischoff and Kobbelt [7]. This enables us to quickly detect topological changes caused by the application of morphological operators (see Section 5.3), even if the two other Betti numbers still need to be computed globally. $\beta_1(\mathcal{V})$ is computed thanks to previous relations $\chi = \beta_0(\mathcal{V}) - \beta_1(\mathcal{V}) + \beta_2(\mathcal{V})$ and $\chi = k_0(\mathcal{V}) - k_1(\mathcal{V}) + k_2(\mathcal{V}) - k_3(\mathcal{V})$. In order to efficiently compute χ (without keeping track of faces, edges, etc.), we exploit the fact that χ is additive, as did for instance [7] – given two sets A and B, $\chi(A \cup B) = \chi(A) + \chi(B) - \chi(A \cap B)$. Since each vertex of $\mathcal{V}$ belongs to 8 voxels (remember that no voxel of $\mathcal{V}$ belongs to the external "layer" of the initial voxelization), χ is the sum of the local Euler characteristics around each vertex of the voxelization divided by 8. The local Euler characteristic around a vertex v is defined as $k_0(\mathcal{V}, v) - k_1(\mathcal{V}, v) + k_2(\mathcal{V}, v) - k_3(\mathcal{V}, v)$, with $k_0(\mathcal{V}, v), k_1(\mathcal{V}, v), k_2(\mathcal{V}, v)$ and $k_3(\mathcal{V}, v)$ denoting respectively the number of vertices, edges, faces and voxels that both belong to $\mathcal{V}$ and are incident to v. It can be computed using a lookup table, since only 256 2×2 voxel configurations can occur (in fact, up to isomorphism, we only have 22 different configurations). Moreover, since χ is additive, we do not need to compute the Euler characteristic around each vertex at each step of our algorithm. Each time we add or remove voxels, we only need to update local Euler characteristics around corresponding ver-

tices. Once χ is computed, we immediately have the number of tunnels in our volume: $\beta_1(\mathcal{V}) = \beta_0(\mathcal{V}) + \beta_2(\mathcal{V}) - \chi$.

5.2 Morphological operators

In order to track down areas of the object where topology is wrong (that is to say, irrelevant handles creating tunnels or connecting different components, or on the contrary missing tunnels or bridges between several parts of a connected component), we use morphological operators. Basic operators are *erosion* and *dilation*. The erosion operator $\mathcal{E}$ transforms the set of voxels $\mathcal{V}$ into the set $\mathcal{E}(\mathcal{V}) = \{V \in \mathcal{V}$, all 26-neighbours of V are also in $\mathcal{V}\}$. The dilation operator $\mathcal{D}$ transforms $\mathcal{V}$ into the set $\mathcal{D}(\mathcal{V}) = \{$ voxels $V \in \mathcal{V}$ or V is a 26-neighbour of some voxel of $\mathcal{V}\}$ [7]. Two combinations of these two operators are called *opening* and *closing*: $\mathcal{O} = \mathcal{D} \circ \mathcal{E}$ and $\mathcal{C} = \mathcal{E} \circ \mathcal{D}$. Erosion and opening can expand holes and disconnect parts, while dilation and closing can close holes and connect previously disconnected parts of the volume. Figure 4 shows these four operators applied on an example. Note that they can be defined using either the 6- or the 26-neighbourhood relationship; we choose to use the 26-neighbourhood relationship because it generates larger modifications to the set $\mathcal{V}$. This choice has no influence on the neighbourhood relationship defined later for the connected components of the set.

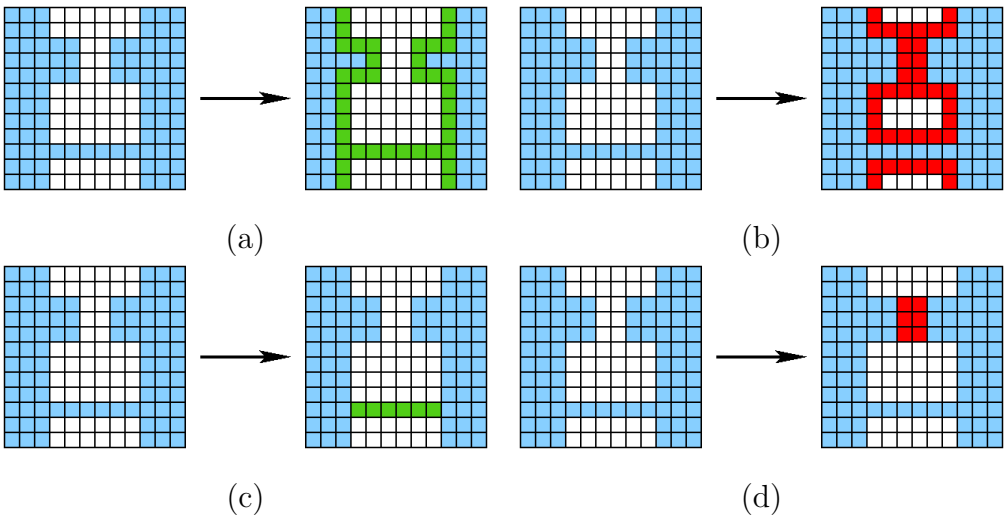


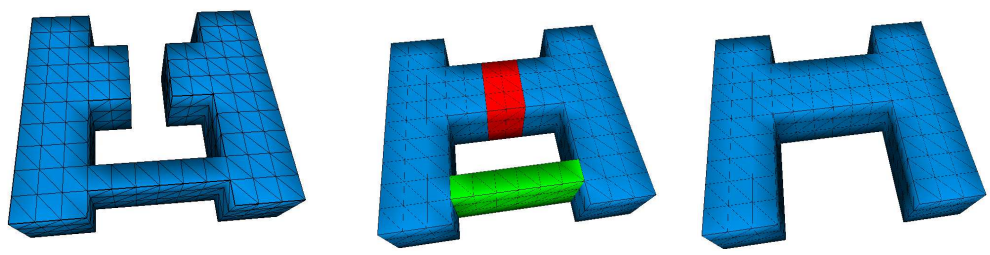
Fig. 4. Morphological operators applied on a 2D set of blue pixels: (a) erosion, (b) dilation, (c) opening, (d) closing. Removed pixels are in green while added pixels are in red.

To go further and detect bigger topologically critical areas, we can iterate this process. We call *opening of order* n (noted $\mathcal{O}^n$) a sequence of n erosions followed by n dilations, and *closing of order* n (noted $\mathcal{C}^n$) a sequence of n dilations followed by n erosions, $n \geq 1$.

Note that we choose to use opening and closing instead of erosion and dilation because they avoid shrinkage or expansion of the model. Erosion and dilation are faster to compute, but they usually shrink or expand the model.

5.3 Algorithm

We start from the voxel set $\mathcal{V}$ (the discrete membrane plus its interior volume). The discrete membrane is computed as described in section 4 (see Figure 5 (a)). In order to detect topologically critical areas, we apply openings and closings to $\mathcal{V}$ (the order n is selected by the user). We then compute the set of voxels which belong to $\mathcal{V}$ and not to $\mathcal{O}^n(\mathcal{V})$ and the set of voxels which belong to $\mathcal{C}^n(\mathcal{V})$ and not to $\mathcal{V}$. We cluster voxels of these two sets in 26-connected components (because we use the 26-neighbouring relationship for our set of voxels). For each component K, we then compute the Betti numbers of the new set of voxels $\mathcal{V}\backslash K$ or $\mathcal{V} \cup K$ (remember that the new Euler characteristic can be computed simply by adding or removing local Euler characteristics of vertices of K to or from the Euler characteristic of $\mathcal{V}$), and compare it to the Betti numbers of $\mathcal{V}$. If one of them changes, we have detected a topologically critical area, which we call a *critical component* of the voxel set. In this case, K is labelled with a special tag: "candidate for removal" or "candidate for addition". The discrete membrane together with the critical components are displayed in a visualization interface, in which the user can select to remove and/or add some critical components to the voxel set (Figure 5 (b) and (c)). Each time a critical component is removed or added, the new topology is displayed.



(a) (b) (c)

Fig. 5. (a) Discrete membrane. (b) Discrete membrane with critical components: candidate voxels for removal are shown in green and candidate voxels for addition are shown in red. (c) Voxel set after the user chose to add the red component and to remove the green one. In this example, we applied an opening and a closing of order 1.

In case the user does not have a guess about the opening and closing order n he should apply, or in cases he wants to remove topological artifacts with various sizes, he can apply this algorithm several times, with different values

for n. For instance, small topological errors can be corrected first, with a small value for n; then large ones can be detected with a larger value for n. Figure 16 shows an example of this process.

6 Isosurface computation

Once we get a discrete volume with the desired topology, we convert it to a surface by using a Marching Cubes-like algorithm. We use the dual of the voxelization as the grid; each vertex of this grid is labeled as *interior* to the surface if it corresponds to a voxel of $\mathcal{V}$, otherwise it is considered as *outside* the surface. The volume of the resulting surface roughly corresponds to the volume of $\mathcal{V}$. This volume may thus be a bit greater than the volume inside the input mesh, because $\mathcal{V}$ includes all voxels intersecting the mesh. However, the surface is then shrunk during the smoothing step (see section 6.2), thus reducing this volume.

6.1 Topology preservation

The standard Marching Cubes algorithm [33] is known to generate topological inconsistencies, due to ambiguous configurations, which are called *X-faces* and *X-cubes* in [2] (see Figure 6). X-cube corresponds to pattern number 4 of the original Marching Cubes look-up table [33], while X-faces appear in patterns number 3, 6, 7, 10, 12 and 13. In our case, each X-face is related to two voxels (either of $\mathcal{V}$ or of $\mathcal{V}^C$) which are edge-connected. Each X-cube is related to two voxels which are only vertex-connected.

Since we defined connections between voxels of $\mathcal{V}$ using the 26-neighbourhood relationship (thus using the 6-neighbourhood relationship for $\mathcal{V}^C$, see Section 5), we need to connect interior vertices, within each X-face and each X-cube. This corresponds to situations described on Figure 7, which lead to cases 3.1, 3.2, 4.1.1, 4.1.2, 6.1.1, 6.2, 7.1, 7.4.1, 10.1.1 and its opposite configuration, 12.1.1 and its opposite configuration, and 13.1 and its opposite configuration, in Chernyaev's advanced look-up table ([13]; see Figure 8 of this paper for the corresponding local triangulations).

6.2 Smoothing

The previous stage generates a 2-manifold whose vertex coordinates have been estimated in a very simple way: each vertex lies in the midpoint of an edge

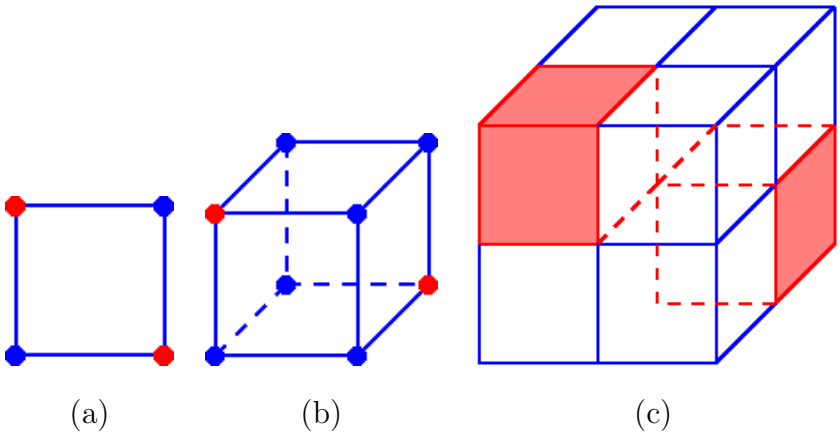


Fig. 6. Marching Cubes ambiguous cases (from [2]). (a) X-face: two opposite vertices correspond to voxels of $\mathcal{V}$ (*interior vertices*), while the two others correspond to voxels of $\mathcal{V}^C$ (*outside vertices*). (b) X-cube is the only Marching Cubes ambiguous configuration without any X-face. (c) Corresponding voxelization of (b): voxels of $\mathcal{V}$ (in red) are 26-connected, but not 6-connected.

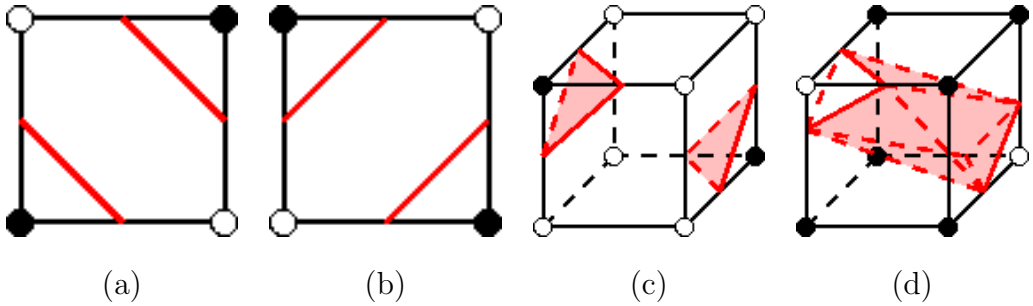


Fig. 7. Solving Marching Cubes ambiguous cases. White vertices correspond to voxels of $\mathcal{V}$, while black vertices correspond to voxels of $\mathcal{V}^C$.

of the grid. Since we assume the input mesh is noisy, it seems reasonable not to rely on the original vertex positions. In case a smooth mesh is expected as output, some postprocessing is required. The smoothing method must fulfill the following requirements:

- it should not require additional information (such as expected normals),as those produced by the discrete Marching Cubes are not suitable;
- it should preserve features as much as possible while correctly smoothing the sharp edges introduced by the previous method;
- most importantly, it must neither change the topology of the mesh nor create new singularities, such as auto-intersections.

In our implementation, we have chosen to apply the bilateral mesh denoising method of [20], which is fast and satisfies the previous conditions. In practice, no singularity is created as long as the smoothing does not destroy geometrical features; the strength of the smoothing can be controlled with very simple parameters. Only one parameter has been kept in our implementation – the

number of iterations. The normal to the surface at a vertex is computed using the 2-ring neighborhood of the vertex, because by the discretereconstruction technique used, normals are also discretized. The neighborhood used for the computation of the other parameters is set to the 1-ring neighborhood; this is a valid approximation because the aspect ratio of the faces is, by construction, bounded over the mesh.

6.3 Local computation

In case only a part of the input mesh was voxelized, we need to compute a surface that will be merged with the input one. Let $\mathcal{M}$ be the input mesh, and let $\mathcal{S}$ be the selected part of $\mathcal{M}$ that is voxelized into the set $\mathcal{V}$. We recall that additional voxels are added to $\mathcal{V}$ to fill tunnels. Once the Marching Cubes algorithm has been applied, triangles corresponding to at least one of these added voxels are removed first. The boundary of the computed surface $\mathcal{S}'$ is made of one or several (edge) loops, since we added connected, genus-0, sets of voxels. More precisely, the boundary of $\mathcal{S}'$ should be made of as many loops as the boundary of $\mathcal{S}$. Finally, we merge $\mathcal{S}'$ to $\mathcal{M}\backslash\mathcal{S}$ by stitching both boundaries, using pairs of loops. Several algorithms, such as [39,44,29], can be applied to stitch boundaries. In our case we use the following simple user-assisted method, which yields satisfactory results (see Figure 14).

We assume that boundaries we want to stitch are sets of closed lines which need to be matched by pairs. Each closed line is defined as a set of k vertices $v_0, v_1, \ldots, v_{k-1}$, together with the set of edges $(v_0, v_1), \ldots, (v_{k-2}, v_{k-1}), (v_{k-1}, v_0)$. Let us denote $L = (v_0, v_1, \ldots, v_{k-1})$ and $L' = (w_0, w_1, \ldots, w_{l-1})$ two closed lines to be matched. We suppose these closed lines do not intersect; if this is not the case we remove a strip of boundary triangles of $\mathcal{S}'$ so that the new connected component of the boundary L does not intersect L'. We then create new triangles between L and L'. First, each of the two closed lines is cut into developable pieces. This is done manually, usually in a few seconds. We may cut each line into more pieces than required, in order to be sure new triangles subsequently computed from various pieces will not intersect. An efficient heuristic is to cut lines around sharp angles.

Let us now suppose that the subset $(v_i, \ldots, v_j)$ of L needs to be matched with the subset $(w_{i'}, \ldots, w_{j'})$ of L'. We project these two pieces onto a common plane $\mathcal{P}$, which is defined by its normal $v_i v_j \times v_i w_{i'}$ and the point $\frac{v_i + w_i}{2}$ through which it passes. Orthogonal projections of the vertices to this plane are denoted as $v_i^p, \ldots, v_j^p, w_{i'}^p, \ldots, w_{j'}^p$. We then compute a constrained Delaunay triangulation of this set of points, enforcing $(v_i^p, v_{i+1}^p), \ldots, (v_{j-1}^p, v_j^p), (w_{i'}^p, w_{i'+1}^p), \ldots, (w_{j'-1}^p, w_{j'}^p), (v_i^p, w_{i'}^p)$ and $(v_j^p, w_{j'}^p)$ to be edges of this triangulation. Neighbourhood relationships between vertices induced by this triangulation enables us to finally create new

triangles that stitch $(v_i, \ldots, v_j)$ to $(w_{i'}, \ldots, w_{j'})$.

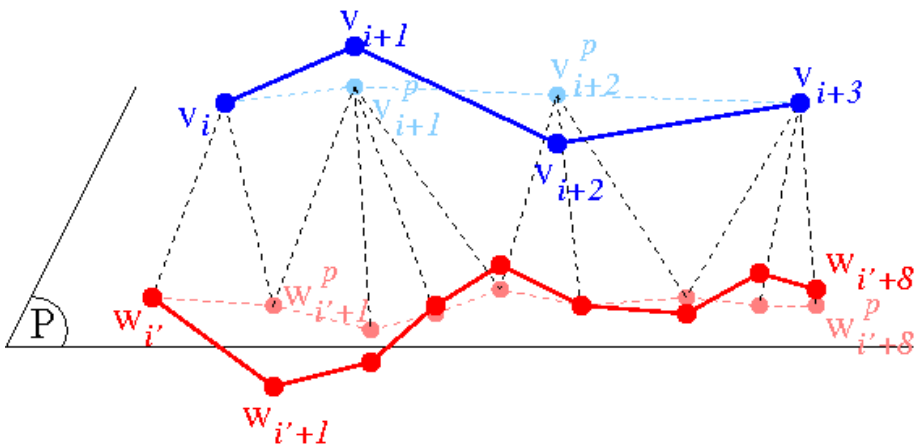


Fig. 8. Stitching algorithm: lines to be stitched $v_i, \ldots, v_j$ and $w_{i'} \ldots w_{j'}$ are projected onto a common plane, then a constrained Delaunay triangulation is computed.

Although this method is quite simple, it yields suitable results in our experiments. Nevertheless more sophisticated, automatic techniques are available (see e.g. [39,44,29]).

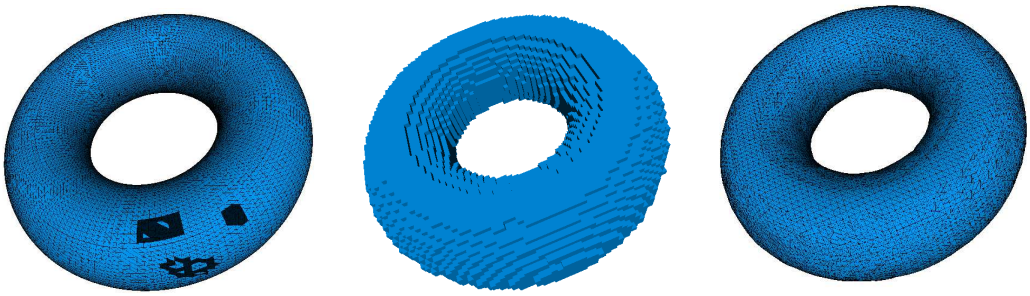
7 Results and discussion

Figure 2 shows the entire process on a model with all types of singularities. This model is a soup of triangles, some of them overlapping, with a hole-like region in the background. The computation of the discrete membrane leads to a voxel set with one connected component with no tunnels. A closing of order 2 enables us to create a tunnel, which leads to a torus-shaped final 2-manifold surface: one connected component, genus 1.

7.1 Repair of combinatorial and geometric singularities

Models with complex holes can be repaired with our method, as can be seen on Figure 9. Very noisy models with holes can also be transformed into smooth 2-manifolds, such as the Buddha model in Figure 15 (a).

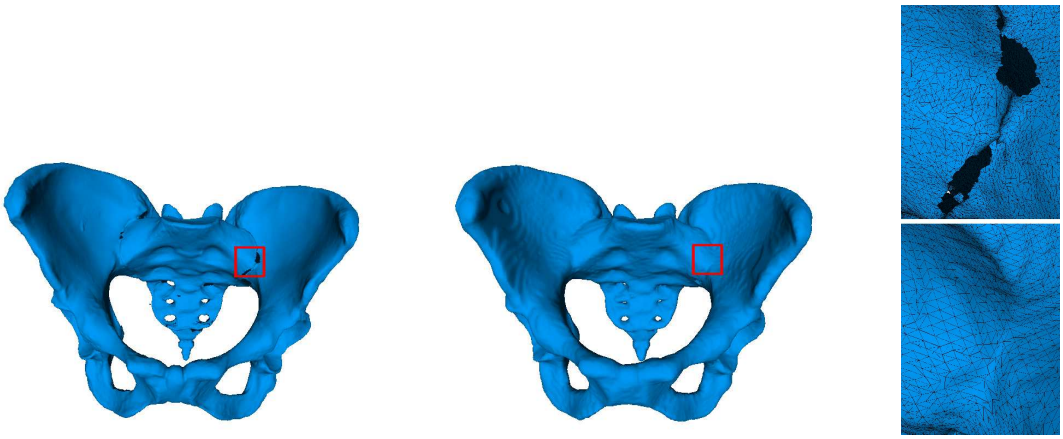
Figures 10 and 11 show how our algorithm can repair noisy meshes acquired with a laser scan. Both input meshes contain many small holes, some of them with islands, and even outliers. In both cases our algorithm produces a two-manifold mesh, with the correct topology (one connected component, genus 9 for the pelvis model, one connected component and genus 0 for the statue). Outliers have been removed from the pelvis model by applying an order 1



(a) (b) (c)

Fig. 9. (a) Input model, containing complex holes with islands within. (b) Computed discrete membrane. (c) 2-manifold result.

opening, and holes have been filled by applying order 1 and order 2 closings. On the statue model, an order 1 closing was sufficient to recover the correct topology.



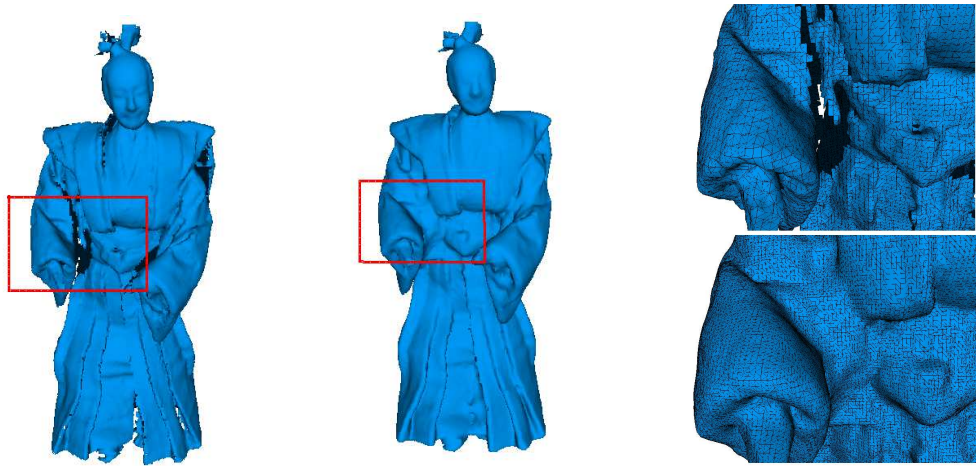
(a) (b) (c)

Fig. 10. (a) Input model. (b) Output mesh. (c) Close up.

7.2 Repair of topological singularities

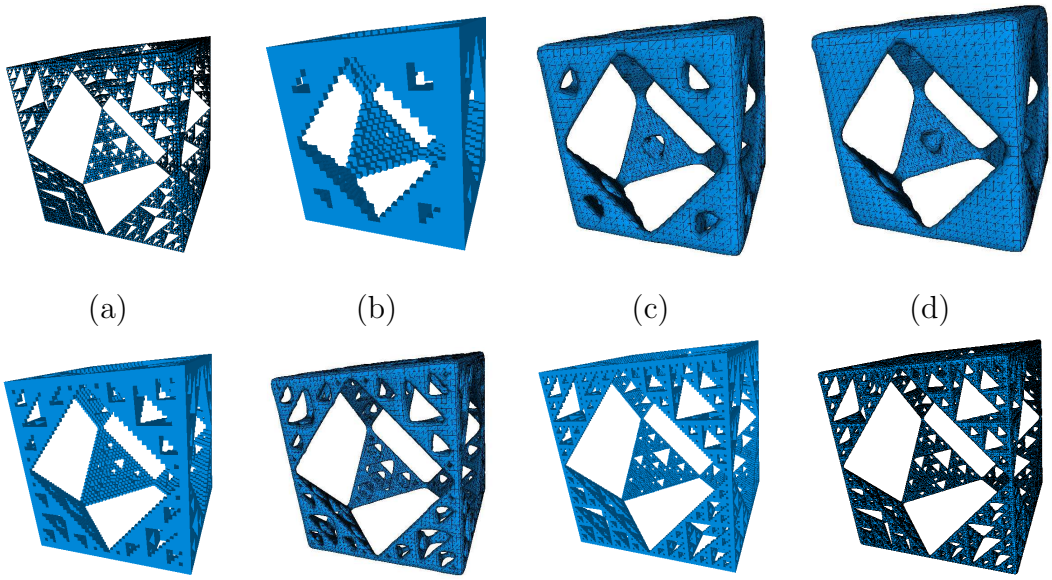
As can be noticed on Figure 12, the resolution chosen for the voxelization has a major influence on the resulting topology: topological features smaller than the voxel size are not recovered. Thus, choosing an appropriate voxelization size can save time for the user, by automatically filling the smallest holes during the discrete membrane computation.

The same model can be repaired in different ways, depending on the expected topology, as shown on Figures 13 and 15. On the statue model, an opening of order 3 is necessary to modify the topology. A closing of order 2 was applied on the noisy Buddha model to fill the two holes on its left side. No morphological



(a) (b) (c)

Fig. 11. (a) Input model. (b) Output mesh. (c) Close up.



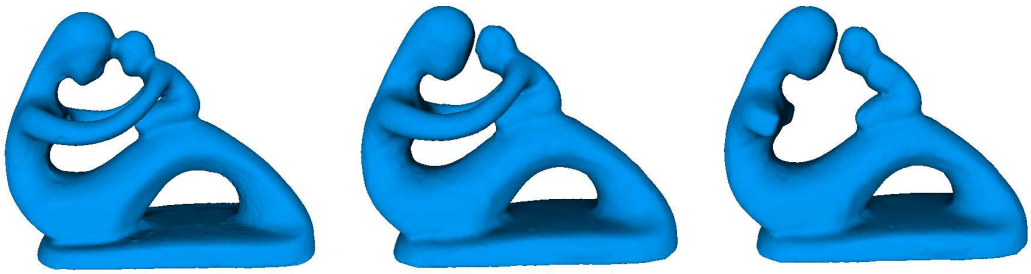
(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 12. (a) Input model: approximation of a Sierpinski fractal. (b) $30 \times 30 \times 30$ voxelization. (c) Output mesh (no morphological operator applied). (d) Output mesh (one closing applied). (e) $50 \times 50 \times 50$ voxelization. (f) Output mesh (no morphological operator applied). (g) $100 \times 100 \times 100$ voxelization. (h) Output mesh (no morphological operator applied).

operator is necessary to recover the topology (genus = 6) of the original non noisy model.

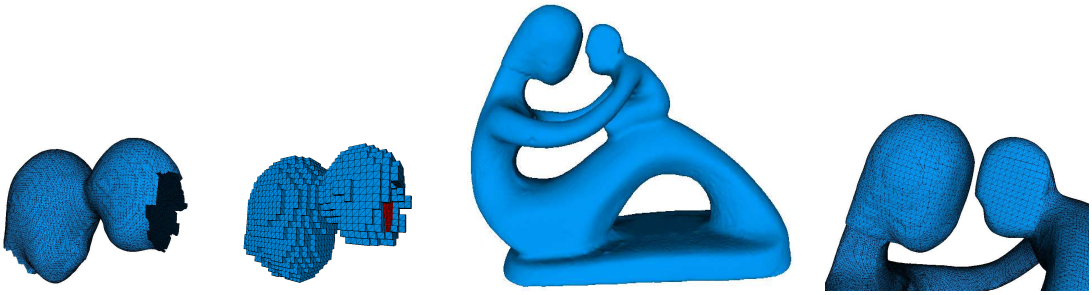
Figure 14 shows an example of a local topological repair on the model depicted in Figure 13. Selecting only a small part of the input model saves time at all stages (see Table 2). It thus allows to select a finer voxelization size, in order



(a) (b) (c)

Fig. 13. (a) Input model, with genus 4. (b) Genus 3 output mesh. (c) Genus 1 output mesh.

to get a more accurate final surface with respect to the initial mesh geometry.



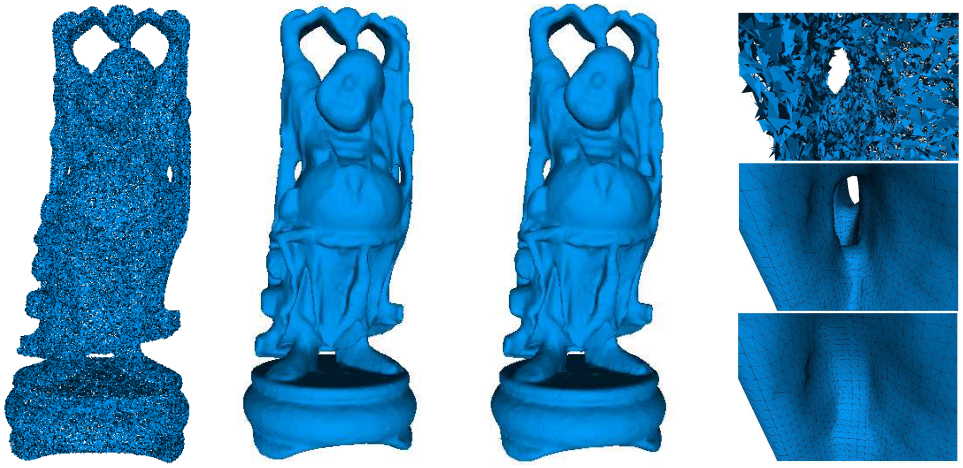
(a) (b) (c) (d)

Fig. 14. (a) Selected zone on the input model. (b) Voxelization, with added voxels in red. (c,d) Output mesh. Compare with Figure 13 (b).

Figure 16 shows how our method can be applied for interactive topology correction. In this example, we merge all the bones of the foot into one connected component. After voxelizing the model, we start by computing the order 1 opening and closing (b). We select some 6-connected components of voxels to be added to the set (in yellow), while we discard others (in red). Green voxels refer to connected components proposed for removal. Obviously, we select none of them. Once the desired voxels have been added to the set, we compute the order 2 opening and closing (c). Once again we choose to add some of the proposed connected components to the voxel set. This leads us to a final set made of only one 6-connected component; thus we can now convert it back to a surface (d).

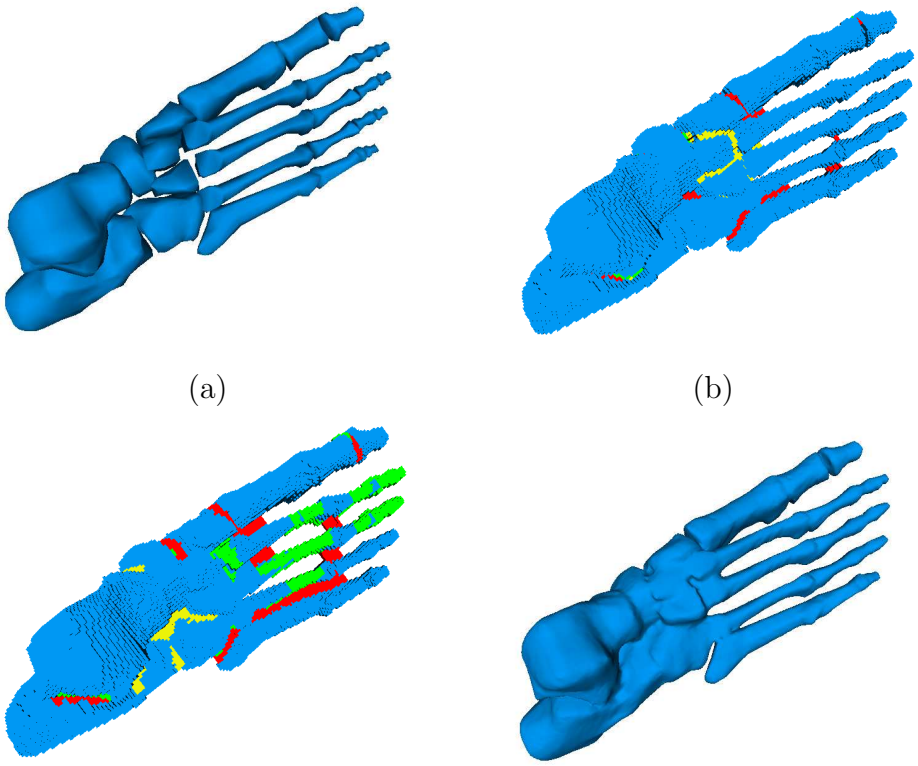
7.3 Number of critical connected components

Table 1 gives the number of computed critical components for some models shown in Figures 10 to 18. This number can be large, therefore inspecting all critical components one by one, as described before, can be tedious and



(a) (b) (c) (d)

Fig. 15. (a) Input model: noisy Buddha triangle soup. (b) Genus 6 output mesh. (c) Genus 4 output mesh. (d) Close-up on the left side (back view).



(a) (b)

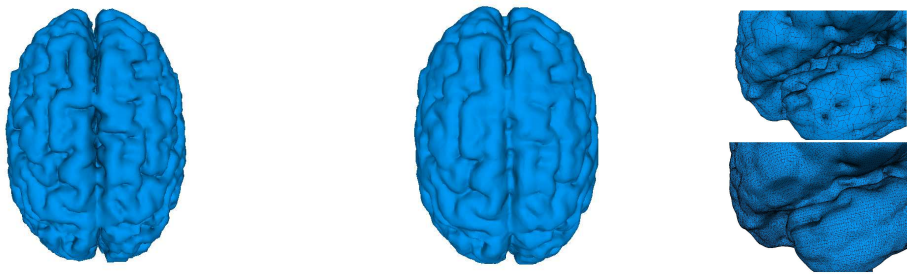
(c) (d)

Fig. 16. (a) Input model. (b) After the order 1 opening and closing. (c) After the order 2 opening and closing. (d) Output mesh.

Model	$\mathcal{O}^1$	$\mathcal{C}^1$	$\mathcal{O}^2$	$\mathcal{C}^2$	$\mathcal{O}^3$	$\mathcal{C}^3$	**Total**
Pelvis	4	32	–	10	–	–	46
Statue 1	–	26	–	–	–	–	26
Statue 2	–	–	–	–	3	–	3
Buddha	–	–	–	5	–	–	5
Foot	21	7	16	5	–	–	49
Brain	17	16	–	–	–	–	33
Paperclip	–	–	3	1	–	–	4

Table 1
Number of critical connected components for some models. $\mathcal{O}^1$ means after applying an order 1 opening, $\mathcal{C}^1$ after an order 1 closing, etc.

time consuming. In case the desired topology is simple (e.g. one connected component with genus 0) or, more generally, in case the location of critical components is not important, our algorithm can be applied without user intervention, since for each critical component we know exactly how it affects the topology. Figure 17 illustrates this point: the initial cortex model has genus 45, while we want a genus 0 mesh. We automatically set all critical components which reduce the genus to be selected. Doing so, we get a final mesh with genus 0 after applying only the order 1 opening and closing.



(a) (b) (c)

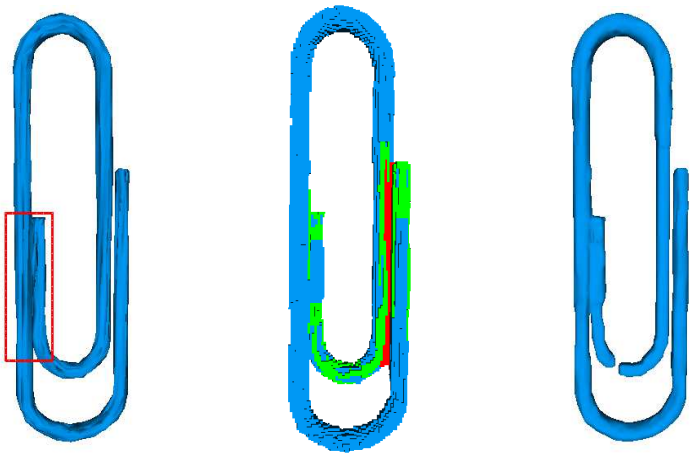
Fig. 17. (a) Input model, with genus 45. (b) Genus 0 output mesh. (c) Close up.

Nevertheless, please note that this simple idea cannot be used in case the desired genus is not 0.

7.4 Failure case

Our algorithm can fail in some cases, especially when trying to change the topology in thin, elongated areas. For instance, see the noisy paperclip model in Figure 18 (a). The original model is supposed to have genus 0, being just a winded stem. Hence, we would like to repair this mesh around the area

highlighted in red. Unfortunately, an order 1 opening is not able to detect any critical component. Meanwhile, the three critical components that an order 2 opening finds encompass a large set of voxels (see on Figure 18 (b); one changes the genus, while the two other break the model into two connected components). This is because of the thin, tubular shape of the model. When applying erosion, we hardly control the area where the voxel set is broken into several components. It may even happen that almost all voxels disappear at the same time.



(a) (b) (c)

Fig. 18. (a) Input mesh with the wrong genus (1 instead of 0) due to noise. (b) After the order 2 opening and closing. (c) Selecting a critical component for deletion yields the correct genus, but the result is not the one sought.

7.5 Timings

Table 2 gives computation times (in seconds) for the all models presented in this paper, on a desktop PC with a 2.13 GHz Pentium 4 processor. We also give user interaction times. 10 iterations were performed for the smoothing stage, except on the pelvis model (5 iterations only). No opening or closing was performed for the torus and the bunny models, since the discrete membrane alone yielded the desired topology. Timings for the pelvis and the foot models include both order 1 and order 2 morphological operators. Interaction times are given for comparison purpose only, since they greatly depend on the visualization interface and the framerate.

The most time-consuming stage is the application of morphological operators. This time can be prohibitive in some cases. Fortunately, it can often be shortened. For instance, many models do not have any internal cavity, and morphological operators hardly create one. In these cases, the computation of the number of cavities can be turned off, resulting in a significant speed up of

Model	**Nb of faces**	**Vox. size**	**1**	**2**	**3**	**Total**	**User**
Torus	25 300	66x23x66	< 1	0	3	< 4	0
Pelvis	500 000	120x100x136	93	669	12	774	20
Statue 1	141 000	285x123x100	64	3491	34	3589	170
Sierpinski	32 000	30x30x30	< 1	< 1	2	< 4	40
Sierpinski	32 000	50x50x50	1	0	5	6	0
Sierpinski	32 000	100x100x100	34	0	23	57	0
Statue 2 (genus 3)	483 000	113x83x45	3	30	7	40	5
Statue 2 (genus 1)	483 000	113x83x45	3	30	5	38	10
Statue 2 (local)	42 500	39x28x21	< 1	< 1	1	< 3	5
Buddha (genus 6)	293 000	100x234x100	129	0	27	156	0
Buddha (genus 4)	293 000	100x234x100	129	2206	29	2364	10
Foot	4 200	216x75x84	16	962	39	990	80
Brain	290 000	134x159x177	21	4014	34	4069	0
Paperclip	2 900	54x180x20	< 1	6	4	< 11	5

Table 2
Computation times (in seconds) for the models of Figures 9 to 18. Stage 1 is the discrete membrane computation, stage 2 is the application of morphological operators and stage 3 is the isosurface computation and smoothing. Statue 1 corresponds to the model of Figure 11, and Statue 2 to the model of Figures 13 and 14.

the process. In our experiments, from 30 to 15s for the second statue model (Figure 13), from 21min 56s to 3min 26s for the foot bone model, and even from 36min 46s to 6min 3s for the Buddha model.

7.6 A guide on user intervention

We now sum up the different ways the user can control the mesh repair process with the proposed method.

- **Voxelization size**: this parameter can be set automatically (see Section 4.2). The larger the size, the slower the algorithm but also the finer the geometrical and topological features that can be recovered. In case the user does not have any clue, we advice to test the voxelization stage with 3 to 5 different sizes, then to choose the smallest one which generates the discrete membrane with sufficient detail.
- **Local application**: in case the model has a huge number of faces, but only very localized singularities to be repaired, we suggest to use our method

locally, then to stitch the result to the non-modified part of the mesh (see Section 6.3).

- **Order of the morphological operators**: this parameter controls the size of the topological singularities that can be detected. Usually, orders lower than 3 are sufficient to recover most singularities. They can be used one after the other.
- **A priori knowledge about the model's topology**: the application of the morphological operators to the voxel set is the most time-consuming stage of the algorithm, but it can be shortened if the user has some information about the desired topology. For instance, the computation of the number of cavities can be turned off if it is known that the model does not have any, or the repair can be done without user intervention if the desired genus is 0 (see Section 7.3).
- **Mesh smoothness**: The third and last user-controlled parameter is related to the bilateral mesh denoising algorithm we use for mesh smoothing. In our experiments, setting this parameter to 10 iterations has been sufficient in all cases. Yet, this is not critical since the smoothing stage is not the most time-consuming one.

We thus propose various ways to control the behaviour of our algorithm. However, it should be recalled that our method is not perfect and can fail in some cases, as discussed in Section 7.4.

8 Conclusion

In this paper we have presented a method to repair meshes with combinatorial, geometrical or topological singularities, or all of them. This method is fast and produces 2-manifolds whose topology is controlled by the user. It runs in four stages:

(1) creation of a discrete volumetric model, which is a 3-manifold with boundary;
(2) application of morphological operators (openings and closings) to detect topologically critical areas;
(3) selection by the user of the areas to remove or add to the model;
(4) reconstruction of a smooth 2-manifold;

Apart from the selection of the topologically critical areas, the user can control some parameters (one for each other stage):

(1) voxelization size;
(2) strength of the morphological operators to apply;
(3) strength (number of iterations) of the surface smoothing stage.

Even if this method is better suited for smooth models because of the final smoothing stage, it has no requirement of the input model, which can be as simple as a triangle soup. A local application of this algorithm is possible, in case most parts of the input mesh do not need repair. It speeds up computation, but needs the user to cleverly select the area to repair.

Possible enhancements of this work include:

- trying to use an octree for the first stage, to speed up the computation of the final voxel set, especially in case the whole input model is selected for repair;
- investigating ways to better control the size and shape of computed topologically critical components;
- developing methods to automatically display these components from relevant viewpoints in the visualization interface;
- modifying the surface reconstruction stage in order to possibly fit some geometrical features of the input model, if the user wants to (e.g. sharp edges). This could be done by using the exact intersection point between the input model and each edge of the grid, instead of the midpoint of the edge. This can only be done when this intersection point is available and reliable; the key issue is to find when it is the case.

Acknowledgements

Part of this work was done while the first author was visiting the Universitat Politècnica de Catalunya in Barcelona with a Lavoisier grant from French Ministry of Foreign Affairs. The Buddha model is courtesy of the Stanford 3D Scanning Repository, the hip model is courtesy of Cyberware, and the pelvis, statue and brain models are courtesy of the AIM@SHAPE Digital Shape WorkBench. The authors would like to thank the anonymous reviewers for their valuable comments, which helped greatly improve the paper, and Nassim Jibai and Kartic Subr for proof-reading.

References

[1] Andújar C, Brunet P, Ayala D. Topology-reducing surface simplification using a discrete solid representation. ACM Transactions on Graphics 2002; 21(2):88–105.

[2] Andújar C, Brunet P, Chica A, Navazo I, Rossignac J, Vinacua À. Optimizing the topological and combinatorial complexity of isosurfaces. Computer-Aided

Design 2005; 37(8):847–857.

[3] Andújar C, Brunet P, Fairen M, Cebollada V. Error-Bounded Simplification of Topologically-Complex Assemblies. Workshop on Multiresolution and Geometric Modelling 2003, p. 355–366.

[4] Attene M, Falcidieno B. ReMESH: an interactive environment to edit and repair triangle meshes. Proceedings of the IEEE International Conference on Shape Modeling and Applications (SMI) 2006, p. 271-276.

[5] Attene M. A lightweight approach to repairing digitized polygon meshes. The Visual Computer 2010; 26.

[6] Barequet G, Duncan C, Kumar S. RSVP: a geometric toolkit for controlled repair of solid models. IEEE Transactions on Visualization and Computer Graphics 1998; 4(2):162–177.

[7] Bischoff S, Kobbelt L. Isosurface reconstruction with topology control. Proceedings of Pacific Graphics 2002, p. 246–255.

[8] Bischoff S, Kobbelt L. Structure preserving CAD model repair. Computer Graphics Forum (Eurographics Proceedings) 2005; 24(3):527–536.

[9] Bischoff S, Pavic D, Kobbelt L. Automatic restoration of polygon models. ACM Transactions on Graphics 2005; 24(4):1332–1352.

[10] Borodin P, Novotni M, Klein R. Progressive gap closing for mesh repairing. Advances in Modelling, Animation and Rendering, Springer-Verlag; 2002, p. 201–213.

[11] Botsch M, Pauly M, Kobbelt L, Alliez P, Levy B, Bischoff S, Rössl C. Geometric modeling based on polygonal meshes. SIGGRAPH 2007 Course Notes.

[12] Campen M, Kobbelt L. Exact and robust (self-)intersections for polygonal meshes. Computer Graphics Forum (Eurographics Proceedings) 2010; 29(3).

[13] Chernyaev EV. Marching Cubes 33: construction of topologically correct isosurfaces. Technical Report CERN CN 95-17, CERN, 1995.

[14] Cormen T, Leiserson C, Rivest R, Stein C. Introduction to algorithms, 2nd edition. MIT Press, 2001.

[15] Davis J, Marschner SR, Garr M, Levoy M. Filling holes in complex surfaces using volumetric diffusion. Proceedings of the Symposium on 3D Data Processing, Visualization, and Transmission 2002; p. 428–438.

[16] Dey TK, Guha S. Computing homology groups of simplicial complexes in $\mathbb{R}^3$. Journal of the ACM 1998; 45:266–287.

[17] Eisemann E, Décoret X. Single-pass GPU solid voxelization for real-time applications. Proceedings of Graphics Interface 2008; p. 73–80.

[18] El Sana J, Varshney A. Topology simplification for polygonal virtual environments. IEEE Transactions on Visualization and Computer Graphics 1998;4(2):133–144.

[19] Esteve J, Brunet P, Vinacua À. Approximation of a cloud of points by shrinking a discrete membrane. Computer Graphics Forum 2005; 24(4):791–807.

[20] Fleishman S, Drori I, Cohen-Or D. Bilateral mesh denoising. ACM Transactions on Graphics (SIGGRAPH Proceedings) 2003; 22(3):950–3.

[21] Friedman J. Computing Betti numbers via combinatorial Laplacians. Algorithmica 1998; 21(4):331–346.

[22] Guéziec A, Taubin G, Lazarus F, Horn W. Cutting and stitching: converting sets of polygons to manifold surfaces. IEEE Transactions on Visualization and Computer Graphics 2001; 7(2):136–151.

[23] Guskov I, Wood Z. Topological noise removal. Proceedings of Graphics Interface 2001; p. 19–26.

[24] Haumont D, Warzée N. Complete polygonal scene voxelization. Journal of Graphics Tools 2002; 7(3):27–41.

[25] Ju T. Robust repair of polygonal models. ACM Transactions on Graphics (SIGGRAPH Proceedings) 2004; 23(3):888–895.

[26] Ju T. Fixing geometric errors on polygonal models: a survey. Journal of Computer Science and Technology, 2009; 24(1):19–29.

[27] Ju T, Zhou QY, Hu SM. Editing the topology of 3D models by sketching. ACM Transactions on Graphics (SIGGRAPH Proceedings) 2007; 26(3):42.

[28] Kriegeskorte N, Goebel R. An efficient algorithm for topologically correct segmentation of the cortical sheet in anatomical MR volumes. NeuroImage 2001; 14:329–346.

[29] Lai HC, Lai JY. A partial mesh replacement technique for design modification in rapid prototyping. Computers and Industrial Engineering 2009.

[30] Lee CN, Poston T, Rosenfeld A. Holes and genus of 2d and 3d digital images. Graphical Models and Image Processing 1993; 55(1):20–47.

[31] Li X, Han CY, Wee W. On surface reconstruction: a priority driven approach. Computer-Aided Design 2009;41(9):626–640.

[32] Liepa P. Filling holes in meshes. Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing 2003; p. 200–5.

[33] Lorensen W, Cline H. Marching cubes: a high resolution 3D surface construction algorithm. Computer Graphics (SIGGRAPH Proceedings) 1987; 21(4):163–170.

[34] Min P. [binvox] 3D mesh voxelizer. `http://www.cs.princeton.edu/~min/binvox/`. Accessed Sept. 3rd, 2009.

[35] Murali T, Funkhouser T. Consistent solid and boundary representations from arbitrary polygonal data. Proceedings of the Symposium on Interactive 3D Graphics 1997; p. 155–162.

[36] Nooruddin FS, Turk G. Simplification and repair of polygonal models using volumetric techniques. IEEE Transactions on Visualization and Computer Graphics 2003; 9(2):191–205.

[37] Pauly M, Mitra NJ, Wallner J, Pottmann H, Guibas L. Discovering structural regularity in 3D geometry. ACM Transactions on Graphics (SIGGRAPH Proceedings) 2008; 27(3).

[38] Podolak J, Rusinkiewicz S. Atomic volumes for mesh completion. Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing 2005; p. 33–42.

[39] Rocchini C, Cignoni P, Ganovelli F, Montani C, Pingi P, Scopigno R. The Marching Intersections algorithm for merging range images. The Visual Computer 2004; 20:149–164.

[40] Rossignac J, Cardoze D. Matchmaker: manifold B-Reps for non-manifold r-sets. Proceedings of the ACM Symposium on Solid Modeling and Applications 1999; p. 31–41.

[41] Sharf A, Alexa M, Cohen-Or D. Context-based surface completion. ACM Transactions on Graphics (SIGGRAPH Proceedings) 2004; 23(3):878–887.

[42] Verdera J, Caselles V, Bertalmío M, Sapiro G. Inpainting surface holes. Proceedings of the IEEE International Conference on Image Processing 2003; p. 903–6.

[43] Wood Z, Hoppe H, Desbrun M, Schröder P. Removing excess topology from isosurfaces. ACM Transactions on Graphics 2004; 23(2):190–208.

[44] Zaharescu A, Boyer E, Horaud R. TransforMesh: A topology-adaptive mesh-based approach to surface evolution. Proceedings of the Asian Conference on Computer Vision 2007; LNCS 4844:166–175.

[45] Zhou QY, Ju T, Hu SM. Topology repair of solid models using skeletons. IEEE Transactions on Visualization and Computer Graphics 2007; 13(4):675–685.

$\therefore$

A.2 AN ITERATIVE ALGORITHM FOR HOMOLOGY COMPUTATION ON SIMPLICIAL SHAPES

Dobrina Boltcheva, David Canino, Sara Merino Aceituno, Jean-Claude Léon, Leila De Floriani, Franck Hétroy

Computer-Aided Design 43 (11), Elsevier, 2011. Presented at the SIAM Conference on Geometric & Physical Modeling (GD/SPM) 2011.

An iterative algorithm for homology computation on simplicial shapes

Dobrina Boltcheva[a,*], David Canino[b], Sara Merino Aceituno[a], Jean-Claude Léon[a], Leila De Floriani[b], Franck Hétroy[a]

[a] *Grenoble University, Laboratoire Jean Kuntzmann, INRIA, 655, av. de l'Europe, 38334, Montbonnot, France*
[b] *University of Genova, Department of Computer Science, Via Dodecaneso, 35, 16146, Genova, Italy*

Abstract

We propose a new iterative algorithm for computing the homology of arbitrary shapes discretized through simplicial complexes, We demonstrate how the simplicial homology of a shape can be effectively expressed in terms of the homology of its sub-components. The proposed algorithm retrieves the complete homological information of an input shape including the Betti numbers, the torsion coefficients and the representative homology generators.

To the best of our knowledge, this is the first algorithm based on the *constructive* Mayer-Vietoris sequence, which relates the homology of a topological space to the homologies of its sub-spaces, i.e. the sub-components of the input shape and their intersections. We demonstrate the validity of our approach through a specific shape decomposition, based only on topological properties, which minimizes the size of the intersections between the sub-components and increases the efficiency of the algorithm.

Keywords: computational topology; simplicial complexes; shape decomposition; $\mathbb{Z}$-homology; Mayer-Vietoris sequence; generators

1. Introduction

Recently, the problem of computing the topological features of a shape has drawn much attention because of its applications in several disciplines, including shape analysis and understanding, shape retrieval, and finite element analysis [15, 20, 38]. Unlike geometric features (such as curvature) which are only invariant under rigid transformations, topological features are invariant under continuous deformations. Thus, they provide global quantitative and qualitative information about a shape, such as the number of its connected components, the number of holes and tunnels. Topological features are the core descriptors to extend geometric modelers with non-manifold shapes processing. For instance, the generation of simulation models still lacks capabilities for processing non-manifold shapes, like idealized representations [36]. Homological information on arbitrary shapes can strongly support new modeling capabilities, because constructive modeling techniques are often used. Also, topological features are especially important in high dimensional data analysis, where pure geometric tools are usually not sufficient.

The most common way to discretize a shape is through a simplicial complex. Simplicial complexes are easy to construct and manipulate, and compact data structures have been developed to encode them efficiently [7]. Simplicial homology is one of the most useful and algorithmically computable topological invariants. It characterizes a simplicial complex of dimension n through the notion of *homological descriptor*. Homological descriptors are defined in any dimension $k \leq n$ and are related to the non-trivial k-cycles in the complex which have intuitive geometrical interpretations up to dimension 2. In dimension 0, they are related to the connected components of the complex, in dimension 1, to the tunnels and the holes, and in dimension 2, to the shells surrounding voids or cavities.

Here, we propose a new algorithmic approach for homology computation on arbitrary shapes represented by finite simplicial complexes. Our framework is based on the *constructive homology theory* discussed in [32, 34, 33]. It provides a tool, the *constructive Mayer-Vietoris sequence*, which offers an elegant way for computing the homology of a simplicial complex from the homology of its sub-complexes and of their intersections. This leads to a *modular* algorithm for homology computation, that we call *Mayer-Vietoris (MV) algorithm*. Here, we show that our algorithm is more efficient than the classical one based on the reduction of the incidence matrices to a canonical form, known as the *Smith Normal Form (SNF)* [1, 30].

We demonstrate the validity of our approach through a decomposition of an n-dimensional simplicial complex, called the *Manifold-Connected (MC) decomposition* and proposed in [26] for 2-dimensional simplicial complexes. In our experiments, we demonstrate that the MC decomposition minimizes the size of the intersection between sub-complexes, and, thus, it is especially useful for the homol-

*Corresponding author
Email addresses: dobrina.boltcheva@inrialpes.fr (Dobrina Boltcheva), canino@disi.unige.it (David Canino), sm851@cam.ac.uk (Sara Merino Aceituno), jean-claude.leon@grenoble-inp.fr (Jean-Claude Léon), deflo@disi.unige.it (Leila De Floriani), franck.hetroy@grenoble-inp.fr (Franck Hétroy)

ogy computation in our constructive approach.

The remainder of the paper is organized as follows. In Section 2 we review some background notions on simplicial homology. In Section 3, we discuss related work on homology computation. In Section 4, we describe the MC decomposition, a graph-based data structure for encoding it and an algorithm for computing it. In Section 5, we introduce the main concepts from constructive homology, and, in Section 6, we describe the Homological Smith Reduction, the key tool for our MV algorithm. In Section 7, we provide a detailed description of the MV algorithm. In Section 8, we present experimental results based on our implementation of the homology computation algorithm. Finally, in Section 9, we draw some concluding remarks.

2. Background Notions on Simplicial Homology

Simplicial homology exploits the combinatorial structure of simplicial complexes and reformulates the homological problem into an algebraic one. In this section, we introduce some basic notions on simplicial homology needed throughout the paper. See [1, 30, 25] for more details.

Simplicial complexes. A simplex $\sigma = [v_0, \ldots, v_k]$ is the convex hull of a set V of affinely independent points in $\mathbb{R}^N$: here, k is the *dimension* of σ, which is called a $k-$ *simplex*. For every non-empty subset $T \subseteq V$, the simplex σ' spanned by T is called a *face* of σ, and σ' is a *proper face* of σ if T is a proper subset of V. A *simplicial complex* X is a collection of simplices such that all the faces of any simplex in X are also in X and the intersection of two simplices is either empty or a face of both. The largest dimension of any simplex in X is the *dimension* of X, denoted as $dim(X)$. A subset Y of a simplicial complex X is a *subcomplex* of X if Y is itself a simplicial complex. Each k-simplex of a simplicial complex X can be *oriented* by assigning a *linear ordering* to its vertices. The *boundary* of an oriented k-simplex is defined as the alternate sum of its incident (k-1)-simplices: $d_k([v_0, \ldots, v_k]) = \sum_{i=0}^{k}(-1)^i[v_0, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k]$. A fundamental property of the boundary operator is that the boundary of every boundary is null, $d_{k-1}d_k = 0$, for all $k > 0$.

Chain-complexes. Given an *oriented simplicial* complex X, simplicial homology builds an algebraic object $C_*(X)$, called *chain-complex*, on which the homological problem for X is resolved using linear algebra.

Let $n = dim(X)$. A *k-chain* is defined for each dimension $k \in [0, \ldots, n]$ as $a^k = \sum \lambda_i \sigma_i^k$, where $\lambda_i \in \mathbb{Z}$ are the coefficient assigned to each k-simplex σ_i^k. The kth *chain group*, denoted as $C_k(X)$, is formed by the set of k-chains together with the addition operation, defined by adding the coefficients simplex by simplex. There is a chain group for every integer k, but for a complex in $\mathbb{R}^n$, only the groups for $0 \leq k \leq n$ may not be trivial. These chain groups are Abelian and finitely generated, thus, the set of *oriented* k-simplices forms a basis for $C_k(X)$. In the following, we will refer to this basis as the *canonical* basis.

Each boundary operator d_k can be linearly extended to k-chains, $a^k = \sum \lambda_i \sigma_i^k$, as sum of the boundaries of the simplices in the chain: $d_k(a^k) = \sum \lambda_i d_k(\sigma_i^k)$. The *chain-complex*, denoted as $C_*(X) = (C_k, d_k)_{k \in \mathbb{N}}$, is the sequence of the chain groups $C_k(X)$ connected by the boundary operator d_k:

$$(C_*, d_*) : 0 \xleftarrow{0} C_0 \xleftarrow{d_1} C_1 \xleftarrow{d_2} \ldots \xleftarrow{d_{n-1}} C_{n-1} \xleftarrow{d_n} C_n \xleftarrow{0} 0$$

The chain-complex $C_*(X)$, associated with a simplicial complex of finite dimension n, can be encoded as a pair (B_k, D_k) for each $0 \leq k \leq n$, where $B_k = [\sigma_0^k, \ldots, \sigma_l^k]$ is the canonical basis of C_k and $D_k = [\eta_{j,i}^k]$ is an integer matrix, called the *incidence matrix*, which expresses the boundary operator with respect to B_{k-1} and B_k:

$$\eta_{j,i}^k = \begin{cases} 0 & \text{if } \sigma_j^{k-1} \text{ is not in the boundary of } \sigma_i^k; \\ 1 & \text{if } \sigma_j^{k-1} \text{ is in the boundary of } \sigma_i^k; \\ -1 & \text{if } -\sigma_j^{k-1} \text{ is in the boundary of } \sigma_i^k. \end{cases}$$

Homology groups. Given a chain-complex $C_*(X)$, homology groups are derived from two specific subgroups of the chain groups defined by the boundary operators:

$$Z_k = \ker d_k = \{c \in C_k \mid d_k(c) = 0\}$$

$$B_k = img\ d_{k+1} = \{c \in C_k \mid \exists a \in C_{k+1} : c = d_{k+1}(a)\}$$

We say that a k-cycle in Z_k *bounds* if it is also in B_k. Two cycles are *homologous* if they differ by a cycle that bounds. The collections Z_k of k-cycles and B_k of k-boundaries form subgroups in C_k: $B_k \subseteq Z_k \subseteq C_k$. For each $k \in [0, n]$, the kth *homology group* of X is defined as the quotient of the cycle group over the boundary group, i.e., $H_k = Z_k/B_k$. Thus, the elements of the homology group are equivalence classes of k-cycles which are not k-boundaries. Since $C_k(X)$ is an Abelian group, then it is isomorphic to:

$$H_k(X) = \overbrace{\mathbb{Z} \oplus \ldots \oplus \mathbb{Z}}^{\text{free group}} \oplus \overbrace{\mathbb{Z}/\lambda_1\mathbb{Z} \oplus \ldots \oplus \mathbb{Z}/\lambda_p\mathbb{Z}}^{\text{torsion group}}$$

The number of occurrences of $\mathbb{Z}$ in the free part is the number of elements of H_k with infinite order and is called the kth *Betti number*, β_k. It can also be seen as the maximal number of independent k-cycles that do not bound. The values $\lambda_1, \ldots, \lambda_p$ satisfy two conditions: (i) $\lambda_i \geq 2$ and (ii) λ_i divides λ_{i+1}, for each $i \in [1, p)$. They correspond to the *torsion coefficients*. A set of homologous k-cycles can be associated with each group $\mathbb{Z}/\lambda_i\mathbb{Z}$ of H_k. These k-cycles are not the boundary of any $(k+1)$-chain, but if taken λ_i times, they become the boundary of any $(k+1)$-chain. We will call these cycles *weak-boundaries*.

For all k, there exists a finite number of elements of H_k from which we can deduce all elements of H_k. Let H_k be a homology group generated by q independent equivalence classes $C_1 \ldots C_q$, then any set $\{g_1, \ldots, g_q | g_1 \in$

$C_1, \ldots, g_q \in C_q\}$ is called a set of *generators* for H_k. We can denote a homology group in terms of its generators as $H_k = [g_1, \ldots, g_q]$. We refer the complete homology information of a simplicial complex X (generators, Betti numbers and torsion generators) as the $\mathbb{Z}$-*homology* of X.

Mayer-Vietoris sequence. The Mayer-Vietoris sequence is an algebraic tool which allows us to study the homology of a space X by splitting it into two subspaces A and B such that $A \cap B \neq \emptyset$, for which the homology groups are easier to compute. This sequence relates the chain-complex of the union $(A \cup B)_*$ to the chain-complexes of the disjoint sum $A_* \oplus B_*$ and the intersection $(A \cap B)_*$:

$$0 \xleftarrow{0} (A \cup B)_* \xleftarrow{j} (A \oplus B)_* \xleftarrow{i} (A \cap B)_* \xleftarrow{0} 0$$

This sequence is *exact*, i.e., the kernel of each homomorphism is equal to the image of the previous one, $Img(i) = Ker(j)$. Therefore, we have the following relations: $(A \cap B)_* \cong Ker(j)$ and $(A \cup B)_* \cong (A \oplus B)_*/Img(i)$.

As demonstrated in [30], we can build a long exact sequence of their homology groups, starting from the short exact sequence of the chain-complexes:

$$\ldots \longleftarrow H_{k-1}((A \cap B)_*) \xleftarrow{\partial} H_k((A \cup B)_*) \xleftarrow{j} H_k((A \oplus B)_*) \xleftarrow{i} H_k((A \cap B)_*) \xleftarrow{\partial} H_{k+1}((A \cap B)_*) \longleftarrow \ldots$$

In some cases, the homology of the union can be deduced from this long exact sequence of homology groups, but it is not always possible to decide. This problem is known as the *extension problem*. Moreover, there is no way to give the generators of the homology group, because this method is *non-constructive* [34]. Thus, *classical* Mayer-Vietoris sequence is known as a purely theoretical tool and is useful only for computations by hand.

Smith Normal Form (SNF) algorithm. For each k such that $1 \leq k \leq n$, the SNF algorithm transforms the incidence matrix D_k into its Smith normal form N_k [30]:

$$N_k = \begin{array}{c} \\ s_0^{k-1} \\ \vdots \\ s_p^{k-1} \end{array} \begin{array}{c} \begin{array}{ccc} s_0^k & \ldots & s_l^k \end{array} \\ \begin{pmatrix} 0 & \boldsymbol{\lambda} & 0 \\ 0 & 0 & Id \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

where $\boldsymbol{\lambda}$ is a diagonal matrix with $\lambda_r, \ldots, \lambda_1 \in \mathbb{Z}$ in the diagonal such as $\lambda_i > 1$ and λ_i divides λ_{i+1}, $\forall i \in [1, r)$. Each incidence matrix D_k is initially expressed into the *canonical* basis β_c^{k-1} and β_c^k of the chain groups C_{k-1} and C_k. At the end of the algorithm, matrix N_k is expressed into different basis β_s^{k-1} and β_s^k, called the *Smith basis*.

The input incidence matrix can be expressed as $D_k = P_{k-1} N_k P_k^{-1}$, where matrix P_k encodes the basis changes $P_k : C_k[\beta_s] \rightarrow C_k[\beta_c]$. Initially, $P_k = Id$, but during the transformation every elementary operation on the rows and the columns of the boundary matrix D_k is translated into an operation on matrix P_k which encodes the change of the basis. Thus, P_k tell us how to express an element of the *Smith basis* in terms of the *canonical basis* of C_k.

The homology is computed using two consecutive incidence matrices in Smith Normal forms, denoted as N_k, and N_{k+1}. The rank of the sub-group $Z_k = \ker N_k$ is equal to the number of zero-columns of N_k, which correspond to the k-cycles. The rank of $B_k = \text{img } N_{k+1}$ is equal to the number of non-zero rows of N_{k+1}. The *generators*, expressed in the canonical basis, are obtained by computing the image of each generator γ_i from the *Smith basis* through the matrix P_k.

3. Related Work

The classical approach to compute the $\mathbb{Z}$-homology of a simplicial complex of finite dimension is based on the Smith Normal Form (SNF) algorithm [1, 30]. Although this method is theoretically valid in any dimension and for any kind of simplicial complex, it has some inherent limitations due to the size of the incidence matrices and to the high complexity of the reduction algorithm. The best available reduction algorithms have super-cubical complexity [35, 14] and they are suitable only for small simplicial complexes. Another well-known problem is the appearance of huge integers during the reduction [24].

In the literature several optimizations of the SNF algorithm have been developed. *Stochastic methods* [21] are efficient on sparse integer matrices, but they do not provide the homological generators. *Deterministic methods* [28, 35] perform the computations modulo an integer chosen by a determined criterion, but the information about the torsion coefficients is lost with this strategy. Another way to improve the computation time is to *reduce* the input complex without changing its topology by applying iterative simplifications, and by computing the homology when no more simplifications are possible. This *reduction approach* has been mainly investigated in the context of homology computation from 3D voxel images [27, 6, 29, 31]. Other reduction approaches apply discrete Morse theory [18] to homology computation since in many applied situations one expects the Morse complex built on the original simplicial complex to be much smaller than this latter.

Another approach for homology computation is based on *persistent homology* [15]. In this framework, the input simplicial complex is filtered (according to any real function) in order to study which homological attributes appear, disappear and are maintained through nesting. The pertinent information is encapsulated by a pairing of the critical values of the function which are visualized by points forming a diagram in the plane. Since the filtration is done by adding only one simplex at a time, it can be interpreted as a special case of the *Mayer-Vietoris sequence*. These methods are usually designed for simplicial complexes with dimensional restrictions in most cases. The original algorithm in [16] computes the pairs from an ordering of the simplices in a triangle mesh and exhibits a cubic worst-case time in the size of the complex. In [5], an algorithm that maintains the pairing in worst-case linear time per transposition in the ordering has been

presented. A nearly linear algorithm for computing only the Betti numbers (the ranks of the homology groups) for simplicial 3-complexes is proposed in [9]. In [10], an algorithm is proposed for computing the homological generators of manifold simplicial complexes, embedded in the 3D Euclidean space, which is then extended to arbitrary simplicial complexes through a thickening process. The algorithm presented in [11] extracts two types of possible 1-cycles which identify handles and tunnels on 2-manifold surfaces. In [23], another method has been proposed for computing the non-contractible 1-cycles on smooth compact 2-manifolds. The shape of the computed generators has been addressed in [39, 4]. However, the persistence of a feature depends highly on the chosen filtering function and it is still an open problem to find geometrically meaningful functions on non-manifold simplicial complexes. In [17], a first attempt for a Mayer-Vietoris formula for persistent homology has been proposed with an application to shape recognition in the presence of occlusions. However, this work is based on the classical version of the Mayer-Vietoris sequence, and the proposed formula cannot be used in practice since it does not lead to an algorithm.

Finally, there exist also a few methods based on *constructive homology*, introduced in [33], which provides an original *algorithmic* approach for computing homology. Concepts borrowed from constructive homology have been used in [2, 22] for homology computation on images. To the best of our knowledge, none of the existing algorithms uses the *constructive* Mayer-Vietoris sequence which provides an effective strategy for computing the homology generators of an arbitrary simplicial complex from the homology of its sub-complexes.

4. The Manifold-Connected Decomposition

In this section, we describe a decomposition of a simplicial complex, called the *Manifold-Connected (MC)* decomposition, which we use as the basis for performing homology computation. The MC decomposition has been introduced in [26] for two-dimensional simplicial complexes, but it can be defined in arbitrary dimensions.

Let us consider a d-dimensional regular simplicial complex X. Recall that a *regular (or dimensionally homogeneous)* simplicial complex is a complex in which all top simplices are d-dimensional, where a *top simplex* is a simplex which does not belong to the boundary of any other simplex in X. We introduce some definitions and concepts needed for the definition of manifold-connected complex and component.

A $(d-1)$-simplex τ in a regular simplicial d-complex X is said to be a *manifold* simplex if and only if there exists at most two d-simplices in X incident in τ. Two d-simplices σ and σ' in X are said to be *manifold-connected* if and only if there exists a path P joining σ and σ' formed by d-simplices such that any two consecutive d-simplices in P are adjacent through a manifold $(d-1)$-simplex. A regular d-complex in which every pair of n-simplices are manifold-connected is a *manifold-connected* complex.

Any combinatorial manifold, i.e., any simplicial complex with a manifold domain, is clearly manifold-connected, but the reverse is not true. Thus, the class of manifold-connected complexes is a decidable superset of the class of combinatorial manifolds. Note that the class of d-manifolds is not decidable for $d \geq 6$ [8].

The manifold-connectivity relation between the top d-simplices in a regular d-complex X defines an equivalence relation. The *manifold-connected* components of X are the equivalence classes of the top d-simplices of X with respect to the manifold-connectivity relation. The collection of all manifold connected components in X forms the *Manifold-Connected (MC)* decomposition. Any two or more components in the MC decomposition of a simplicial d-complex X may have a common intersection which is a sub-complex of X of dimension lower than d.

Any arbitrary (non-regular) simplicial n-complex Y is uniquely decomposed into a collection of maximal regular complexes Y_d formed by top simplices of the same dimension $d \leq n$. Figure 1 shows an example of the decomposition of an arbitrary simplicial 3-complex into maximal regular sub-complexes Y_2 and Y_3.

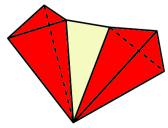


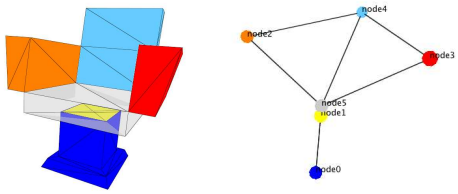
Figure 1: Decomposition of a 3-complex into maximal regular sub-complexes Y_2 (in yellow) and Y_3 (in red).

Thus, the MC decomposition of an n-complex Y is the collection of the MC decompositions of the maximal regular sub-complexes of Y. As a consequence, the MC decomposition of a complex Y is unique. Figure 2(a) shows an example of the MC decomposition of a regular simplicial 2-complex. The six MC-components of dimension 2 are connected through chains of non-manifold edges.

4.1. Encoding the MC decomposition

We have developed a representation for the MC decomposition suitable for homology computation. For this purpose, we need to efficiently access the intersection of pairs of MC-components, and to have a unique vertex ordering for all MC-components to be able compute the chain-complexes. Thus, we propose a graph-based data structure which encodes the MC decomposition as a graph, called the *Homology MC-graph (Homo-MC graph)*, denoted as $\mathcal{H} = (\mathcal{N}, \mathcal{A})$. A node in $\mathcal{N}$ corresponds to an MC-component, while an arc in $\mathcal{A}$ describes the intersection of *two* MC-components. Figure 2(b) shows the Homo-MC graph describing the MC decomposition in Figure 2(a).

The data structure based on the Homo MC-graph consists of two layers: the top layer is the encoding of the Homo MC-graph, while the second layer describes the simplicial complex Y and is currently implemented through the *Incidence Simplicial (IS)* data structure [7]. The IS data structure encodes all simplices in Y and allows a di-



(a) (b)

Figure 2: An example of a non-manifold shape (a) and a graphical representation of its Homo-MC graph. Each node of the Homo-MC graph is identified by a color and it is graphically represented by its center of gravity.

rect access to each simplex of Y in constant time. Moreover, it encodes the relations among a k-simplex and its bounding simplices of dimension $k-1$ and among a k-simplex and the simplexes of dimension $k+1$ in its co-boundary. Each node C in the Homo-MC graph has a list of references to the top simplices of the corresponding complex in C. In this way, we ensure that all the MC-components have the same vertex ordering provided by Y. Similarly, each arc a of the Homo-MC graph contains a reference for each simplex belonging to the intersection described by a.

4.2. Building the Homology MC-Graph

The computation of the Homo-MC graph for an arbitrary simplicial n-complex Y consists of two steps: first, the MC-components are identified, and then the arcs of the Homo-MC graph are computed.

The detection of the MC-components is performed according to the following steps:

1. retrieve the maximal regular sub-complexes Y_k (with $k \leq n$) of Y formed by the top k-simplices in Y which are adjacent along $(k-1)$-simplices;
2. for each sub-complex Y_k, perform a traversal starting from an unvisited top k-simplex σ and retrieve all top k-simplices which are reachable from σ by visiting manifold $(k-1)$-simplices and their incident top k-simplices. All top k-simplices visited by starting from σ belong to the same k-dimensional MC-component, identified by an integer label C. Mark with C all sub-simplices of each top simplex in the MC-component;
3. create a node of the Homo-MC Graph for each MC-component.

At the end, each simplex σ in Y is marked with a list of integer labels $l_\sigma = \{C_1, \ldots, C_s\}$, which denote the MC-components containing σ. A simplex $\bar{\sigma}$ marked with several labels is a *singularity*. The arcs of the Homo-MC graph are then retrieved as follows:

1. for each non-manifold singularity $\bar{\sigma}$, generate all pairs of integer labels (C_i, C_j) in $l_{\bar{\sigma}}$ and store the tuples $(\bar{\sigma}, C_i, C_j)$ in an array A;
2. sort the tuples in A by using the lexicographic order of labels: tuples related to the same pair of labels are stored in consecutive locations in A;
3. generate an arc of the Homo-MC Graph for each unique pair of nodes identified at step 2.

5. Constructive Homology

Constructive Homology theory has been developed in order to solve the *non-constructiveness* of classical homology from its roots [32]. Within this framework, based on *constructive mathematics* [37], the homological concepts are reformulated into concepts with a computational nature, thus yielding to effective implementable algorithms. The theory has been developed to handle homology computations over chain-groups of infinite dimension [33, 34] and its validity has been proven using functional programming [13]. The Mayer Vietoris algorithm we present here is an application of constructive homology. In this section, we introduce the main concepts and theorems which are used in our MV algorithm. For more details, see also [3].

Our method is based on two key concepts: the *reduction*, which is a relation between two chain-complexes with equivalent homologies, and the *cone of a morphism*, which is a particular way to represent the morphism relating two chain-complexes as a new chain-complex. We use also two main constructive theorems: the *Short Exact Sequence (SES) theorem*, which provides the *constructive* version of the Mayer-Vietoris sequence, and the *Cone Reduction theorem*, which provides an efficient way to access the homology of the union of two simplicial complexes only from their *reduced* chain-complexes.

The reduction relates two chain-complexes with equivalent homologies in such a way that, if the homology of one of them is known, then the homology of the other can be found through reduction. Intuitively, it relates a *large* chain-complex $\widehat{C}_*$ to a *small* one C_*, which contains the same homological information in the most compact way.

$$\rho = \begin{tikzcd} \widehat{C}_* \arrow[loop right, "h"] \arrow[d, shift left, "f"] \\ C_* \arrow[u, shift left, "g"] \end{tikzcd}$$

Figure 3: A reduction.

Definition 5.1 (Reduction). *A reduction $\rho : \widehat{C}_* \Rrightarrow C_*$ is a diagram as shown in Figure 3, where $\widehat{C}_*$ and C_* are two chain-complexes; f and g are chain-complex morphisms satisfying $fg = id_{C_*}$; $h : \widehat{C}_{n-1} \to \widehat{C}_n$ is an homotopy operator satisfying the relations $gf + dh + hd = id_{\widehat{C}_*}$ and $fh = hg = hh = 0$.*

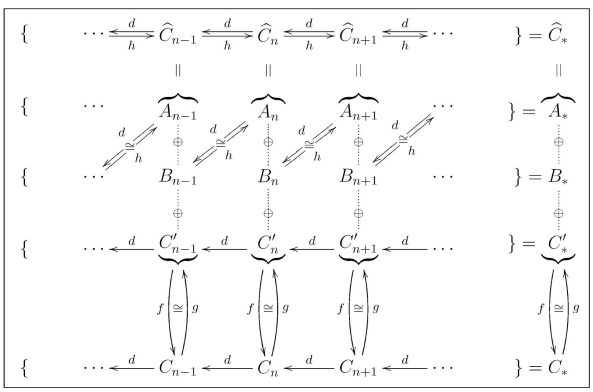


Figure 4: Reduction diagram (courtesy of [34]).

The reduction is a compact and convenient form for the diagram presented in Figure 4. It is equivalent to a decomposition, where every chain group $\widehat{C}_k$ is decomposed into three components: $\widehat{C}_k = A_k \oplus B_k \oplus C'_k$. Note that there exists a bijection between A_{k+1} and B_k through the boundary operator d and the homotopy operator h for every finite dimension k. Therefore, component A_{k+1} is a collection of $(k+1)$-cycles such that their boundaries are the elements in B_{k-1}. We call these cycles *pre-boundaries*. Component B_k is a collection of k-cycles known as k-boundaries. Component C'_k is a copy of C_k and, thus, $C'_* \cong \widehat{C}_*$. In summary, the large chain-complex $\widehat{C}_*$ is the direct sum of one small chain-complex C'_* and $A_* \oplus B_*$, where the last component does not play any role from a homological point of view. Given a chain-complex C_*, we call *trivial reduction*, the reduction where the small chain-complex is C_* itself, the morphism f and g are the identity morphisms, and the homotopy operators h are 0 morphisms.

Definition 5.2 (**Reduction equivalence**). *A reduction equivalence $C_* \Longleftrightarrow D_*$ between two chain-complexes C_*, D_*, is a pair of reductions connecting C_* and D_* through a third chain-complex $\widehat{C}_*$, as shown in Figure 5.*

The concept of *reduction equivalence* is used to relate three chain-complexes: the object of interest C_*, whose homology has to be computed, a very small homologically equivalent object D_* and a large object $\widehat{C}_*$, also equivalent. The homology information of object C_* is contained in the very small object D_*, while the big object $\widehat{C}_*$ is required to link C_* and D_*. Such an equivalence implies that the homology groups of the D_* and C_* are isomorphic.

$$\epsilon = \begin{tikzcd} & \widehat{C}_* \arrow[loop above, "lh"] \arrow[loop above, "rh"] \arrow[dl, shift left, "lg"'] \arrow[dr, shift left, "rf"] \\ C_* \arrow[ur, shift left, "lf"] & & D_* \arrow[ul, shift left, "rg"] \end{tikzcd}$$

Figure 5: A reduction equivalence.

A cone of a morphism can simply be seen as a way to represent a morphism relating two chain-complexes as a new chain-complex. Informally, such a representation makes it possible to build an homologically equivalent object from the morphisms used to relate them.

Definition 5.3 (**Cone of a morphism**). *Let $f : X_* \to Y_*$ be a chain-complex morphism between two chain-complexes X_* and Y_*. The cone of the morphism f is a chain-complex, denoted $Cone(f)_*$. For each dimension $Cone(f)_k := Y_k \oplus X_{k-1}$ and the boundary operator is given by the matrix:*

$$D_{Cone(f)_k} := \begin{bmatrix} D_{Y_k} & f_{k-1} \\ 0 & -D_{X_{k-1}} \end{bmatrix}.$$

The matrices D_{Y_k} and $D_{X_{k-1}}$ are the incidence matrices of the chain-complexes Y_* and X_*. The groups Y_k and X_{k-1} are considered as disjoint and a basis of $Cone(f)_k$ is formed by a basis of Y_k and a basis of X_{k-1}.

Definition 5.4 (**Constructive exact short sequence of Mayer-Vietoris**). *Let A, B be two simplicial complexes with non-empty intersection $A \cap B$, then the following diagram defines a short exact sequence of their chain-complexes.*

$$0 \longleftarrow (A \cup B)_* \underset{j_A \ominus j_B}{\overset{\nu}{\rightleftarrows}} A_* \oplus B_* \underset{i_A \oplus i_B}{\overset{\rho}{\rightleftarrows}} (A \cap B)_* \longleftarrow 0$$

$$\begin{array}{rccc} i = i_A \oplus i_B : & (A \cap B)_* & \longrightarrow & A_* \oplus B_* \\ & \sigma & \longmapsto & (\sigma, \sigma) \\ j = j_A \ominus j_B : & A_* \oplus B_* & \longrightarrow & (A \cup B)_* \\ & (\sigma, \tilde{\sigma}) & \longmapsto & \sigma - \tilde{\sigma} \\ \nu : & (A \cup B)_* & \longrightarrow & A_* \oplus B_* \\ & \sigma & \longmapsto & (\sigma|_A, -\sigma|_B + \sigma|_{A \cap B}) \\ \rho : & A_* \oplus B_* & \longrightarrow & (A \cap B)_* \\ & (\sigma, \tilde{\sigma}) & \longmapsto & \tilde{\sigma}|_{A \cap B} \end{array}$$

The following theorem is the basic result on which our algorithm is based. It allows us to establish an homological equivalence between the cone of the morphism inclusion $i : (A \cap B)_* \to A_* \oplus B_*$ and the chain-complex of the union $(A \cup B)_*$.

Theorem 5.5 (**Short Exact Sequence (SES) theorem**). *The constructive short exact sequence of Mayer-Vietoris provides the reduction shown in Figure 6.*

$$\begin{tikzcd} Cone(i)_* \arrow[loop above, "\rho"] \arrow[d, shift left, "j"] \\ (A \cup B)_* \arrow[u, shift left, "\nu - \rho \; d_*^{A \oplus B} \; \nu"] \end{tikzcd}$$

Figure 6: SES reduction.

Thus, if we know the homology of the cone of the morphism inclusion i, then we can retrieve the homology of $(A \cup B)_*$ by computation of the image of each element of the homology of $Cone(i)_*$ by morphism $f = j_A \ominus j_B$. However, chain-complex $Cone(i)_*$ is much larger than the chain-complex of the union $(A \cup B)_*$ and it would be extremely inefficient to compute the homology directly on this huge object.

The *Cone Reduction Theorem* gives us another reduction of chain-complex $Cone(i)_*$ and allows us to build a *reduction equivalence* between the chain-complex of the union $(A \cup B)_*$ and one very small chain-complex which is homologically equivalent to $Cone(i)_*$.

Theorem 5.6 (**Cone Reduction Theorem**). *Let $i : (A \cap B)_* \to (A \oplus B)_*$ be a chain-complex morphism and two reductions $(A \oplus B)_* \Rrightarrow EA_* \oplus EB_*$ and $(A \cap B)_* \Rrightarrow E(A \cap B)_*$. Then, we can define a reduction $\rho = (f_c, g_c, h_c) : Cone(i)_* \Rrightarrow Cone(Ei)_*$, as shown in Figure 7, where:*

$$f_c = \begin{bmatrix} f_{A \oplus B} & (-f_{A \oplus B})(i)(h_{A \cap B}) \\ 0 & f_{A \cap B} \end{bmatrix}$$

$$g_c = \begin{bmatrix} g_{A \oplus B} & (-h_{A \oplus B})(i)(g_{A \cap B}) \\ 0 & g_{A \cap B} \end{bmatrix}$$

$$h_c = \begin{bmatrix} h_{A \oplus B} & (h_{A \oplus B})(i)(h_{A \cap B}) \\ 0 & -h_{A \cap B} \end{bmatrix}.$$

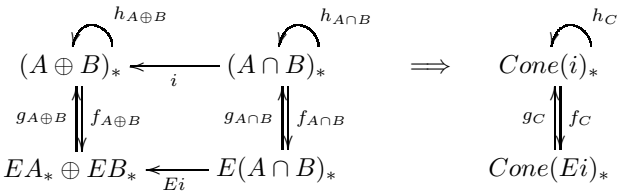


Figure 7: Cone Reduction theorem.

Note that the reduction $(A \oplus B)_* \Rrightarrow EA_* \oplus EB_*$ is simply defined as the formal sum of the reductions of the sub-complexes A and B.

By definition, chain-complex $Cone(Ei)_*$ is given by: $Cone((f_{A\oplus B})\circ(i)\circ(g_{A\cap B}))_* := EA_*\oplus EB_*\oplus E(A\cap B)_{*-1}$, where the chain-complexes EA_*, EB_* and $E(A\cap B)_*$ are the *reduced* chain-complexes of, respectively, A_*, B_* and $(A\cap B)_*$, and contain only their homological information. Therefore, we can efficiently compute the homology on this small chain-complex, by using the SNF algorithm.

As a consequence, we obtain the reduction equivalence shown in Figure 8, which demonstrates that the chain-complex $(A \cup B)_*$ has the same homology as the chain-complex $Cone(Ei)_*$. Therefore, the Betti numbers and the torsion coefficients of the union $(A \cup B)_*$ are provided directly from the homology of the chain-complex $Cone(Ei)_*$. The homological generators of $H_k((A \cup B)_*)$ can be obtained by computing the image of each cycle $c \in H_k(Cone(Ei)_*)$ by the morphisms $(j \circ g_c)(c)$.

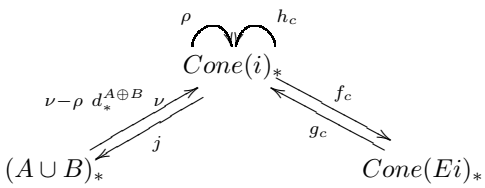


Figure 8: Cone reduction equivalence

Finally, we need to introduce the *Cone Equivalence theorem*, which will be useful for the MV algorithm, described in Section 7.

Theorem 5.7 (**Cone Equivalence Theorem**). *Let $i : (A \cap B)_* \to (A \oplus B)_*$ be a chain-complex morphism between two chain-complexes and two reduction equivalences, as shown in Figure 9. Thus, we can define the reduction equivalence:*

$$Cone(i) \overset{\rho_l}{\Lleftarrow} Cone(\widehat{i}) \overset{\rho_r}{\Rrightarrow} Cone(Ei)$$

with $\widehat{i} = (lg')\ i\ (lf)$ *and* $Ei = (rf')\ (lg')\ i\ (lf)\ (rg)$

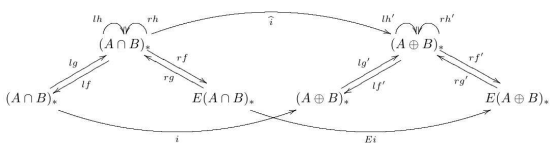


Figure 9: Cone equivalence theorem.

6. Homological Smith Reduction

In this section, we introduce a specific kind of reduction, which we call the *Homological Smith Reduction*. It will be used to encode the homology of each sub-complex of the input complex in our Mayer-Vietoris algorithm.

Given a simplicial complex X of finite dimension n, this reduction relates its chain-complex, X_*, with a very small chain-complex, EX_*, which contains only the homological information of X_*. This information is computed through the SNF algorithm, which transforms each incidence matrix D_k into its Smith Normal Form N_k. To describe chain-complex EX_*, we need a basis for each dimension and a boundary matrix. The basis is defined as a subset of Smith basis of X_*, while the boundary matrix is a sub-matrix of matrix N_k. The morphisms f, g and h relating the chain-complexes are defined from the matrices of the basis changes P_k, which is also restricted. Thus, we need first to classify the elements of the *Smith basis* provided by the SNF algorithm in order to find the basis of the small chain-complex EX_*.

Basis classification. Here, we illustrate the basis classification algorithm through the example shown in Figure 10. Let N_k and N_{k+1} be two consecutive incidence matrices in Smith Normal Form. We assume that $\beta_s^k = \{\gamma_1, \ldots, \gamma_l\}$ is the Smith basis, in which the columns of N_k and the rows of N_{k+1} are expressed.

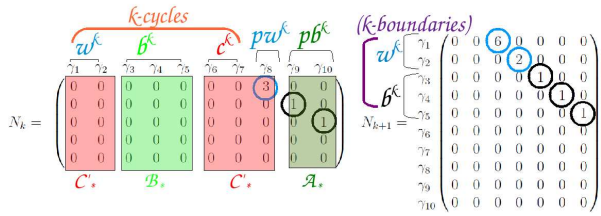


Figure 10: Smith basis classification $\beta_s^k = \{w^k, b^k, c^k, pw^k, pb^k\}$.

Consider now the sub-basis of the k-cycles $\ker d_k = [\gamma_1, \ldots, \gamma_7]$, which corresponds to the zero columns of N_k. This basis is formed by the union of three sub-basis, defined as follows. First, the sub-basis $\boldsymbol{w^k} = \{\gamma_1, \gamma_2\}$ is composed of the elements corresponding to the *weak-boundaries* which are associated with the torsion coefficients. They correspond to the rows of N_{k+1} with coefficient $\lambda_i > 1$. Second, the sub-basis $\boldsymbol{b^k} = \{\gamma_3, \gamma_4, \gamma_5\}$ is composed of the elements corresponding to the *boundaries* and can be retrieved through the rows of N_{k+1} with coefficient equal to 1. Finally, the remaining kernel basis corresponds to the non-trivial k-cycles $\boldsymbol{c^k} = \{\gamma_6, \gamma_7\}$.

We complete the basis classification with the k-chains which are not k-cycles. The elements corresponding to the columns of N_k with coefficients equal to 1 are called *pre-boundaries*, $\boldsymbol{pb^k} = \{\gamma_9, \gamma_{10}\}$. These chains do not carry any homological information. The elements corresponding to the columns of N_k with coefficients $\lambda_i > 1$ are called *pre-weak boundaries*, $\boldsymbol{pw^k} = \{\gamma_8\}$ and they are related to the torsion coefficients.

The basis classification for vertices and for simplices of dimension n must be treated as special cases, since the boundary morphisms at dimension 0 and $n+1$ are zero morphisms. Therefore, in the basis of dimension n there

are only cycles, pre-boundaries, and possibly pre-weak-boundaries but not weak-boundaries nor boundaries. In the vertex basis, there are only cycles and boundaries.

Reduced chain-complexes. The basis classification allows us to construct the *reduced* chain-complex EX_* from the original one X_*. Notice that the basis classification is equivalent to the decomposition of each original chain-group X_k into three sub-groups: $X_k = A_k \oplus B_k \oplus C'_k$, as shown in Figure 10.

Component $A_k = [\boldsymbol{pb}^k]$ is generated by the k-chains which do not play any role in homology computation. The chain-group $B_k = [\boldsymbol{b}^k]$ is generated by the k-cycles which are known to be boundaries. Note that, the subgroup generated by the pre-boundaries $[\boldsymbol{pb}^k]$ is isomorphic to the subgroup generated by the boundaries $[\boldsymbol{b}^{k-1}]$, since the identity sub-matrix relates them. In summary, the homology of X_k is given by the *reduced* chain-complex $EX_k = C'_k = [\boldsymbol{w}^k, \boldsymbol{c}^k, \boldsymbol{pw}^k]$. For each dimension $1 \leq k \leq n$, the boundary matrix EN_k of EX_k is:

$$EN_k := \begin{array}{c} \\ \boldsymbol{w}^{k-1} \\ \boldsymbol{c}^{k-1} \\ \boldsymbol{pw}^{k-1} \end{array} \begin{array}{c} \begin{array}{ccc} \boldsymbol{w}^k & \boldsymbol{c}^k & \boldsymbol{pw}^k \end{array} \\ \begin{pmatrix} 0 & 0 & \boldsymbol{\lambda} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array}.$$

It is immediate to prove that $EN_{k-1}EN_k = 0 \quad \forall k$, so EC_* is effectively a chain-complex.

Definition 6.1 (Homological Smith Reduction)**.** *Let X_* be a chain-complex X_*, then its Homological Smith Reduction is*

$$\rho = \begin{array}{c} X_* \circlearrowleft h \\ g \Big\uparrow\Big\downarrow f \\ EX_* = [\boldsymbol{w}^*, \boldsymbol{c}^*, \boldsymbol{pw}^*] \end{array}$$

with:
$f : X_* \to EX_*, f_k = (P_k)^{-1}|_{\boldsymbol{w}^k, \boldsymbol{c}^k, \boldsymbol{pw}^k}\}$
$g : EC_* \to X_*, g_k = P_k|_{\boldsymbol{w}^k, \boldsymbol{c}^k, \boldsymbol{pw}^k}\}$
$h : X_* \to X_{*+1}, h_k = (P_k)|_{pb^k} * (P_{k-1})^{-1}|_{b^{k-1}}$

The chain-complex morphisms f and g are inverse isomorphisms between EX_* and a subchain of X_* that contains all the homological information of X_*.

The restriction of the homotopy operator $h_k : B_k \to A_{k+1}$ and the restriction of the boundary operator $d_k : A_{k+1} \to B_k$ are isomorphisms between boundaries and pre-boundaries. This means that, given a boundary $\sigma^k \in B_k$, h_k gives us the $(k+1)$-chain of A_{k+1} for which σ is the boundary. Intuitively, the homotopy operator h captures only the information about the boundaries and their pre-boundaries. It can be seen as the *constructive* version of the definition of boundary. The algorithm for computing the *Homological Smith Reduction* of a chain-complex X_* is summarized in Algorithm 1.

7. The Mayer-Vietoris Algorithm

In this section, we first introduce the algorithm based on the constructive Mayer-Vietoris sequence, which computes the homology of the union of two simplicial complexes. Then, we explain how this algorithm can be applied on a Homo-MC graph thus resulting in the iterative Mayer-Vietoris homology computation algorithm.

Algorithm 1 Building the Homological Smith Reduction

Input: A chain-complex X_*.
Output: The reduction $X_* \Rrightarrow EX_*$.
1: **For all** $1 \leq k \leq dim(X)$ **do**:
2: Compute the SNF of the incidence matrix $D_k = P_{k-1} N_k P_k^{-1}$.
3: Classify the Smith basis : $[w^k, b^k, c^k, pw^k, pb^k]$
4: Build the reduction by cutting the matrices:
$EX_k := N_k|_{w^k, c^k, pw^k}$
$f_k := (P_k)^{-1}|_{w^k, c^k, pw^k}$
$g_k := (P_k)|_{w^k, c^k, pw^k}$
$h_k := (P_k)|_{pb^k} * (P_{k-1})^{-1}|_{b^{k-1}}$
5: **End for**

7.1. Homology computation of the union of two complexes

Here, we explain how the constructive version of the Mayer-Vietoris sequence, introduced in Section 5, yields to an effective algorithm for homology computation. We illustrate this algorithm through the example in Figure 11, where a simplicial complex X is decomposed onto two MC-components A and B with non-empty intersection $A \cap B$, formed by two isolated vertices.

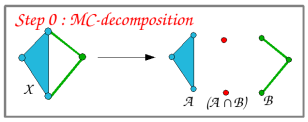


Figure 11: The MC-decomposition of a simplicial complex X into two MC-components A and B.

The first step of the algorithm consists in computing the Homological Smith Reductions of the three input complexes A, B and $A \cap B$, as explained in Section 6. We obtain reductions $A_* \Rrightarrow EA_*$, $B_* \Rrightarrow EB_*$ and $(A \cap B)_* \Rrightarrow E(A \cap B)_*$, as shown in Figure 12. Recall that the *reduced* chain-complexes EA_*, EB_* and $E(A \cap B)_*$ are homology equivalent to the large chain-complexes A_*, B_* and $(A \cap B)_*$ but contain *only* the homological information.

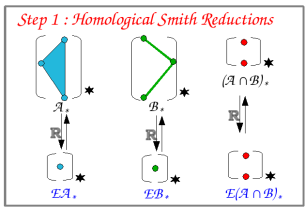


Figure 12: Homological Smith Reductions of A_*, B_* and $(A \cap B)_*$, associated to the example in Figure 11.

The second step of the algorithm builds the *constructive* Mayer-Vietoris sequence, as shown in Figure 13(a), and computes morphisms i, j, ρ and ν from the input simplicial complexes, following definition 5.4.

At this point, we can apply the SES theorem (theorem 5.5) which builds a reduction between the cone of

the inclusion morphism i and the chain-complex associated with the union of the sub-complexes A and B: $Cone(i)_* \Rrightarrow (A \cup B)_*$, as illustrated in Figure 13(b). This means that the chain-complex of the cone has the same homology as the chain-complex of the union. However, as indicated by the direction of the reduction, the size of the former is larger (in terms of number of simplices) than that of the latter, and computing the homology of the union through this large chain-complex would be extremely inefficient.

The fourth step of the algorithm applies the cone reduction theorem (theorem 5.6) in order to build a new reduction of the chain-complex of cone $Cone(i)_* \Rrightarrow Cone(Ei)_*$. This reduction establishes a homological equivalence between the large chain-complex $Cone(i)_*$ and the chain-complex associated with the cone of the inclusion morphism Ei, which relates the *reduced* chain-complexes $EA_* \oplus EB_*$ and $E(A \cap B)_*$, as shown in Figure 13(c). Note that chain-complex $Cone(Ei)_*$ can be efficiently computed from the reduced chains $EA_* \oplus EB_*$ and $E(A \cap B)_*$, following definition 5.3.

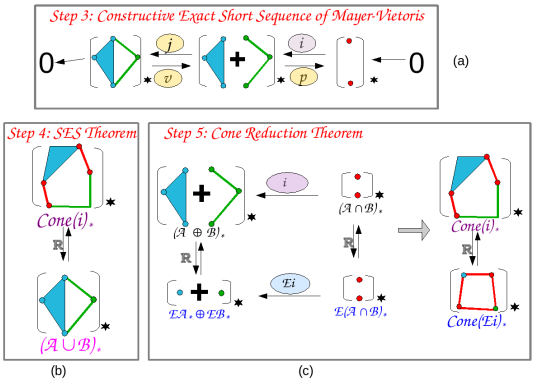


Figure 13: Main steps of our algorithm, applied to the example in Figure 11. We use (a) the Constructive Mayer-Vietoris sequence, (b) the SES theorem, and (c) the Cone Reduction theorem.

At this point, we establish the *reduction equivalence* $(A \cup B)_* \Longleftrightarrow Cone(Ei)_*$ from the two last reductions. This means that chain-complex $Cone(Ei)_*$ has the same homology as the chain-complex of the union $(A \cup B)_*$, since they are related through the large chain-complex $Cone(i)_*$, as shown in Figure 14(a). However, chain-complex $Cone(Ei)_*$ is much smaller (in terms of number of simplices) than the chain-complex of the union, since it contains only the homological information of sub-complexes A, B and $A \cap B$.

The next step of the algorithm computes the homology of the small chain-complex $Cone(Ei)_*$ through the SNF algorithm, obtaining the Homological Smith Reduction $Cone(Ei)_* \Rrightarrow ECone(Ei)_*$, as shown in Figure 14(b).

Finally, the algorithm composes the last two reductions. It outputs a *reduction equivalence* between the chain-complex of the union $(A \cup B)_*$ and chain-complex $ECone(Ei)_*$, as shown in Figure 14(c). From the reduction equivalence we can extract the required homological information. The Betti numbers and the torsion coefficients of the union can be directly accessed in $ECone(Ei)_*$, while the generators of the union are obtained by computing the image of the generators of $ECone(Ei)_*$ through the morphisms of the reduction equivalence.

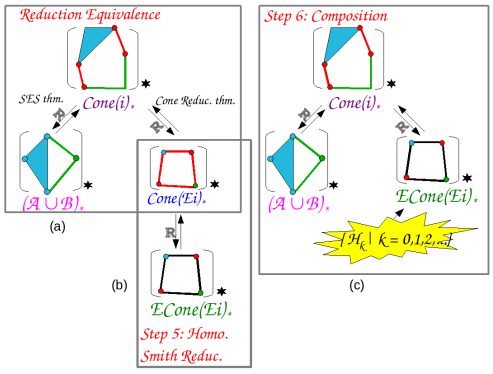


Figure 14: Last steps of our algorithm for computing homologies of the union of two complexes A and B.

The main steps of our algorithm for computing the homology of the union of two complexes are summarized in Algorithm 2.

Algorithm 2 Homology of the union of two complexes

Input: three simplicial complexes A, B and $A \cap B \neq \emptyset$.
Output: the reduction equivalence:
$(A \cup B)_* \Lleftarrow Cone(i) \Rrightarrow ECone(Ei)_*$

1: Compute the morphisms i, j, ρ and ν of the Constructive Mayer-Vietoris sequence for the chain-complexes $(A \oplus B)_*$, $(A \cap B)_*$ and $(A \cup B)_*$.
2: Build the reduction of the morphism inclusion i, provided by the SES Theorem: $Cone(i)_* \Rrightarrow (A \cup B)_*$.
3: Build the reduction of the morphism inclusion i, provided by the Cone Reduction Theorem: $Cone(i)_* \Rrightarrow Cone(Ei)_*$.
4: Compute the Homological Smith Reduction of the reduced chain-complex: $Cone(Ei)_* \Rrightarrow E(Cone(Ei))_*$.
5: Compose the last two reductions: $Cone(i)_* \Rrightarrow E(Cone(Ei))_*$

7.2. Mayer-Vietoris Iterative Algorithm

In this section, we introduce our Mayer-Vietoris iterative algorithm for computing the homology of a simplicial complex X, starting from its Homo-MC graph G_X. This algorithm iteratively computes the homology of the union of two MC-components A and B connected through an arc in the Homo-MC graph and merges the two components. It terminates when the graph consists of a single node. We will use Algorithm 2, introduced in subsection 7.1, to compute the homology of the sub-complex $A \cup B$.

However, before proceeding we need to find a way to reuse the *reduction equivalence* provided as output by Algorithm 2. Thus, we slightly modify the second step of this algorithm and we associate a *reduction equivalence* with each sub-complex. For each MC-component N, the algorithm computes a reduction equivalence $N_* \Lleftarrow N_* \Rrightarrow E(N)_*$, where the right reduction is the Homological Smith Reduction, which is computed as previously through Algorithm 1. The left reduction is simply the *trivial* reduction of N, at the beginning. Note that it is pos-

sible to compute the reduction equivalence for each node in G_X as a pre-processing step and in parallel. As a consequence, the modified Algorithm 2 should use, at step 4, the cone equivalence theorem (theorem 5.7), instead of the cone reduction one.

At each step, the algorithm collapses the arc between two components A and B in the Homo-MC graph and generates a new component AB (node in the graph) by merging the lists of the top simplices. It also updates the arcs incident in A and B, which become incident in AB. Then, it associates the reduction equivalence, computed by Algorithm 2, with the new component AB in order to make it reused in the subsequent steps. The algorithm repeats this operation until there is only one node in the graph. When the algorithm stops, the last node corresponds to the input simplicial complex X and its $\mathbb{Z}$-homology is retrieved from the reduction equivalence associated with this node, as performed in the final step of Algorithm 2.

Algorithm 3 summarizes the main steps of our Mayer-Vietoris iterative algorithm for homology computation.

Algorithm 3 Mayer-Vietoris iterative computation

Input: the Homo-MC graph G of a simplicial complex X.
Output: homology information for X

1: Initialize the reduction equivalence for all nodes in G_X
2: **while** there is more than one arc in G_X **do:**
3: Select a random arc $a = (n_A, n_B)$ in G_X.
4: Apply Algorithm 2 to n_A, n_B, and $n_A \cap n_B$
5: Create a new node $n_{AB} = n_A \cup n_B$
6: Associate the new reduction equivalence with n_{AB}
7: **End while**
8: Extract the $\mathbb{Z}$-homology of X from the unique node in G_X

Note that any sub-complex AB resulting the union of two MC-components A and B is not manifold-connected. Thus, the decomposition we obtain at any intermediate step is not an MC-decomposition. However, the intersections of the components is still composed by a limited number of non-manifold simplices.

8. Experimental Results

In this section, we present qualitative and quantitative results about the MC-decomposition and our Mayer-Vietoris (MV) algorithm. We have tested our algorithms on some datasets freely available from [19], on a computer with a $3,2$ Ghz Intel i7 processor and 16 Gb of RAM.

We first demonstrate one of the most important properties of the MC-decomposition, which is critical for the efficiency of the homology computation. Recall that our MV algorithm operates on decomposed shapes and computes the homology of the input model from the homology of the components and the homology of their intersection complexes through Mayer-Vietoris sequences. Thus, in order to reduce the redundancies during the homological computation, it is mandatory to use a decomposition which minimizes the size of the intersections (in terms of number of simplices). As shown in Table 1, the size of the intersection complexes (which correspond to the singularities shared by two components) is very small in comparison with the size of the input complex, and it never exceeds 5% of this size. This fact makes the MC-decomposition perfectly suitable as a basis for the MV algorithm.

Shape	s_0	s_1	s_2	S	N	A	MS	MA
armchair	43	125	88	256	6	8	32%	3%
twist	$1K$	$4K$	$2K$	$7K$	4	5	65%	0.8%
two-twist	$1K$	$5K$	$3K$	$9K$	8	13	45%	0.9%
carter	$4K$	$12K$	$8K$	$24K$	28	40	45%	0.6%

Table 1: Statistics about the Homo MC-Graph. Here, we analyze non-manifold shapes formed by s_0 vertices, s_1 edges and s_2 triangles: their corresponding Homo MC Graphs has N nodes and A arcs. It is interesting to compare the size MS of the largest MC-component and the size MA of the largest intersection between two MC-components, both expressed as a percentage of the total number of simplices $S = s_0 + s_1 + s_2$ in the input complex.

Our MV algorithm has been designed for computing the *complete* homological information for an arbitrary shape, including not only the Betti numbers, but also the generators, and the torsion coefficients, if there are any.

Figure 15 shows the MC decomposition of three non-manifold 2-simplicial complexes and some of the generators for the homology groups H_1 and H_2, computed by the MV algorithm. Note that the *twist* model, Figure 15(a), is isomorphic to a torus (in wired grey) in which there is an other embedded 2-cycle (in blue). The *two-twist* model, Figure 15(d), is equivalent to two intersecting tori, corresponding to the yellow and the wired grey 2-cycles, with one embedded shell, corresponding to the blue 2-cycle. The *carter* model has a very complicated topology. Some of its 1-cycles and 2-cycles are shown in Figure 15(f).

We have decided to compare our MV algorithm to the classical SNF algorithm, which is the most general method for computing the $\mathbb{Z}$-homology. Recall that the SNF algorithm operates on the entire model and computes the incidence matrices from the entire shape, while our MV algorithm uses the SNF algorithm to compute the homology on the MC-components. Our current implementation encodes the classical SNF algorithm, without any optimization: in any case, it is possible to use any other version of this algorithm, provided it keeps track of the basis changes during the reduction of the matrices.

Our experimental results, summarized in Table 2, tend to prove that the MV algorithm is a reasonable tool for computing the $\mathbb{Z}$-homology on simplicial shapes, requiring less space than the SNF algorithm, and providing a relevant speed-up to the computation.

The key point in our storage analysis is the size of the incidence matrices, which we have to be reduced. Recall that an incidence matrix D_k of dimension k relates the chain-groups C_{k-1} and C_k and it requires $I_k = s_k \times s_{k-1}$ integer values (encoded on 4 bytes), where s_j denotes the number of j-simplices in the input simplicial complex. The SNF algorithm needs D_1 and D_2, and thus its storage cost

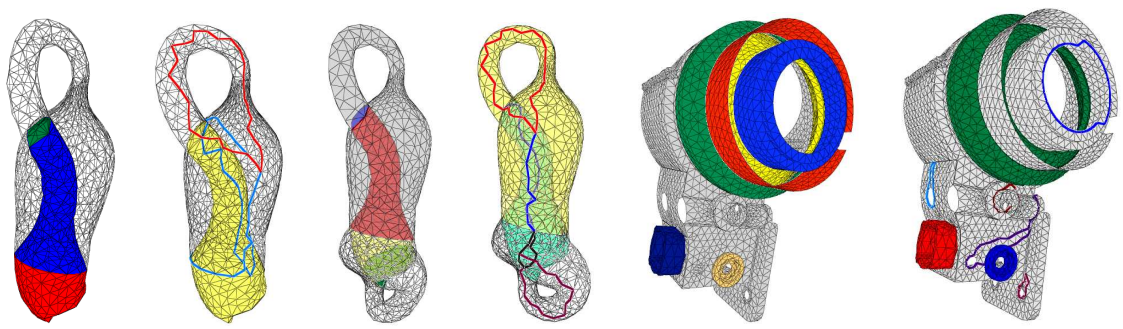


Figure 15: Examples of MC-decomposition and homology generators computed with our MV algorithm on some non-manifold 2-simplicial complexes. (a) The MC decomposition, (b) two 1-cycles (in red and blue), and two 2-cycles (in grey and yellow) for the *twist* model. (c) The MC decomposition, (d) four 1-cycles (in red, blue, black and purple), and three 2-cycles (in yellow, blue and grey) for the *two-twist* model. (e) The MC decomposition and (f) some free generators of H_1 and H_2 for the *carter* model.

Shape	$(\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2)$	$\mathcal{I}_1$	$\mathcal{I}_2$	$\mathcal{SNF}_s$	$\mathcal{SNF}_t$	$\mathcal{MI}_1$	$\mathcal{MI}_2$	$\mathcal{MV}_s$	$\mathcal{MV}_t$	$\mathcal{R}_s$	$\mathcal{R}_t$
armchair	$(\mathbb{Z}, 0, \mathbb{Z}^5)$	0.2 Mb	0.4 Mb	0.6 Mb	60 ms	0.03 Mb	0.04 Mb	0.07 Mb	19 ms	88%	3.2
twist	$(\mathbb{Z}, \mathbb{Z}^2, \mathbb{Z}^2)$	16 Mb	34 Mb	50 Mb	2.2×10^6 ms	7 Mb	14 Mb	21 Mb	1.4×10^6 ms	55%	1.6
two-twist	$(\mathbb{Z}, \mathbb{Z}^4, \mathbb{Z}^3)$	26 Mb	54 Mb	80 Mb	1.2×10^7 ms	7 Mb	14 Mb	21 Mb	3×10^6 ms	73%	3.8
carter	$(\mathbb{Z}, \mathbb{Z}^{27}, \mathbb{Z}^5)$	190 Mb	377 Mb	567 Mb	7.7×10^7 ms	41 Mb	75 Mb	116 Mb	1.7×10^7 ms	79%	4.5

Table 2: Comparisons in terms of timings and storage cost between the SNF and the MV algorithms, which compute the $\mathbb{Z}$-homology $(\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2)$ for some non-manifold simplicial complexes. Columns $\mathcal{I}_1$ and $\mathcal{I}_2$ indicate the size (in Mb) of the incidence matrices for the entire shape. Columns $\mathcal{SNF}_s$ and $\mathcal{SNF}_t$ show respectively the storage cost (in Mb) and the timing (in ms) required by the SNF algorithm. Columns $\mathcal{MI}_1$ and $\mathcal{MI}_2$ show the size (in Mb) of the incidence matrices for the largest MC component. Columns $\mathcal{MV}_s$ and $\mathcal{MV}_t$ show respectively the storage cost (in Mb) and the timing (in ms) required by the MV algorithm. We also provide the reduction of storage cost $\mathcal{R}_s$ (expressed as a percentage of $\mathcal{SNF}_s$), and the ratio $\mathcal{R}_t$ between $\mathcal{SNF}_t$ and $\mathcal{MV}_t$.

$\mathcal{SNF}_s$ is $\mathcal{O}(\mathcal{I}_1 + \mathcal{I}_2)$. Conversely, our MV algorithm computes, at each step, the homology of the union of only two sub-complexes $\mathcal{A}$, $\mathcal{B}$ and their intersection $\mathcal{A} \cap \mathcal{B}$. Since the size of the intersection complex is usually very small, we can ignore it. Thus, we operate only on the incidence matrices for the components $\mathcal{A}$ and $\mathcal{B}$, namely $\mathcal{D}_k^{\mathcal{A}}$ and $\mathcal{D}_k^{\mathcal{B}}$, with $k = 1, 2$. In the following, we respectively indicate their size as $\mathcal{I}_k^{\mathcal{A}}$ and $\mathcal{I}_k^{\mathcal{B}}$. Thus, the storage cost of each step is $\mathcal{O}(\mathcal{I}_1^{\mathcal{A}} + \mathcal{I}_2^{\mathcal{A}} + \mathcal{I}_1^{\mathcal{B}} + \mathcal{I}_2^{\mathcal{B}})$. If we consider the size of incidence matrices for the largest MC-component, respectively indicated as $\mathcal{MI}_1$ and $\mathcal{MI}_2$, then the storage cost of the MV algorithm $\mathcal{MV}_s$ becomes $\mathcal{O}(\mathcal{MI}_1 + \mathcal{MI}_2)$. This fact demonstrates that the MV algorithm requires much less space than the SNF algorithm: in our tests, we obtained a reduction of at least 55% of $\mathcal{SNF}_s$ (see column $\mathcal{R}_s$ in Table 2). We also provide timing comparisons between our MV algorithm and the SNF one, demonstrating that we obtain a relevant speed-up with our approach. In our tests, the MV algorithm is at least 1.6 times faster than the SNF algorithm (see column $\mathcal{R}_t$ in Table 2).

However, advantages introduced by the MV algorithm can be slightly reduced in some cases, as shown in Table 2 for the *twist* model. Since the Homo-MC Decomposition does not impose any limitation on the components size, it is possible to obtain a large MC-component. Thus, computing the homology of this MC-component through the SNF algorithm is time consuming.

This issue could be overcome in different ways. For instance, we can reduce the size of the MC-components through a special simplification algorithm, which preserves its topology and handles its singularities (i.e., non-manifold simplices). In this way, the homology generators of the component can be computed only on the simplified version. However, if this component has to be merged with another MC-component by the MV algorithm in a later iteration, the generators have to be expressed in the original (non-simplified) MC-component, in order to ensure the consistency of the computation duharing the union. It is also possible to gain in efficiency by using one of the existing optimizations of the SNF algorithm for sparse integer matrices. As noted before, our framework does not depend on the SNF algorithm selected and it can work with any optimized version of this latter.

9. Concluding Remarks

We have introduced a new algorithmic approach for homology computation on arbitrary non-manifold shapes discretized through simplicial complexes. Our algorithm is based on the constructive version of the Mayer-Vietoris sequence, which allows us to compute the $\mathbb{Z}$-homology of a simplicial complex from the homologies of its sub-complexes. We call this algorithm as *Mayer-Vietoris (MV)*

algorithm. The starting point of the MV algorithm is the MC-decomposition of the shape, which minimizes the intersections between the manifold-connected components. Combined with this decomposition, the MV algorithm has been proven to be more efficient (in terms of storage and timings) than the classical SNF algorithm, which operates on the entire input model.

In the future, we are planning to improve our current implementation of the SNF algorithm, which will allow us to increase the efficiency of the homology computation on each MC-component and, thus, process very large models. We are also planning to investigate how to combine our algorithm with a different approach for computing the homology of the MC-components, since these latter can be viewed as almost manifold complexes, and thus efficient geometric algorithms for homology computation on manifold shapes could be applied.

We are also planning to investigate different strategies for improving the geometric properties of the generators in order to provide the shortest set of loops that generate the homology groups. One possible solution is to minimize their length, by associating a metric to the homology basis, as recently introduced in [12].

Acknowledgements. We thank Professor Francis Sergeraert for the many helpful discussions on Constructive Homology. We thank the anonymous reviewers for their helpful suggestions.

References

[1] Agoston, M.K., 1976. Algebraic topology: a first course. M. Dekker Publisher, New York, USA.

[2] Alayrangues, S., Damiand, G., Fuchs, L., Lienhardt, P., Peltier, S., 2009. Homology computation on cellular structures in image context, in: CTIC, Austria. pp. 19–28.

[3] Boltcheva, D., Merino, S., Léon, J.C., Hétroy, F., 2010. Constructive Mayer-Vietoris algorithm: computing the homology of unions of simplicial complexes. Technical Report. INRIA-7471.

[4] Chen, C., Freedman, D., 2010. Measuring and computing natural generators for homology groups. Computational Geometry: Theory & Applications 43, 169–181.

[5] Cohen-Steiner, D., Edelsbrunner, H., Morozov, D., 2006. Vines and vineyards by updating persistence in linear time, in: Symposium on Computational geometry, SCG '06, pp. 119–126.

[6] Damiand, G., Peltier, S., Fuchs, L., 2006. Computing homology for surfaces with generalized maps: Application to 3d images., in: ISVC'06, Nevada, USA. pp. 235–244.

[7] De Floriani, L., Hui, A., Panozzo, D., Canino, D., 2010. A dimension-independent data structure for simplicial complexes, in: IMR, Chattanoga, TN, USA. pp. 403–420.

[8] De Floriani, L., Mesmoudi, M.M., Morando, F., Puppo, E., 2003. Decomposing non-manifold objects in arbitrary dimensions. Graph. Models 65, 2–22.

[9] Delfinado, C.J.A., Edelsbrunner, H., 1993. An incremental algorithm for betti numbers of simplicial complexes, in: SoCG'93, ACM, New York, NY, USA. pp. 232–239.

[10] Dey, T.K., Guha, S., 1996. Computing homology groups of simplicial complexes in $\mathbb{R}^3$, in: Proc. 28th ACM Symp. Comp. Theory (STOC 1996).

[11] Dey, T.K., Li, K., Sun, J., Cohen-Steiner, D., 2008. Computing geometry-aware handle and tunnel loops in 3d models. ACM Transactions on Graphics 27, No. 45.

[12] Dey, T.K., Sun, J., Wang, Y., 2010. Approximating loops in a shortest homology basis from point data, in: Proc. 26th Annu. Sympos. Comput. Geom. (SOCG 2010), pp. 166–175.

[13] Dousson, X., Rubio, J., Sergeraert, F., Siret, Y., 2008. The kenzo program. `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/`.

[14] Dumas, J.G., Heckenbach, F., Saunders, B.D., Welker, V., 2003. Computing simplicial homology based on efficient smith normal form algorithms. Alg., Geom., and Soft. Syst. , 177–207.

[15] Edelsbrunner, H., Harer, J., 2010. Computational topology, an introduction. AMS, Providence, Rhode Island.

[16] Edelsbrunner, H., Letscher, D., Zomorodian, A., 2002. Topological persistence and simplification. Discrete & Computational Geometry 28, 511–533.

[17] Fabio, B., Landi, C., Medri, F., 2009. Recognition of occluded shapes using size functions, in: ICIAP'09, Springer-Verlag, Berlin, Heidelberg. pp. 642–651.

[18] Forman, R., 1998. Morse theory for cell complexes. Advances in Mathematics 134, 90–145.

[19] Geometry and Graphics Group, 2009. Non-manifold Meshes Repository. `http://indy.disi.unige.it/nmcollection/`.

[20] Ghrist, R., Muhammad, A., 2005. Coverage and hole-detection in sensor networks via homology, in: IPSN'05, IEEE Press, Piscataway, NJ, USA. p. No. 34.

[21] Giesbrecht, M., 1996. Probabilistic computation of the smith normal form of a sparse integer matrix. Algorithmic Number Theory. Lecture Notes in Computer Science 1122, 173–186.

[22] González-Díaz, R., Jiménez, M.J., Medrano, B., Real, P., 2009. Chain homotopies for object topological representations. Discrete Applied Mathematics 157, 490–499.

[23] Gotsman, C., Kaligosi, K., Mehlhorn, K., Michail, D., Pyrga, E., 2007. Cycle bases of graphs and sampled manifolds. Computer Aided Geometric Design 24, 464–480.

[24] Hafner, J.L., McCurley, K.S., 1991. Asymptotically fast triangularization of matrices over rings. SIAM Journal on Computing 20, 1068–1083.

[25] Hatcher, A., 2002. Algebraic Topology. Cambr. Univ. Press.

[26] Hui, A., De Floriani, L., 2007. A two-level topological decomposition for non-manifold simplicial shapes, in: SPM, Beijing, China. pp. 355–260.

[27] Kaczynski, T., Mrozek, M., Slusarek, M., 1998. Homology computation by reduction of chain complexes. Computers & Mathematics with Applications 35-4, 59–70.

[28] Kannan, R., A., B., 1979. Polinomial algorithms for computing the smith and hermite normal forms of an integer matrix. SIAM Journal of Computing 8, 499–507.

[29] Mrozek, M., Pilarczyk, P., Żelazna, N., 2008. Homology algorithm based on acyclic subspace. Computers and Mathematics with Applications 55, 2395–2412.

[30] Munkres, J., 1999. Algebraic Topology. Prentice Hall.

[31] Peltier, S., Ion, A., Kropatsch, W.G., Damiand, G., Haxhimusa, Y., 2009. Directly computing the generators of image homology using graph pyramids. Image & Vision Computing 27, 846–853.

[32] Sergeraert, F., 1994. The computability problem in algebraic topology. Advances in Mathematics 104, 139–155.

[33] Sergeraert, F., 1999. Constructive algebraic topology. SIGSAM Bull. 33, 13–25.

[34] Sergeraert, F., Rubio, J., 2006. Constructive homological algebra and applications. `http://www-fourier.ujf-grenoble.fr/~sergerar/Papers/`.

[35] Storjohann, A., 1996. Near optimal algorithms for computing smith normal forms of integer matrices, in: ISSAC'96, ACM, New York, NY, USA. pp. 267–274.

[36] Thakur, A., Banerjee, A.G., Gupta, S.K., 2009. A survey of cad model simplification techniques for physics-based simulation applications. Comput. Aided Des. 41, 65–80.

[37] Troelstra, A.S., van Dalen, D., 1988. Constructivism in Mathematics, an Introduction. Studies in Logic and the Foundations of Mathematics, North-Holland.

[38] Zhu, X., Sarkar, R., Gao, J., 2009. Topological data processing for distributed sensor networks with morse-smale decomposition, in: IEEE Infocom, pp. 2911–2915.

[39] Zomorodian, A., Carlsson, G., 2008. Localized homology. Computational Geometry: Theory & Applications 41, 126–148.

$\therefore$

A.3 JUST NOTICEABLE DISTORTION PROFILE FOR FLAT-SHADED 3D MESH SURFACES

Georges Nader, Kai Wang, Franck Hétroy-Wheeler, Florent Dupont
IEEE Transactions on Visualization and Computer Graphics, 2016.

Just Noticeable Distortion Profile for Flat-Shaded 3D Mesh Surfaces

Georges Nader, Kai Wang, Franck Hétroy-Wheeler, and Florent Dupont

Abstract—It is common that a 3D mesh undergoes some lossy operations (*e.g.*, compression, watermarking and transmission through noisy channels), which can introduce geometric distortions as a change in vertex position. In most cases the end users of 3D meshes are human beings; therefore, it is important to evaluate the visibility of introduced vertex displacement. In this paper we present a model for computing a Just Noticeable Distortion (JND) profile for flat-shaded 3D meshes. The proposed model is based on an experimental study of the properties of the human visual system while observing a flat-shaded 3D mesh surface, in particular the *contrast sensitivity function* and *contrast masking*. We first define appropriate local perceptual properties on 3D meshes. We then detail the results of a series of psychophysical experiments where we have measured the threshold needed for a human observer to detect the change in vertex position. These results allow us to compute the JND profile for flat-shaded 3D meshes. The proposed JND model has been evaluated via a subjective experiment, and applied to guide 3D mesh simplification as well as to determine the optimal vertex coordinates quantization level for a 3D model.

Index Terms—Just noticeable distortion, human visual system, psychophysical experiments, contrast sensitivity function, contrast masking, 3D mesh

1 Introduction

Three-dimensional (3D) meshes may be subject to various lossy operations such as compression and watermarking. These operations introduce geometric distortions in form of vertex displacement. Since computer graphics applications are intended towards human subjects, it is important to study the visibility of those distortions, taking into account the properties of the human visual system. Geometric measures like Hausdorff distance or root mean square error (RMS) [1], [2] reflect the physical variation of the mesh geometry. They do not correlate with the human vision [3] and therefore cannot be used to predict whether a distortion is visible or not. The visibility of the geometric distortions is also affected by the lighting conditions, the viewpoint, the surface's material and the rendering algorithm. Figure 1 gives two examples of how the scene parameters can affect the visibility of vertex noise. A slightly visible noise is injected onto the 3D model (Fig. 1.(a)). When the viewing distance is increased, the noise becomes invisible (Fig. 1.(b)). When the light direction is changed from front to top-left, the noise on the mesh becomes more visible (Fig. 1.(c)).

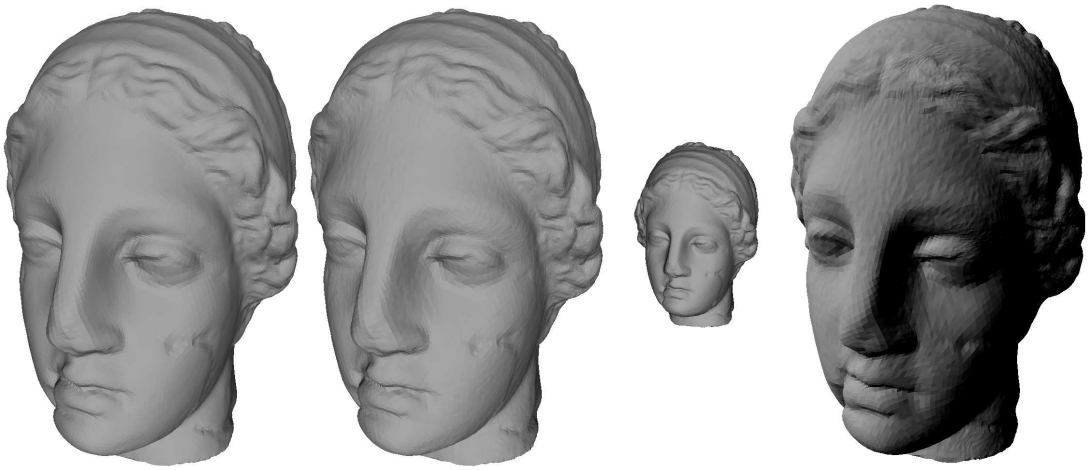
The Just Noticeable Distortion (JND) threshold refers to the threshold above which a distortion becomes visible to the majority of observers [4]. If a distortion is below the JND value, it can be considered as invisible. JND models reflect the local properties of the human visual system, in particular the *contrast sensitivity function* and *contrast masking*. On one hand, in image/video processing, JND models of 2D images and videos have proven to be helpful for various applications such as evaluating the image visual fidelity [5] and optimizing image compression [6], [7], [8]. On the other hand, while perceptually driven graphics are popular within the computer graphics community [3], [9], [10], there has been little effort given to study the visibility of vertex displacement and further to compute a JND profile for 3D models.

This is exactly what this paper proposes. More specifically, our contributions are the following:

1) We define local perceptual properties that are appropriate for a "bottom-up" evaluation of vertex displacement visibility on 3D meshes.
2) We design and conduct psychophysical experiments to study properties of the human visual system when observing a flat-shaded 3D mesh.
3) We propose a JND profile for 3D meshes that takes into consideration the various circumstances of mesh usage (display size, scene illumination, viewing distance).

The rest of this paper is organized as follows: Section 2 briefly introduces the properties of the human visual system that are essential to compute the JND profile, and discusses existing work on perceptually driven graphics techniques. Section 3 explains how perceptual properties are evaluated on a 3D mesh and presents a series of psychophysical experiments that were carried out in order to measure the visibility threshold and their results. Section 4 describes our method to compute the JND profile for a 3D mesh. In Section 5 we evaluate the proposed JND model's performance via subjective experiments. In Section 6 we apply our JND profile to guide mesh simplification and to automatically determine the optimal vertex coordinates quantization level. Finally, we discuss the limitations and conclude the paper.

- *G. Nader and F. Dupont are with Université de Lyon, LIRIS UMR 5205 CNRS, France. E-mail: {georges.nader, florent.dupont}@liris.cnrs.fr*
- *K. Wang is with CNRS and Univ. Grenoble Alpes, GIPSA-Lab, F-38000 Grenoble, France. E-mail: kai.wang@gipsa-lab.grenoble-inp.fr*
- *F. Hétroy-Wheeler is with Univ. Grenoble Alpes, LJK, F-38000 Grenoble, France and with Inria. E-mail: franck.hetroy@grenoble-inp.fr*



original (a) (b) (c)

Fig. 1. (a) The noise injected onto the original 3D mesh is slightly visible. (b) Increasing the viewing distance makes the noise invisible. (c) Changing the light direction from front to top-left increases the perceived intensity of the noise.

2 Related Work

Before presenting the existing works on perceptually driven graphics, we introduce the local properties of the human visual system on which a large number of the existing methods rely.

2.1 Local Properties of the Human Visual System

The visibility of a visual pattern depends on its local *contrast* and its *spatial frequency* [11]. Local contrast refers to the change of light intensity over the light intensity of its surrounding. The spatial frequency is defined as the size of light patterns on the retina. In the field of visual perception, the spatial frequency is expressed in terms of cycles per degree (cpd) which represents the number of light patterns on the retina in 1 degree of the visual angle. To show how local contrast and spatial frequency affect the visibility of a pattern, we briefly introduce the notions of *contrast sensitivity* and *contrast masking*. Both these concepts describe the basic human vision mechanisms and were largely exploited for the development of many image processing methods [12], [13] especially for JND profiles of 2D images [4]. We refer to [11], [14] for a more detailed treatment of the human visual perception.

Contrast sensitivity. A visual pattern can be detected by the visual system only if the amount of contrast in the pattern is above some threshold. This visibility threshold varies under different spatial frequencies of the visual pattern. This is mainly due to the optics in the eye and the size of the photoreceptors on the retina. The reciprocal of this detection threshold is the contrast sensitivity. The contrast sensitivity function (CSF) describes the visibility threshold with respect to the spatial frequency. The CSF represents the visual system's band-pass filter characteristics when it comes to contrast sensitivity. In general, it exhibits a peak at around 2 to 5 cpd then drops off to a point where no detail can be resolved. The shape of the CSF (peak location and drop off slope) depends on the nature of the visual stimulus (complexity, periodicity) [15]. Since the CSF is the main component in many perceptual models [12], [13], there has been a great interest for measuring it in different circumstances [16]. Most notably, let us cite the Modelfest project [17], in which 10 laboratories collaborated on measuring the contrast threshold for 43 different types of visual stimuli.

Contrast masking. Contrast masking is the change in visibility of a stimulus (target) due to the presence of another stimulus (mask). The visibility of the target depends on different factors, in particular the spatial frequency and the contrast of the mask and the target. Legge and Foley [18] studied the contrast threshold necessary to detect the target when varying the contrast and frequency of the mask. One important observation was that this contrast threshold is increased when the contrast of the mask is increased. The effects of contrast masking can be described by a curve which possesses two asymptotic regions: the first one with a slope of zero and the second one with a positive slope of about 0.6 to 1 (depending on the stimulus) [18]. The zero slope occurs for mask contrast values below the mask's visibility threshold as given by the CSF, indicating that there is no masking effect. By contrast, as the mask contrast value increases beyond the CSF threshold in the second asymptotic region, the threshold for detecting the target increases too.

Despite being essential for the development of perceptual image processing algorithms, the *contrast sensitivity function* and the *contrast masking* have, to the best of our knowledge, never been carefully studied in the 3D setting.

2.2 Perceptually Driven Graphics

Over the last two decades, perceptually driven methods have drawn more and more attention in the computer graphics community. These methods have proven to be useful for evaluating the quality of 3D models and optimizing graphics applications. Perceptually driven methods can be divided into two groups: *top-down* methods and *bottom-up* ones.

Top-down methods. Top-down techniques do not rely on

the exact internal mechanism of the human visual system. Such techniques rather propose hypotheses, which are usually difficult to prove, about the overall behavior of the visual system in order to estimate how a specific visual artifact is perceived. Based on the observation that visual artifacts are less visible on rough regions than on smooth ones of a 3D mesh [19], several perceptual metrics have for instance been proposed [20], [21], [22]. Other features used by such top-down metrics include surface curvature [23], [24] and dihedral angle [25]. Perceptual methods have also been used to guide mesh simplification [26], [27], [28], [29] and compression [30]. In the previously mentioned methods, the perceptual analysis is carried out on the geometry of a 3D mesh. In general, these methods are neither easily applicable to models of different properties (size, details and density) nor capable of adapting to varying circumstances of mesh usage (display characteristics, scene illumination and viewing distance). In addition, they mainly study distortions that are above the visibility threshold. Near-threshold vertex displacement visibility is not the focus of these methods.

Bottom-up methods. A bottom-up approach explicitly takes into account the mathematical models describing the mechanisms of the human visual system. Bottom-up methods have been popular in computer graphics. They are usually based on the concepts of *contrast sensitivity* and *contrast masking* (Section 2.1). One of the most popular methods in image processing is Daly's Visual Difference Predictor (VDP) [31]. This algorithm outputs a map describing the visibility of the difference between two images. The VDP has recently been extended to 2D HDR images in [32]. Based on Daly's VDP algorithm, Ramasubramanian *et al*. [33] computed a 2D threshold map in which each pixel contains the value of the minimum detectable difference. This map is used to guide global illumination computations. This threshold model has later been improved in [34]. Bottom-up analysis has also been used for mesh simplification applications [35], [36], [37]. However, the visibility analysis in those methods is still carried out in the 2D space of rendered images.

In this paper, different from all the methods mentioned in the previous paragraph that conduct the visibility analysis in a 2D space, we present a method for studying the visibility of vertex displacement in the 3D space. This method allows us to define a model for computing a JND profile for 3D meshes. To the best of our knowledge, the only existing JND-like model for a 3D mesh is the one of Cheng *et al*. [38], [39]. However, their goal and approach are very different from those of our method. Their goal was to evaluate the visibility of removing a group of vertices from one level of detail (LOD) to another. To do so, they used a top-down approach where they assumed that the visibility of removing vertices is related to the change in distance to a predefined model skeleton. The limitation of this work is that the JND depends on the predefined skeleton and is only applicable for evaluating LOD techniques. By contrast, our method for computing the JND model is based on a bottom-up experimental study of the properties of the human visual system and explicitly takes into account its internal mechanisms. The proposed JND model can cope with different mesh properties (size, density) and different possible usage of a mesh (illumination, display characteristics).

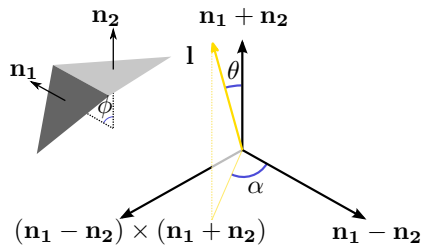


Fig. 2. The contrast between adjacent faces is computed using the angle between their normals and the spherical coordinates of the light direction in the local coordinate system defined by the face normals.

3 Local Perceptual Properties and Psychophysical Experiments

Existing top-down perceptual methods use surface roughness [20], [21], surface curvature [23], [24], dihedral angle [25] and face normal [40] as perceptually-relevant features of a 3D mesh. In this section we define new local perceptual properties for 3D meshes (*i.e.*, local contrast and spatial frequency) that are appropriate for a bottom-up evaluation of vertex displacement visibility. These perceptual properties allow us to study the effects of the contrast sensitivity and the contrast masking in the 3D setting. In the following, we start by explaining how the local contrast (Section 3.1) and the local spatial frequency (Section 3.2) are evaluated on a 3D mesh. We then present a series of psychophysical experiments that were conducted to measure the detection threshold (Section 3.3). In this study we consider 3D meshes that are rendered with a flat shading algorithm. We also limit our study to diffuse surfaces illuminated by a white directional light source.

3.1 Local Contrast Estimation

As explained above, contrast is the measure of difference of luminance. In the case of a flat-shaded rendering, each face of the 3D mesh is attributed a luminance value proportional to the cosine of the angle between its normal and the light direction. The luminance of a face is given by:

$$L = \max\left(\mathbf{l}\cdot\mathbf{n}, 0\right), \tag{1}$$

where $\mathbf{n}$ is the unit face normal and $\mathbf{l}$ is the light direction. The Michelson contrast between two adjacent faces can then be defined by:

$$c = \frac{\parallel L_1 - L_2 \parallel}{L_1 + L_2} = \frac{\parallel \max\left(\mathbf{l}\cdot\mathbf{n_1}, 0\right) - \max\left(\mathbf{l}\cdot\mathbf{n_2}, 0\right) \parallel}{\max\left(\mathbf{l}\cdot\mathbf{n_1}, 0\right) + \max\left(\mathbf{l}\cdot\mathbf{n_2}, 0\right)}, \tag{2}$$

where $\mathbf{n_1}$ and $\mathbf{n_2}$ are the normals of the two adjacent faces. Under the circumstances where the inner products between the light direction and the two face normals are both positive, the above equation yields to the following equation:

$$c = \parallel \cos\alpha \times \tan\theta \times \tan\frac{\phi}{2} \parallel, \tag{3}$$

where α and θ are the spherical coordinates of the light direction in the local coordinate system defined by $\mathbf{n_1}-\mathbf{n_2}$, $\mathbf{n_1}+\mathbf{n_2}$ and their outer product (see Fig. 2). ϕ is the angle between the normals of the two faces.

Equation (3) shows how the contrast is affected by surface

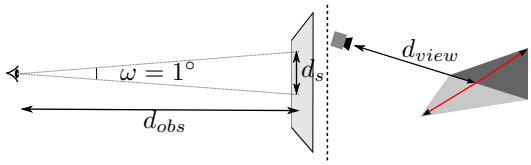


Fig. 3. The spatial frequency of a visual stimulus depends on the observer's distance (d_{obs}) to the screen and the view distance (d_{view}) of the virtual camera to the 3D model.

geometry and the scene illumination. The term $\tan\frac{\phi}{2}$ indicates the impact of surface geometry on the local contrast. On one hand, if the surface is locally smooth ($\phi \approx 0°$), then the local contrast is minimal. On the other hand, if the surface is locally rough ($\phi \gg 0°$), then the local contrast tends to be high. In addition, the term $\cos\alpha \times \tan\theta$ describes how the light direction affects the local contrast. A grazing light direction will maximize the value of the contrast where θ is close to $90°$ and α is close to $0°$ or $180°$, while a light direction close to the normal direction ($\theta \approx 0°$) makes the contrast minimal.

3.2 Local Frequency Estimation

Spatial frequency refers to the size of a visual stimulus on the retina. It is expressed in cycles per degree (cpd) [11]. The spatial frequency is affected by the physical size of the object and the observer's distance to the object. In our case, the visual stimulus is displayed on a screen and consists of the difference in luminance between a pair of adjacent faces. The perceived size of this stimulus depends then on the display's properties (resolution and size), the observer's distance to the display, the position of the model in the virtual 3D world and the size of the faces (see Fig. 3). As a consequence, in order to compute the spatial frequency we first need to evaluate the number of pixels that are occupied by the pair of adjacent faces. This is achieved by applying the perspective projection to the opposing vertices of the pair of adjacent faces. We then convert the computed number of pixels to cpd using the following approximation:

$$f = \frac{d_s}{n_{px}/ppcm} = \frac{2d_{obs} \times \tan\frac{1}{2}\frac{\pi}{180}}{n_{px}/ppcm} \approx \frac{d_{obs}}{n_{px}/ppcm} \times \frac{\pi}{180}, \quad (4)$$

where d_s is the spread of 1 cpd on the screen (see Fig. 3), d_{obs} is the observer's distance to the screen in cm, n_{px} is the number of pixels occupied by the visual stimulus obtained by applying perspective projection, and $ppcm$ is the number of pixels in 1 cm of the screen, computed as:

$$ppcm = \frac{\sqrt{r_X^2 + r_Y^2}}{s}, \quad (5)$$

with r_X the horizontal resolution in pixels, r_Y the vertical one and s the diagonal length of the screen in cm. $n_{px}/ppcm$ represents the size of the displayed stimulus in cm. The density of the 3D mesh is related to the size of its faces. This implies that a dense mesh will display high frequency visual stimuli while a coarse mesh will show low frequency ones under the same observation condition.

3.3 Threshold Measurement

Having defined the local contrast and spatial frequency on the 3D mesh, we now explain how to measure the local contrast threshold required to detect a change in the mesh.

3.3.1 Contrast Sensitivity Function

As detailed above, the contrast sensitivity function describes the human visual system's ability to detect contrast at different frequencies. The threshold given by the CSF refers to the amount of contrast required to distinguish a stimulus from its uniform surrounding (surrounding contrast is 0).

Fig. 4. Visual stimulus to measure the contrast sensitivity. Left: the reference plane. Right: a regular plane where a vertex is displaced.

Stimulus. In order to be able to measure the CSF in the 3D setting, the natural visual stimulus consists of a vertex displaced from the surface of a regular plane whose contrast is 0 (Fig. 4). The displacement of the vertex alters the normal of the adjacent faces and thus changes the contrast. In order to measure the threshold of different frequencies we change the density of the plane, which alters the size of its faces. The threshold is measured for 8 spatial frequencies (1.12, 2, 2.83, 4, 5.66, 8, 11.30 and 16 cpd, also considered in the Modelfest project [17]). The plane is tilted by $20°$ to give the observer a 3D feel.

Experimental setup. Experiments took place in a laboratory environment. The stimuli were displayed on an Asus 23-inch display in a low illuminated room. Screen resolution was 1920×1080. The stimuli were observed from a distance of 1 m, which allowed us to measure the threshold for frequencies between 1 and 16 cpd.

Method. Two planes were displayed side by side on the screen, one of which exhibits a displayed vertex in its central area. The participants were then asked to answer by Yes or No whether they can see any difference between the displayed planes. According to our experience, this method is faster and less tiring for inexperienced subjects than 2AFC methods. We used the QUEST procedure [41] with a fixed number of trials (20 trials) to find the threshold. Each participant repeated the experience 4 times each on a different day. An additional "dummy" frequency, whose data were not taken into account, was included at the beginning of each session to stabilize the subject's answers. In order to avoid any bias, frequency order was randomized for each observer in each session. No user interaction was allowed.

Participants. 5 subjects (3 males and 2 females) participated in our experiments. All had normal or corrected-to-normal vision and were 22 to 26 years old. One of the participants was experienced in perceptual subjective evaluations and the other 4 were inexperienced.

Results. The results of this experiment are shown in Fig. 5. The displacement of a vertex causes a variation in contrast for multiple face pairs. We save the maximum contrast between the affected face pairs. The left panel of Fig. 5 plots the

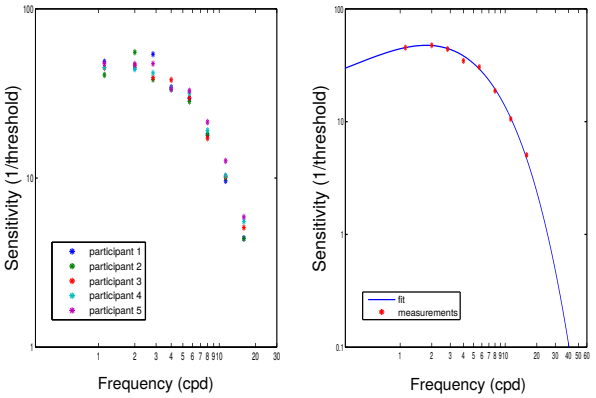


Fig. 5. Left: plot of the mean sensitivity for each observer over each frequency. Right: plot of the subjects' mean sensitivity over each frequency fitted using Manos and Sakrison's mathematical model.

mean sensitivity for each observer over each frequency. The plot shows a high consistency between the participants: All of them exhibit a peak in sensitivity at 2 cpd and the drop off in sensitivity on either side of the peak is similar for all participants. The right panel of Fig. 5 shows the subjects' mean sensitivity over each frequency, fitted using Mannos and Sakrison's mathematical model [42] that is defined by:

$$\mathrm{csf}(f) = \left(1 - a + \frac{f}{f_0}\right) e^{-f^p}, \tag{6}$$

with $a = -15.13$, $f_0 = 0.0096$ and $p = 0.64$. The fit predicts a peak in sensitivity at around 2 cpd that drops rapidly at high frequencies. At low frequencies the drop in sensitivity is much slower that the one measured with a 2D contrast grating [15], [17]. This is due to the aperiodic nature of the visual stimulus [15]. Equation (6) shows the relation between the density of a mesh and the visibility of a local contrast alteration. As the density increases from a very low value, it would be slightly easier for the human visual system to notice the local contrast alteration on the mesh's surface until it reaches a certain limit (around 2 cpd) beyond which it would be harder to detect contrast alteration as the density increases.

3.3.2 Contrast Masking

Contrast masking refers to the ability of the human visual system to discriminate between two visible visual stimuli (a *target* and a *mask*). Since the visibility of a visual stimulus depends on its spatial frequency, the contrast masking threshold is different at each frequency. However, if we normalize the threshold values by the mask's CSF value, then the resulting threshold will be independent of the stimulus's spatial frequency [31]. Therefore, measuring the masking effect can be done by only changing the contrast value of a mask signal without the need to pay too much attention to its spatial frequency. Nevertheless, in our preliminary tests, we measured the normalized contrast masking effects for 3 different frequencies and found that the results were indeed the same (as stated in [31]), showing that measuring the contrast masking effect can be done independently from the spatial frequency of the visual stimulus.

Stimulus and method. In order to be able to measure the threshold relative to the contrast masking effect, the visual stimulus needs to exhibit a visible initial contrast (*i.e.*, above the CSF value). We then increase the initial contrast and measure the value needed to notice that change. In other words, if c is the initial contrast (mask signal) and c' is the increased value, we measure $\Delta c = c' - c$ (target signal) needed to discriminate between c and c'. Similarly to the method used for measuring the CSF, two models were displayed on the screen and the participants were supposed to decide whether they saw the difference between the two objects or not. The stimulus consists of a vertex displaced from a sphere approximated by a subdivided icosahedron (Fig. 6). The icosahedron is subdivided 3 times, which makes the contrast between two adjacent faces (stimulus of about 2 cpd) visible for an observer. This initial contrast represents the mask signal. Varying the light direction modifies the value of the initial contrast between two adjacent faces. We measured the threshold relative to 7 mask contrasts that were log-linearly spaced from 0.6 to 4 times the CSF threshold.

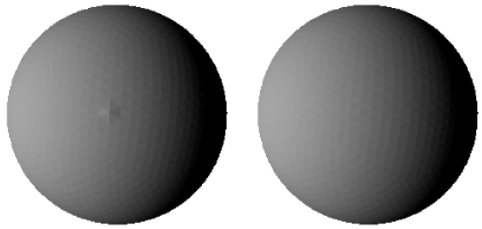


Fig. 6. Visual stimulus for measuring contrast masking. Left: a sphere approximated by an icosahedron subdivided 3 times from which a vertex is displaced. Right: the reference sphere.

Experimental setup. The same experimental setup and the same method than for the CSF measurement experiment previously described have been used. The same 5 subjects also participated in the contrast masking experiments.

Results. The results of this experiment are shown in Fig. 7. The left panel plots for every participant the mean normalized threshold over the normalized contrast mask. For mask contrasts below the visibility threshold (normalized contrast mask lower than 1), the measured normalized threshold is close to 1. This indicates that the measured threshold refers to the one given by the CSF and that no masking has occurred. For mask contrasts above the visibility threshold, the measured normalized threshold is above the one given by CSF and lies close to the asymptotic region with a slope near 0.7. The right panel of Fig. 7 shows the subjects' mean threshold over each mask contrast fitted using Daly's mathematical masking model [31] that is defined by:

$$\mathrm{masking}(\tilde{c}) = \left(1 + (k_1 \times (k_2 \times \tilde{c})^s)^b\right)^{1/b}, \tag{7}$$

with $\tilde{c}$ the normalized threshold, and the fitted values $k_1 = 0.0078$, $k_2 = 88.29$, $s = 1.00$ and $b = 4.207$. The fit exhibits the two asymptotic regions that characterize the contrast masking effect with a transition between the two regions at the CSF visibility threshold. To some extent, Eq. (7) shows how an increase in surface roughness can hide local geometric distortions on the mesh's surface. A rough

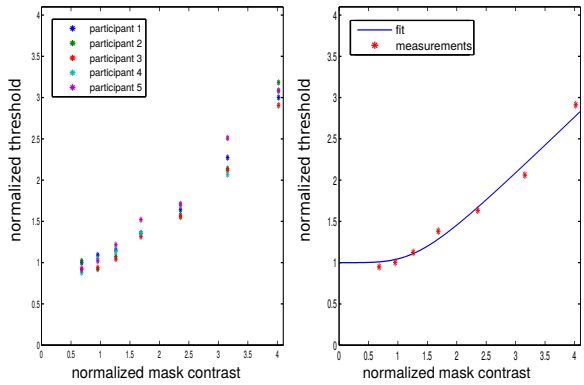


Fig. 7. Left: plot of the normalized mean threshold for each observer over normalized mask contrast. Right: plot of the subjects' mean normalized threshold over each normalized mask contrast, fitted using Daly's mathematical contrast masking model.

surface in general exhibits more contrast and thus is more likely to be able to mask the visibility of a local distortion.

3.3.3 Contrast Visibility Threshold

Having measured the effects of the contrast sensitivity function and the contrast masking, we can now compute the threshold T needed to detect the difference between a pair of adjacent faces. To do so, we first evaluate the spatial frequency f and the local contrast c of the initial face pair (Eqs. (4) and (3)). We then normalize the computed contrast by the corresponding CSF value (Eq. (6)) and finally obtain the threshold using the masking model (Eq. (7)). The threshold T is expressed using the following equation:

$$T = \frac{\text{masking}(c \times \text{csf}(f))}{\text{csf}(f)}, \tag{8}$$

where c is the initial local contrast and f is the corresponding spatial frequency. Accordingly, if a local geometric distortion causes a change in contrast that is higher than the computed threshold then it is classified as visible. In the next section we explain how the JND profile for a 3D mesh is obtained using the computed threshold T.

4 Just Noticeable Distortion Profile

The JND refers to the threshold beyond which a change in contrast becomes visible for the average observer. The JND profile that we propose in this section allows us to get this threshold for an arbitrary displacement direction.

4.1 Overview

In the 3D setting, the JND is evaluated by computing the maximum displacement each vertex can tolerate. On one hand, a vertex displacement in a given direction will probably cause a change in the normals of adjacent faces and a change in local density. On the other hand, we showed in Section 3 that the face normals and the local density affect the contrast and the spatial frequency, respectively. This means that the displacement of a vertex probably alters the local perceptual properties. The visibility of this alteration

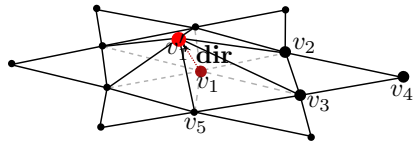


Fig. 8. The displacement of a vertex v_1 in a direction $\mathbf{dir}$ causes a change in contrast and spatial frequency for surrounding pairs of faces *sharing a common edge* in 1-ring and 2-ring of the displaced vertex.

can be evaluated using the perceptual models presented in Eqs. (6), (7) and (8). In this section, we present a numerical method for computing the maximum displacement beyond which the local distortion can be detected by an average human observer.

4.2 Visibility of Adjacent Face Pairs

The displacement of a vertex alters the local perceptual properties, *i.e.*, contrast and spatial frequency, of the surrounding pairs of adjacent faces (Fig. 8). In order to get the maximum displacement a vertex can tolerate, we need to evaluate the perceptual effect of any displacement on the set of its surrounding pairs of faces. In the following, we show how the visibility of a change in local perceptual property is evaluated. To this end, we first compute the change in contrast and spatial frequency and then evaluate the probability of detecting this change.

4.2.1 Change in Contrast

The displacement of a vertex v_1 in a direction $\mathbf{dir}$ causes the normals of its adjacent faces to change. This change in normals causes a variation in contrast for the surrounding pairs of adjacent faces. Therefore, evaluating the change in contrast requires evaluating the change in normal direction of these adjacent faces. For example, having two adjacent faces $\{v_1, v_3, v_2\}$ and $\{v_2, v_3, v_4\}$ (see Fig. 8) with normals $\mathbf{n_1}$ and $\mathbf{n_2}$ respectively, we express the new normal $\mathbf{n_1'}$ after displacing v_1 in a direction $\mathbf{dir}$ with a magnitude d by:

$$\begin{aligned} \tilde{\mathbf{n}}_\mathbf{1}' &= (v_1 - v_2) \times (v_3 - v_2) + d \cdot (\mathbf{dir} \times (v_3 - v_2)), \\ \mathbf{n_1'} &= \frac{\tilde{\mathbf{n}}_\mathbf{1}'}{\| \tilde{\mathbf{n}}_\mathbf{1}' \|}. \end{aligned} \tag{9}$$

Since none of the vertices of the second face $\{v_2, v_3, v_4\}$ is displaced, its normal direction does not change. For the cases where the displacement of v_1 causes changes in the normal directions of both faces (*e.g.*, the pair of adjacent faces $\{v_1, v_3, v_2\}$ and $\{v_1, v_5, v_3\}$ in Fig. 8), their new normals are evaluated similarly, according to an adaptation of Eq. (9). The new contrast between adjacent faces is then evaluated using Eq. (3), with the new face normal(s).

4.2.2 Change in Spatial Frequency

Moving the vertex v_1 in the direction $\mathbf{dir}$ may cause a change in spatial frequency as well, because the size of the adjacent face pairs might be altered. Computing the new spatial frequency requires evaluating the distance between the opposing vertices v_1' and v_4, v_1' being the position of

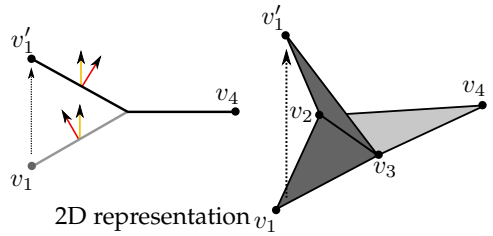


Fig. 9. When the displacement of a vertex alters the convexity of two adjacent faces, the contrast might remain the same as long as the angle between the light direction (yellow arrow) and the face normal (red arrow) does not change.

v_1 after its displacement. The new distance between the opposing vertices is expressed as:

$$\| v_4 - v_1' \| = \sqrt{\| v_4 - v_1 \|^2 + d^2 - 2d \times (v_4 - v_1') \cdot \mathbf{dir}}. \tag{10}$$

To obtain the spatial frequency in cpd, we apply the perspective projection to $(v_4 - v_1')$ in order to get the number of pixels that the face pair occupies on the screen and then apply Eq. (4).

4.2.3 Detection Probability

Having expressed how the displacement of a vertex in an arbitrary direction affects the local contrast and the spatial frequency of the surrounding pairs of adjacent faces, we now explain how to determine whether this change is visible or not. To do so, we compute the probability of detecting this change. The method for computing this probability is inspired by Daly's VDP [31].

The so-called *psychometric function* describes the probability for the human visual system to detect a visual stimulus. A common choice for the psychometric function is given by the following equation:

$$p(\Delta c) = 1 - e^{-(\Delta c / T)^{\beta}}, \tag{11}$$

where T is the threshold as computed in Eq. (8) and β is the slope of the psychometric function. β is set to 3.5 across many psychophysical experiments and perceptual studies [43]. Using the data collected from our psychophysical experiments, we evaluated the detection probability for contrasts near the measured threshold. The computed probabilities are fitted to the psychometric function and we obtain β of about 3.6. This fitted value will be used in our calculation. Δc is the change in contrast which corresponds to contrast difference before and after the displacement of a vertex and is evaluated as:

$$\Delta c = \begin{cases} \| c' - c \| & \text{if } \mathrm{sgn}(\mathbf{n_1} \cdot (v_4 - v_3)) \text{ does not change,} \\ c' + c & \text{if } \mathrm{sgn}(\mathbf{n_1} \cdot (v_4 - v_3)) \text{ changes,} \end{cases} \tag{12}$$

where c and c' are respectively the contrast of the adjacent faces before and after the vertex displacement. We test whether the vertex displacement causes a switch in sign of $\mathbf{n_1} \cdot (v_4 - v_3)$, which implies a change in convexity between the adjacent faces. This allows us to detect the ambiguous case as shown in Fig. 9, where the displacement does not induce a change in the "conventional" contrast between the adjacent faces.

4.3 Vertex Displacement Threshold

In order to compute the threshold beyond which the displacement of a vertex v in a direction $\mathbf{dir}$ is visible, we proceed by the following steps. First, a list of the adjacent pairs of faces that are affected by the displacement of v is built. For each pair of faces, we start by computing their original perceptual properties and the corresponding contrast threshold using Eqs. (3), (4) and (8). In particular, the display and observation parameters are the inputs of the JND algorithm, therefore the proposed JND profile can be adaptively computed for different viewing distances and display sizes. Then we gradually increase the displacement magnitude of v and compute the change in frequency and contrast (Eqs. (10) and (9)) at each step. This allows us to evaluate the probability of detecting the vertex displacement (Eq. (11)) for each of the adjacent face pairs at different displacement steps. Note that when the displacement causes a change in spatial frequencies, we take into account the most sensitive frequency that results in a higher detection probability. Finally, the threshold is attributed to the displacement magnitude where the detection probability reaches a certain threshold for at least one of the face pairs. In practice we set the probability threshold at 0.95. To better understand this process, let us consider the two vertices v_1 and v_2 in Fig 10. Both vertices are displaced in their normal direction. The first vertex v_1 is situated on a rough region (initial contrast of all surrounding pairs of adjacent faces $>$ CSF threshold) and the second vertex v_2 on a smooth region (initial contrast $<$ CSF threshold). The displacement of v_1 and v_2 barely affects the spatial frequency of the surrounding face pairs as can be seen in the left plots. The middle plots show how displacing v_1 and v_2 in the normal direction affects the local contrast. The probability of detecting this change in contrast is shown in the right plots. These plots show that v_2 is more sensitive and can tolerate less displacement than v_1. This is due to the different initial contrasts of the two vertices. The initial contrasts around v_1 is above the CSF threshold. This implies that the visibility threshold is increased due to the masking effect, which explains the slow increase in detection probability. For v_2 all initial contrasts are below the CSF threshold. No masking should occur which means that once the contrast is above the CSF threshold the displacement should be visible. This is exactly what we observe. When the contrast of *"face pair 4"* reaches the CSF level then the detection probability becomes close to 1.

In the description above, we explain how to compute the displacement threshold by brute-force incremental step searching only for clarity purposes. In practice, we instead use a half-interval search to find the threshold (as described in Algorithm 1), which is simple yet very fast and accurate. In our tests we have set the visibility threshold th to 0.95, the precision p to 0.005 and the parameter $very_high_value$ to $1/10th$ of the mesh bounding box. In order to compute the value of $visibility$, we call the psychometric function (Eq. (11)) which again requires the evaluation of the change of contrast (Eq. (12)) and the contrast threshold (Eq. (8)).

Computing the displacement threshold requires an estimation of the spatial frequency and the local contrast. This makes the obtained threshold dependent on the display

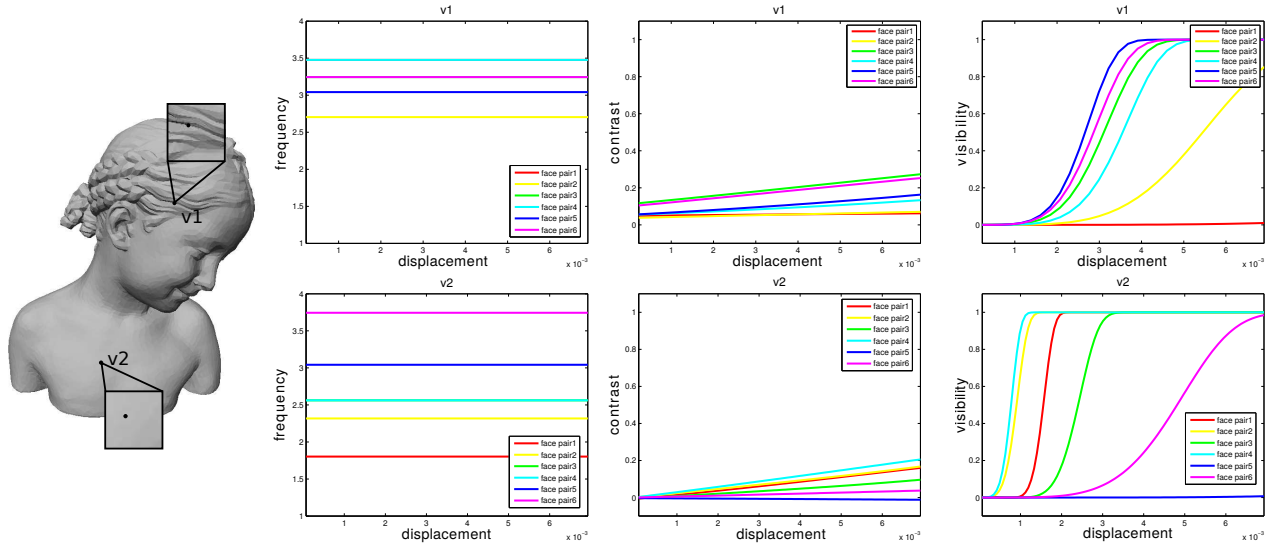


Fig. 10. The evolution of the local perceptual properties and visibility, for two displaced vertices v_1 and v_2 on the Bimba model. Left plots show the change in frequency, middle ones show the change in contrast and the right ones show the detection probability, of different pairs of affected adjacent faces of the two vertices. Note that some of the faces have the same spatial frequency, so the color curves overlap in the left plots.

Algorithm 1: Half-interval search algorithm.

Data: **v**: vertex, **dir**: noise direction, **l**: light direction, th: visibility threshold, p: precision
Result: $dist$: displacement threshold

```
min = 0;
max = very_high_value;
dist = max;
visibility = compute_visibility(v, dir, l, dist);
while || visibility − th || > p do
    dist = (max - min) / 2 + min;
    visibility = compute_visibility(v, dir, l, dist);
    if visibility > th then
        max = dist;
    else
        min = dist;
    end
end
```

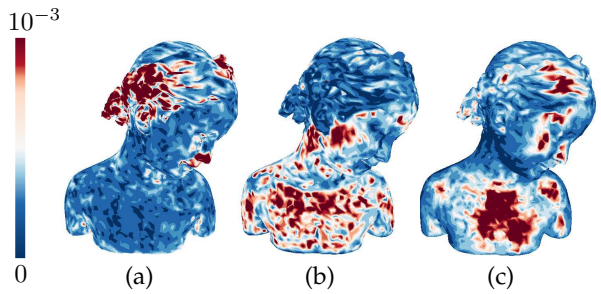


Fig. 11. The JND profile for the Bimba model under different circumstances. (a) The threshold is computed with respect to a displacement in the normal direction of each vertex in a light independent mode. (b) The threshold is computed with respect to a displacement in the tangent direction in a light independent mode. (c) The threshold is computed with respect to a displacement in the normal direction of each vertex with a light fixed in front of the model.

parameters (size and resolution), the observer's distance to the display and scene's illumination. However, in an interactive setting where the light source is fixed relative to the viewpoint, the light direction varies with respect to the 3D mesh. It is therefore important to compute the displacement threshold independently of the light direction.

We hereby propose a *light independent mode* for computing the displacement threshold. To do so, we simply compute the threshold according to multiple light directions and then choose the smallest one. Since the contrast between two faces is defined only when the dot product between the light direction and the normals is positive, then we consider that the possible light directions are located and sampled on the local half sphere around the displaced vertex. In practice, it is not necessary to densely sample the entire half sphere; Eq. (3) implies that if the light direction is close to the normal, the contrast varies slowly. This means that the displacement threshold is bigger for light directions that are close to the normal direction than for light directions close to the tangent direction. This suggests that it would only be necessary to sample the half sphere near the tangent direction.

In order to obtain the JND profile of a 3D mesh, we compute for each of the vertices the displacement threshold relative to a certain direction. Figure 11 shows the JND profile for a mesh under different circumstances. Figure 11.(a) displays the JND profile relative to a displacement in the normal direction in a light independent mode. Due to the effects of contrast masking, the rough region of the model can tolerate more noise than the smooth part. This is not the case when the JND is computed relative to a displacement in the tangent direction (Fig. 11.(b)) where the smooth part can tolerate more displacement. This is because a displacement in the tangent direction for a smooth vertex will barely alter the normal of the surrounding faces and thus the local contrast will not be affected by the displacement. Figure 11.(c) shows the JND profile relative to a displacement in the normal direction when the light source is fixed. As expected, we can see that the obtained threshold is maximal when the surface normals are in the same direction of the light.

5 Subjective Validation

In order to test the performance of a Just Noticeable Distortion profile, it is common in the image or video JND context to perform a subjective experiment [44], [45], [46] where a JND modulated random noise is added to the images or videos. The participants should then rate the visibility of the displayed noise. A JND model should be able to inject noise into the image or video while keeping it invisible; the best JND model being the one that is able to add the largest amount of invisible noise. We have conducted a subjective experiment where we have tested the performance of the proposed JND model. We compared the visibility of noise on altered 3D meshes, which were obtained by adding three different types of noise to an initial mesh. The three types of noise are:

- uniform random noise without any modulation;
- random noise modulated by the surface roughness;
- random noise modulated by the proposed JND model.

Surface roughness is an important candidate to test our JND model against since it is accepted in the computer graphics community that noise is less visible in rough regions [19]. We also have reached this conclusion when the noise is in the normal direction (Fig. 11.(a)).

5.1 Mesh Alteration

We injected noise into 3D meshes according to the following equation:

$$v_i^{'} = v_i + \mathrm{rnd} \times M\left(v_i\right) \times \mathbf{dir}_i, \tag{13}$$

where v_i is the ith vertex of the initial mesh and $v_i^{'}$ is the corresponding noisy vertex. **dir** is the noise direction. rnd is a random value equal to either $+1$ or -1 and $M\left(v_i\right)$ represents the magnitude of the noise for v_i. It is defined as:

$$M\left(v_i\right) = \begin{cases} \beta_{unif} & \text{uniform noise,} \\ \beta_{rough} \times \mathrm{lr}\left(v_i\right) & \text{roughness modulated noise,} \\ \beta_{jnd} \times \mathrm{jnd}\left(v_i\right) & \text{JND modulated noise,} \end{cases} \tag{14}$$

where β_{unif}, β_{rough} and β_{jnd} regulate the global noise energy for each of the noise injection methods. $\mathrm{lr}\left(v_i\right)$ is the local surface roughness as defined in [21] and $\mathrm{jnd}\left(v_i\right)$ is the JND value computed as explained in Section 4.3. In order to allow user interaction during the experiment, the JND value was computed independently from any light direction.

For the subjective experiments we injected noises of two different energy levels: $\beta_{jnd} = 1$ and $\beta_{jnd} = 2$. These levels correspond to a near-threshold noise and to a supra-threshold noise, respectively. For $\beta_{jnd} = 1$ the injected noise is supposed to be difficult to notice while for $\beta_{jnd} = 2$ the noise is expected to be visible. We then fix β_{unif} and β_{rough} such that for the meshes altered using our JND model, the maximum root mean square error (MRMS) [1], [2], a widely used purely geometric distance, is the biggest for each noise level. Indeed, the objective here is to show that our JND model is able to inject the highest amount of noise onto the mesh among the three methods, while producing the least visible one. In addition, we tested the performance of the JND model for noise in a random direction for each vertex and that in the normal direction for each vertex. To see the effects of light direction we ran the experiment twice: once with the light source in front of the model and another time with the light on top left of the model.

5.2 Method

Procedure. The subjective experiment followed the "adjectival categorical judgment method" [47]. This procedure consists of displaying two 3D meshes side by side, the reference on the left and the noisy one on the right. The participants were asked to rate the visibility of the noise on a discrete scale from 0 to 5, 0 being the score attributed when the noise cannot be seen and 5 when the noise is clearly visible. 5 "dummy" models were included at the beginning of each session to stabilize subjective scores. The models were presented in a randomized order. To avoid any memory-based bias, two meshes derived from the same reference model were never displayed consecutively.

Settings. The experiment was conducted in a low illuminated environment. We used a 23-inch Asus screen with a 1920×1080 resolution to display the 3D models. The participants viewed the models from a distance of 50 cm. During the experiment, the two displayed meshes had a synchronized viewpoint and subjects could freely rotate around the displayed meshes. To encourage close examination of the displayed mesh, no score could be registered before 10 seconds of interaction occur. The initial viewpoint was manually set for all models. The light source was fixed with reference to the camera position. A front and a top-left light directions were used.

Participants. 12 subjects (7 females and 5 males) participated in these experiments. All of them had normal or corrected-to-normal vision and were between the age of 20 and 29.

5.3 Results

After collecting the subjective scores, we have computed the mean score over each of the noise types. "JND 1" and "JND 2" refer to the models obtained by modulating the random noise with our JND model for near-threshold and supra-threshold levels, respectively. "Rough 1" and "Rough 2" refer to the ones obtained using the surface roughness measure and "Unif 1" and "Unif 2" to the ones with uniform random noise. Figure 12 displays the results of the subjective experiments. Plots (a) to (c) present the results for the noise in the normal direction and plots (d) to (e) the results for the noise in a random direction. Figures 12.(a) and 12.(d) show that the noise on the "JND 1" models was indeed difficult to detect as the mean subjective score is about 0.45. Interestingly, the participants rated "Unif 1" and "Rough 1" models similarly to "JND 2" which refers to the supra-threshold noise level models that contain twice the noise of "Unif 1" and "Rough 1". Plots (b) and (e) also show that "JND 1" models were perceived almost identically both under front and top-left illumination conditions. This is not the case for "Unif 1" and "Rough 1" models where the grazing light direction of the top-left illumination made the noise more apparent. It is also important to note that the visibility of the noise for "JND 1" models was identical for all models. This is not the case for "Rough 1" and "Unif

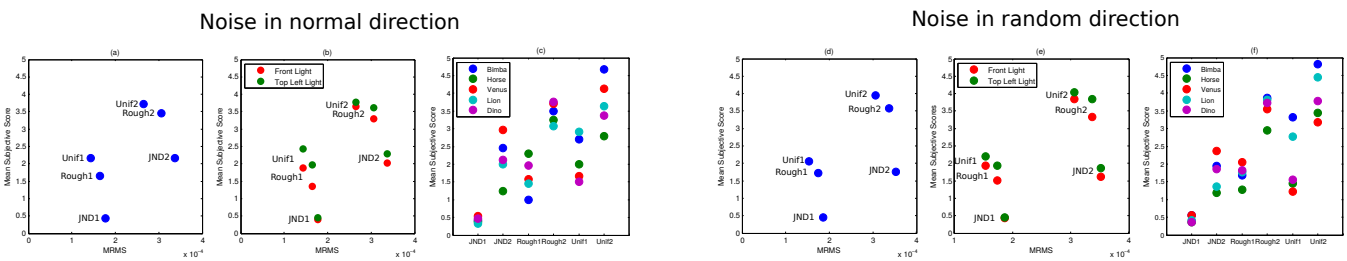
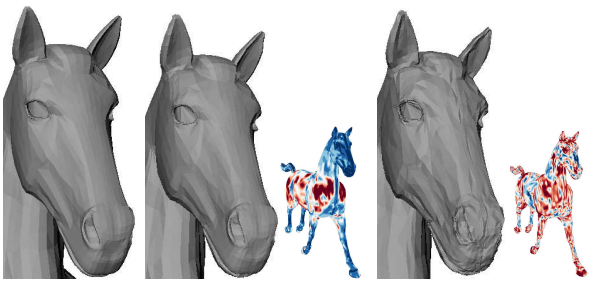


Fig. 12. Mean subjective score values versus MRMS distance values. Plots (a) and (d) present, for different noise injections, the mean subjective scores over all test models and the two illumination settings. Plots (b) and (e) show the difference in mean subjective scores between the experiments in the two illumination settings. Plots (c) and (f) compare the mean subjective scores for the different models used in the experiments.



Original (a) JND 1 (b) Rough 1

Fig. 13. The proposed JND model takes into account the density of mesh when computing the visibility threshold. When adding random noise modulated by the JND profile, the noise will be added to the coarse regions and avoid the dense area where the noise will be easily visible (a). This is not the case when adding random noise modulated by the surface roughness (b).

1" where the visibility of noise varied a lot for different models (see Figs. 12.(c) and 12.(f)). This is mainly due to the difference in mesh density between the models; high density models are in general more sensitive to noise than low density ones.

The main advantage of the proposed JND model over the surface roughness measures is that it adapts to the mesh characteristics (density, size), noise direction and scene illumination. Figure 13 illustrates the importance of mesh density. The Horse is a model with mostly smooth regions, the rough regions are packed in the head's features. In addition, the head is densely sampled while the body is coarsely sampled. The JND model avoids adding noise in the dense head and takes advantage of the coarse body, while surface roughness measures are not able to detect the difference in sampling. The noise is thus rather injected in the dense head features, which makes it visible.

These results show that the proposed JND model is indeed able to add the largest amount of invisible noise onto the mesh surface among the three methods. Furthermore, the proposed JND model can accurately predict the visibility threshold for 3D meshes, taking into account the noise direction, the mesh characteristics and the scene illumination. However, the proposed model cannot accurately describe how the supra-threshold noise visibility (or annoyance) is perceived since it has not been designed for this purpose; the noise was perceived differently for each model in "JND 2" (Figs. 12.(c) and (f)).

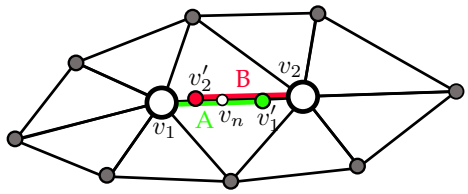


Fig. 14. If $\mathbf{v_1 v_2}$ and $\mathbf{v_1' v_2'}$ are in opposite directions, then the edge (v_1, v_2) can be collapsed to v_n without causing any visible distortion.

6 Applications

The JND models of 2D images and videos have been used extensively throughout the literature to guide and perceptually optimize several image and video processing algorithms [6], [7], [8]. In this section, we show how the proposed 3D JND profile can be integrated to mesh processing algorithms. We used the proposed JND profile to guide the simplification of 3D meshes and to automatically select the optimal vertex coordinates quantization level.

6.1 JND-Driven Mesh Simplification

The goal of mesh simplification algorithms is to reduce the number of vertices in a mesh to a certain degree by iteratively applying a simplification step (edge collapse or vertex removal). Mesh simplification is usually used to efficiently display highly detailed models or to create multiple levels of details (LOD) of a mesh, so it is required that the simplified mesh preserves the geometric features of the model as much as possible. To do so, a simplification cost is assigned to each of the mesh edges (or vertices), then the simplification step is applied to the edge (or vertex) with the lowest cost and finally the costs are updated prior to the next iteration. Several perceptual methods have been proposed to compute the simplification cost. However, existing perceptual methods either carry out the perceptual analysis on the rendered image [35], [36], [37] or rely on a top-down estimation of saliency [26], [28], [29]. Moreover, none of the existing algorithms propose a method to automatically control the quality of the resulting output; the simplification is usually carried out until a manually prescribed number of edges or vertices is reached.

We use our JND model to define both the simplification cost for each edge and a stopping criterion that automatically controls the quality of the simplified mesh.

Edge cost. In an edge collapse operation, an edge (v_1, v_2) is removed and is replaced by a vertex v_n (Fig. 14). This can

be seen as if the vertices v_1 and v_2 moved towards the new vertex v_n. Using our JND model we analyze the visibility of displacing v_1 and v_2 along the edge (v_1, v_2). Let A (resp. B) be a part of (v_1, v_2) bounded by v_1 and v_1' (resp. v_2 and v_2') (see Fig. 14) where v_1' (resp. v_2') is the vertex obtained by displacing v_1 (resp. v_2) by exactly the JND value in the direction of $\mathbf{v_1v_2}$ (resp. $\mathbf{v_2v_1}$). This means that replacing v_1 (resp. v_2) by a vertex belonging to A (resp. B) will not cause any visible distortion. In order to apply an edge collapse that is invisible to a human observer, we need to find a new vertex v_n such that $v_n \in A \cap B$. This requires that the vectors $\mathbf{v_1v_2}$ and $\mathbf{v_1'v_2'}$ should be in opposite directions so that $A \cap B \neq \emptyset$. Otherwise, if $\mathbf{v_1v_2}$ and $\mathbf{v_1'v_2'}$ are in the same direction, then we have $A \cap B = \emptyset$, making the distortion caused by the edge collapse visible. This analysis leads us to define the simplification cost of an edge by:

$$c = \frac{\mathbf{v_1v_2}.\mathbf{v_1'v_2'}}{||\mathbf{v_1v_2}||^2}. \tag{15}$$

The value of our simplification cost c varies between $[-1, 1]$. If $c < 0$ then the collapse operation does not affect the visual fidelity of the model. If $c > 0$ then the edge collapse will be visible. Figure 15 shows the simplification cost on a cube where we have injected a random noise on each of its sides. The simplification cost of the edges belonging to the top side is below 0 as the injected noise is under the JND threshold.

Vertex placement. Having defined the simplification cost of an edge, we now should decide how the position of the new vertex v_n is computed. In order to get the "optimal" position we have found that minimizing the following quadratic energy produce very good results:

$$\arg\min \left\{ \left(\frac{||\mathbf{v_1v_n}||}{\mathrm{jnd}_{\mathrm{v_1}}} \right)^2 + \left(\frac{||\mathbf{v_2v_n}||}{\mathrm{jnd}_{\mathrm{v_2}}} \right)^2 \right\}, \tag{16}$$

where $\mathrm{jnd}_{\mathrm{v_1}}$ (resp. $\mathrm{jnd}_{\mathrm{v_2}}$) is the JND threshold of v_1 (resp. v_2) in the direction of $\mathbf{v_1v_2}$ (resp. $\mathbf{v_2v_1}$). This yields to:

$$||\mathbf{v_1v_n}|| = ||\mathbf{v_1v_2}|| \times \frac{\mathrm{jnd}_{\mathrm{v_1}}^2}{\mathrm{jnd}_{\mathrm{v_1}}^2 + \mathrm{jnd}_{\mathrm{v_2}}^2}, \tag{17}$$

where $||\mathbf{v_1v_n}||$ and $||\mathbf{v_2v_n}||$ represent respectively the distances by which v_1 and v_2 are being displaced. The idea behind minimizing this quadratic energy is to make the displacement of v_1 and v_2 adaptive to their corresponding JND values.

Stopping criterion. The value of the defined simplification cost varies between $[-1, 1]$. For edges with a cost greater than 0 the collapse operation will be visible. So in order to have a simplified mesh that is visually similar to the original version, we collapse all the edges whose cost is less than or equal to 0. This allows us to define a stopping criterion which consists in stopping the simplification process once all edges have a simplification cost above 0. Figure 16 shows a highly dense 3D mesh. The model is then simplified with the JND-driven simplification method. The resulting simplified mesh (Fig. 16.(a)) has 80% less vertices and is visually very similar to the original version. Removing 5% more vertices beyond the JND level introduces slightly visible distortions to the model (Fig. 16.(b)). In addition, simplifying the model using Lindstrom and Turk's method [48] (edge collapse with a different cost) to the same number of vertices as the JND-driven simplification also results in slightly visible distortions (Fig. 16.(c)). The mesh LOD results of the simplification application can be found in the supplementary material submitted along with the manuscript.

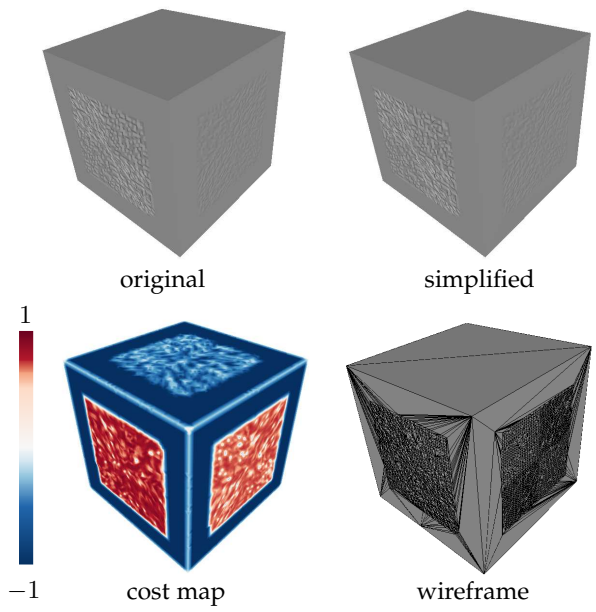


Fig. 15. A random noise of different intensities is injected to different sides of a dense cube mesh. The noise on the top side is below the JND threshold. On the right side, the noise is barely visible as it is just above the JND threshold and on the left side is injected a visible noise. The JND driven simplification process will keep all of the visible noise and simplify the top side with noise that is below the visibility threshold.

6.2 Vertex Coordinates Quantization

Vertex coordinates quantization is an important step in many mesh processing algorithms, especially compression. This operation may introduce visible distortion to the original mesh. It is thus important to find the optimal quantization level (in bits per coordinate, bpc), which is different for each mesh due to differences in geometric complexities and details. We define the optimal quantization level as the one with the highest quantization noise energy that remains visually indistinguishable from the original mesh.

The proposed JND model provides a simple and automatic way to determine the optimal quantization level independently of the nature of the mesh. The idea is to compute a score allowing us to compare the model's JND profile to the magnitude of introduced noise. To do so, we start by computing the displacement vectors as:

$$\mathbf{disp}_i = v_i^{'} - v_i, \tag{18}$$

where $v_i^{'}$ and v_i are the ith vertices of respectively the distorted mesh and the original one. The direction of $\mathbf{disp}_i$ represents the quantization noise direction. We then compute the JND profile of the original mesh with respect to the computed displacement direction. We finally compute

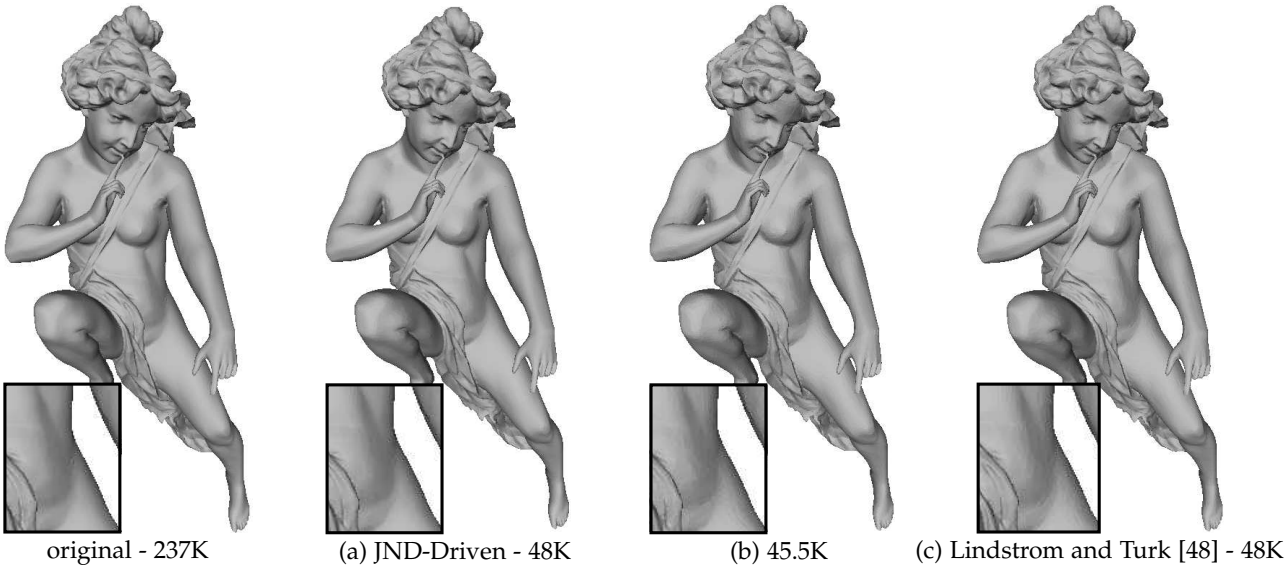


Fig. 16. (a) The JND-driven mesh simplification process outputs a model that is visually very similar to the original model. (b) Removing 5% more vertices will introduce slightly visible distortions to the simplified model. (c) The simplified model by using the method of Lindstrom and Turk [48] to the same number of vertices as the JND-driven simplification. (b) and (c) contain slightly visible distortions, especially on the belly and thighs.

the score as the mean of the ratio between the norm of the displacement vector and the JND value:

$$S = \frac{1}{n} \sum_{i=1}^{n} \frac{\| \mathbf{disp}_i \|}{\mathrm{jnd}\,(v_i)}, \tag{19}$$

where n is the number of vertices in the mesh. This score allows us to test whether the introduced distortion is visible. If $S < 1$, the noise magnitude is globally below the visibility threshold, which means that the distortion is not visible. On the other hand if $S > 1$, the distortion becomes visible as the noise magnitude is above the visibility threshold. Figure 17 shows the JND comparison scores versus the level of coordinates quantization for three different meshes. According to the defined score the optimal quantization level is respectively 12, 11 and 10 bpc for the Venus, Head and Bimba models. These results are consistent with human observations as shown in Fig. 18. Figure 17 shows also the FMPD [21] scores versus the level of coordinates quantization for the three meshes. We cannot define a proper threshold on the FMPD [21] scores that gives the same optimal quantization levels for the Venus, Head and Bimba models. One possible explanation is that the FMPD metric has difficulties in producing consistent evaluation results on meshes of different densities and geometric complexities.

7 Limitations

One of the limitations of the proposed JND model is that it currently only works for diffuse surface that is illuminated by a white directional light and rendered with a flat shading algorithm. This is due to the simplified contrast definition under that circumstance which is proposed in Section 3.1. However, the JND threshold is based on the estimation of visibility which is obtained using low-level proprieties of human visual system (CSF and contrast masking) and relies heavily on an estimation of local contrast. This means that extending the JND model to different types of surfaces and

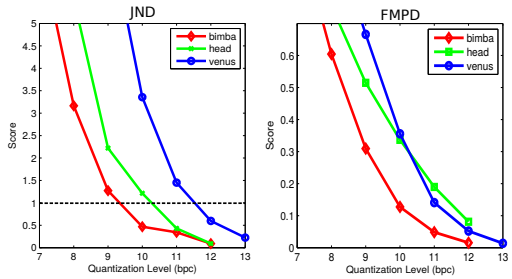


Fig. 17. Left: The JND comparison score versus the quantization levels (in bpc) of three meshes. Right: The FMPD [21] score versus the quantization levels (in bpc) of the three meshes.

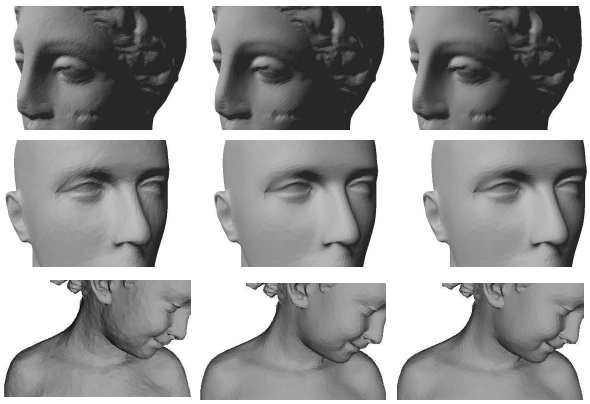


Fig. 18. Quantized meshes with different quantization levels. The middle mesh corresponds to the optimal quantization level (12, 11 and 10 bpc for Venus, Head and Bimba respectively). The right mesh corresponds to a one bit higher than the optimal level while the one on left corresponds to a one bit lower. (For better comparison between the models please refer to the electronic version of this manuscript.)

lights such as specular surfaces and point light illumination requires a generalization of the definition of contrast under the corresponding surface and lighting condition, probably via an appropriate analysis of the rendering algorithm. This will be the focus of our future work.

Another limitation of the proposed JND model is that it relies on low-level properties of the human visual system such as the contrast sensitivity function and contrast masking. These low-level properties allow us to predict whether a distortion is visible or not. If a distortion is visible, the current JND model cannot accurately predict to which extent this distortion affects the visual fidelity of the model. Taking this into consideration requires adding to the model some higher-level properties of the human visual system such as entropy masking [49] or the free energy principle [50]. We think that both of these properties could be properly defined by analyzing the local contrast of the surface in a certain neighborhood.

8 Conclusion and Future Work

In this paper, we have presented a model for computing a Just Noticeable Distortion (JND) profile for flat-shaded 3D meshes. The proposed model takes into consideration the different mesh properties (size, density) and the varying circumstances of mesh usage (display parameters, light direction, viewing distance). Our JND profile is based on an experimental study of the local perceptual properties of the human visual system, *i.e.*, the local contrast and the spatial frequency. These perceptual properties have been defined for 3D meshes. They were then used to experimentally measure the effects of the *contrast sensitivity function* and *contrast masking* when the displacement of a vertex occurs. The results of these experiments have been utilized to evaluate the probability of detecting the displacement of a vertex in an arbitrary direction, which allows us to define a JND profile for flat-shaded 3D meshes. We have tested the performance of the proposed JND model via a subjective experiment where the participants had to rate the visibility of JND modulated random noise added to a series of models. The results show that our model can accurately predict the visibility threshold of vertex noise.

We have used the proposed JND model to guide the simplification of 3D meshes. The JND-driven simplification method relies on a perceptual simplification cost assigned to each edge, and it can automatically stop the simplification process in order to obtain a visually very similar simplified mesh. Finally, we have proposed a method to automatically obtain the optimal vertex coordinates quantization level.

Our future work will first focus on generalizing the contrast definition of 3D meshes. This will broaden the usage of the proposed JND model to include smooth shaded surface and different types of illumination. We will also work to add higher aspects of the human visual system to the JND, which will allow us to predict the visibility/annoyance of supra-threshold geometric distortions. Interestingly, there have also been several studies on the perception of dynamic 3D meshes recently [51], [52]. By incorporating the dynamic aspects of the human visual system we may be able to extend the JND model to dynamic meshes.

Acknowledgments

We would like to thank all the subjects who participated in the subjective experiments. This work is supported by the ARC6 program of the "Région Rhône-Alpes" through the PADME project. We thank the anonymous reviewers for their very helpful comments and suggestions.

References

[1] P. Cignoni, C. Rocchini, and R. Scopigno, "Metro: Measuring error on simplified surfaces," *Computer Graphics Forum*, vol. 17, no. 2, pp. 167–174, 1998.

[2] N. Aspert, D. Santa-Cruz, and T. Ebrahimi, "MESH: Measuring error between surfaces using the Hausdorff distance," in *Proc. of IEEE International Conference on Multimedia & Expo*, 2002, pp. 705–708.

[3] M. Corsini, M.-C. Larabi, G. Lavoué, O. Petřík, L. Váša, and K. Wang, "Perceptual metrics for static and dynamic triangle meshes," *Computer Graphics Forum*, vol. 32, no. 1, pp. 101–125, 2013.

[4] W. Lin, "Computational models for just-noticeable difference," in *Digital Video Image Quality and Perceptual Coding*, H. R. Wu and K. R. Rao, Eds. London, UK: CRC Press, 2006, pp. 281–303.

[5] W. Lin, L. Dong, and P. Xue, "Visual distortion gauge based on discrimination of noticeable contrast changes," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 7, pp. 900–909, 2005.

[6] C.-H. Chou and Y.-C. Li, "A perceptually tuned subband image coder based on the measure of just-noticeable-distortion profile," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 6, pp. 467–476, 1995.

[7] Z. Liu, L. J. Karam, and A. B. Watson, "JPEG2000 encoding with perceptual distortion control," *IEEE Transactions on Image Processing*, vol. 15, no. 7, pp. 1763–1778, 2006.

[8] Z. Wei and K. Ngan, "Spatio-temporal just noticeable distortion profile for grey scale image/video in DCT domain," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 3, pp. 337–346, 2009.

[9] R. W. Fleming and M. Singh, "Visual perception of 3D shape," in *Proc. of ACM SIGGRAPH Courses*, 2009, pp. 24:1–24:94.

[10] G. Lavoué and M. Corsini, "A comparison of perceptually-based metrics for objective evaluation of geometry processing," *IEEE Transactions on Multimedia*, vol. 12, no. 7, pp. 636–649, 2010.

[11] B. A. Wandell, *Foundations of Vision*. Sunderland, MA, USA: Sinauer Associates, 1995.

[12] A. Beghdadi, M.-C. Larabi, A. Bouzerdoum, and K. M. Iftekharuddin, "A survey of perceptual image processing methods," *Signal Processing: Image Communication*, vol. 28, no. 8, pp. 811–831, 2013.

[13] W. Lin and C.-C. J. Kuo, "Perceptual visual quality metrics: A survey," *Journal of Visual Communication and Image Representation*, vol. 22, no. 4, pp. 297–312, 2011.

[14] S. E. Palmer, *Vision Science: Photons to Phenomenology*. Cambridge, MA, USA: MIT Press, 1999.

[15] C. T. Blakemore and F. Campbell, "On the existence of neurones in the human visual system selectively sensitive to the orientation and size of retinal images," *The Journal of Physiology*, vol. 203, no. 1, pp. 237–260, 1969.

[16] D. G. Pelli and P. Bex, "Measuring contrast sensitivity," *Vision Research*, vol. 90, pp. 10–14, 2013.

[17] A. B. Watson and A. J. Ahumada, "A standard model for foveal detection of spatial contrast ModelFest experiment," *Journal of Vision*, vol. 5, no. 9, pp. 717–740, 2005.

[18] G. E. Legge and J. M. Foley, "Contrast masking in human vision," *Journal of Optical Society of America*, vol. 70, no. 12, pp. 1458–1471, 1980.

[19] G. Lavoué, "A local roughness measure for 3D meshes and its application to visual masking," *ACM Transactions on Applied Perception*, vol. 5, no. 4, pp. 1–23, 2009.

[20] M. Corsini, E. Drelie Gelasca, T. Ebrahimi, and M. Barni, "Watermarked 3D mesh quality assessment," *IEEE Transactions on Multimedia*, vol. 9, no. 2, pp. 247–256, 2007.

[21] K. Wang, F. Torkhani, and A. Montanvert, "A fast roughness-based approach to the assessment of 3D mesh visual quality," *Computers & Graphics*, vol. 36, no. 7, pp. 808–818, 2012.

[22] L. Dong, Y. Fang, W. Lin, and H. S. Seah, "Objective visual quality assessment for 3D meshes," in *Proc. of International Workshop on Quality of Multimedia Experience*, 2014, pp. 1–6.
[23] G. Lavoué, "A multiscale metric for 3D mesh visual quality assessment," *Computer Graphics Forum*, vol. 30, no. 5, pp. 1427–1437, 2011.
[24] F. Torkhani, K. Wang, and J.-M. Chassery, "A curvature-tensor-based perceptual quality metric for 3D triangular meshes," *Machine Graphics & Vision*, vol. 23, no. 1–2, pp. 59–82, 2014.
[25] L. Váša and J. Rus, "Dihedral angle mesh error: a fast perception correlated distortion measure for fixed connectivity triangle meshes," *Computer Graphics Forum*, vol. 31, no. 5, pp. 1715–1724, 2012.
[26] C. H. Lee, A. Varshney, and D. W. Jacobs, "Mesh saliency," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 659–666, 2005.
[27] X. Chen, A. Saparov, B. Pang, and T. Funkhouser, "Schelling points on 3D surface meshes," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 29:1–29:12, 2012.
[28] J. Wu, X. Shen, W. Zhu, and L. Liu, "Mesh saliency with global rarity," *Graphical Models*, vol. 75, no. 5, pp. 255–264, 2013.
[29] R. Song, Y. Liu, R. R. Martin, and P. L. Rosin, "Mesh saliency via spectral processing," *ACM Transactions on Graphics*, vol. 33, no. 1, pp. 1–17, 2014.
[30] S. Marras, L. Váša, G. Brunnett, and K. Hormann, "Perception-driven adaptive compression of static triangle meshes," *Computer-Aided Design*, vol. 58, pp. 24–33, 2015.
[31] S. Daly, "The visible differences predictor: An algorithm for the assessment of image fidelity," in *Digital Images and Human Vision*, A. B. Watson, Ed. Cambridge, MA, USA: MIT Press, 1993, pp. 179–206.
[32] R. Mantiuk, K. J. Kim, A. G. Rempel, and W. Heidrich, "HDR-VDP-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions," *ACM Transactions on Graphics*, vol. 30, no. 4, pp. 40:1–40:13, 2011.
[33] M. Ramasubramanian, S. Pattanaik, and D. Greenberg, "A perceptually based physical error metric for realistic image synthesis," in *Proc. of ACM SIGGRAPH*, 1999, pp. 73–82.
[34] G. Ramanarayanan, J. Ferwerda, B. Walter, and K. Bala, "Visual equivalence: towards a new standard for image fidelity," *ACM Transactions on Graphics*, vol. 26, no. 3, pp. 76:1–76:11, 2007.
[35] N. Williams, D. Luebke, J. D. Cohen, M. Kelley, and B. Schubert, "Perceptually guided simplification of lit, textured meshes," in *Proc. of ACM Symposium on Interactive 3D Graphics*, 2003, pp. 113–121.
[36] L. Qu and G. W. Meyer, "Perceptually guided polygon reduction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 5, pp. 1015–1029, 2008.
[37] N. Menzel and M. Guthe, "Towards perceptual simplification of models with arbitrary materials," *Computer Graphics Forum*, vol. 29, no. 7, pp. 2261–2270, 2010.
[38] I. Cheng and P. Boulanger, "A 3D perceptual metric using just-noticeable-difference," in *Proc. of Eurographics Short Papers*, 2005, pp. 97–100.
[39] I. Cheng, R. Shen, X. D. Yang, and P. Boulanger, "Perceptual analysis of level-of-detail: The JND approach," in *Proc. of IEEE International Symposium on Multimedia*, 2006, pp. 533–540.
[40] Y. Yang, N. Peyerimhoff, and I. Ivrissimtzis, "Linear correlations between spatial and normal noise in triangle meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 1, pp. 45–55, 2013.
[41] A. B. Watson and D. G. Pelli, "QUEST: a Bayesian adaptive psychometric method." *Perception & Psychophysics*, vol. 33, no. 2, pp. 113–120, 1983.
[42] J. Mannos and D. J. Sakrison, "The effects of a visual fidelity criterion of the encoding of images," *IEEE Transactions on Information Theory*, vol. 20, no. 4, pp. 525–536, 1974.
[43] M. J. Mayer and C. W. Tyler, "Invariance of the slope of the psychometric function with spatial summation," *Journal of Optical Society of America A*, vol. 3, no. 8, pp. 1166–1172, 1986.
[44] A. Liu, W. Lin, M. Paul, C. Deng, and F. Zhang, "Just noticeable difference for images with decomposition model for separating edge and textured regions," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 11, pp. 1648–1652, 2010.
[45] Y. Zhao, Z. Chen, C. Zhu, Y.-P. Tan, and L. Yu, "Binocular just-noticeable-difference model for stereoscopic images," *IEEE Signal Processing Letters*, vol. 18, no. 1, pp. 19–22, 2011.
[46] J. Wu, G. Shi, W. Lin, A. Liu, and F. Qi, "Just noticeable difference estimation for images with free-energy principle," *IEEE Transactions on Multimedia*, vol. 15, no. 7, pp. 1705–1710, 2013.
[47] International Telecommunication Union, Rec. BT.500: Methodology for the Subjective Assessment of the Quality of Television Pictures, 2012.
[48] P. Lindstrom and G. Turk, "Fast and memory efficient polygonal simplification," in *Proc. of IEEE Visualization Conference*, 1998, pp. 279–286.
[49] L. Dong, Y. Fang, W. Lin, C. Deng, C. Zhu, and H. S. Seah, "Exploiting entropy masking in perceptual graphic rendering," *Signal Processing: Image Communication*, vol. 33, pp. 1–13, 2015.
[50] K. Friston, "The free-energy principle: a unified brain theory?" *Nature Reviews Neuroscience*, vol. 11, no. 2, pp. 127–138, 2010.
[51] L. Váša and V. Skala, "A perception correlated comparison method for dynamic meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 2, pp. 220–230, 2011.
[52] F. Torkhani, K. Wang, and J.-M. Chassery, "Perceptual quality assessment of 3D dynamic meshes: Subjective and objective studies," *Signal Processing: Image Communication*, vol. 31, pp. 185–204, 2015.

Georges Nader received his B.S. degree in 2011 from the Lebanese University (Lebanon) and his M.S degree in 2013 from the "Université de Bordeaux" (France). He is now a Ph.D. student at the "Université Claude Bernard Lyon 1" (France) under the supervision of Prof. Florent Dupont, Dr. Kai Wang and Dr. Franck Hétroy-Wheeler. His research interests include perceptually driven computer graphics and geometry processing.

Kai Wang received the Ph.D. degree in Computer Science in 2009 from the University of Lyon, Lyon, France. Following a ten-month postdoctoral position at Inria Nancy, he joined in 2011 GIPSA-Lab, Grenoble, France, as a full-time CNRS researcher. His current research interests include multimedia security and surface analysis.

Franck Hétroy-Wheeler received an Eng. degree and a M.S. degree in applied mathematics both in 2000 from the University of Grenoble Alpes, as well as a PhD in computer science from the same university in 2003. Since 2004 he has been an assistant professor at the University of Grenoble Alpes (Grenoble INP - Ensimag). He has been a member of the Morpheo team at Inria Grenoble since 2011, where he conducts research on shape analysis and understanding, digital geometry and topology processing.

Florent Dupont received his B.S. and M.S. degree in 1990, and his Ph.D. in 1994 from "Institut National des Sciences Appliquées" of Lyon, France. He became Associate Professor in 1998. He is now Professor in the Multiresolution, Discrete and Combinatorial Models (M2DisCo) team of the LIRIS Laboratory in the "Université Claude Bernard Lyon 1", France. His technical research concerns 3D digital image processing and computational geometry.

— Supplementary Material —
Just Noticeable Distortion Profile for Flat-Shaded 3D Mesh Surfaces

Georges Nader, Kai Wang, Franck Hétroy-Wheeler, and Florent Dupont

This Supplementary Material is organized as follows. Section 1 presents the results of additional psychophysical experiments that were carried out in order to verify the accuracy and robustness of the measured thresholds obtained by our original experiments. Section 2 provides some details about the accuracy and performance of computing the JND threshold, especially for the light independent mode. Section 3 presents an additional subjective experiment that validates the proposed JND profile. In Section 4 we give some details and show additional results for the mesh simplification application, including the generation of mesh levels of details (LODs) and a subjective validation. Section 5 provides some comparison results with mesh perceptual quality metrics for the application of optimal vertex coordinates quantization, as well as an additional subjective validation. Finally, in Section 6 we discuss the difference between the proposed JND model and mesh saliency measures.

1 Psychophysical Experiments

We have conducted additional psychophysical experiments in order to make sure that our previous contrast sensitivity function (CSF) and contrast masking measurements were accurate and robust. The results of this new set of experiments show that the previous measurements are indeed accurate and stable. We present the results of the new experiments in this Supplementary Material and not in the manuscript because we have validated the JND profile in Section 5 of the manuscript using the models fitted by the data from the first set of experiments. In fact, it would be quite time consuming to redo the subjective validation using the new models, and the corresponding results would be very close to the ones presented in the manuscript. Therefore, we would like to present in the manuscript the original results, which however are proven to be accurate and stable.

1.1 Experimental Procedure

We used the same experimental procedure as described in Section 3 of the manuscript. We display two models on the screen one of which has a displaced vertex. The subjects have to answer by "yes" or "no" whether they see a difference between the two models on the screen. The magnitude of the vertex displacement is then regulated using the QUEST procedure [WP83]. 5 new subjects participated in the experiment. None of them was a participant in any of our previous experiments.

1.2 Results

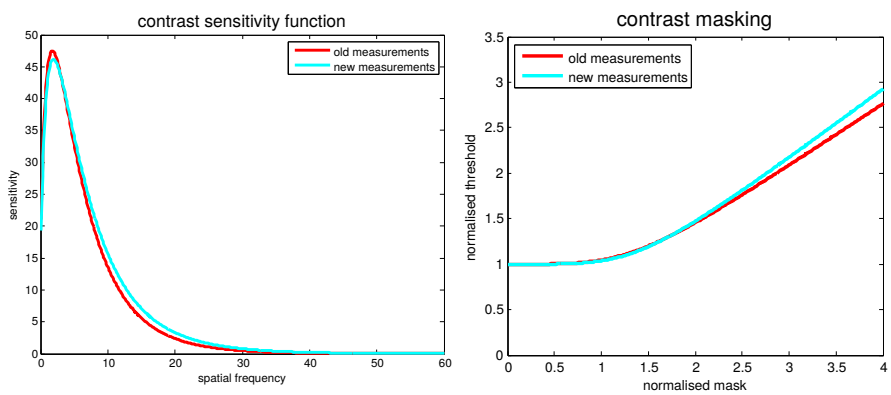


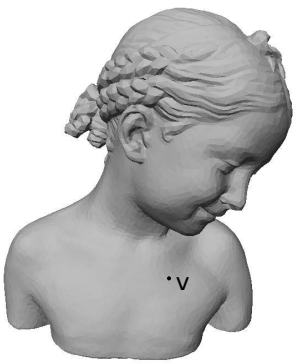
Figure 1: Plot of the fitted contrast sensitivity function and contrast masking models for the data obtained by the old and new experiments.

Figure 1 shows the fitted models from the old and new sets of experiments. The new fit of the CSF model is obtained using Eq. (6) of the manuscript with $a = -13.59$, $f = 0.001$ and $p = 0.62$ while the old fit was computed with $a = -15.13$, $f = 0.0096$ and $p = 0.64$. For the contrast masking model, the new fit is obtained from Eq. (7) of the manuscript with $k_1 = 0.006$, $k_2 = 90.66$, $s = 1.05$ and $b = 4.53$ while the old fit was computed with $k_1 = 0.0078$, $k_2 = 88.29$, $s = 1.00$ and $b = 4.207$.

2 Light Independent JND and Algorithm Speed

Here we present some details that would help readers efficiently implement the proposed JND model in particular for the light independent mode (indeed we plan to freely deliver a reference implementation in the near future). We will also report some theoretical and practical results concerning execution time of the JND computation.

2.1 Threshold Accuracy



local light direction (α, θ)	JND threshold
$(10, 85)$	0.00195312
$(10, 55)$	0.003125
$(10, 25)$	0.0078125
$(100, 85)$	0.0015625
$(100, 55)$	0.00390625
$(100, 25)$	0.0069725
$(190, 85)$	0.00107422
$(190, 55)$	0.00234375
$(190, 25)$	0.0046875
$(280, 85)$	0.00107422
$(280, 55)$	0.0021875
$(280, 25)$	0.004375
$(0, 0)$	0.015625

Figure 2: The JND threshold of a vertex v computed for different light directions.

The algorithm presented in Section 4.3 of the paper computes the JND threshold for a given light direction. However, in an interactive setting where the light source is fixed relative to the viewpoint, the light direction varies with respect to the 3D mesh. It is therefore important to compute the displacement threshold independently of the light direction.
To do so, we compute the threshold according to multiple light directions and then choose the smallest one. The light independent threshold can then be seen as the one corresponding to the worst possible illumination (*i.e.*, the light direction that makes distortions the most visible). The set of all possible light directions belongs to the sphere around a vertex. However, the contrast between two faces is only defined when the dot product between the light direction and the normals is positive. This means that the set of all possible light can be reduced to the local half sphere in the direction of the unit normal. In practice, we do not need to densely sample all the half sphere. Figure 2 shows the JND threshold obtained from different light directions belonging to the half sphere of a vertex v. We notice that as the light direction approaches the base of the half sphere, the threshold gets smaller. This implies that the worst possible illumination is at the most of time found near the base of the half sphere. This observation can also be deduced from Eq. (3) of the manuscript. Through our testing we notice that it is actually not necessary to densely sample the half sphere in order to obtain an accurate solution. It is observed that the algorithm begins to converge to an accurate JND value with 8 samples as it can be seen in Fig. 3, where the normalized root mean square error (RMSE) is computed with regard to the JND profile obtained with 64 light direction samples (shown in the rightmost of Fig. 3). In practice, in order to obtain the results presented in Sections 4 and 5 of the manuscript, we have used the 12-points sampling, as shown in Fig. 2 (excluding the point $(0, 0)$), which ensures a very good trade-off between threshold accuracy and algorithm speed.

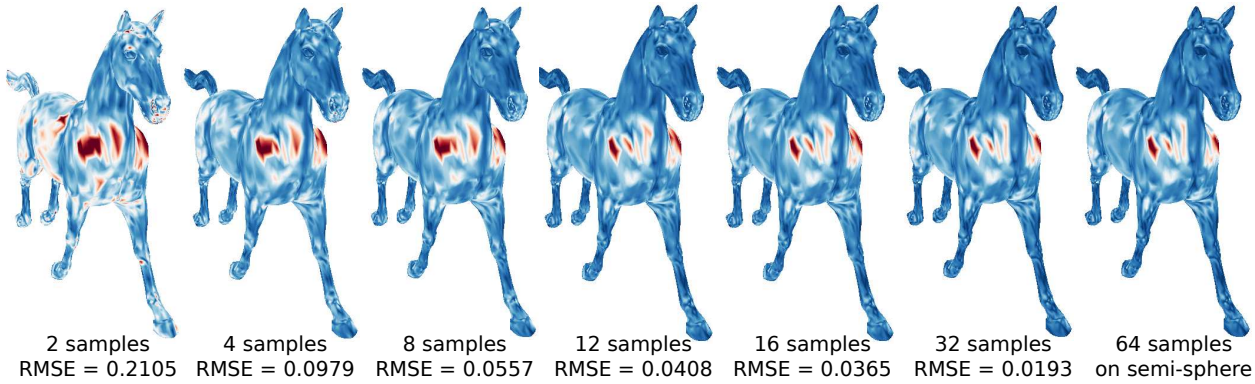


Figure 3: The effect of the number of light samples on the JND computation accuracy. The illustrated JND profiles are computed for noise in the normal direction for each vertex.

2.2 Algorithm Complexity

A theoretical analysis of the proposed JND algorithm shows that the complexity of computing the light independent JND for one vertex is equivalent to:

$$O\left(L \times \log\left(\frac{x_{max}}{x_{precision}}\right)\right), \tag{1}$$

where L is the number of light samples and x_{max} and $x_{precision}$ are respectively the upper displacement bound and the precision used in the half-interval search algorithm (Algorithm 1 of the paper).
This means that the complexity for computing the JND profile of a mesh is :

$$O\left(V \times L \times \log\left(\frac{x_{max}}{x_{precision}}\right)\right), \tag{2}$$

where V is the number of vertices. This shows that as the number of vertices increases the execution time should increase in a linear way at a rate relative to the number of light samples and the precision of the search procedure.

2.3 Execution Time

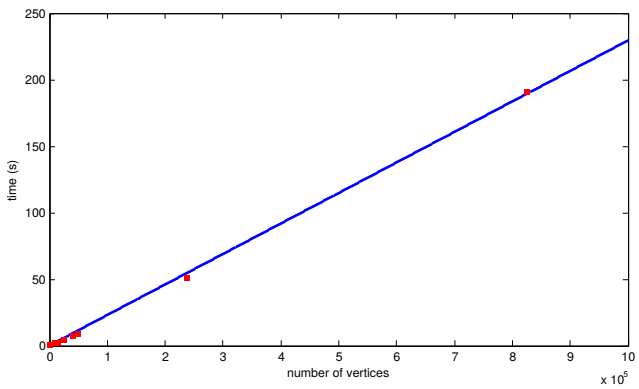


Figure 4: JND profile execution time.

Having adopted a half-interval search algorithm makes finding the JND threshold a very efficient operation. On average computing the JND threshold for a vertex in the light independent mode takes about 7×10^{-4} s. We have used an HP EliteBook 8570w with an i7-37400QM cpu (4 cores) and 16GB of RAM in our computation.
When the number of vertices increases, we have observed that the execution time increases approximately in a linear way (see Fig. 4, also analyzed in Section 2.2 of this Supplementary Material). In addition, since the JND threshold of a vertex is independent from the threshold of other vertices then the JND profile on a mesh can be computed in a parallel way. Using OpenMP, the algorithm performs about three to five times

faster. For a model with 237K vertices, the JND profile took about 52 seconds to compute. Indeed, the vertex displacement threshold searching consumes the most part of the computation time. However, with the simple but efficient half-interval search, this step can be accomplished in a very reasonable time and thus does not harm the efficiency of the whole algorithm.

3 JND Validation

We have conducted an additional subjective experiment in order to validate the proposed JND model. In the manuscript we have presented a subjective experiment where the participants rate the visibility of an injected noise on scale of 0 to 5. It showed that the models with a JND modulated noise were rated the lowest on the visibility scale but meanwhile could tolerate the biggest amount of distortions. However, in the following new experiment we gradually increase the intensity of the injected noise until the subject notices it.

3.1 Experimental Procedure

Similarly to the experiment presented in Section 5 of the manuscript, the noise is injected into the 3D meshes according to the following equation:

$$v_i^{'} = v_i + \text{rnd} \times M\left(v_i\right) \times \vec{dir}_i, \tag{3}$$

where v_i is the ith vertex of the initial mesh and $v_i^{'}$ is the corresponding noisy vertex. $\vec{dir}$ is the noise direction. rnd is a random value equal to either $+1$ or -1 and $M\left(v_i\right)$ represents the magnitude of the noise for v_i. It is defined as:

$$M\left(v_i\right) = \begin{cases} \beta_{unif} & \text{uniform noise,} \\ \beta_{rough} \times \text{lr}\left(v_i\right) & \text{roughness modulated noise,} \\ \beta_{jnd} \times \text{jnd}\left(v_i\right) & \text{JND modulated noise,} \end{cases} \tag{4}$$

where β_{unif}, β_{rough} and β_{jnd} regulate the global noise intensity for each of the noise injection methods. $\text{lr}\left(v_i\right)$ is the local surface roughness as defined in [WTM12] and $\text{jnd}\left(v_i\right)$ is the JND value computed as explained in Section 4 of the manuscript.

The idea behind this experiment is to find the minimum noise intensity (β_{unif}, β_{rough} and β_{jnd}) starting from which the participants notice the noise in the model. To do so, we have adapted the same experimental procedure that we have used to measure the local contrast threshold in the studies of CSF and contrast masking. Two models were displayed on the screen, one of which has noise injected. The subjects had to answer by either "yes" or "no" whether they saw the noise on one of the model. The intensity of the noise (β_{unif}, β_{rough} and β_{jnd}) is then adjusted using the QUEST procedure [WP83]. The subjects were allowed to interact with the displayed models by rotating the camera around them. 5 new subjects participated in the experiment.

3.2 Results

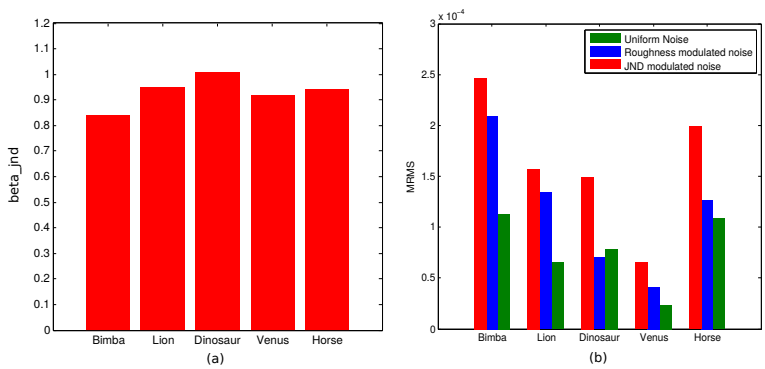


Figure 5: (a) Plot of the measured noise intensity relative to the JND modulated noise. (b) Plot of the MRMS induced by noise injection for three different types of noise at the same visibility level.

Figure 5 displays the results of this subjective experiment. Plot (a) shows the mean measured intensity required to make JND modulated noise visible on a 3D mesh. We see that the measured β_{jnd} is close to 1 for

all of the models, meaning that the proposed JND profile is able to accurately detect the threshold beyond which a noise is visible. Plot (b) shows that the MRMS value of the mesh model with JND modulated noise of just noticeable level is higher than those of the corresponding models with uniform noise or roughness modulated noise at the same visibility level. This means that the JND model is able to tolerate the highest amount of noise among the three candidates, which is what we expected.

4 JND-Driven Mesh Simplification

Integrating the JND model into the mesh simplification process allows us to define a perceptual simplification cost for the edge collapse operation. In order to obtain a simplified model that is visually very similar to the input, we run the simplification process until all the edges have a cost whose value is greater than 0. Moreover, the defined JND profile takes into consideration the size of the display, the viewing distance, and the position of the model in the virtual 3D world. This means that the JND-driven simplification can be useful for generating model level of details (LOD) as it will automatically stop the simplification at different stages for different settings. Figure 6 shows some mesh LODs generated by the JND-driven simplification method at different viewing distances.

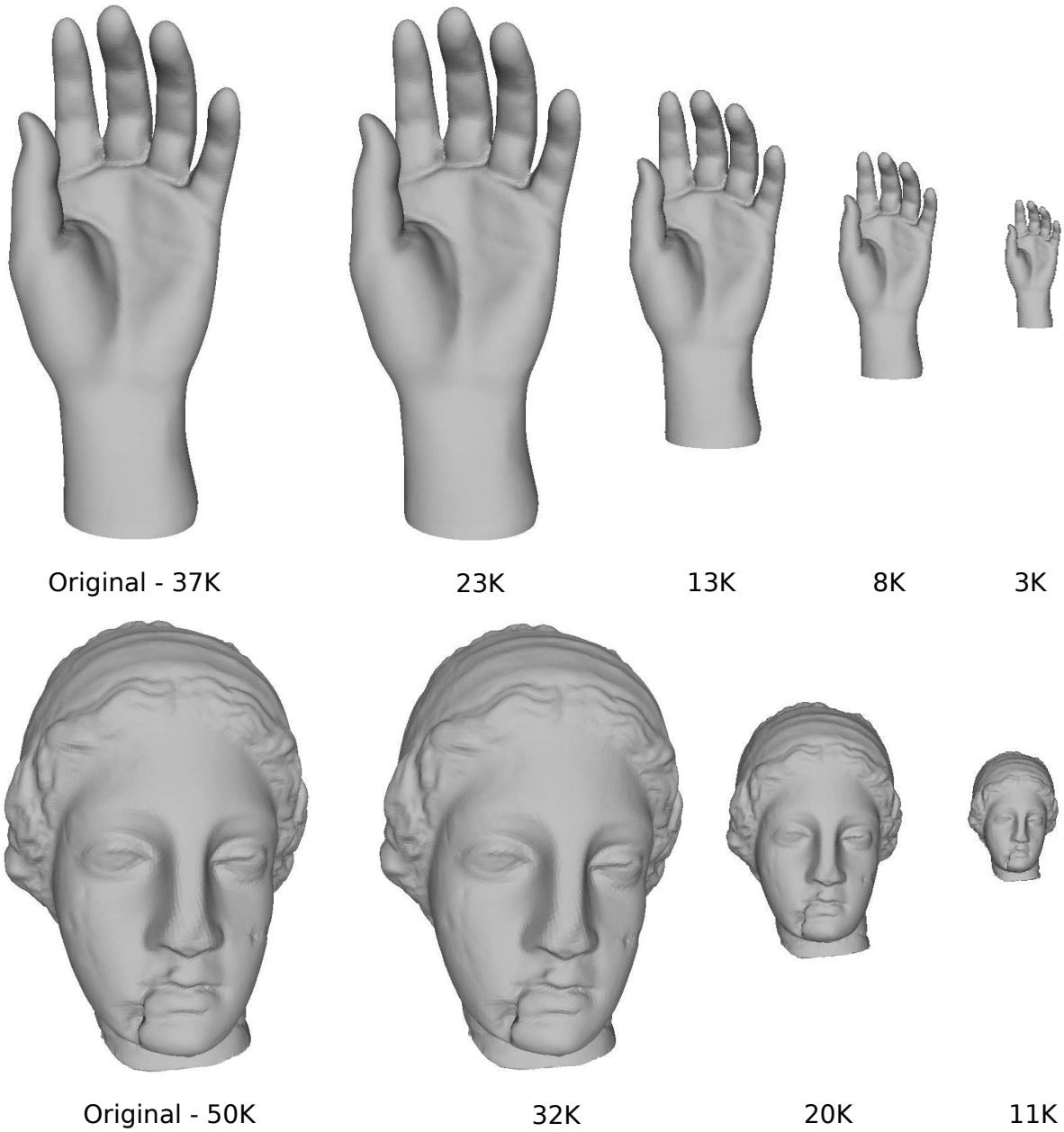


Figure 6: LODs generated using the JND-driven simplification method at different viewing distances. Different adaptive numbers of vertices are obtained under different viewing distances.

4.1 Subjective Validation

We have performed a subjective experiment in order to verify that the proposed JND-driven simplification outputs a simplified mesh that is visually very similar to the original one. 10 subjects participated in the experiment. We have adopted the same experimental procedure as the JND validation experiment presented in Section 5 of the manuscript. We displayed two models on the screen and the participants had to rate the visibility of the distortion on a scale from 0 to 5. 0 corresponds to the case where the subject cannot see any distortion and 5 to the case where the noise is clearly visible. For this experiment we included three models (Angel, Venus and Hand) with three simplified versions each: a JND-driven simplified mesh, a mesh that is further simplified to 5% beyond the JND level and a mesh simplified with Lindstrom and Turk's method to the same number of vertices as the JND-driven simplified mesh. The results of this experiment are shown in Fig. 7. It is clear that the geometric distortion that is due to the simplification process is not visible for the models simplified with the JND-driven method. The average subjective score is below 0.2 for the three models meaning that about 80% of the participants were unable to notice the distortion. For the models where we removed 5% more vertices than the JND level the geometric distortion is visible as the average subjective score given by the participants is greater than 1. As for meshes simplified with Lindstrom and Turk's method to the same number of vertices as the JND level, the mean subjective score is above 0.85 for the three models meaning that most subjects were able to see the distortions introduced by the simplification process.

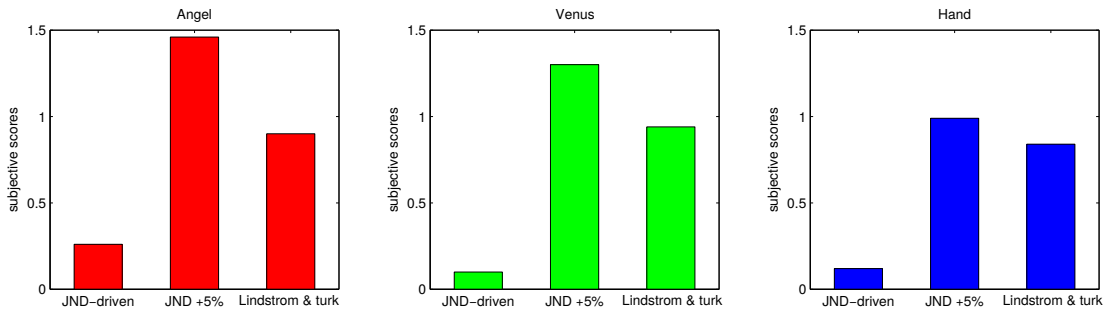


Figure 7: Plot of the subjective scores of the simplified meshes obtained by three different methods.

4.2 Implementation Details

We have implemented a prototype of the proposed perceptual mesh simplification using CGAL's mesh simplification module [Cac15]. The JND-driven simplification algorithm can be summarized as follows. The collapse cost is first computed for all the edges of a mesh and then added to a priority queue. As long as the stopping criterion has not been reached, we pick the edge with the lowest cost and perform the edge collapse operation. After each collapse operation the algorithm will update the simplification costs of the affected edges of the last edge collapse. Accordingly, we update the priority queue. A rather straightforward theoretical complexity analysis shows that the complexity of this family of mesh simplification algorithms is dominated by the complexity of managing the internal data-structure that handles the order by which the edges are being collapsed. This means that both the proposed perceptual mesh simplification and Lindstrom and Turk's simplification methods perform similarly since they are implemented using the same edge-collapse-based mesh simplification framework. From a computation perspective the only difference between the two methods consists in the computation of the edge cost. For a suitable number of light samples (in mesh simplification application we use 8 samples, number from which the JND computation starts to converge, see Section 2.1 of this Supplementary Material), the execution time for computing both the JND-based and Lindstrom and Turk's edge costs is quite similar (see Table 1 of this Supplementary Material).

Table 1: Execution time (in seconds) for computing the JND-based (with 8 light direction samples) and Lindstrom and Turk's edge costs

# of edges	9124	16056	50000	162641
JND-based	0.4	0.76	1.93	3.65
Linstrom and Turk's	0.31	0.65	1.89	3.39

5 Vertex Coordinates Quantization

For the application to optimal vertex coordinates quantization, we compare the JND scores to those of FMPD [WTM12], MSDM2 [Lav11] and MRMS. The comparison results are shown in Fig. 8 of this Supplementary Material (The comparison results of JND scores and FMPD are also illustrated in Fig. 17 of the paper). As shown by this figure, MRMS is not correlated with human perception. For FMPD and MSDM2, it is not possible to define a proper threshold that gives the correct and perceptually relevant optimal quantization levels for all the three models. One possible explanation is that FMPD and MSDM2 have difficulties in producing consistent evaluation results on meshes of different densities and geometric complexities. In addition, the main advantage of the JND model is that it does not require manually defining a threshold. Instead, it can automatically determine the optimal quantization level.

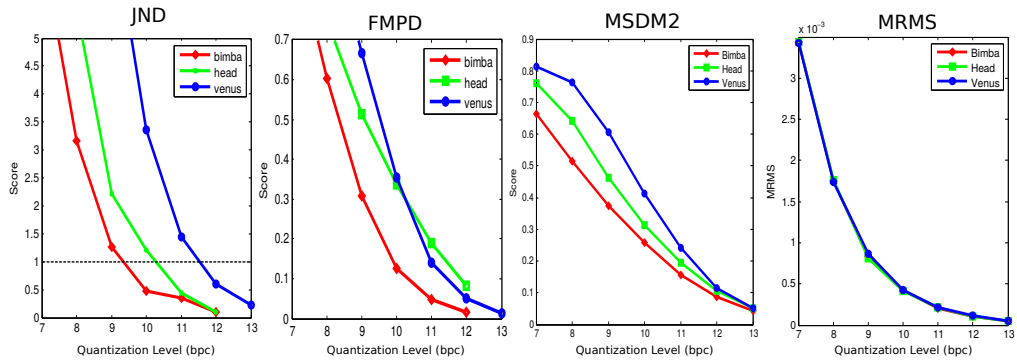


Figure 8: Comparison between the JND scores and those of FMPD [WTM12], MSDM2 [Lav11] and MRMS, versus the vertex coordinates quantization level (in bpc, bits per coordinate).

5.1 Subjective Validation

We have performed a subjective experiment in order to verify that the optimal quantization level obtained using the JND profile is indeed the one beyond which quantization noise becomes visible. 10 subjects participated in the experiment. We have adopted the same experimental procedure as the JND validation experiment presented in Section 5 of the paper. We displayed two models on the screen and the participants had to rate the visibility of the distortion on a scale from 0 to 5. 0 corresponds to the case where the subject cannot see any distortion and 5 to the case where the noise is clearly visible. For this experiment we included three models with five quantization levels each: the optimal quantization level, two immediate higher levels and two immediate lower levels. The results of this experiment are shown in Fig. 9. It is clear that the geometric distortion that is due to vertex quantization becomes visible when the quantization level in bcp become even 1 bit lower that the optimal level. For the quantization levels that are higher than the optimal one, the quantization noise is invisible as the participants rated its visibility by 0. As for the optimal quantization level the average subjective score is between 0 and 1 meaning that some participants were able to barely see the distortions while others were unable to notice it.
As a final remark concerning vertex coordinates quantization application, it is worth mentioning that the coordinates of the original models are never quantized and are represented by high-precision floating numbers.

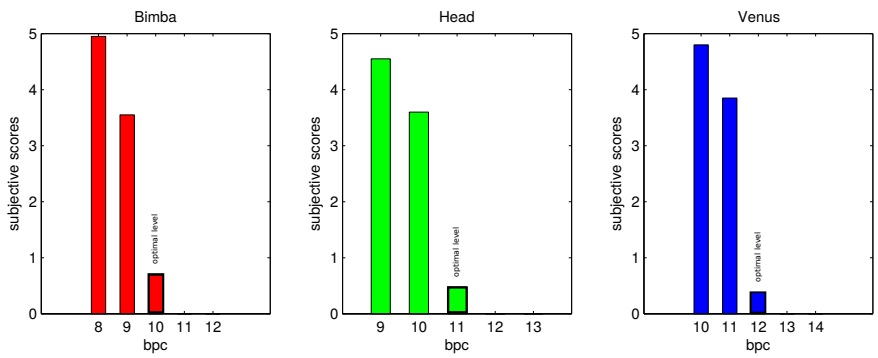


Figure 9: Plot of the subjective scores of quantized meshes with different levels of bpc.

6 JND vs Saliency

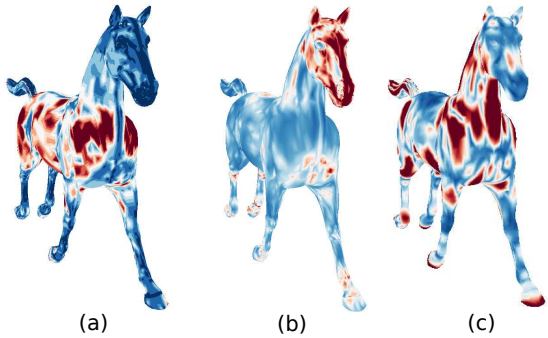


Figure 10: (a) JND profile relative to the normal direction in a light independent mode. (b) JND profile relative to a tangent direction in a light independent mode. (c) Mesh saliency value computed with the method of Lee *et al.* [LVJ05]

Mesh saliency has been the basis of many perceptual geometry processing methods [LVJ05, WSZL13, SLMR14]. By definition saliency is a measure of whether an area is visually attractive for the human visual system or not. For example, Fig 10.(c) shows that the Horse's neck, extremities of its legs and part of its head are the most visually important features (saliency computed by the method of Lee *et al.* [LVJ05]). A human observer will most likely focus his attention on these areas while observing the Horse model.

On the contrary, the proposed JND model computes the threshold beyond which a distortion becomes visible using low-level properties of the human visual system such as the contrast sensitivity function and the contrast masking effect. The JND profile in Fig. 10 indicates that the Horse's body can tolerate the most noise in the normal direction. This is because in that region the mesh is coarse, so the visibility threshold is higher due to the CSF property of the human visual system. By contrast, if the noise is in a tangent direction, then the head can tolerate the most noise. This is because in that area the geometry is relatively flat, so the displacement of a vertex in a tangent direction does not cause any change in contrast.

It is clear from the example in Fig. 10 that the saliency and JND profile measure different properties of a 3D mesh. The former points out the visually important regions that are more likely to be observed by a human being, while the latter computes the threshold beyond which a displacement of a vertex becomes visible. Integrating the JND model into a geometry processing application will allow us to (automatically) control the visibility of the introduced distortion. In JND model the main components are low-level properties of the human visual system such as CSF and contrast masking, while in mesh saliency higher-level properties such as visual attention should be taken into account.

However, it would be interesting and possible to use the low-level properties studied in the proposed JND model for the purposes of mesh saliency derivation, since a better understanding of the low-level properties would be helpful for the development of accurate higher-level properties. In particular, a salient region is by definition an area that stands out from its surrounding. It can be attributed to regions where a big change of local contrast occurs. Such regions usually attract human's visual attention. Having defined a measure of contrast in Section 3 of the manuscript, we think that it would be possible to use it in order to define a saliency measure. In addition, in perceptually oriented mesh processing, it would be beneficial to combine both low- and high-level properties of the human visual system (*e.g.*, both the concept of JND and that of mesh saliency), so as to achieve better performance or to reach a good trade-off.

References

[Cac15] F. Cacciola. Triangulated surface mesh simplification. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.7 edition, 2015.

[Lav11] G. Lavoué. A multiscale metric for 3D mesh visual quality assessment. *Computer Graphics Forum*, 30(5):1427–1437, 2011.

[LVJ05] C. H. Lee, A. Varshney, and D. W. Jacobs. Mesh saliency. *ACM Transactions on Graphics*, 24(3):659–666, 2005.

[SLMR14] R. Song, Y. Liu, R. R. Martin, and P. L. Rosin. Mesh saliency via spectral processing. *ACM Transactions on Graphics*, 33(1):1–17, 2014.

[WP83] A. B. Watson and D. G. Pelli. QUEST: a Bayesian adaptive psychometric method. *Perception & Psychophysics*, 33(2):113–120, 1983.

[WSZL13] J. Wu, X. Shen, W. Zhu, and L. Liu. Mesh saliency with global rarity. *Graphical Models*, 75(5):255–264, 2013.

[WTM12] K. Wang, F. Torkhani, and A. Montanvert. A fast roughness-based approach to the assessment of 3D mesh visual quality. *Computers & Graphics*, 36(7):808–818, 2012.

$\therefore$

A.4 A HIERARCHICAL APPROACH FOR REGULAR CENTROIDAL VORONOI TESSELLATIONS

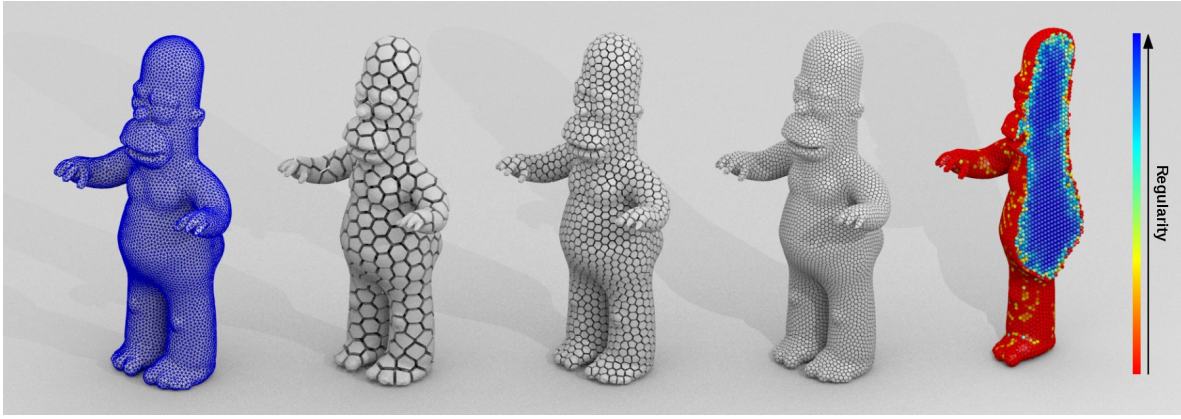
Li Wang, Franck Hétroy-Wheeler, Edmond Boyer
Computer Graphics Forum 35 (1), Wiley, 2016.

Volume xx (200y), Number z, pp. 1–15

A Hierarchical Approach for Regular Centroidal Voronoi Tessellations

L. Wang, F. Hétroy-Wheeler and E. Boyer

Univ. Grenoble Alpes & Inria & CNRS, LJK, F-38000 Grenoble, France



(a) (b) (c) (d) (e)

Figure 1: *Hierarchical CVT computation in 3D. (a) Input: a 3D object bounded by a triangulated mesh. (b,c,d) Successive CVTs computed using our approach, with* 546*,* 4375 *and* 35000 *cells respectively. (e) A cut of Homer shows that most of the interior Voronoi cells present high regularity values.*

Abstract

In this paper we consider Centroidal Voronoi Tessellations (CVTs) and study their regularity. CVTs are geometric structures that enable regular tessellations of geometric objects and are widely used in shape modeling and analysis. While several efficient iterative schemes, with defined local convergence properties, have been proposed to compute CVTs, little attention has been paid to the evaluation of the resulting cell decompositions. In this paper, we propose a regularity criterion that allows us to evaluate and compare CVTs independently of their sizes and of their cell numbers. This criterion allows us to compare CVTs on a common basis. It builds on earlier theoretical work showing that second moments of cells converge to a lower bound when optimising CVTs. In addition to proposing a regularity criterion, this paper also considers computational strategies to determine regular CVTs. We introduce a hierarchical framework that propagates regularity over decomposition levels and hence provides CVTs with provably better regularities than existing methods. We illustrate these principles with a wide range of experiments on synthetic and real models.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling —Curve, surface, solid, and object representations I.5.3 [Pattern Recognition]: Clustering—

1. Introduction

Centroidal Voronoi Tessellations (CVTs) are specific Voronoi diagrams in which each site is located at the cen-

troid of the associated Voronoi cell. CVTs yield tessellations of 2D or 3D domains that have strong regularity properties. Consequently, they are widely used to represent shapes and structures in various scientific domains that include quantization, sensor networks, crystallography and shape modeling, among others (see e.g. [DFG99]). In this paper, we consider CVTs for 2D and 3D regions that are bounded by polygonal curves and 2-manifold meshes, respectively. Inspired by a recent work from Quinn et al. [QSL*12], we study CVT optimality with respect to the spatial regularity of the cells.

Over the last decades, even though many authors have considered the construction of CVTs, little effort has been devoted to the evaluation of their regularities. For many applications, such as climate modeling [JRG11] and shape tracking [AFB15], the regularity of the cell decomposition is crucial to ensure uniform local behaviour. Interestingly, for a few applications such as rendering and stippling, regularity should be avoided [BSD09]. Nonetheless, a measure is desirable to assess the regularity of a CVT. This is true also when comparing CVT decompositions of different shapes or of a single shape but with different decomposition levels. To the best of our knowledge, such a regularity measure has not yet been proposed for CVTs. In practice, they are usually evaluated using the CVT energy function (see Section 2.1), which integrates distances within cells. However, while this energy accounts for the compactness of the cells [LWL*09], it is a metric that depends both on the number of cells and on the volume of the shape.

In this paper, we build on the theoretical work of Conway and Sloane [CS82] and propose a regularity criterion based on the normalized second order moments of the cells. We show that this regularity criterion is linked to the CVT energy function but is dimensionless and therefore enables global evaluations as well as comparisons. We also consider computational strategies to build regular CVTs and we introduce a hierarchical approach that provides CVTs with more regularity than state-of-the-art methods. Our strategy is based on a subdivision scheme that preserves cell regularity and the (local) optimality of CVTs on unbounded domains. This scheme tends to propagate cell regularity through hierarchy levels when applied to bounded domains. We demonstrate the efficiency of this framework with an in-depth evaluation that includes sensitivity analysis, comparisons with previous work and analyses of the convergence speed and computation time.

The remainder of this paper is organized as follows. In Section 2, we review fundamental ideas and related work on CVTs. We introduce our CVT regularity criterion and detail our hierarchical framework in Sections 3 and 4, respectively. Section 5 evaluates the approach.

2. Background and Related Work

2.1. Centroidal Voronoi tessellation

Given a finite set of n points $X = \{x_i\}_{i=1}^n$, called *sites*, in a m-dimensional Euclidean space $\mathbb{E}^m$, the *Voronoi cell* or *Voronoi region* Ω_i [Aur91, For92, OBSC00] of x_i is defined as follows:

$$\Omega_i = \{x \in \mathbb{E}^m \mid \|x - x_i\| \leq \|x - x_j\|, \forall j \neq i\}.$$

The partition of $\mathbb{E}^m$ into Voronoi cells is called a *Voronoi tessellation*.

Some of the Voronoi cells are not closed. However, in many applications, only the intersection of the Voronoi cells with an input 2D or 3D object Ω are required. A *clipped Voronoi tessellation* [YWLL13] is the intersection between the Voronoi tessellation and the object. A *clipped Voronoi cell* is thus defined as:

$$\Omega_i = \{x \in \Omega \mid \|x - x_i\| \leq \|x - x_j\|, \forall j \neq i\}.$$

A *centroidal Voronoi tessellation* [DFG99] is a special type of clipped Voronoi tessellation where the site of each Voronoi cell is also its centre of mass. Let the clipped Voronoi cell Ω_i be endowed with a density function ρ such that $\rho(x) > 0 \; \forall x \in \Omega_i$. The centre of mass $\hat{x}_i$, also called the centroid, of Ω_i is defined as follows:

$$\hat{x}_i = \frac{\int_{\Omega_i} \rho(x) x \, \mathrm{d}\sigma}{\int_{\Omega_i} \rho(x) \, \mathrm{d}\sigma},$$

where $\mathrm{d}\sigma$ is the area differential.

CVTs are widely used to discretize 2D or 3D regions. In that respect, CVTs are optimal quantizers that minimise a distorsion or quantization error $E : \mathbb{E}^{nm} \to \mathbb{R}$ defined as:

$$E(X) = \sum_{i=1}^{n} F_i(X) = \sum_{i=1}^{n} \int_{\Omega_i} \rho(x) \|x - \hat{x}_i\|^2 \, \mathrm{d}\sigma. \qquad (1)$$

CVTs correspond to local minima of the above function E, also called CVT energy function [DFG99]. By definition, an *optimal CVT* achieves the global minimum of this function. Yet finding an optimal CVT appears difficult since the energy function is usually non linear and non convex [LWL*09, LSPW12]. The function E measures the quantization error of a Voronoi tessellation and expresses, to some extent, the compactness of the cells [LWL*09]. However, it does not quantify how regular a tessellation is since it depends on the dimensions of the original region as well as the number of cells considered. As stated in the introduction, our objective in this paper is the ability to quantify a CVT decomposition independently of the shape, size and cell number.

It should be noticed that although Delaunay tetrahedrizations are dual to the Voronoi tessellations, optimizing tetrahedral decompositions is a different problem without guaranties over the dual Voronoi tesselations. As pointed out

by [ACSYD05], duals of CVTs are actually Delaunay tetrahedrizations that may contain badly shaped or irregular tetrahedra. In this paper we focus on CVT decompositions of volumetric shapes instead of Delaunay tetrahedrizations.

2.2. CVT methods

Existing CVT computation strategies are mostly iterative optimization that rely on two fundamental steps: (i) find the initial locations for the given number of sites; (ii) optimize the site locations by minimizing the CVT energy function E (Eq. (1)). They differ then by the initialization they consider and by the optimization approach they implement.

2.2.1. Initialisation

The initial position of the sites has a strong influence on the convergence speed and on the result quality. Different methods have been considered in the literature.

Random Sampling. The idea is to sample the initial site locations randomly inside the object. This simple and fast method is widely used. However, neither the speed of convergence nor the quality of the result can be guaranteed. Other sampling methods can be used to improve these criteria, such as farthest point sampling or Ward's method [MS06].

Greedy Edge-Collapsing. Moriguchi and Sugihara proposed a method which applies a greedy edge-collapsing decimation on the input object and uses the decimated mesh vertices as the initial site locations [MS06]. As pointed out by Quinn et al. [QSL*12], this method can be time-consuming, and the sites may not be regularly positioned if the object is not described by a regular mesh. Consequently, the quality of the resulting CVT can be even worse than using random sampling.

Hammersley Sampling. Quinn et al. suggest to use Hammersley sequences to generate the initial site locations [QSL*12]. Hammersley sequences have correlated positions, which means that the probability of a site being at some position depends on the positions of its neighbours. Unfortunately, the Hammersley sequence generation algorithm as described in [QSL*12] can only place the sites in a square in 2D or a cube in 3D. As a result, the number of sites in the region tessellation is difficult to control.

2.2.2. Iterative scheme

Most of the strategies update the site locations using the Lloyd's gradient descent method [Llo82]. At each iteration, this method moves the current sites to the centroid locations of the corresponding clipped Voronoi cells. This is the continuous equivalent to the k-means clustering algorithm in the discrete case. It has been proved that this leads the CVT energy function to reach a local minimum [DFG99]. Convergence speed can anyway be slow since the site locations may oscillate around a local minimum.

To speed up convergence, Du et al. proposed a Lloyd-Newton method [DE06] which is equivalent to minimizing the sum of distances between the sites and the centroids of the corresponding Voronoi cells. Unfortunately, the resulting tessellation is not always a proper CVT since it is not necessarily a local minimum of E. In an influential work, Liu et al. prove that CVT energy function has C^2 smoothness almost everywhere, except for some non-convex parts of the object [LWL*09]. According to this property, quasi-Newton methods can be used to minimize the CVT energy function. This leads to fast and effective updates in practice.

Another strategy worth mentioning here is the stochastic approach of Lu et al. [LSPW12]. In this iterative approach, the site locations are perturbed once a local mimimum of the energy function is reached and the algorithm is then launched again. The global minimum can theoretically be reached after an infinite number of iterations. In practice, convergence is still slow (see Section 5.3).

In this work, we focus on the regularity of CVTs rather than on the iterative scheme adopted to minimize the CVT energy function itself. The hierarchical algorithm we propose does not depend on this scheme but contributes with respect to the initialization step, in a way similar to the stochastic approach in [LSPW12]. In practice, we use a quasi-Newton approach in our implementation and in all the methods we used for comparisons because of its fast convergence.

3. Regularity Criterion

As mentioned before, the energy function E (Eq. (1)) measures the quantization error. It provides therefore a way to compare CVTs when the shape under consideration and the number of CVT sites are the same. However, this energy function does not evaluate the regularity of the cells and cannot be used for comparison when the number of cells, the shape or the size differ. In this section, we build on theoretical results on optimal quantizers to propose a measure for cell regularity.

3.1. Dimensionless second moment of a cell

In a seminal work, Gersho [Ger79] stated the conjecture that, for a sufficiently large number of sites, all cells of a distortion-minimising CVT are congruent to some polytope H, with the possible exception of regions touching the boundary of the tessellated object, where the polytope H only depends on the dimension m. Gersho's conjecture was proved in two dimensions [New82], the Voronoi cells being regular hexagons in that case. A weaker version of Gersho's conjecture was also proved in three dimensions [BS83]. It says that among all lattice-based CVTs (i.e., regular CVTs, where sites are located on a regular grid), the body-centered cubic (BCC) lattice is the optimal one. The BCC lattice has its sites displayed on a regular cubic grid, with additional

sites at the centre of each cube. The Voronoi tessellation of a BCC lattice is called a bitruncated cubic honeycomb. Each of its cells is a truncated octahedra. Thus, Voronoi cells are truncated octahedra for optimal lattice-based CVTs in 3D. Our criterion is based on Gersho's conjecture and also on the following work of Conway and Sloane [CS82].

Motivated by the design of quantizers of m-dimensional Euclidean spaces, Conway and Sloane [CS82] define, for a given polytope $P \in \mathbb{E}^m$, the value:

$$G(P) = \frac{1}{m} \frac{\int_P \|x - \hat{x}\|^2 \, \mathrm{d}x}{(\int_P \mathrm{d}x)^{(m+2)/m}}, \tag{2}$$

with $\hat{x}$ the centroid of P.

$G(P)$ is called the *dimensionless second moment of* P. It is a measure that depends neither on the dimension m nor on the volume $\int_P \mathrm{d}x$ of P, only on its shape. In contrast, the CVT energy function E (Eq. (1)) reflects the average *unnormalised second moment* of Voronoi cells.

3.2. Regularity measure

Using Gersho's conjecture in the unbounded case, Conway and Sloane showed that the lower bound of $G(P)$ for any space-filling polytope in two dimensions is that of the hexagon:

$$G_2 = \frac{5}{36\sqrt{3}} = 0.0801875\ldots$$

Similarly, in three dimensions, and with unbounded lattices, the optimal lattice-based CVT being the Voronoi tessellation of a BCC lattice, the optimal quantizer is the truncated octahedron with the lower-bound G_3:

$$G_3 = \frac{19}{192\sqrt[3]{2}} = 0.0785433\ldots$$

Consequently, for a sufficiently large number of sites and with the exception of the boundary regions, an optimal CVT should present cells with values of G close to the optimal value G_m. Thus, G is a measure of the regularity of a CVT cell since in the limit, with an infinite number of sites, all cells should reach the value G_m. Note here that we assume a large number of cells and that this reasoning does not apply to the boundary cells, for which the optimal quantizers are not necessarily hexagons (truncated octahedra in 3D) nor necessarily space-filling polytopes. However, under the assumption that boundary cells are largely inferior to interior cells, the distribution of the values of G is a good indicator of the regularity of cells for a given CVT where the regularity is defined with respect to the dimensionless moment G. In our experiments, we consider the average value of G over a CVT:

$$\overline{G}(X) = \frac{1}{n} \sum_{i=1}^{n} G(\Omega_i). \tag{3}$$

3.3. Relation to the CVT energy

Under the assumption of a uniform density function ρ and using the definition (2), the CVT energy E (Eq. (1)) can be rewritten as:

$$E(X) = \sum_{i=1}^{n} mV(\Omega_i)^{(m+2)/m} G(\Omega_i),$$

where $V(\Omega_i) = \int_{\Omega_i} \mathrm{d}x$. Using Gersho's conjecture with unbounded lattices, optimal CVTs present in that case similar cells with volumes V/n and hence:

$$E(X) \sim mn \left(\frac{V}{n}\right)^{(m+2)/m} \overline{G}(X).$$

Thus optimizing the CVT energy E is equivalent to optimizing the average value $\overline{G}$ of G with infinite lattices. Knowing the theoretical optimal quantizer G_m in that case, we can even deduce that the value of an optimal CVT energy:

$$E_m = mn \left(\frac{V}{n}\right)^{(m+2)/m} G_m \tag{4}$$

These are theoretical values for infinite lattices. Nevertheless, with bounded shapes, our experiments show that optimal CVTs converge asymptotically to these values. Note anyway that although the CVT energy E and the regularity $\overline{G}$ are related by the above expression, optimizing $\overline{G}$ directly is inefficient since $\overline{G}$ is dimensionless and therefore ambiguous with respect to the cell sizes. The interest of $\overline{G}$ lies in the comparison between CVTs with different numbers of sites or of different volumes, which is not possible with E.

4. Hierarchical Centroidal Voronoi Tessellation

We now propose a new algorithm to compute CVTs that exploit regularity aspects. As shown by Lu et al. [LSPW12], computing a CVT with a small number of sites is more likely to be regular than with a large number of sites. Thus, we choose to follow a hierarchical approach, creating a coarse regular tessellation from a small number of sites and then refining it while preserving the regularity (see Figure 2).

Although local subdivision schemes have already been proposed for mesh generation (e.g. [TAD07, TWAD09] for 2D triangle and tetrahedral meshes, respectively), we do not know of a previous global hierarchical approach for CVT computation. In addition to the acceleration of the convergence speed, which is well known in many fields (a hierarchical sampling approach is for instance described for quantization in [GG91]), such an approach gives guarantees about the CVT regularity.

Our input is a 2D or 3D object, represented by its boundary: a closed polygonal curve in the first case and a manifold mesh without boundary in the second case. We also ask the user to provide the target number n of sites in the final CVT and the desired number s of subdivisions. From n and s

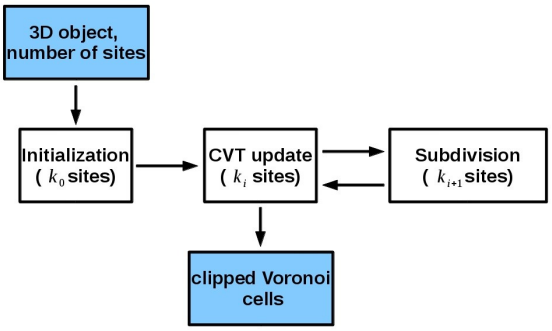


Figure 2: *Overview of our hierarchical approach.*

we derive an initial number k_0 of sites, as explained in Section 4.1. We elaborate on the choice of these parameters in Section 5.

We first create an initial CVT of the object with k_0 sites, using a standard algorithm, see Section 4.3. We then subdivide this tessellation, as explained in Section 4.1. This generates a new tessellation with k_1 sites, which is not Centroidal Voronoi. These sites are then moved towards the centroids of their cells to generate a new CVT with k_1 sites, as described in Section 4.2. We iterate the subdivision - CVT update process s times, until the desired number $k_s = n$ of sites has been reached.

We now detail the three stages of our approach by starting with the subdivision scheme.

4.1. Subdivision

The idea behind our subdivision scheme is to preserve the (local) optimality of the CVT. For example for the 2D case, a CVT is locally optimal with respect to our regularity criterion when its sites form an hexagonal lattice (see Figure 3 (a)), as explained in Section 3. Hence, our goal is to add sites such that the new set of sites keeps forming an hexagonal lattice. In this way, the new CVT will also be locally optimal with respect to regularity in the same area. In non-optimal areas, sites will move and possibly align to form a locally optimal lattice. Thus, iterating the subdivision - CVT update process tends to increase the area where the CVT is optimal for regularity, as shown on Figure 5. With a large number of subdivision s, most interior cells are expected to be regular.

Let X be a set of sites of a given CVT. To subdivide this CVT, we compute its dual Delaunay triangulation and add the centre of each Delaunay edge to X. As shown on Figures 3 and 4, this preserves the local optimality of the CVT.

4.1.1. Number of sites

The previous subdivision scheme does not account for the desired number n of sites. To set up the initial number of

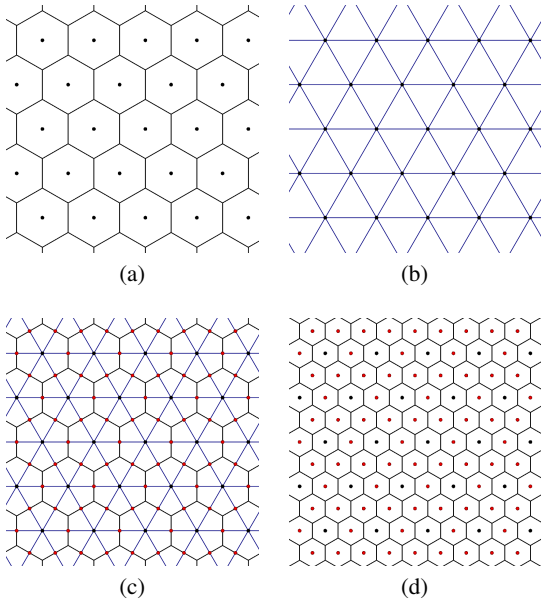


Figure 3: *Subdivision scheme (2D case). (a) Locally optimal CVT: the sites form an hexagonal lattice. (b) Delaunay triangulation of the sites. (c) Subdivision: sites are added in the centre of each edge of the Delaunay triangulation (in red). (d) The new set of sites also forms an hexagonal lattice.*

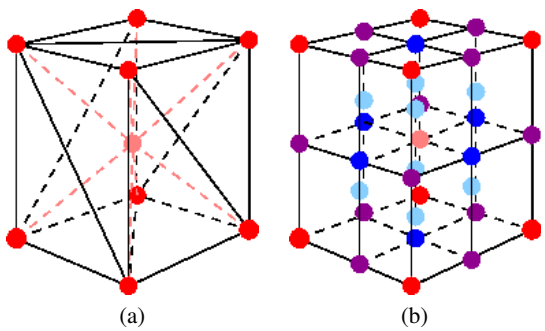
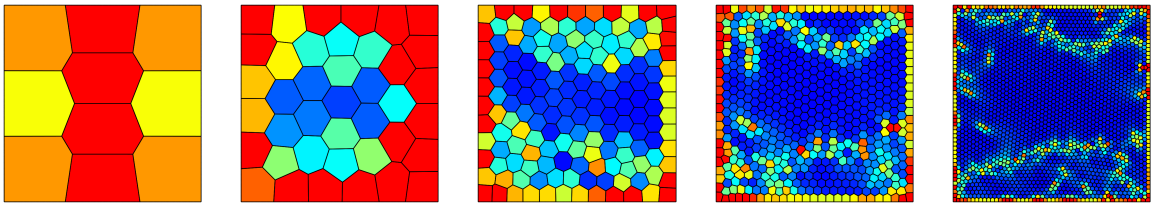


Figure 4: *Subdivision scheme (3D case). (a) Delaunay triangulation of the sites, which form a BCC lattice. (b) Subdivision: sites are added in the centre of each edge of the Delaunay triangulation (in blue and purple). The new set of sites also forms a BCC lattice.*



(a) (b) (c) (d) (e)

Figure 5: *Hierarchical CVT computation. From an initial CVT with $k_0 = 10$ sites (left), successive subdivisions and updates lead to CVTs with $k_1 = 40$, $k_2 = 160$, $k_3 = 640$ and $k_4 = 2560$ sites (from left to right). The cell regularity measure $G_m(\Omega_i)$ is colour-coded from blue (regular) to red (far from regular). Note how regular areas grow.*

sites k_0 such that it reaches the value n after s subdivisions, we proceed in the following way.

Let X be an hexagonal lattice, that is a to say an optimal 2D CVT, with k_i sites. Our subdivision scheme generates a new tessellation with $k_{i+1} = k_i + \frac{6k_i}{2} = 4k_i$ sites, since a new site is created on each of the six edges of a cell, an edge is shared by two cells and there are k_i cells. In the optimal 3D case (BCC lattice), the same reasoning shows that $k_{i+1} = k_i + \frac{14k_i}{2} = 8k_i$, since a site is added on each of the 14 faces of a truncated octahedron. The maximum number of iterations to reach n from a small number k_0 of sites in these ideal cases is thus $s = \lfloor \log_4(n) \rfloor$ and $\lfloor \log_8(n) \rfloor$, respectively.

Thus, if $s \geq \lfloor \log_a(n) \rfloor$, with $a = 4$ in the 2D case and $a = 8$ in the 3D case, we change s to $\lfloor \log_a(n) \rfloor$. We then define s numbers $b_1, \ldots, b_s$ such that $b_s = n$ and $b_i = \lfloor b_{i+1}/a \rfloor$, $1 \leq i \leq s-1$. b_i represents the target number of sites after i iterations. We also define $k_0 = \lfloor b_1/a \rfloor$. After the i-th subdivision, we check the new number k_i of sites. In case of an optimal CVT, $k_i = b_i$. Otherwise, $k_i \leq b_i$. If k_i is smaller than b_i, we randomly sample $b_i - k_i$ new sites inside the *boundary* Voronoi cells. When a new site is inserted into a boundary cell, this cell is then removed from the list of candidate boundary cells for next insertions. This way, the regularity is empirically preserved as much as possible since sites are inserted in different boundary cells, be avoiding many new sites to be neighbours to each other. We thus have b_i sites after the i-th iteration, which are as regularly sampled as possible. This will improve the speed of the CVT update computation, which we describe in the next section.

As an example, Table 1 gives the number k_i of sites obtained after each subdivision and the number $b_i - k_i$ of sites added in the boundary cells, for CVTs depicted in Figures 5, 8 and 9.

4.2. CVT update

Once a new set of sites is defined, any CVT computation method can be used to move these sites towards the centroid of their Voronoi cells. In practice, we use the L-BFGS quasi-Newton algorithm, since it is known to be one of the fastest methods [LWL*09]. As explained in the previous section, the sites where the previous CVT was optimal are not moved, thanks to our subdivision scheme. As a consequence, although the number of sites has increased, the CVT computation is very fast (see Section 5.5 for a discussion).

Once the sites are moved to their new positions and the tessellation is computed, we clip it to the boundary mesh. Our clipping algorithm, detailed below, guarantees that the boundary of the tessellation is a triangulated mesh.

4.2.1. Clipping algorithm

Computing a clipped Voronoi tessellation of an arbitrary 3D object, usually described by its meshed boundary surface, is not an easy problem. Yan et al. [YWLL13] have proposed an algorithm to compute clipped Voronoi diagrams of 3D objects described by tetrahedral meshes. This algorithm consists of two main steps: detection of boundary sites by computing surface restricted Voronoi diagram [ES94, YLL*09] and computation of the intersection between the Voronoi cells of boundary sites and the input tetrahedral mesh using Sutherland's clipping algorithm [SH74]. Recently, Lévy proposed another efficient method based on iterative convex clipping [LÍ4]. This method expresses the clipping problem as a 3D volume intersection problem but also requires a tetrahedral mesh as input. When the input 3D object is given as a closed triangle mesh, a 3D constraint Delaunay triangulation must be computed first [She98, She08]. This is a complex problem which has many degenerate cases and usually requires additional (Steiner) points to ensure the existence of a solution. The complexity highly depends on the quality of the input surface triangle mesh [Eri03]. Inspired by [ZBH11], we overcome this problem and propose an algorithm that exploits a 2D constrained Delaunay triangulation to determine triangles on the input mesh that intersect a given Voronoi cell, without the need of a tetrahedral mesh inside the shape.

Our algorithm first triangulates the polygonal boundaries of the Voronoi cells. In case of an infinite Voronoi cell, the infinite rays edging the cell are replaced by finite length segments, with a length greater than the diameter of the input

Object	n	s	k_0	k_1	b_1-k_1	k_2	b_2-k_2	k_3	b_3-k_3	k_4	b_4-k_4	k_5	b_5-k_5
Figure 8	1033	4	4	5	11	49	15	233	25	995	38	/	/
Figure 5	2560	4	10	27	13	145	15	609	31	2535	25	/	/
Figure 9	10025	5	9	23	16	130	26	568	58	2373	133	9780	245

Table 1: *Number of sites after each subdivision, and number of sites randomly inserted in boundary cells.*

object. The boundaries of the cell, now finites, are then triangulated. Since the boundary of the 3D object is given as a triangulated mesh, the clipping problem now reduces to the computation of triangle-triangle intersections. Once such intersections have been found, we set them as constraints. Constraints are represented as line segments. The intersection I of two triangles is processed according to the following rules:

- Case 1: if I is a point, ignore it.
- Case 2: if I is a line segment, add it to the constraints.
- Case 3: if I is a triangle, add its three edges to the constraints.
- Case 4: otherwise, I is a polygon, construct segments using adjacent vertices of this polygon and add them to the constraints.

These cases are illustrated in Figure 6. The interior of each intersected triangle of either the cell boundary or the mesh is then robustly triangulated using a 2D constrained Delaunay triangulation.

Our clipping algorithm is summarized below (Algorithm 1). Figure 7 depicts its main steps.

```
Data: cell C, 3D object O bounded by a triangulated
      mesh M
Result: clipped cell C'
C_T := TriangulateBoundary(C);
I := Intersection(C_T, M);
if I not empty then
    T := IntersectedTriangles(C_T, I);
    for each triangle t of T do
        T_1 := ConstrainedDelaunay(t, I);
        for each triangle t_1 of T_1 do
            if IsInside(t_1,O) then
                Add(t_1,C');
            end
        end
    end
    T := IntersectedTriangles(M, I);
    for each triangle t of T do
        T_2 := ConstrainedDelaunay(t, I);
        for each triangle t_2 of T_2 do
            if IsInside(t_2,C) then
                Add(t_2,C');
            end
        end
    end
end
else
    C' := C;
end
```

Algorithm 1: Clipping algorithm.

4.3. Initialisation

Before starting the subdivision - CVT update process, we create a first coarse CVT from the input object, with a number k_0 of sites. Our aim is to get an initial CVT with as-large-as-possible optimal areas, since our subdivision scheme can only make these areas grow, as explained in Section 4.1. We propose two different possible initialisations, each of them having different benefits.

A first straightforward idea to initialise the hierarchical CVT computation is to create a CVT using random sampling and a L-BFGS quasi-Newton algorithm to update the positions of the sites. This approach is fast and easy to implement. However, the constructed CVT with k_0 sites may be far from being regular.

Another idea to create a coarse but regular CVT is to sample the k_0 sites on a optimal lattice (hexagonal lattice in 2D and BCC lattice in 3D) which includes the input object. The density of the lattice should be chosen so that there are k_0 sites inside the object. As stated by [QSL*12], it is very hard to control the number of sites inside the object because the density depends on both the size and the shape of the input object. However, tuning the density of the lattice to reach the proper number of sites inside the object is easier with a small number k_0 of sites than with a large number n sites. When applicable, this leads to an optimal tessellation, except at the boundary of the object. In addition, most sites do not need to be moved to create a CVT. Thus, this approach is fast and generates regular CVT cells, except on the boundary of the object.

Both initialisation methods are evaluated in the next section. For the random initialisation, 10 runs are performed for each test, and the median value is taken as the result.

5. Evaluation

We now provide a thorough evaluation of our approach. We first analyse the effect of our hierarchical approach over the regularity of the generated CVT, by discussing the influence

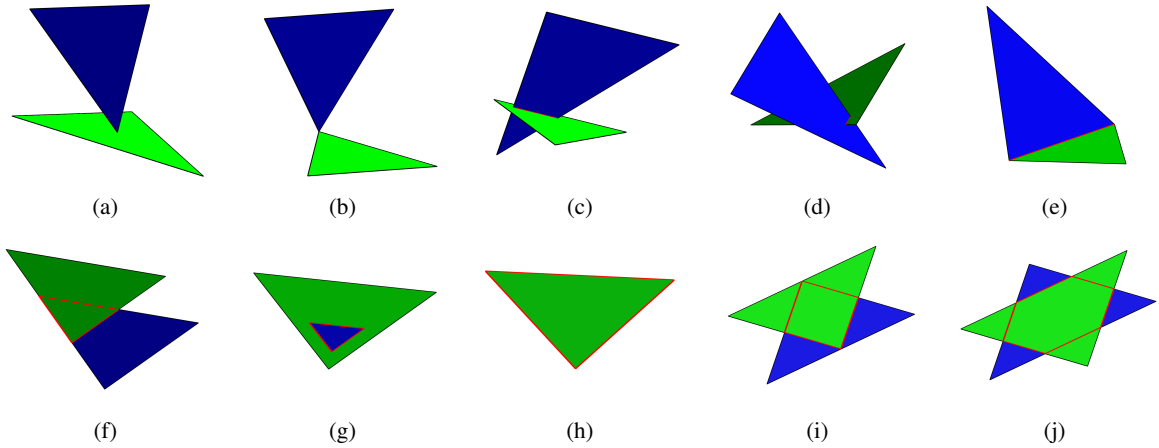


Figure 6: *Different intersection cases. Constraints (line segments) are shown in red. (a, b) Case 1. (c, d, e) Case 2. (f, g, h) Case 3. (i, j) Case 4. (b,e,h) represent singularities.*

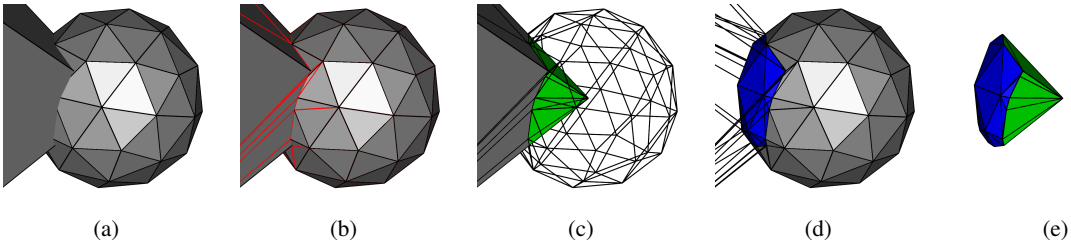


Figure 7: *Clipping algorithm. (a) Input: a Voronoi cell and a 3D object (here: a closed ball) bounded by a mesh. (b) Constrained Delaunay triangulations of the boundary of the cell and of the mesh. (c) In green: boundary of the cell inside the closed ball. (d) In blue: part of the mesh inside the cell. (e) Result: the clipped cell is bounded by the green and the blue triangulations.*

of the two parameters described in Section 4: initial number of sites and number of subdivisions. We compare the regularity of 2D and 3D CVTs generated with our hierarchical approach to CVTs computed with previous work (see Section 2.2). We also provide some details about computation time. In all figures, CVT cells Ω_i are colour-coded according to the cell regularity measure $G_m(\Omega_i)$ defined in Section 3: cells with a high dimensionless second moment are coloured in red, while cells with a low dimensionless second moment are coloured in blue.

5.1. Sensitivity to the initial number of sites

As stated in Section 4, the idea that drives our hierarchical approach is to first create a large regular area in a coarse tessellation, and then to preserve and possibly widen this area when subdividing. An example is shown in Figure 5. The average value of the cell regularity measure $G_2(\Omega_i)$ over all cells Ω_i expresses that the tessellation becomes more regular over subdivision, see Table 2. If n and s are large enough, we expect most of the interior cells of a CVT to be regular, see Figure 1 for an example.

CVT	(a)	(b)	(c)	(d)	(e)
Number of sites	10	40	160	640	2560
$\overline{G}_2(\Omega_i) \times 10^{-2}$	8.458	8.246	8.154	8.099	8.066

Table 2: *Average regularity measure for CVTs depicted in Figure 5 (*1033 *sites, random sampling initialisation).*

On the same 2D square example, we create two other CVTs with 1033 and 2560 sites, respectively, using also random sampling initialisation and the same number of subdivisions ($s = 4$). We obtained an average cell regularity measure of 8.086×10^{-2} and 8.066×10^{-2}, respectively. This shows that the greater the number n of sites, the smaller the average cell regularity measure.

5.2. Sensitivity to the number of subdivisions

A CVT with a few big cells is likely to contain less different regular areas than a CVT of the same object with more, thus smaller, cells. Since our subdivision scheme preserves regular areas, a CVT generated with a large number s of sub-

divisions is more regular than a CVT with the same number n of sites but generated with a small s, as shown in Table 3. As a consequence, we suggest to set s as large as possible.

s	0	1	2	3	4
$\overline{G}_2(\Omega_i) \times 10^{-2}$	8.079	8.063	8.060	8.056	8.054

Table 3: *Average regularity measure for CVTs of a square with* 10000 *sites generated using our hierarchical approach (random sampling initialisation) with different number of subdivisions.*

In the case that an optimal lattice sampling is used as initialisation, it is preferable to start from a small number of subdivisions, since we only have one large regular area whatever the lattice size. Actually, $s = 0$ correspond to the optimal lattice sampling, as shown for instance in Figures 8 (f) and 9 (i). However, as stated before, the larger n, the more difficult it is to build such a lattice with a prescribed number of sites.

5.3. Comparison to previous work

We test our approach against previously mentioned methods in a simple 2D square. To check which method gives the most regular CVT, we first compute an hexagonal lattice with approximately 1000 sites. As stated above, it is difficult to accurately set the number n of sites. We were able to set $n = 1033$. We then compute CVTs with 1033 sites using the following methods:

- random sampling and L-BFGS update;
- Hammersley sampling [QSL*12] and L-BFGS update;
- global Monte-Carlo optimisation [LSPW12];
- our hierarchical approach with random sampling initialisation step (4 subdivisions, which is the maximum possible);
- our hierarchical approach with a lattice sampling initialisation step (1 subdivision).

For the global Monte-Carlo optimisation, we have used the parameter values advised in [LSPW12]. In particular, 200 updates have been performed.

Qualitative results are shown on Figure 8. The average values of the cell regularity measure $G_2(\Omega_i)$ over all cells Ω_i, as well as the CVT energy function values, are given in Table 4. Remember that, as explained in Section 3, an optimal cell has a dimensionless second moment value of $G_2 = \frac{5}{36\sqrt{3}} = 0.08018\ldots$

The hexagonal lattice is not optimal because of its boundary cells which are not hexagonal. Among other methods, the stochastic approach of [LSPW12] and our hierarchical framework give similar results. The main difference in practice between these two methods is the computation time: the global Monte-Carlo minimisation is about 100 times slower than our approach (207.08 seconds instead of 2.16).

We then test a geometrically more complex object: a five-branches star. We also increase the number of sites. As in the previous case, we first start by computing an hexagonal lattice with approximately 10000 sites. The lattice contains exactly $n = 10025$ sites. We discard the stochastic approach of [LSPW12] since it is too slow in this case. For the Hammersley sampling, we construct a bounding box of the object, and try different numbers of sites until we exactly obtain 10025 sites inside the object. It must be mentioned that several attempts were necessary since the number of sites inside the object does not necessarily increase when more sites are generated in the bounding box. For the hierarchical approach, we use 5 subdivisions after the random sampling initialisation (the maximum possible) and only one after lattice sampling initialisation.

Results are shown on Figure 9 and in Table 5. These results are in accordance with the results for the square. Our hierarchical approach performs better than the previous initialisation methods for both criteria. Moreover, lattice sampling initialisation gives better results than random sampling initialisation by almost reaching the regularity of a clipped hexagonal lattice.

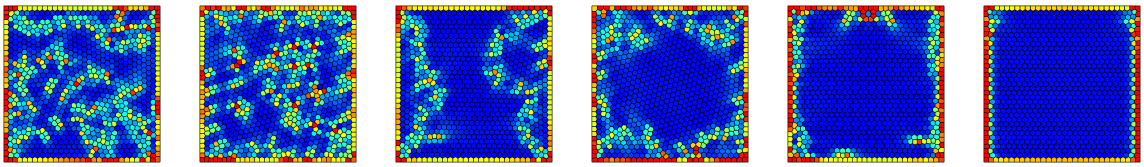
In Figure 10 and Table 6 we compare CVTs computed on a simple closed 3D ball using random sampling, Hammersley sampling, our hierarchical approach with random sampling initialisation and our hierarchical approach with lattice sampling initialisation. It was not possible, in this case, to construct a BCC lattice with a prescribed and sufficiently large number of sites. Two different numbers n of sites were tested. We used the maximum number of subdivisions for the hierarchical approach with random sampling initialisation: two in the $n = 1000$ sites case and three in the $n = 5000$ sites case.

Remember that in this 3D case, the optimal cell regularity measure value is $G_3 = \frac{19}{192\sqrt[3]{2}} = 0.0785433\ldots$ This example shows that our hierarchical approach performs better than other initialisation methods, in case the number n of sites is high enough. In case not, the number of boundary cells is too high with respect to the number of interior cells for the Gersho's conjecture to apply in practice.

Other 3D CVTs with $50k$, $80k$ and $100k$ sites, computed with a standard random sampling initialisation + L-BFGS update, our hierarchical approach using a random sampling initialisation and our hierarchical approach with a lattice sampling initialisation, are shown in Figure 11. It was not possible to create an Hammersley initialisation and a BCC lattice with the correct number of sites inside the objects in these cases.

5.4. Convergence speed

The main parameter in most CVT computation methods is the number of iterations of the algorithm. A local minimum of the CVT energy function is asymptotically reached, but



(a) (b) (c) (d) (e) (f)

Figure 8: *CVTs with* 1033 *sites. (a) Random sampling + L-BFGS update. (b) Hammersley sampling [QSL*12] + L-BFGS update. (c) Global Monte-Carlo [LSPW12]. (d) Our approach, random sampling initialisation. (e) Our approach, lattice sampling initialisation. (f) Hexagonal lattice.*

CVT	(a)	(b)	(c)	(d)	(e)	(f)
$\overline{G}_2(\Omega_i)$	0.08101	0.08107	0.08075	0.08086	0.08077	0.08066
Energy function	25.140	25.160	25.041	25.082	25.045	25.002

Table 4: *Average regularity measure and CVT energy function value for CVTs depicted in Figure 8.*

little is known about how many iterations are necessary before convergence. We decided to investigate this for the previously described methods (except the global Monte-Carlo minimisation of [LSPW12]). The evolution of both the average of cell regularity measures $\overline{G}_2 = \frac{1}{n}\sum_{i=1}^{n} G_2(\Omega_i)$ and the CVT energy function value with respect to the number of iterations of the CVT L-BFGS update are displayed in Figure 12 for the star shape. For our hierarchical approach, this means the number of iterations of the last update (updates for coarser CVTs were done until the usual stopping criterion $\frac{\|g\|}{\|X\|} < 10^{-10}$ is reached, with g the gradient and X the vector of site coordinates, see [QSL*12]).

As shown in Figure 12, all methods behave the same for both measures. This was expected in our case (see Section 3.3). The hexagonal lattice converges the fastest and to the smallest value. Then our hierarchical approach, combined with a lattice sampling initialisation, gives the best results. It is interesting to notice that, because of the non hexagonal boundary cells, no approach reaches the theoretical optimal values ($G_2 = 0.08018\ldots$ and $E_2 = 0.221\ldots$, computed using Eq. (4)).

5.5. Computation time

Our algorithm is implemented in C++ and we use the CGAL library [cga] for 2D constrained and 3D Delaunay triangulations, and the libLBFGS library [ON10] for L-BFGS computation. All computations were performed on an Intel Xeon E5-2643 with 3.30 GHz CPU.

Computation times for our clipping method are shown in Table 7. In particular, we have tested this method on complex and badly triangulated objects and scenes to show its efficiency and robustness, see Figure 13.

Computation times for our hierarchical approach with random sampling initialisation (4 subdivisions) and for a standard method combining a random sampling initialisation and L-BFGS updates are shown in Table 8. Both methods are computationally equivalent in 2D, but ours is faster in 3D. This can be explained by the fact that in our approach the first CVT is computed with a small number of sites, which is very fast, while the next ones quickly converge since most of the sites do not move much.

Object	Fig.	#V (k)	#T (k)	Sites (k)	Time (s)
Homer	1	10	20	35	28.66
Ball	10	0.5	1	1	0.55
Bunny	11	10	20	50	30.06
Kitten	11	10	20	80	54.57
Armadillo	11	173	346	100	103.06
Ballgame	13	12.4	24.8	10	17.92
Dancer	13	15.1	30.2	5	10.52
Dragon	13	100	200	0.1	1.89
Dragon	13	100	200	1	6.14
Dragon	13	100	200	10	21.26
Dragon	13	100	200	100	99.09
CAD model	13	182	364	100	269.67

Table 7: *Computation times for clipped Voronoi diagrams.*

Other initialisation methods (Hammersley sampling and lattice sampling) are usually more time-consuming since finding the right density for a given number n of sites inside the object is difficult in practice. As stated before, the computation time for the stochastic approach of [LSPW12] depends on the number K of perturbations allowed. For a standard value $K = 200$, we found it to be very time consuming (207.08s in the case of the 2D square with $n = 1000$ sites).

6. Conclusions and Future Work

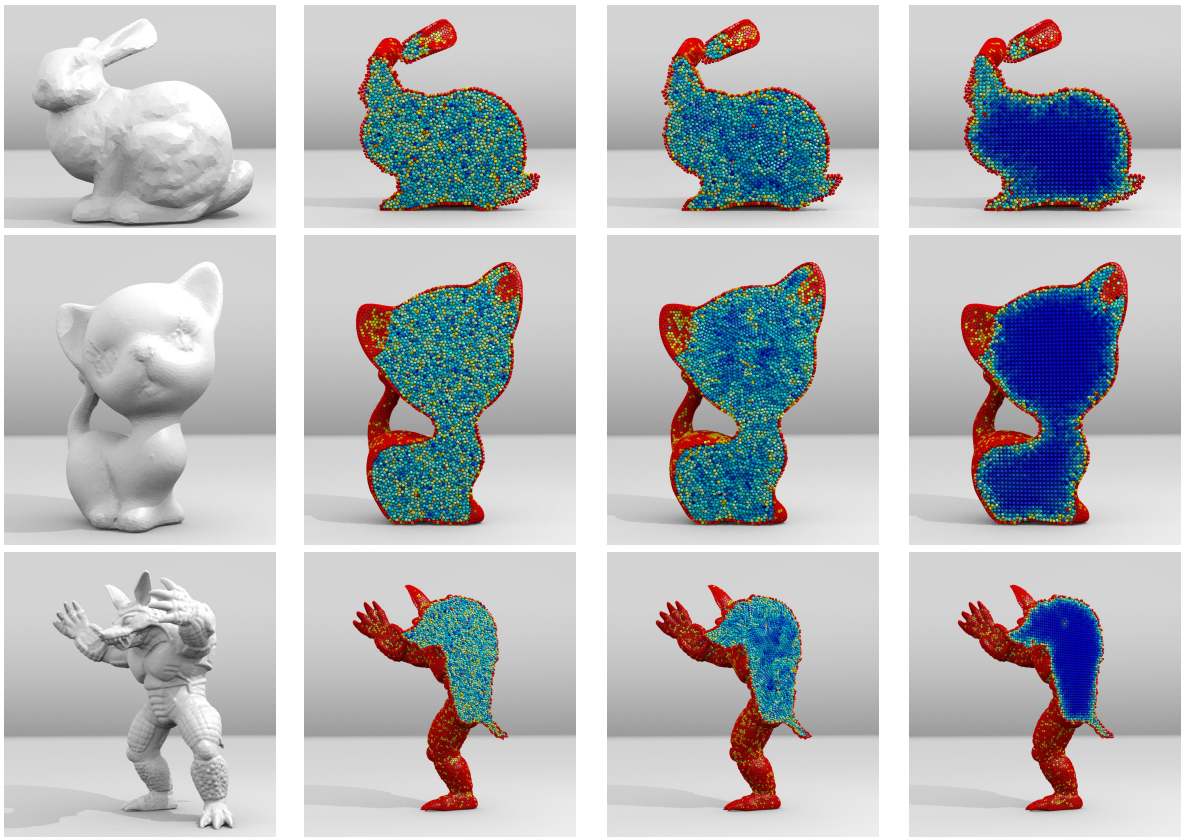
We have introduced different contributions to CVT in two and three dimensional spaces. A regularity criterion was de-

CVT	(a)	(c)	(e)	(g)	(i)
$\overline{G}_2(\Omega_i) \times 10^{-2}$	8.089	8.074	8.068	8.054	8.049
Energy function $\times 10^{-1}$	2.234	2.229	2.227	2.222	2.220

Table 5: *Average regularity measure and CVT energy function value for CVTs depicted in Figure 9.*

CVT	(a)	(c)	(e)	(g)	(b)	(d)	(f)	(h)
$\overline{G}_3(\Omega_i) \times 10^{-2}$	8.025	8.025	8.022	8.022	7.980	7.976	7.975	7.965
Energy function $\times 10^{-3}$	4.324	4.325	4.324	4.325	1.471	1.470	1.470	1.468

Table 6: *Average regularity measure and CVT energy function value for CVTs depicted in Figure 10.*



1

Figure 11: *More examples of comparaisons between a standard approach and our hierarchical one. From left to right: Input object, Random sampling + L-BFGS update, Our approach with random sampling initialisation, Our approach with lattice initialisation.*

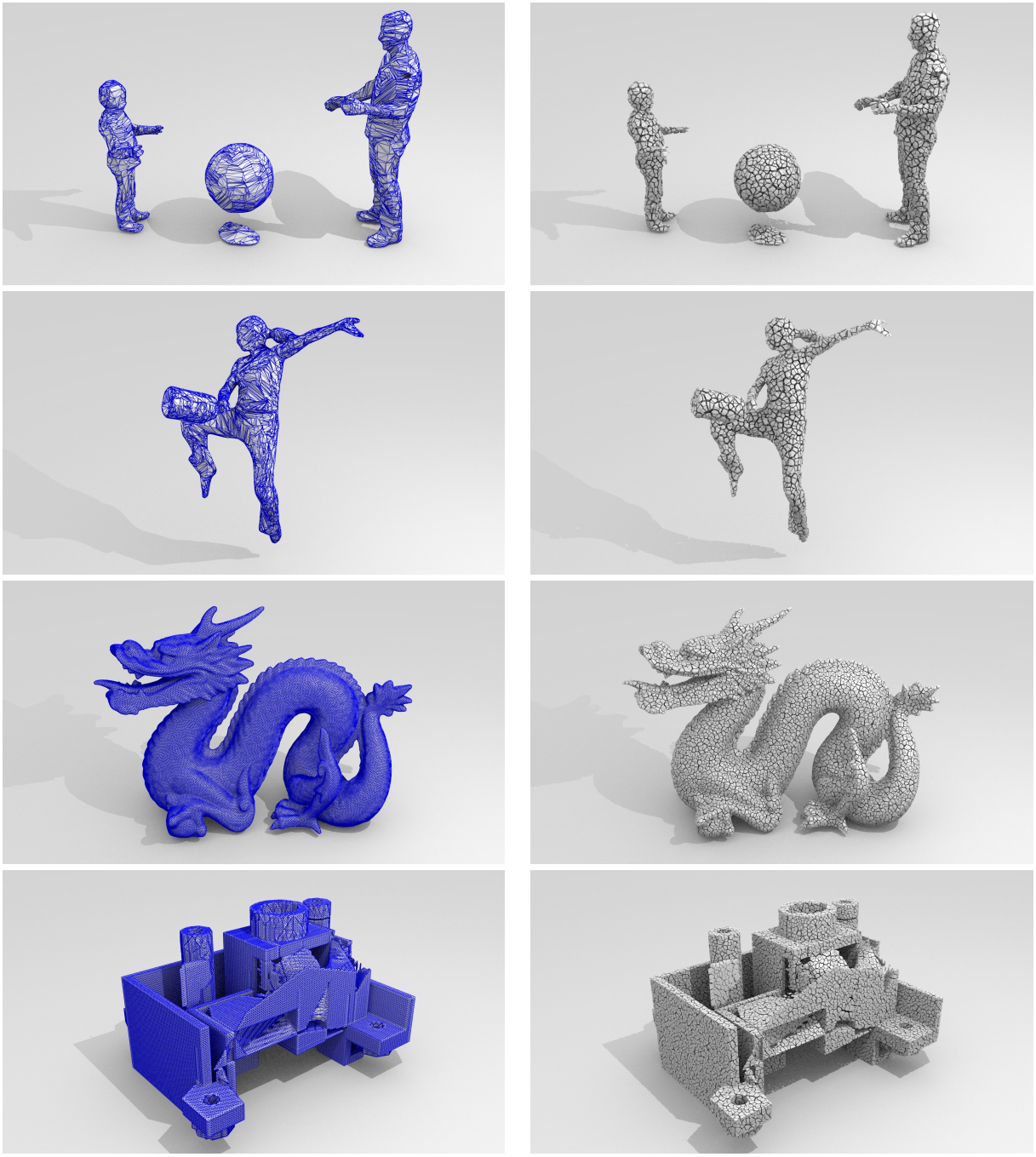


Figure 13: *More examples of clipped (non Centroidal) Voronoi diagrams. Left: input triangulations. Right: clipped Voronoi diagrams.*

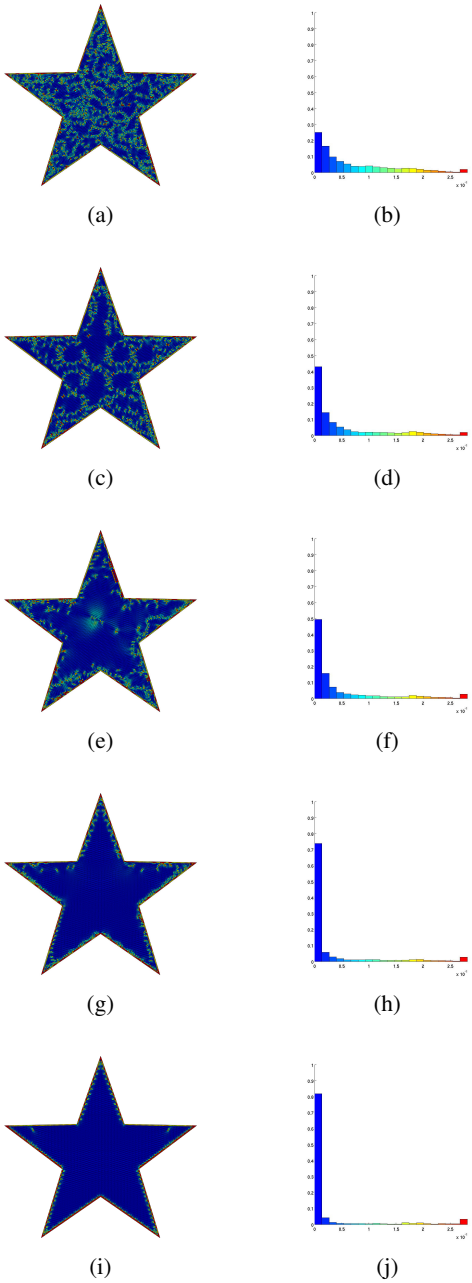


Figure 9: *(a,c,e,g,i) CVTs with* 10025 *sites. (b,d,f,h,j) Corresponding regularity histograms: each bin indicates how many cells share a regularity measure comprised between its boundary values. (a,b) Random sampling + L-BFGS update. (c,d) Hammersley sampling [QSL*12] + L-BFGS update. (e,f) Our approach, random sampling initialisation. (g,h) Our approach, lattice sampling initialisation. (i,j) Hexagonal lattice.*

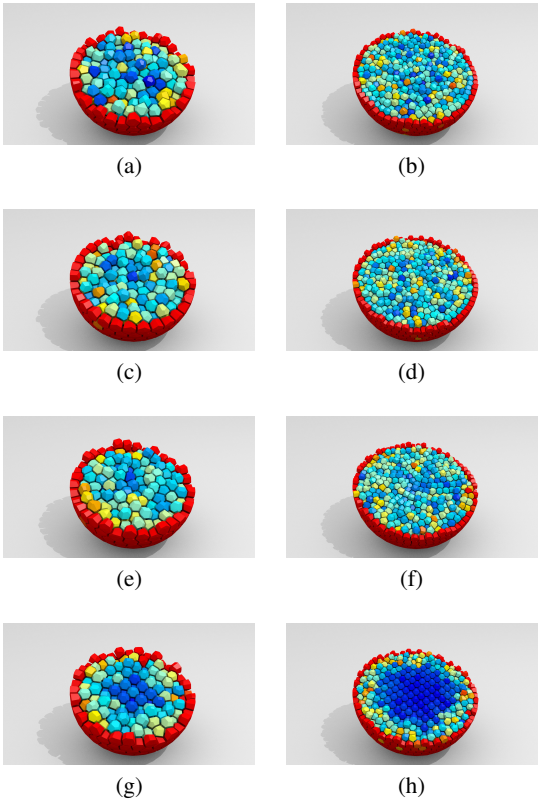
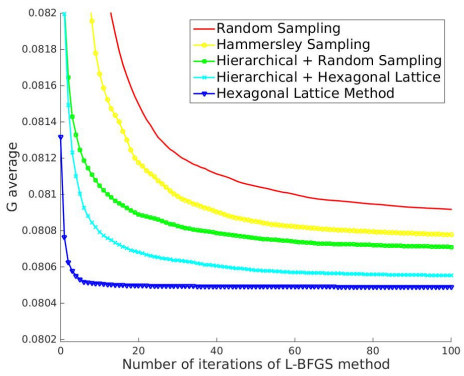


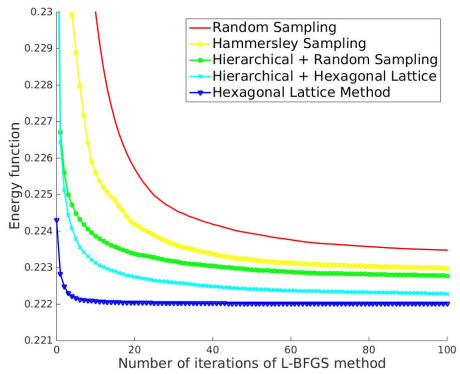
Figure 10: *(a,c,e,g) CVTs with* 1000 *sites in a sphere. (b,d,f,h) CVTs with* 5000 *sites. (a,b) Random sampling + L-BFGS update. (c,d) Hammersley sampling [QSL*12] + L-BFGS update. (e,f) Our approach, random sampling initialisation. (g,h) Our approach, lattice sampling initialisation.*

Object	Fig.	Sites	Method	Time (s)
Square	8	1000	Standard	2.84
			Hierarchical	2.16
Square	8	5000	Standard	10.85
			Hierarchical	9.50
Star	9	2000	Standard	3.56
			Hierarchical	4.91
Star	9	10000	Standard	26.94
			Hierarchical	26.97
Ball	10	1000	Standard	521.86
			Hierarchical	205.82
Ball	10	5000	Standard	1484.69
			Hierarchical	665.01
Homer	1	50000	Standard	15720
			Hierarchical	7860

Table 8: *Computation times for CVTs of objects depicted in Figures 8, 9, 10 and 1.*



(a)



(b)

Figure 12: *Average cell regularity (a) and CVT energy function value (b) with respect to the number of iterations of the CVT update, for the star shape displayed in Figure 9.*

fined to evaluate the quality of CVTs. We also proposed a hierarchical approach for generating CVTs with increased regularities with respect to existing methods as well as a new solution to clip Voronoi tessellations in 3D. Our approach can be used for CVTs in higher dimensional spaces, combined with an adapted Voronoi clipping algorithm such as [LÍ4], although Gersho's conjecture has not been proven in this case. In future work, we may also consider extensions of the approach to generalised CVTs such as for instance weighted diagrams, power diagrams or L_p CVTs [LL10].

Acknowledgements

This work is partly supported by the ANR (project ANR-10-BLAN-0206 MORPHO). We would like to thank Lin Lu for the code of [LSPW12], and Bruno Lévy and the reviewers for their remarks. The CAD model (Figure 13) is courtesy of Jean-Claude Léon.

References

[ACSYD05] ALLIEZ P., COHEN-STEINER D., YVINEC M., DESBRUN M.: Variational tetrahedral meshing. *ACM Transactions on Graphics 24* (2005), 617–625. 3

[AFB15] ALLAIN B., FRANCO J.-S., BOYER E.: An Efficient Volumetric Framework for Shape Tracking. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)* (2015). 2

[Aur91] AURENHAMMER F.: Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR) 23* (1991), 345–405. 2

[BS83] BARNES E. S., SLOANE N. J. A.: The optimal lattice quantizer in three dimensions. *SIAM Journal on Algebraic Discrete Methods 4*, 1 (1983). 3

[BSD09] BALZER M., SCHLÖMER T., DEUSSEN O.: Capacity-constrained point distributions: A variant of lloyd's method. *ACM Transactions on Graphics 28*, 3 (2009), 86:1–86:8. doi:10.1145/1531326.1531392. 2

[cga] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org. 10

[CS82] CONWAY J. H., SLOANE N. J. A.: Voronoi regions of lattices, second moments of polytopes, and quantization. *IEEE Transactions on Information Theory 28* (1982), 211–226. 2, 4

[DE06] DU Q., EMELIANENKO M.: Acceleration schemes for computing centroidal voronoi tessellations. *Numerical Linear Algebra with Applications 13* (2006), 173–192. 3

[DFG99] DU Q., FABER V., GUNZBURGER M.: Centroidal voronoi tessellations: Applications and algorithms. *SIAM review 41* (1999), 637–676. 2, 3

[Eri03] ERICKSON J.: Nice point sets can have nasty delaunay triangulations. *Discrete & Computational Geometry 30*, 1 (2003), 109–132. doi:10.1007/s00454-003-2927-4. 6

[ES94] EDELSBRUNNER H., SHAH N. R.: Triangulating topological spaces. In *Proceedings of the tenth annual symposium on Computational geometry* (1994), pp. 285–292. 6

[For92] FORTUNE S.: Voronoi diagrams and delaunay triangulations. *Computing in Euclidean geometry 1* (1992), 193–233. 2

[Ger79] GERSHO A.: Asymptotically optimal block quantization. *IEEE Transactions on Information Theory 25* (1979), 373–380. 3

[GG91] GERSHO A., GRAY R. M.: *Vector quantization and signal compression*. Kluwer Academic Publishers, 1991. 4

[JRG11] JU L., RINGLER T., GUNZBURGER M.: Voronoi tessellations and their application to climate and global modeling. In *Numerical Techniques for Global Atmospheric Models*, Lauritzen P., Jablonowski C., Taylor M., Nair R., (Eds.). Springer, 2011, pp. 313–342. 2

[LÍ4] LÉVY B.: Restricted voronoi diagrams for (re)-meshing surfaces and volumes. In *8th International Conference on Curves and Surfaces* (2014). 6, 14

[LL10] LÉVY B., LIU Y.: Lp centroidal voronoi tessellation and its applications. *ACM Transactions on Graphics 29*, 119 (2010). 14

[Llo82] LLOYD S. P.: Least squares quantization in pcm. *IEEE Transactions on Information Theory 28* (1982), 129–137. 3

[LSPW12] LU L., SUN F., PAN H., WANG W.: Global optimization of centroidal voronoi tessellation with monte carlo approach. *IEEE Transactions on Visualization and Computer Graphics 18* (2012), 1880–1890. 2, 3, 4, 9, 10, 14

[LWL*09] LIU Y., WANG W., LÉVY B., SUN F., YAN D.-M., LIU L., YANG C.: On centroidal voronoi tessellation - energy smoothness and fast computation. *ACM Transactions on Graphics 28*, 101 (2009). 2, 3, 6

[MS06] MORIGUCHI M., SUGIHARA K.: A new initialization method for constructing centroidal voronoi tessellations on surface meshes. In *3rd International Symposium on Voronoi Diagrams in Science and Engineering, 2006* (2006), pp. 159–165. 3

[New82] NEWMAN D. J.: The hexagon theorem. *IEEE Transactions on Information Theory 28* (1982), 137–139. 3

[OBSC00] OKABE A., BOOTS B., SUGIHARA K., CHI S. N.: *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley, 2000. 2

[ON10] OKAZAKI N., NOCEDAL J.: libLBFGS: a library of Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS), 2010. http://www.chokkan.org/software/liblbfgs/. 10

[QSL*12] QUINN J., SUN F., LANGBEIN F. C., LAI Y.-K., WANG W., MARTIN R. R.: Improved initialisation for centroidal voronoi tessellation and optimal delaunay triangulation. *Computer-Aided Design 44* (2012), 1062–1071. 2, 3, 7, 9, 10, 13

[SH74] SUTHERLAND I. E., HODGMAN G. W.: Reentrant polygon clipping. *Communications of the ACM 17* (1974), 32–42. 6

[She98] SHEWCHUK J. R.: A condition guaranteeing the existence of higher-dimensional constrained delaunay triangulations. In *SCG '98 Proceedings of the fourteenth annual symposium on Computational geometry* (1998), pp. 76–85. 6

[She08] SHEWCHUK J. R.: General-dimensional constrained delaunay and constrained regular triangulations, i: Combinatorial properties. *Discrete & Computational Geometry 39* (2008), 580–637. 6

[TAD07] TOURNOIS J., ALLIEZ P., DEVILLERS O.: Interleaving delaunay refinement and optimization for 2d triangle mesh generation. In *Proceedings of the 16th International Meshing Roundtable* (2007), pp. 83–101. doi:10.1007/978-3-540-75103-8_5. 4

[TWAD09] TOURNOIS J., WORMSER C., ALLIEZ P., DESBRUN M.: Interleaving delaunay refinement and optimization for practical isotropic tetrahedron mesh generation. *ACM Transactions on Graphics 28*, 75 (2009). 4

[YLL*09] YAN D.-M., LÉVY B., LIU Y., SUN F., WANG W.: Isotropic remeshing with fast and exact computation of restricted voronoi diagram. *Computer Graphics Forum 28* (2009), 1445–1454. 6

[YWLL13] YAN D.-M., WANG W., LÉVY B., LIU Y.: Efficient computation of 3d clipped voronoi diagram for mesh generation. *Computer-Aided Design 45* (2013), 843–852. 2, 6

[ZBH11] ZAHARESCU A., BOYER E., HORAUD R.: Topology-adaptive mesh deformation for surface evolution, morphing, and multi-view reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence 33* (2011), 823–837. 6

$\therefore$

A.5 HARMONIC SKELETON FOR REALISTIC CHARACTER ANIMATION

Grégoire Aujay, Franck Hétroy, Francis Lazarus, Christine Depraz
ACM-SIGGRAPH/Eurographics Symposium on Computer Animation, 2007.

Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2007)
D. Metaxas and J. Popovic (Editors)

Harmonic Skeleton for Realistic Character Animation

Grégoire Aujay[1] Franck Hétroy[1] Francis Lazarus[2] Christine Depraz[1]

[1]EVASION - LJK (CNRS, INRIA and Univ. Grenoble)
[2]GIPSA-Lab (CNRS and Univ. Grenoble)

Abstract

Current approaches to skeleton generation are based on topological and geometrical information only; this can be insufficient for realistic character animation, since the location of the joints does not usually match the real bone structure of the model. This paper proposes the use of anatomical information to enhance the skeleton. Using a harmonic function, this information can be recovered from the skeleton itself, which is guaranteed not to have undesired endpoints. The skeleton is computed as a Reeb graph of such a function over the surface of the model. Starting from one point selected on the head of the character, the entire process is fast, automatic and robust; it generates skeletons whose joints can be associated with the character's anatomy. Results are provided, including a quantitative validation of the generated skeletons.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism: Animation

1. Introduction

A common technique for animating a 3D model consists of creating a hierarchical articulated structure, named skeleton (or IK skeleton), whose deformation drives the deformation of the associated model. The location and displacement of the skeleton's joints dictate how the model moves (see Figure 1 for an example). A skeleton attached to a 3D model (usually represented as a mesh) can be either created by hand or computed. In the case of the *realistic* animation of a character (be it a human, an animal or a made-up monster), the first option is most often chosen by artists, although it is a time-consuming task which needs a skilled user. Indeed, professional artists may create an initial skeleton relatively quickly, but often need to make many adjustments during the rigging process because the skin is very sensitive to the exact location of the skeleton's joints: they often have to go back and forth several times between skeleton skinning and testing animation before getting it right. Automatic or semi-automatic methods have several drawbacks: they often allow little control over the result, they can produce noisy skeletons with unwanted joints, and most importantly they rely on the topology and the geometry of the model only, which is not sufficient for realistic animation where the anatomy of the model does not completely match its geometry. For instance, in most cases the spine of a character is close to its back, while the corresponding axis in computer-generated skeletons is usually centered within the body (see Figure 12). Moreover, animation skeletons may have some joints which do not match any anatomical part of the model but are useful for animation purpose (e.g., on the head, see Figure 2).

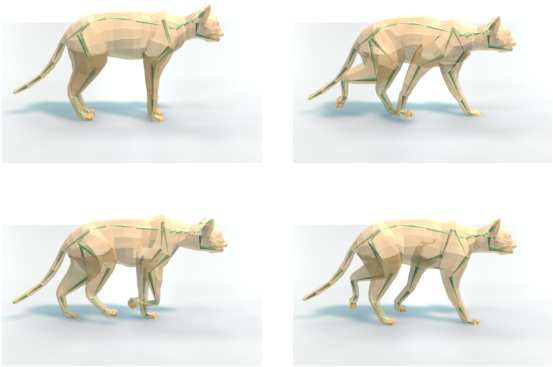


Figure 1: *Walking cat. These images are taken from an animation created using our harmonic skeleton (see the video): joints deformation drives the mesh deformation.*

This paper explains how to automatically, robustly and efficiently compute skeletons adapted to realistic character an-

Copyright © 2007 by the Association for Computing Machinery, Inc.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions Dept, ACM Inc., fax +1 (212) 869-0481 or e-mail permissions@acm.org .
SCA 2007, San Diego, California, August 04 - 05, 2007
© 2007 ACM 978-1-59593-624-4/07/0008 $ 5.00

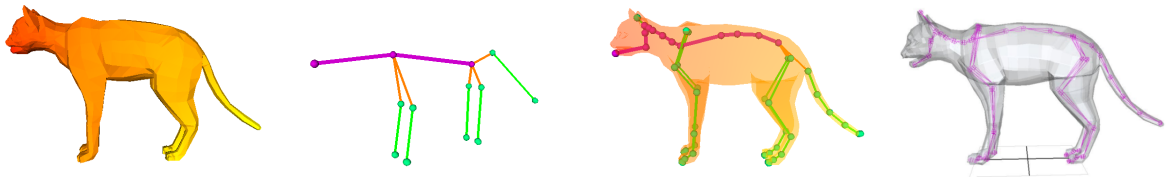


Figure 2: *From left to right: a cat model, the computed harmonic graph with its symmetry axis, the computed harmonic skeleton compared to a previously handmade animation skeleton.*

imation, starting from a single point selected by the user on the model. Generated skeletons match the ones that are created by hand by professionals in most biped and quadruped cases. Moreover, they carry anatomical information (that is to say, we know which joint corresponds to which part of the model), allowing a semantic decomposition of the input meshes.

1.1. Related work

Numerous algorithms have been proposed to compute skeletons of 3D shapes from their geometry. Bloomenthal and Lim [BL99] were among the first ones to point out that these geometric skeletons can be converted to IK skeletons and then used for animation purposes. However, to be useful for animation, skeletons should be structured as graphs, whose nodes correspond to the joints and whose edges correspond to their hierarchy. This discards two-dimensional skeletons such as the Medial Axis [Blu67].

Graph-like skeleton generation algorithms start either from the boundary surface [KT03, LWM*03, DS06, LKA06, TVD06] of the input model, or from its inner volume [GS01, WP02, CSYB05]. Methods working on a volumetric representation of a model have a major drawback: only features with a size greater than the voxel size can be taken into account. This often leads to computationally expensive algorithms.

Katz and Tal [KT03] extract a skeleton from a meshed model using a hierarchical decomposition of this mesh into meaningful parts. Generated skeletons are star-shaped (they contain a root joint, located in the center of mass of the model, from which all other joints derive) and thus are not suited for realistic animation. Lien et al. [LKA06] generate shape decomposition and skeleton simultaneously; the skeleton is computed using centroids and principal axes of the shape's components, which gives a skeleton with geometrically but not necessarily anatomically meaningful positions. The same problem appears with Dey and Sun's robust skeleton computation from the Medial Axis [DS06]. Liu et al. [LWM*03] propose to use a repulsive force field to position the joints. This method is quite slow (as reported in the paper, it takes several minutes to compute the skeleton for a model containing about 10,000 triangles), and does not guarantee that the result will capture all desired features.

Following Shinagawa et al. [SKK91], several authors have proposed to use a mathematical tool called the *Reeb graph* to capture the model's topology, before possible refinements to capture its geometry. A Reeb graph is defined with respect to a mathematical function, and the result highly depends on the choice of this function. In the next section, we precisely define the Reeb graph and then list some existing methods using this mathematical notion.

The algorithm we propose takes as input a triangle mesh. It first computes a Reeb graph of this mesh, in a fast and robust way (that is to say, the graph's leaves are only the desired ones). This abstract graph is then refined and embedded in the 3D space in order to be useful for realistic character animation; this is made possible thanks to a semantic decomposition of the model, given by the graph. Our algorithm computes the skeleton of a model with several hundred of thousand faces in no longer than a few seconds on a low-end computer.

1.2. Mathematical background

Let $f : M \to \mathbb{R}$ be a function defined over a 2-manifold M with or without boundary (that is to say, a surface for which each point has a neighboorhood homeomorphic to a disk or half-disk). Level sets of f are the sets $f^{-1}(u) = \{x \in M, f(x) = u\}$. Each of these sets, if it exists, can be connected or not. For instance, on Figure 3, where f is a height function, $f^{-1}(u)$ is connected for low and high values of u, but is made of several connected components for values around -0.7, 0 and 0.7. For some special values, the number of connected components of the level set changes: these values are called *critical values*, and the corresponding points x on the surface are called *critical points*. The *Reeb graph* of f [Ree46] is a graph whose nodes correspond to these critical points, and which encodes the connectivity between them (see Figure 3). In particular, notice that the leaves of the Reeb graph exactly match the local maxima and minima of f. Mathematically speaking, the Reeb graph of f is defined as the quotient space $M/\sim$, with $\sim$ the following equivalence relation on M:

$$x_1 \sim x_2 \iff \begin{cases} f(x_1) = f(x_2) \\ \text{and } x_1 \text{ and } x_2 \text{ belong to the same} \\ \text{connected component of } f^{-1}(f(x_1)) \end{cases}$$

More details about these notions can be found in e.g. [FK97].

© Association for Computing Machinery, Inc. 2007.

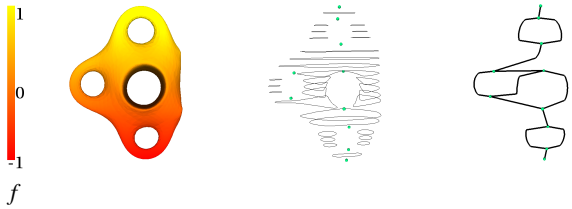


Figure 3: *From left to right: a surface, some level sets of f, the Reeb graph of f.*

A Reeb graph *w.r.t.* a triangulated surface with n edges can be computed in $O(n \log n)$ time [CMEH*03]. However, the choice of the function f is a key issue in revealing information about the surface, and several proposals have been made in order to obtain a relevant graph: the distance on the mesh to a source point [LV99], the integral over the mesh to such a distance (in order to avoid the choice of the source point) [HSKK01], a mapping function that highlights the relevant features [TVD06], etc. Following an idea proposed by Ni et al. [NGH04], we choose to find a "fair" function f, whose extrema will be anatomically significant, by solving Laplace's equation $\triangle f = 0$. Steiner and Fischer did the same [SF01], but their Reeb graph captured neither geometrical nor anatomical features, only the topology of the model.

The main property of such functions f, called *harmonic functions*, is their lack of extrema except at boundary points [NGH04]. They also have the following property:
let M be a compact surface, B_M its boundary and $g : M \to \mathbb{R}$ a function; there exists a unique solution $f : M \to \mathbb{R}$ to the following system, called Laplace's equation with non-homogeneous Dirichlet boundary conditions:

$$\begin{cases} \triangle f(x) = 0 & \forall x \in M \\ f(x) = g(x) & \forall x \in B_M \end{cases} \tag{1}$$

In our case, B_M will be a (disjoint) set of vertices of the mesh, corresponding to anatomically significant parts of the model. We will compute the Reeb graph of the solution $f_{B_M,g}$ to the previous system (1), for some function g which will be described in section 2.1.1; thanks to the property of harmonic functions, the leaves of this graph will exactly match the chosen vertices: in other words, the graph is guaranteed not to be noisy.

1.3. Algorithm overview

The Reeb graph of a function *w.r.t.* a surface is a pair (V,E) with V a set of nodes and $E \subset V \times V$ a set of edges between these nodes. It is minimal in the sense that there is no regular node: each node has either one or at least 3 incident edges. Moreover, nodes do not have 3D coordinates. Thus, in order to construct a skeleton which is suitable for animation from such a graph, we must *embed* it in $\mathbb{R}^3$, that is to say link each node with 3D coordinates. Thus, our method first computes a Reeb graph, then embeds it in $\mathbb{R}^3$. In the following, *node* will refer to the graph while *joint* will refer to the skeleton and *vertex* to the mesh.

Our algorithm runs in seven successive stages:

1. the endpoints of the desired skeleton are chosen by the user or computed (however at least one of them, called the *source node/joint*, must be manually chosen on the head of the character);
2. the harmonic function f solving Laplace's equation with non-homogeneous Dirichlet boundary conditions is computed;
3. the Reeb graph of f is computed with the algorithm described in [CMEH*03];
4. this graph, which we call a *harmonic graph* since f is a harmonic function, is subsequently filtered to recover the symmetry of the character's morphology (i.e., overall structure);
5. starting from the source node, the symmetry axis of the graph is detected;
6. the harmonic graph is refined by inserting regular nodes and embedded in $\mathbb{R}^3$: this gives us the *harmonic skeleton*, which carries anatomical information about the input model (such as "this joint corresponds to the tail");
7. additional heuristics are used in case the model is detected to be a biped or quadruped with sagittally oriented legs (this excludes amphibians, but includes most mammals), in order to fit the IK skeleton that would be manually created by an expert. Although not presented in this paper, equivalent heuristics can be defined for other kinds of characters, such as birds or insects.

The contributions of this paper are the following:

- the computed skeleton is robust: endpoints are exactly the ones that have been chosen, and two meshes representing the same model under two different postures generate equivalent skeletons;
- our algorithm is fast and does not need user intervention, except for the selection of the source joint at the very beginning. However, controlling the skeleton generation is possible, by manually choosing its endpoints or tuning some parameters;
- our method gives a semantic decomposition of the shape (which is used for the embedding process): we know which part of the mesh corresponds to the head, the legs, the trunk and the tail of the character;
- we propose standard skeletons (graphs and their embeddings) for bipeds and quadrupeds with sagittally oriented limbs.

Moreover, in the case of quadrupeds, we have validated our results not only visually but also by comparing parameters with handmade animation skeletons. To our knowledge, this is the first time a quantitative validation is proposed.

The organization of this paper is as follows: section 2 describes stages 1 to 4 of our algorithm, that is to say the computation of the harmonic graph; section 3 explains the construction of the harmonic skeleton from the harmonic graph,

© Association for Computing Machinery, Inc. 2007.

that is to say stages 5 and 6; in section 4, we detail the proposed skeletons for bipeds and quadrupeds; we give results and discuss them in section 5; finally, we conclude in section 6.

2. Harmonic graph

2.1. Graph computation

2.1.1. Finding extrema

The first stage of our algorithm is to choose the endpoints of the skeleton; they will correspond to extremal joints. The user must select one source vertex x_{source} on the head of the character, which will give the source node of the graph. We set $f(x_{source}) = 0$. Other endpoints should match relevant anatomical features of the character that the user wants to animate: hands, feet and possibly tail, ears, etc. These endpoints can be either selected manually, or computed. In the latter case, we try to find vertices x such that the distance $d(x_{source}, x)$ on the mesh is locally maximum. Several methods have been proposed to solve this problem: for instance, Dong et al. [DKG05] choose to solve the Poisson equation $\triangle f = -\|\triangle x\|$; the algorithm proposed by Tierny et al. [TVD06] can also be applied, but it does not use the source vertex, which should be selected afterwards among the detected feature vertices, hence it does not ensure this vertex will be on the head of the character. The same problem arises when computing the average geodesic distance function over the mesh, as did Zhang et al. [ZMT05]. In our implementation, we use a fast and more straightforward solution: g is defined as a geodesic distance to x_{source}; we use Dijkstra's algorithm to compute shortest paths on the mesh from the source vertex to all other vertices, as proposed by Lazarus and Verroust [LV99]. This method, as Dong's, has one drawback: multiple neighboring local extrema can be found in almost flat regions. We propose a solution to cluster these extrema, which will be discussed in section 2.2. For each extremum vertex x (be it manually or automatically chosen), the value $f(x)$ is set to the length of the shortest path from the source vertex, as computed by Dijkstra's algorithm (it could also be set to the value given by Dong's method when using this algorithm). Doing so, the harmonic function f can be seen as a smooth approximated distance to the source vertex over the mesh.

2.1.2. Solving Laplace's equation

Once the boundary conditions to Laplace's equation are set, the system (1) is solved using a classical finite elements method of $P1$ type (the function f, defined for each vertex, is linearly interpolated inside each triangle). Since the assembled matrix is very sparse, computation can be done very efficiently (e.g. using the SuperLU solver [DEG*99]).

2.1.3. Generating the graph

The Reeb graph of f is then computed using Cole-McLaughlin's algorithm [CMEH*03]. This algorithm requires f to be a Morse function: this basically means that two neighboring critical points should have two different values for f. To ensure this property, we check if all vertices on the mesh have different values. If several vertices $x_1, \ldots, x_k$ have the same value $f(x_1) = \ldots = f(x_k)$, we order them and change their values slightly.

2.2. Graph filtering

2.2.1. Recovering the shape's symmetries

Even if the model is symmetric, Cole-McLaughlin's algorithm may generate a non-symmetric graph, because the source vertex may not be located exactly on the symmetry plane or axis. We propose here a simple way to recover these symmetries.

Each node n of the graph G is assigned with the value $f(x)$, where x is the critical vertex on the surface corresponding to n. Now, let us give weights to the edges of G. Let (n_1, n_2) be an edge of G. (n_1, n_2) is balanced by the following weight:

$$w(n_1, n_2) = \frac{|f(n_1) - f(n_2)|}{|\max_{n \in G} f(n) - \min_{n \in G} f(n)|} \tag{2}$$

Considering f as an approximated distance to the source vertex over the mesh (see section 2.1.1), $w(n_1, n_2)$ represents the normalized difference between the distance to the source vertex of two "topologically close" vertices. If $w(n_1, n_2)$ is small, this means that the corresponding vertices x_1 and x_2 are approximately at the same distance to the source vertex, and are also located in the same topological area (they are not necessarily geometrically close to each other). Thus, in order to recover the shape's symmetries, we propose to filter the graph by collapsing every *internal* edge with a weight lower than a given threshold t_1. We do not collapse edges containing a leaf node, since this could remove small features.

Notice that we can recover not only *geometrical* symmetries of the model, but also *morphological* ones: for instance, the octopus model of Figure 4 is not symmetric, geometrically speaking, because its tentacles are not in the same position; it can however be regarded as morphologically symmetric, because these tentacles have the same size and are regularly placed around a symmetry axis. As shown on the same model, we can recover not only symmetries *w.r.t.* a plane but also symmetries *w.r.t.* an axis.

2.2.2. Removing irrelevant extrema

As explained in section 2.1.1, it may happen that too many extremum vertices are computed. In order to remove irrelevant extrema, since extrema correspond exactly to the leaf nodes of the graph, we propose to remove the *external* edges (that is to say edges containing a leaf node) with a weight lower than a given threshold t_2, *together with their nodes*. However, these edges should be removed carefully (see Figure 5): in order to avoid extra deletion of edges, they should first be ordered by increasing weight.

© Association for Computing Machinery, Inc. 2007.

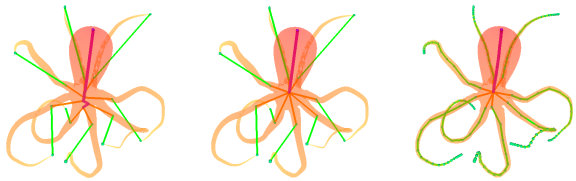


Figure 4: *Left: non-symmetric graph obtained from a model containing a symmetry. Middle: the same graph after filtering ($t_1 = 0.007$). Right: refined harmonic skeleton.*

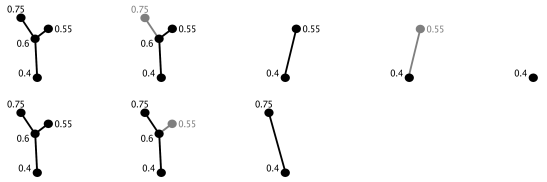


Figure 5: *Deletion of edges whose weight is lower than or equal to $t_2 = 0.15$. Top: without weight ordering. Bottom: with weight ordering.*

Both thresholds t_1 and t_2 can be set by the user, but they can also be computed. Indeed, unwanted edges usually have very small weights compared to the others', since they can be seen as noise while the others are associated with feature sizes. Thus, a statistical analysis upon all the edge's weights can help to set these parameters.

3. Harmonic skeleton

The harmonic graph gives the topological structure of the model. This is not enough to get an animation skeleton: we need to add 3D coordinates to its nodes, which will represent the joints of the skeleton; we may also need to refine the graph. Previous methods constructed the skeleton from a Reeb graph using only topological and geometrical information from the model, which is often not sufficient for realistic animation. We propose to take benefit from anatomical information to design the skeleton; this information will be recovered from the harmonic graph, knowing that the source vertex was chosen on the head of the character. In this section, we explain how to detect symmetries of the model's morphology on the graph and propose a skeleton in general case. In the next section, we show how to improve this general skeleton in the case of biped and quadruped characters.

3.1. Symmetry axis detection

We suppose here that the character's morphology is symmetric. This is often the case: typically, the model has two or four legs, two ears, and the head and the tail (if it exists) are centered with respect to the legs. Thus, the harmonic graph should also be symmetric with respect to an axis (or a node, but a node can be considered as a degenerate case of an axis). We propose here a heuristic to recover this symmetry axis starting from the source node, which is located on this axis since its corresponding vertex is supposed to be on the head of the character.

Finding symmetries on a graph is a NP-complete problem; that is why we must make some hypotheses about the graph to get an efficient algorithm. Several restrictions have been proposed in the graph theory community [DeF99]; we describe here a simple iterative algorithm based on the 3 following assumptions:

1. the source node is located on the symmetry axis;
2. the harmonic graph is actually a tree, i.e. it does not contain any cycle;
3. two subtrees are isomorphic if they have the same depth and if their root nodes have the same degree (that is to say, the same number of child nodes).

The two last hypotheses are relevant for our application, since harmonic graphs are usually simple: they are made of one node for the head, one node for each leg, possibly one extremal node for the tail, for each ear and/or each wing and/or each finger, and that is usually all.

We use n_0 to denote the source node of the harmonic graph, and $(n_0, n_1) = e_0$ as its incident edge: e_0 is on the symmetry axis. n and n' denote nodes of the harmonic graph, whereas e denotes an edge. Our algorithm proceeds as follows:

- $e = (n, n') \leftarrow e_0 = (n_0, n_1)$
- **while** $e \neq NULL$ **loop**
  - add e to the symmetry axis;
  - let $e_1 = (n', n'_1), e_2 = (n', n'_2), \ldots, e_k = (n', n'_k)$ be the incident edges to n', excepting e;
  - for each node n'_j, let T_j be the subtree of G whose root node is n'_j and which does not contain n';
  - store the T_j into sets $\mathcal{S}_1, \ldots, \mathcal{S}_l$ of isomorphic trees, according to assumption number 3;
  - **if** $\exists! \mathcal{S}_i$ which contains only one tree T_i **then** $e \leftarrow e_i = (n', n'_i)$
  - **else** $e \leftarrow NULL$
  - **end if**
- **end loop**

Figure 6 shows the successive steps of the algorithm on an example. It adds edges to the symmetry axis iteratively, discarding subtrees of the harmonic graph that are symmetric *w.r.t.* the computed part of the axis. Note that if several sets $\mathcal{S}_i$ containing one tree exist at the same time the algorithm stops, because it cannot tell which tree has its root on the axis. This algorithm can be applied not only to the harmonic graph G, but also to subtrees of G, in order to find non-principal symmetries. We can thereby obtain a hierarchy of symmetries, like [SKS06].

© Association for Computing Machinery, Inc. 2007.

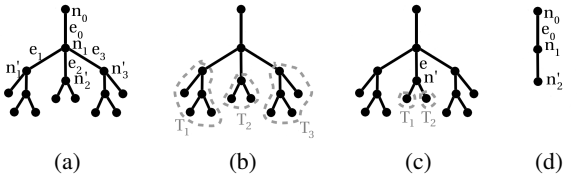


Figure 6: *Symmetry axis detection. (a) Initialization (b) First step: T_1 and T_3 are isomorphic trees, and T_2 is not isomorphic to any other known tree: $e_2 = (n_1, n_2')$ is on the symmetry axis. (c) Second step: T_1 and T_2 are isomorphic, there is no candidate tree to process further on, so the algorithm stops. (d) Detected symmetry axis.*

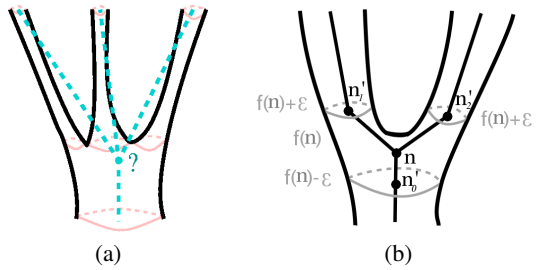


Figure 7: *(a) Some nodes may have more than one corresponding vertex on the mesh. (b) Added regular nodes and possible embedding for internal nodes with 3 incident edges.*

3.2. Simple embedding

Finding an appropriate embedding for each node of the harmonic graph is not a trivial task: even if each extremal node can be embedded onto the corresponding vertex on the mesh, this is not always possible for internal nodes, since they may have more than one corresponding vertex (Figure 7 (a)). Moreover, it is often more relevant to embed an internal node *inside* the model than *on* the surface. Before giving details about how internal nodes will be embedded in $\mathbb{R}^3$, we should explain how regular nodes (nodes with exactly two incident edges) that will be inserted to the graph will be embedded.

Let u be a regular value of f (that is to say a non-critical value), let $f^{-1}(u)$ be its level set, and let C be a connected component of $f^{-1}(u)$. C is a simple closed curve made of segments whose endpoints $p_1, p_2, \ldots, p_k, p_{k+1} = p_1$ intersect the edges of the mesh. We define the *center of* C as the center of mass of these segments [LV99]:

$$center(C) = \frac{\sum_{i=1}^{k} \|p_i p_{i+1}\| \frac{p_i + p_{i+1}}{2}}{\sum_{i=1}^{k} \|p_i p_{i+1}\|} \tag{3}$$

We embed a regular node with value $f(u)$ onto the center of its associated connected component C. This choice is more relevant that the center of mass of the points p_i, since the result is less dependent on the surface's discretization level.

Now, here is the algorithm we propose in order to embed an internal node n:

1. split each incident edge (n, n_i) to n in two, by inserting a new node n_i';
2. assign the value $f(n) + \varepsilon$ or $f(n) - \varepsilon$ to each n_i', depending whether $f(n) < f(n_i)$ or $f(n) > f(n_i)$ (ε should be a small scalar value, lower than the lowest weight among the graph's edges);
3. since each node n_i' is a regular node, embed it as explained before;
4. determine which nodes among these are on the symmetry axis:
   - if there is none, embed n onto the center of mass of the n_i''s embeddings;
   - if there is one, embed n onto the embedding of this node n_k';
   - if there are two (three or more is not possible), choose one of them, embed n onto its embedding and remove the other node from the graph;
5. finally, freeze the new edges (n, n_i'): this means that if one node's embedding is subsequently modified, the other should be modified the same way.

Figure 7 (b) shows the possible embeddings for internal nodes with 3 incident edges. Freezing edges has an important meaning: some degrees of freedom are removed for some joints of our animation skeleton, and freezing allows us to mirror the effect of bones such as the clavicle or the pelvis.

3.3. Joint hierarchy

Embedding the graph's nodes in $\mathbb{R}^3$ is not sufficient to get an applicable animation skeleton: we should also define a joint hierarchy. This can easily be done using the detected symmetry axis on the harmonic graph: the base joint can correspond to any node on this axis, then other joints recursively come from it. Common base joint choices include the head, that is to say the source joint, a node on the symmetry axis with a mean value for f, or the pelvis, which is the last node on the symmetry axis with at least three incident edges.

Once we have set up this hierarchy, we can use our embedded and augmented harmonic graph, which we call *harmonic skeleton*, as animation skeleton: nodes will be used as joints. The direction of the symmetry axis (or more precisely, of its embedding) can be used to set up the initial orientation of each joint. Moreover, additional joints can be added in a very simple way as regular nodes on the graph, with the embedding described in section 3.2. The value for f corresponding to a new joint, and hence its exact location, can be either set up by the user, or computed as the mean value between the two values of the edge's nodes (this is our default choice), or even computed so that the joint fits some geometrical feature

© Association for Computing Machinery, Inc. 2007.

(e.g. local minimum of the gaussian curvature, as proposed by [TVD06]).

4. Adapted embedding for bipeds and quadrupeds

In this section, we explain how the previously computed skeleton can be modified in order to better fit biped or quadruped mammals. Equivalent heuristics can be developed for other kinds of characters. These heuristics rely on semantic information about the model's anatomy associated to each joint of the skeleton, which can be recovered since the source joint corresponds to the head of the character and all skeleton extrema are known (see Figure 8 (a)). First, we propose a heuristic to check if the skeleton corresponds to a biped or a quadruped model.

4.1. Biped/quadruped discrimination

In the case of a biped or quadruped character, the computed harmonic graph should be as described in Figure 8 (a): the symmetry axis should have at least 2 nodes with at least 3 incident edges. The last of these nodes P matches the pelvis, and the previous one S matches the shoulders (we can have others, matching for example the ears). Since P and S have 3 or 4 incident edges, we know from section 3.2 that the ones not on the symmetry axis have been frozen: let P_1, P_2, S_1 and S_2 be their other endpoints; these nodes correspond to the beginning of the leg bones (when the subtree corresponding to the tail is isomorphic to the back legs, P_1 and P_2 are chosen among the three children of P so that $|SP.(PP_1 \times PP_2)|$ is maximum). We can now define 3 unit vectors: the spine direction $Spine = \frac{SP}{\|SP\|}$, a unit vector N_P normal to the triangle PP_1P_2 and a unit vector N_S normal to the triangle SS_1S_2. Since edges PP_1, PP_2, SS_1 and SS_2 are frozen with $f(P_1) \approx f(P_2) \approx f(P)$ and $f(S_1) \approx f(S_2) \approx f(S)$, we say that the model is a quadruped if $|Spine.N_P| \approx 1$ and $|Spine.N_S| \approx 1$, and a biped if $|Spine.N_P| \approx 0$ and $|Spine.N_S| \approx 0$ (see Figure 8 (b) and (c)). In the other cases, we cannot conclude.

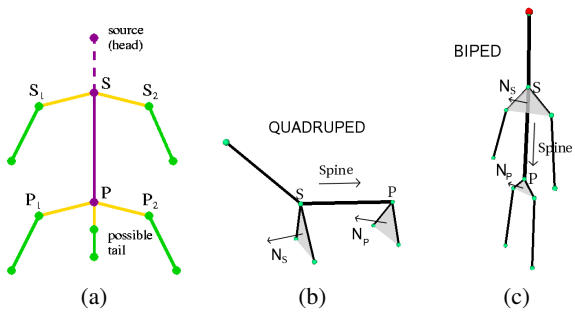


Figure 8: *(a) Minimal harmonic skeleton for a biped or a quadruped model. The symmetry axis is colored in purple and frozen edges are colored in orange. (b,c) Spine, N_P and N_S vectors for quadrupeds and bipeds.*

Actually, this heuristic is well-adapted for most quadrupeds, but not all. Indeed, vertebrate terrestrial quadrupeds can be classified into two groups, according to the orientation of their leg bones (see Figure 9): in the case of amphibians these bones approximately lie in a transversal plane (plane with constant altitude), while in the case of most mammals they lie in a sagittal plane (orthogonal to S_1S_2 and P_1P_2). While our test is adequate for "sagitally oriented" quadrupeds, it can fail for amphibians, for which the result can be the same than for bipeds: $|Spine.N_P| \approx 0$ and $|Spine.N_S| \approx 0$.

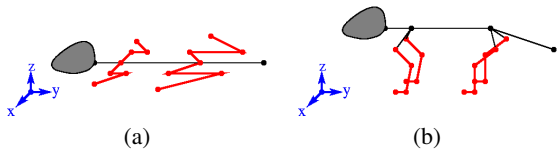


Figure 9: *(a) Schematic skeleton of an amphibian: the leg bones are in a tranversal plane ($z = cst$). (b) Mammal case: they are in a sagittal plane ($x = cst$).*

4.2. Biped embedding

If the character has been detected as a biped, we propose a special refinement of the harmonic skeleton. This refinement starts with the addition of several nodes to the graph:

- the spine, that is to say the edge SP, is subdivided into 4;
- a new node N is inserted on the symmetry axis before S;
- a new node J is inserted before N, and a new edge JM is added from J (M is a new extremum of the graph);
- each arm and each leg is subdivided into 3 edges;
- if there is a tail, it is subdivided into 4 edges.

The goal of this refinement is to match what would have created an artist. The nodes added to each arm will match elbows and wrists, while the nodes added to each leg will match knees and ankles; N will match the base of the neck, J the jaw and M the mouth. Notice that the source node and M do not match any real joint: these are in fact useful to better control the movement of the head and its size. We choose not to add edges for the rib cage, as it is not usually modeled for IK skeletons.

In order to shift some node embeddings and to embed the newly inserted nodes, we first give a reference frame to the model. This reference frame is defined by the previously introduced unit vector $Spine$, the unit vector $\frac{P_1P_2}{\|P_1P_2\|}$ and the unit vector $Spine \times \frac{P_1P_2}{\|P_1P_2\|}$, which gives the front-to-back (or back-to-front) direction. We can then embed newly inserted nodes, such as the nodes of the spine which can be slightly moved backward. To mimick what an artist would do, we have also chosen to unfreeze the SS_1 and SS_2 edges,and to embed S_1 and S_2 ahead of the embedding of S, in order to match clavicles. Regular nodes can be embedded either using a mean Euclidean position or a mean value for f *w.r.t.* the embeddings of their edge's endpoints, or fitting some geometric criterion, such as proposed by [TVD06]. The last solution can be particularly adapted for neck and wrists, which match constrictions of the shape.

© Association for Computing Machinery, Inc. 2007.

4.3. Quadruped embedding

Automatic animation skeleton generation is much less developed for four-footed animals than for bipeds. In order to refine the harmonic skeleton for parasagittally oriented quadrupeds, we based our work on the reference animation skeletons proposed by [RFDC05]. These IK skeletons were constructed by hand, from anatomical references [Cal75]. We add the same nodes to the harmonic graph as for bipeds, except that each front leg is subdivided into 5 edges, each back leg into 4 edges, and instead of having 2 edges between J and S (JN and NS), we have 5: the 4 added nodes will match the first, the second, the fourth and the seventh (which is the last) cervical vertebrae. We also subdivide the edge starting from the source node in 3; the first inserted node J' will match the jaw, while this time J will match the cranium. As for bipeds, M does not match any real joint and is useful to control the head's size and its movement. It will be put on top of the head of the character. We use the same reference frame as for biped embedding; here is how some of the joints are embedded: P is lifted up along the $Spine \times \frac{P_1P_2}{\|P_1P_2\|}$ direction from the simple embedding position (the center of its connected component for $f^{-1}(f(P))$) in order to be close to the back; nodes on SP are also lifted up, and so are S_1, S_2, P_1 and P_2; S is lifted up in order to match the pelvis' height; the first inserted nodes on each leg are moved along the $-Spine$ direction. We found that the best choice to embed the node J was near the neck constriction (actually a bit closer to the source joint); its value for f and exact location depends on the neck length. Finally, a simple solution for J' is along the $-Spine \times \frac{P_1P_2}{\|P_1P_2\|}$ direction from J, close to the chin.

5. Results and validation

Figures 2 and 10 to 12 show harmonic skeletons computed with our method. In these cases extrema have been selected by hand, because automatic computation of the extremal features can be quite slow. Thus, the threshold t_2 has not been used (it has been set to zero). No fine tuning of t_1 has been necessary: for almost all models, setting t_1 between 0.001 and 0.150 is sufficient. Except the selection of the extrema and t_1, the entire process is automatic; no post-processing has been applied.

5.1. Biped and quadruped embeddings

Figure 11 shows the harmonic skeleton computed from a biped model, compared with a standard handmade skeleton (from Autodesk's Maya software). We have not modeled the rib cage, as explained before. As for the other models, the symmetry axis is colored in purple and frozen edges are colored in orange. Even though the graph is more complex than the minimal harmonic graph for a biped (Figure 8 (a)) because we decided to model the fingers, the symmetry axis has been correctly detected. Another biped skeleton is shown on the right of the figure. We have chosen to embed extremal nodes onto corresponding vertices on the mesh, but we could have easily embedded them inside the model instead, using a close but regular value for f and the definition (3) of the center of a connected component.

Results on two quadruped models are shown on figures 2 and 12. The cat's tail is not considered as part of the symmetry axis, since its corresponding subtree on the harmonic graph is isomorphic to the back legs. Our algorithm provides animation skeletons close to the model's anatomy and to traditional IK skeletons. Nevertheless, some joints may need to be slightly displaced for better animation, particularly in the head. It is also noticeable that the very beginning of the tail is actually included in a frozen edge; this is correct since it corresponds to the first coccygeal vertebrae which are indeed attached to the sacrum [Cal75].

Our harmonic skeletons have been used for animation, as can be seen on Figure 1 and on the accompanying video.

5.2. Robustness

Figure 10 shows the robustness of the skeleton generation *w.r.t* the pose, mesh deformation and source vertex location.

Two different poses of the same character generate the same graph, with approximately the same values for f on each node, as long as the model is not stretched from one to the other. The reason is twofold: we are guaranteed that the extremal nodes correspond to the selected or computed extremal vertices, and f can be approximated as a distance *over the mesh* to the source vertex. Then, the embedding is most often the same since it does not depend on the leg orientation, for instance: it depends mostly on the computed reference frame, which is the same except if the back has been bended. It can also depends on the surface's local geometry, if we use constrictions to fix some joints such as the neck and wrists.

If the pose deformation is not isometric, we cannot be sure to get the same harmonic graph, from a theoretical point of view. However, stretching or shortening one leg in a homogeneous way does not change neither the graph nor its embedding, since for instance the ratio forearm length over arm length is not modified.

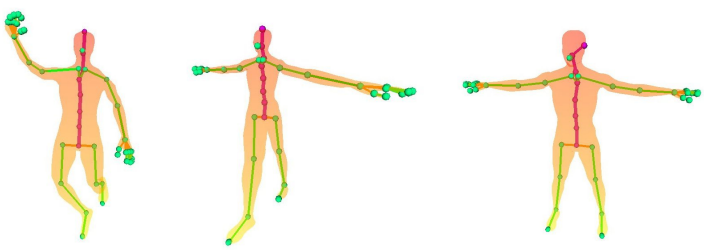


Figure 10: *Robustness of the skeleton generation* w.r.t *the pose (left), mesh deformation (middle) and the source vertex location (right). Compare to Figure 11.*

Our skeleton computation is also very robust *w.r.t* the

© Association for Computing Machinery, Inc. 2007.

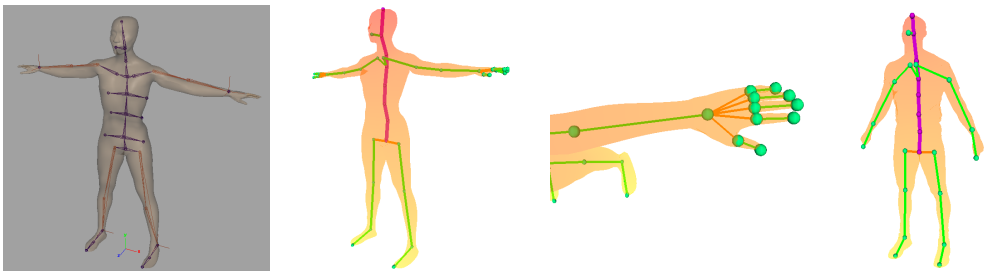


Figure 11: *Comparison on a standard biped model,* `MayaHuman`*, between a standard IK skeleton (left) and our harmonic skeleton (middle left). Middle right: hand close-up; right: harmonic skeleton for another biped model,* `MaleWB`*.*

[Liu et al. 2003] [Lien et al. 2006] [Tierny et al. 2006] Our method Horse anatomy

Figure 12: *Comparison on a horse model between several methods. Images are taken from the papers; the right image is taken from Wikipedia.*

source vertex location, as long as it is chosen on the head: even if it is not on the character's symmetry plane, the symmetry axis of the harmonic graph is recovered; then, since the embedding we propose does not depend on the source vertex location, it does not change.

5.3. Quantitative validation

To prove that our approach is useful, we have carried out a quantitative validation of our results: since [RFDC05] introduced parameters to define quadruped's skeletons (back and front leg height – or similarly spine tilt – and neck length, normalized by the spine length), we compared their values between our skeletons and IK skeletons, handmade from anatomical reference. Results for 6 models are provided in Table 1; in most cases our embedding of nodes S and J is correct, resulting in similar values between harmonic skeletons and IK skeletons for front leg height and neck length. The location of the pelvis is sometimes low in our harmonic skeletons, which explains the greater difference for back leg height.

5.4. Computation time

The Table 2 gives computation times for 5 models on a standard PC with a 2.4 GHz Pentium 4 processor. Even for a dense mesh, our algorithm generates the skeleton in less than 1 minute. The memory requirement is also low: at most 350 MB for a model made of 300,000 faces, 1.5 MB for a model with 15,000 faces (including the storage of the mesh). Most of the time is spent on the harmonic function computation; graph computation is then done in $O(n \log n)$ time for a mesh with n faces [CMEH*03], and embedding is done in nearly linear time because we only compute ray/mesh intersections for some joints in order to get their distance to the mesh, and the number of joints does not depend on the mesh's complexity.

Mesh	Back leg		Front leg		Neck	
	Harmo.	IK	Harmo.	IK	Harmo.	IK
Cat	1.2	1.3	1.2	1.2	0.4	0.4
Cow	1.0	1.1	0.9	0.9	0.3	0.4
Dog	1.3	1.3	1.1	1.2	0.5	0.4
Elephant	1.4	1.6	1.4	1.4	0.3	0.3
Horse	1.3	1.7	1.4	1.6	0.7	1.0
Panther	1.0	1.1	0.9	1.0	0.4	0.5

Table 1: *Parameter comparison between our harmonic skeletons and hand-built IK skeletons.*

Mesh	Nb. faces	Graph	Embedding	Total
Cat	2,566	0.085	0.108	0.193
`MayaHuman`	14,118	0.634	0.139	0.773
Octopus	33,058	1.393	0.061	1.454
Horse	96,966	6.268	3.525	9.793
`MaleWB`	296,272	30.816	5.230	36.046

Table 2: *Computation time (in seconds) for some meshes.*

6. Conclusion

In this paper we have presented a fully automatic method to compute an animation skeleton from a 3D meshed model in

© Association for Computing Machinery, Inc. 2007.

a few seconds after the selection of an initial point. In the case of most bipeds or quadrupeds, this skeleton fits the animation skeleton that would be hand-built by an expert starting from anatomical boards, and is thus adapted for realistic animation. The main idea is to construct the Reeb graph of a harmonic function, which gives the overall morphological structure of the model (especially its symmetry axis), then to refine and embed it using anatomical information. There are two main restrictions on the input mesh: it should be a triangulated 2-manifold (with or without boundary), and, in order to recover the symmetry axis of the shape's morphology, it should not have handles (otherwise the Reeb graph contains cycles). Although the method is fully automatic, the user can control the skeleton generation by tuning a few optional parameters. This tool has been designed both to help artists and to allow non-experts to quickly generate skeletons which can be used for realistic character animation. Computed skeletons can be edited and refined, for instance to add joints that correspond to wings or to the trunk of an elephant.

Given this skeleton generation process, we see three promising research directions. First, each vertex of the mesh is related to the joints of the skeleton, since we have given values for the harmonic function to the graph's nodes, and hence the skeleton's joints; these relations may be used to enhance skinning weights. Second, our semantic decomposition of the graph may also be used to define heuristics that give adapted skinning weights: weights may vary according to the meaning of neighboring joints. It may also help for automatic mesh segmentation into anatomically meaningful regions. Finally, even if not embedded to match the model's anatomy, the harmonic graph may be useful for other applications (e.g. shape matching), since its construction is robust and does not create unnecessary nodes.

Acknowledgments

The authors would like to thank Lionel Revéret for interesting discussions at the beginning of this work. The horse and `MaleWB` models are courtesy of Cyberware. The `MayaHuman` model is courtesy of Autodesk.

References

[BL99] BLOOMENTHAL J., LIM C.: Skeletal methods of shape manipulation. In *Shape Modeling International* (1999).

[Blu67] BLUM H.: A transformation for extracting new descriptors of shape. In *Symposium on Models for the Perception of Speech and Visual Form* (1967), pp. 362–380.

[Cal75] CALDERON W.: *Animal Painting and Anatomy*. Dover, 1975.

[CMEH*03] COLE-MCLAUGHLIN K., EDELSBRUNNER H., HARER J., NATARAJAN V., PASCUCCI V.: Loops in reeb graphs of 2-manifolds. In *Symposium on Computational Geometry* (2003), pp. 344–350.

[CSYB05] CORNEA N., SILVER D., YUAN X., BALASUBRAMANIAN R.: Computing hierarchical curve-skeletons of 3d objects. *The Visual Computer 21*, 11 (2005), 945–955.

[DeF99] DEFRAYSSEIX H.: An heuristic for graph symmetry detection. In *Symposium on Graph Drawing* (1999), pp. 276–285.

[DEG*99] DEMMEL J., EISENSTAT S., GILBERT J., LI X., LIU J.: A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications 20*, 3 (1999), 720–755.

[DKG05] DONG S., KIRCHNER S., GARLAND M.: Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *Computer Aided Geometric Design, Special issue on Geometry Processing 22*, 5 (2005), 392–423.

[DS06] DEY T., SUN J.: Defining and computing curve-skeletons with medial geodesic function. In *Symposium on Geometry Processing* (2006), pp. 143–152.

[FK97] FOMENKO A., KUNII T.: *Topological Modeling for Visualization*. Springer-Verlag, 1997.

[GS01] GAGVANI N., SILVER D.: Animating volumetric models. *Graphical Models 63*, 6 (2001), 443–458.

[HSKK01] HILAGA M., SHINAGAWA Y., KOMURA T., KUNII T.: Topology matching for fully automatic similarity estimation of 3d shapes. In *SIGGRAPH* (2001), pp. 203–212.

[KT03] KATZ S., TAL A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. In *SIGGRAPH* (2003).

[LKA06] LIEN J., KEYSER J., , AMATO N.: Simultaneous shape decomposition and skeletonization. In *ACM Symposium on Solid and Physical Modeling* (2006), pp. 219–228.

[LV99] LAZARUS F., VERROUST A.: Level set diagrams of polyhedral objects. In *ACM Symposium on Solid Modeling* (1999).

[LWM*03] LIU P., WU F., MA W., LIANG R., OUHYOUNG M.: Automatic animation skeleton construction using repulsive force field. In *Pacific Graphics* (2003), pp. 409–413.

[NGH04] NI X., GARLAND M., HART J.: Fair morse functions for extracting the topological structure of a surface mesh. In *SIGGRAPH* (2004), pp. 613–622.

[Ree46] REEB G.: Sur les points singuliers d'une forme de pfaff complètement intégrable ou d'une fonction numérique. *Comptes-Rendus de l'Académie des Sciences 222* (1946), 847–849.

[RFDC05] REVÉRET L., FAVREAU L., DEPRAZ C., CANI M.: Morphable model of quadruped skeletons for animating 3d animals. In *Symposium on Computer Animation* (2005).

[SF01] STEINER D., FISCHER A.: Topology recognition of 3d closed freeform objects based on topological graphs. In *Pacific Graphics* (2001), pp. 82–88.

[SKK91] SHINAGAWA Y., KUNII T., KERGOSIEN Y.: Surface coding based on morse theory. *IEEE Computer Graphics and Applications 11*, 5 (1991), 66–78.

[SKS06] SIMARI P., KALOGERAKIS E., SINGH K.: Folding meshes: Hierarchical mesh segmentation based on planar symmetry. In *Symposium on Geometry Processing* (2006).

[TVD06] TIERNY J., VANDEBORRE J., DAOUDI M.: 3d mesh skeleton extraction using topological and geometrical analyses. In *Pacific Graphics* (2006), pp. 409–413.

[WP02] WADE L., PARENT R.: Automated generation of control skeletons for use in animation. *The Visual Computer 18* (2002).

[ZMT05] ZHANG E., MISCHAIKOW K., TURK G.: Feature-based surface parameterization and texture mapping. *ACM Transactions on Graphics 24*, 1 (2005), 1–27.

© Association for Computing Machinery, Inc. 2007.

$\therefore$

A.6 SIMPLE FLEXIBLE SKINNING BASED ON MANIFOLD MODELING

Franck Hétroy, Cédric Gérot, Lin Lu, Boris Thibert

International Conference on Computer Graphics Theory and Applications (GRAPP), 2009.

SIMPLE FLEXIBLE SKINNING BASED ON MANIFOLD MODELING

Franck Hétroy[1,2], Cédric Gérot[3], Lin Lu[4], Boris Thibert[1]
Franck.Hetroy@imag.fr, Cedric.Gerot@gipsa-lab.inpg.fr, llu@cs.hku.hk, Boris.Thibert@imag.fr

[1] *Université de Grenoble & CNRS, Laboratoire Jean Kuntzmann, Grenoble, France*
[2] *INRIA Grenoble - Rhône-Alpes, Grenoble, France*
[3] *Université de Grenoble & CNRS, GIPSA-Lab, Grenoble, France*
[4] *Department of Computer Science, The University of Hong Kong, Hong Kong, China*

Keywords: skinning, manifold atlas, covering.

Abstract: In this paper we propose a simple framework to compute flexible skinning weights, which allows the creation from quasi-rigid to soft deformations. We decompose the input mesh into a set of overlapping regions, in a way similar to the constructive manifold approach. Regions are associated to skeleton bones, and overlaps contain vertices influenced by several bones. A smooth transition function is then defined on overlaps, and is used to compute skinning weights. The size of overlaps can be tuned by the user, enabling an easy control of the desired type of deformations.

1 INTRODUCTION

Skeletal animation is a widespread technique to deform articulated shapes. It uses a joint hierarchy called *skeleton*; during the animation, joints are translated and/or rotated then each vertex of the shape (usually represented by a mesh) is deformed with respect to the closest joints. The process that describes the skin deformation is called *skinning*. Many skinning techniques attach joint (or bone) weights to each vertex of the mesh; a weight specifies the amount of influence of the corresponding joint on the vertex. Defining proper values for joint weights is often time-consuming for the animator. Usually, weights are defined using the Euclidean distance between the vertices and the joints. A basic painting tool (or equivalent) can be applied manually to quantify which vertices are influenced by a given joint. Careful manual tuning is then required to set up weights that give the desired deformation.

In this paper, we propose a simple framework to automatically compute skinning weights, with a user control on the type of deformation. We get inspiration from the concept of constructive manifold atlas (Grimm and Zorin, 2005). Contrary to piecewise modeling, an atlas allows to construct a surface from pieces of surface which overlap substantially instead of abutting only along their boundaries. As a consequence, when one piece is stretched or moved, the overlapping pieces follow this deformation or motion. We use this idea to compute skinning weights for any shape, proceeding in two steps. Firstly, a covering of the mesh, with regions associated to skeleton bones, is defined (Section 3). This covering can be controlled on the overlapping areas. Secondly, a partition of the unity is defined on this covering for each vertex of the mesh, providing the weights for the skinning (Section 4).

Our weight computation scheme is both simple and fast. Control is easy since only one parameter has to be tuned in order to move from a quasi-rigid deformation to a soft one, and no manually tuned example nor additional tool is required as input. We demonstrate the effectiveness of our framework on a set of examples (Section 5).

2 RELATED WORK

2.1 Flexible skinning

Most skinning weight computation methods try to generate ideal weights for realistic character animation. They can rely on geometric features, such as the medial axis of the object (Bloomenthal, 2002) or a mesh segmentation (Katz and Tal, 2003; Attene et al., 2006), or on example poses (e.g. (Merry et al., 2006; Wang et al., 2007; Weber et al., 2007)). An increasingly popular solution is to solve a heat equation for each joint in order to automatically set the weights associated to this joint (Baran and Popović, 2007; Weber et al., 2007). However, these solutions usually do not allow for flexible skinning.

To the best of our knowledge, only a few skinning methods allow different kinds of deformations. One of them is to

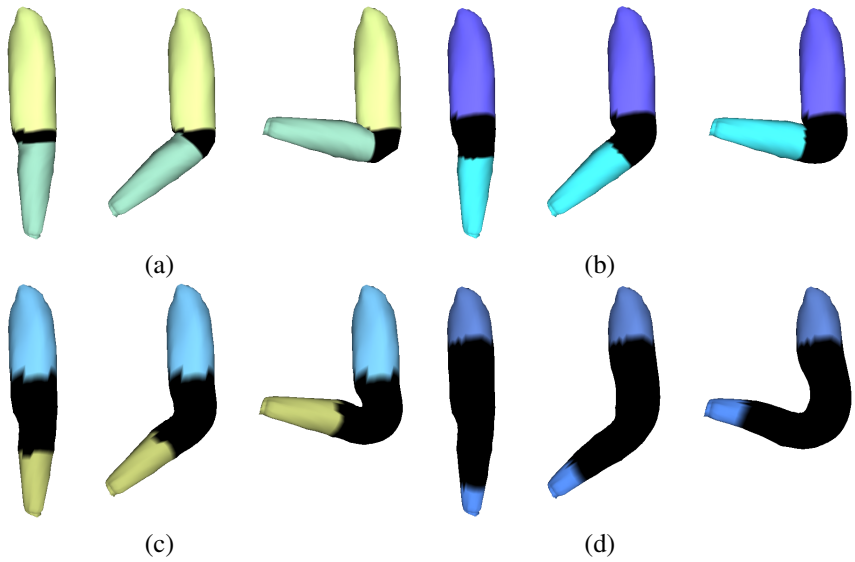


Figure 1: Rest pose, medium deformation and large deformation around an elbow. $K = 0.1, 0.5, 1.0$ and 2.0 for (a), (b), (c) and (d) respectively. Overlap areas are shown in black. Overlaps and weights were computed using a geodesic distance, and deformations were created using the technique of (Kavan et al., 2007).

use spline-aligned deformations instead of the traditional Linear Blend Skinning (LBS), which can be mixed with user-designed deformation styles (Forstmann et al., 2007). Another solution is to compute the set of possible new locations for a vertex deformed with LBS and let the user choose the one he wants (Mohr et al., 2003). Recently, Rohmer et al. proposed a local volume preservation technique which enables the creation of both rubber-like and realistic deformations for organic shapes, depending on the correction map applied to skinning weights (Rohmer et al., 2008). The solution we suggest is more flexible in the sense that any deformation, from quasi-rigid to soft, can be created, and any skinning method can be used: for instance LBS, (Merry et al., 2006; Kavan et al., 2007). It also lies in the general (rigid) skeleton-based animation framework, and do not need the creation of new tools such as spline curves.

Our method can be related to the "mesh forging" approach of Bendels and Klein (Bendels and Klein, 2003), except that we propose a Hermite function as a transition function between two bones, while they let the user draw the function.

2.2 Modeling with an atlas

Surface modeling with an atlas has properties which lends itself to the skinning problem. Indeed constructive manifold definitions (Grimm and Zorin, 2005) represent a surface as a set of blended embedded planar disks. The blending is performed as a convex combination whose weights are defined as a partition of the unity overall the planar disks. Hence the surface is made up with 3D regions which overlap substantially and are glued together. As a consequence, when an embedded planar disk is stretched or moved, the overlapping regions are stretched or moved accordingly. Defining such a set of regions per joint of the skeleton provides a skinning.

However, this construction makes sense only if the planar disks are linked together with transition functions. These functions indicate which embedded points have to be combined together in the blending process. To do so, either a *proto-manifold* associated with a mesh with a large number of pieces (at least one per vertex) is defined (Grimm and Hughes, 1995; Navau and Garcia, 2000; Ying and Zorin, 2004), or a pre-defined manifold with a small number of pieces, but in general not adapted to the particular geometry to be represented is used (Grimm, 2004). These constructions target a global highly-continuous parameterization of the surface. This implies major contraints on the definition of the transition functions. Reversely, an atlas can be constructed from the final surface to be represented. The global parameterization of the surface is used for high-quality sampling, texture mapping or reparameterization (Praun et al., 2000). In this case again, the components of the atlas have to be defined explicitly and with continuity constraints.

Real-time constraints impose to deal with small structures and to consider meshes as C^0-surfaces. Hence, we propose to adapt this parameterization-oriented framework onto a lighter one, sufficient for skinning and providing a better control on the overlapping influences of different skeleton bones than other skinning algorithms.

3 C^0 ATLAS DEFINITION

Our work takes as input a closed mesh and an embedded animation skeleton. As stated in Section 2.2, we adapt the manifold modeling with an atlas onto a lighter framework

sufficient for skinning. Following constructive manifold approach, we decompose the mesh into overlapping regions. Despite the fact that these regions are not necessarily homeomorphic to discs, they will be interpreted as charts with transition functions implicitely defined by the shared faces. In order to control these overlapping areas, we first segment the mesh into a partition of regions associated to skeleton bones, and then stretch these regions onto a covering of the mesh. Note that regions are not restricted to cylindrical shapes with at most two boundaries.

3.1 Initial mesh segmentation

To decompose the mesh into overlapping regions, we need as a preprocess its segmentation into regions associated to skeleton bones. Any skeleton-based segmentation method can be used, such as for instance (Katz and Tal, 2003; de Goes et al., 2008) which also use segmentation to create animations. In our implementation, we use a simple yet robust automatic mesh segmentation algorithm. Our approach is to first find the boundaries of the regions, which should be associated to skeleton joints since regions are associated to skeleton bones. The boundary B associated to joint J is defined as the intersection between the input mesh and a plane P going through J and orthogonal to a plane Q (see Figure 2). Q is defined by the two bones incident to J. In case more than two bones are incident to J (this is for instance the case of the pelvis joint for a human model), we can use the skeleton's hierarchy to select two of them. There is an infinite number of possible planes P, but each one can be defined by its normal n, which lies in the plane Q. In practice we compute a discrete set of planes $P_0, \ldots, P_{k-1}$, by selecting a random n_0 normal vector and then rotating it around J with an angle $2\pi i/k, 1 \leq i < k$. Then we keep the plane such that the length of the corresponding boundary curve B is minimum.

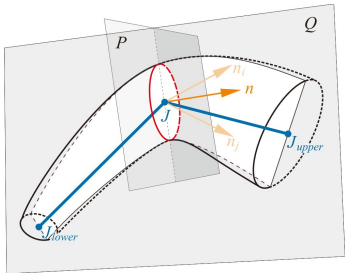


Figure 2: Each boundary is defined with respect to a plane P going through a joint J.

Although this method is quite simple, it provides segmentations which are robust to noise on the input mesh, to the initial pose of the character and to the location of joints, as can be seen on Figure 3. Once again, we emphasize that any other skeleton-based segmentation method can be applied instead of this one, as a pre-processing step for overlap generation.

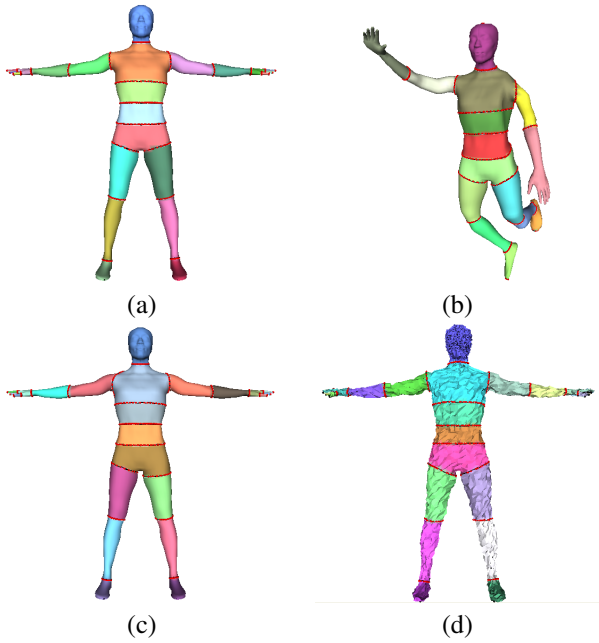


Figure 3: Segmentation results for (a) a human model, (b) the same model with a different pose (and no hand nor arm joint in the skeleton), (c) the same model with different right shoulder and left hip joint locations, and (d) the same model with noised vertex locations.

3.2 Overlap generation

We now describe how we generate a mesh decomposition into overlapping pieces from this segmentation.

Suppose that the mesh M is decomposed into r regions $\{R_j\}_{j=1}^{r}$; we note $\{B_i\}_{i=1}^{b}$ the b boundaries between these regions. Besides, each B_i has 2 adjacent regions denoted by R_{i_1} and R_{i_2}; each R_j has m boundaries, denoted by $\{B_{j,k}\}$, with $\forall j,k, B_{j,k} = B_{k,j}$. Generation of overlaps consists in growing each region R_j into a new region R'_j with a distance criterion: R'_j is connected and $R_j \subset R'_j$. This is done by integrating vertices of neighbouring regions to R'_j. Thus, each boundary $B_{j,k}$ of R_j is modified into a new boundary $B'_{j,k}$ of R'_j, with $B'_{j,k} \neq B'_{k,j}$ (the new boundary of R'_k). Vertices between $B'_{j,k}$ and $B'_{k,j}$ are in the *overlap area* of R'_j and R'_k. Note that a whole region R'_k may belong to the overlap area of a neighbouring region R'_j (see Figure 4).

To compute the overlap areas, we compute for each vertex v of the mesh its distance to all $B_{j,k}$, and we let the user choose a size parameter K. Then, we use the length $L_{j,k}$ of $B_{j,k}$ as the criterion to generate the overlap area between R'_j and R'_k: we mark each vertex with distance to $B_{j,k}$ lower than $K * L_{j,k}/\pi$ as in this overlap area. (Baran and Popović, 2007) claims that the range of a transition between two bones (that is to say, the area of the region where vertices are influenced by both bones) must be roughly

proportional to the distance from the joint to the surface. This corresponds to $K = 0.5$.

In our implementation, the same parameter is used for all areas, but other solutions can be applied: for instance, K can be chosen according to the type of skeleton joint, in case semantic information is attached to joints (Aujay et al., 2007).

Different kinds of distances can be used: Euclidean distance, approximated geodesic distance or distance based on a harmonic function, for instance. We tested several of them and discuss results in Section 5.2.

4 COMPUTATION OF SKINNING WEIGHTS

In the manifold constructive approach, a partition of the unity defined on a proto-manifold is used to blend embedded pieces. In the same way, we define skinning weights as a partition of the unity on the covering defined in Section 3.

We define weights that depend on the mesh covering $\{R'_j\}_{j=1}^r$ defined in Section 3.2. In each extended region R'_j a distance map $d_j(v)$ is specified. It gives to every vertex v of the region R'_j its distance to the boundary of the region (computed as the lowest distance from v to all $B'_{j,k}$). As in Section 3.2, this can be a Euclidean or geodesic distance, or anything else. We tested Euclidean, approximated geodesic and harmonic distances; see Section 5.2 for results and a discussion.

Let δ_j be the maximal distance to the boundary in R'_j: $\delta_j = \max_{v \in R'_j} d_j(v)$. Let $s(l)$ be the cubic function which satisfies the Hermite conditions $s(0) = 0$, $s(1) = 1$, $s'(0) = s'(1) = 0$: $s(l) = -2l^3 + 3l^2$. This cubic function lets us define weights which decrease smoothly towards 0 as the vertex v is closer to the region boundary, providing visually better results (see Section 5.3). However, weights can be defined with any function such that $s(0) = 0$ and $s(1) = 1$.

We define unnormalized weights $\sigma_j(v)$ as $\sigma_j(v) = s(\frac{d_j(v)}{\delta_j})$. Let $I(v)$ be the set of indices of regions the vertex v belongs to $I(v) = \{j \in \{1,\ldots,n\} : v \in R'_j\}$. Normalized weights $\omega_j(v)$ are then defined as $\omega_j(v) = \frac{\sigma_j(v)}{\sum_{i \in I(v)} \sigma_i(v)}$.

Because the regions R'_j define a covering of the surface and s in monotonic from $[0,1]$ onto $[0,1]$, the denominator is never equal to zero and $\omega_j(v) \in [0,1]$. Moreover these well-defined weights define a partition of the unity associated to this covering: for every vertex v of the mesh, $\sum_{j \in I(v)} \omega_j(v) = 1$.

Note that for non-overlapped vertices, $I(v)$ is reduced to a singleton $\{j\}$ and $\omega_j(v) = 1$. For a vertex v belonging to the boundary of a region R'_j, we have $d_j(v) = 0$, thus $\sigma_j(v) = 0$ and $\omega_j(v) = 0$.

5 RESULTS AND DISCUSSION

Some deformation results are shown on Figures 1, 4, 6 and 7. In all cases the Dual Quaternion technique (Kavan et al., 2007) was used to deform the meshes. The segmentation pre-processing step is done in real-time, and so is done the weight computation. Time to compute the overlap areas highly depends on the chosen distance function: it is almost real time using a Euclidean distance, but lasts a few seconds using an approximated geodesic distance, on a low-end PC.

Figure 4 shows the mesh covering defined for two standard models, and examples of deformations that can be generated in a few minutes using our framework. K was set to 0.5 (resp. 0.2) for all joints of the human (resp. Homer) model. Overlaps as well as skinning weights were computed with an approximated geodesic distance, using Dijkstra's algorithm on the mesh's vertices. As input we only used the two mesh models and their corresponding animation skeletons.

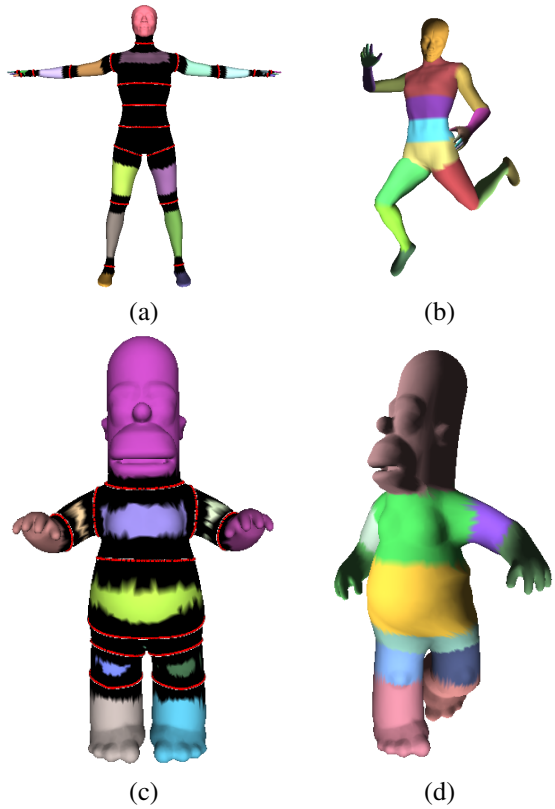


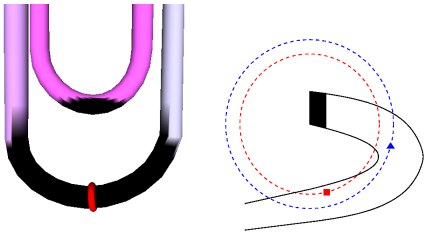
Figure 4: Computed covering (a,c) and deformation (b,d) for two models. Overlap areas are shown in black. Note that some vertices may belong to three or more overlapping areas, especially around the spine and the pelvis.

5.1 Influence of the overlap size

As can be seen on Figure 1, the overlap size $K * L_{j,k}/\pi$ influences the behavior of the deformation around a joint. For a small value of K, only a few number of vertices around the joint are smoothly bended out: the deformation is quasi-rigid. As K becomes larger, the deformation becomes elastic. Thus tuning K allows for various kinds of deformations.

5.2 Choice of the distance function

As stated in Section 3, several distance functions can be used to compute both overlap areas and skinning weights. Using the Euclidean distance is the simplest and fastest solution. However, in some cases it generates artefacts (see Figure 5). For instance, if some part of the input mesh is close to a joint related to other regions, vertices in this part can be wrongly set to be in an overlap area of the joint. This drawback can sometimes be corrected using the skeleton's hierarchy, by preventing vertices from belonging to overlap areas of joints that are far from their bone in the hierarchy, but this is not always possible. Euclidean distance can also generate artefacts for weight computation, in case of curved regions: see for instance Figure 5 (b).



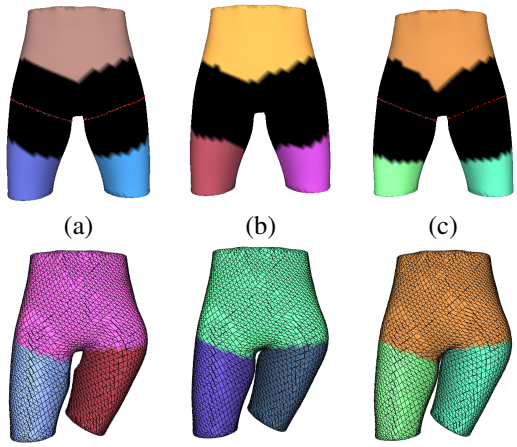
(a) (b)

Figure 5: Artefacts using the Euclidean distance (overlap areas are shown in black). (a) For the overlap generation: an overlap area around a joint can be disconnected. (b) For the weight computation: the point represented by a square is closest to the boundary of the region than the point represented by a triangle.

Figure 6 shows the deformation around a pelvis joint using Euclidean (first row), approximate geodesic (second row) or harmonic (third row) distance. Approximated geodesic distance has been computed with Dijkstra's algorithm. Following an idea from (Aujay et al., 2007), we set two boundary conditions for the computation of the harmonic distance: the points on boundary curves have zero distance and the farthest points to these curves have a distance set to their approximated geodesic distance to these curves. Although the overlap areas between the three regions (waist and both thighs) are quite similar, a small artefact can be noticed for the Euclidean distance, due to the high influence the right thigh has on vertices close to the left thigh/pelvis boundary.

5.3 Choice of the weight function

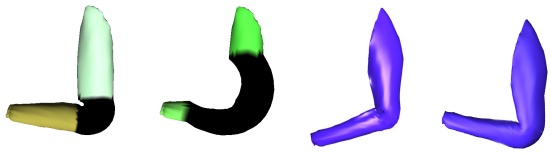
Results of deformations using a linear function instead of s to compute the skinning weights are shown on Figure 7 (a,b). They look much less natural (compare with Figure 1 (b,d)), because of the sharp decrease or increase of influence of bones near the overlap boundaries. On the contrary, our cubic function s increases very slowly around $l = 0$ and $l = 1$, leading to visually better results.



(a) (b) (c)

(d) (e) (f)

Figure 6: Overlap areas (a,b,c) and deformation (d,e,f) around a pelvis joint using Euclidean (a,d), approximated geodesic (b,e) and harmonic (c,f) distance. K was set to 0.5 in the first two cases, and to 0.4 in the harmonic case.



(a) (b) (c) (d)

Figure 7: (a,b) Deformations using a linear function instead of a cubic one with $K = 0.5$ (a) or $K = 2$ (b). We used approximate geodesic distance to compute overlap areas and weights. (c) Deformation using Blender's paint tool. (d) Deformation using harmonic weights (Baran and Popović, 2007).

5.4 Comparison with standard methods

Deformations obtained using two standard weight computation methods and Dual Quaternion technique are shown on Figure 7 (c,d). Using the paint tool (available in common software such as Autodesk's Maya or Blender), it took approximately half an hour to get a relatively decent result. The painted area corresponds to the overlap area shown on Figure 1 (b). The use of a harmonic function (Baran and Popović, 2007) is as fast as our technique, but do not allow for accurate control over the size of the deformed region.

6 CONCLUSION

We have presented a simple way to compute flexible skinning weights for skeleton-based animation, based on the concept of manifold modeling. Starting from a segmentation of the input mesh into regions corresponding to skeleton bones, we generate overlaps by extending each region around joints. Size of these overlaps is controlled by a simple parameter, that can be user-chosen or automatically computed. Then, vertices belonging to an overlap area are influenced by bones related to all regions that overlap. Skinning weights are defined using a simple smooth function based on the distance to the overlap boundary.

Results show that this framework allows to create from quasi-rigid to soft deformations, depending on the overlap size. Using a geodesic distance instead of a Euclidean one to create overlaps and compute skinning weights is more time-consuming, but avoids some artefacts. We believe our method can be especially useful for non-expert animators, since it is simple (only one parameter is to set) and fast to use.

Further work includes anatomic information into the overlapping width definition. Such information can be derived from semantic information associated with skeleton (Aujay et al., 2007). Besides, providing a skinning framework for multiresolution animated meshes, founded on our pseudo-parameterization on the initial mesh, would be a further development in the similarity with manifold parameterization.

ACKNOWLEDGEMENTS

This work was partially supported by the IMAG, ELESA and INRIA through the MEGA project and the ANR through the MADRAS project (ANR-07-MDCO-015). Part of this work was done while Lin Lu was visiting INRIA with an INRIA Internship grant.

REFERENCES

Attene, M., Spagnuolo, M., and Falcidieno, B. (2006). Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer*, 22(3):181–193.

Aujay, G., Hétroy, F., Lazarus, F., and Depraz, C. (2007). Harmonic skeleton for realistic character animation. In *Symposium on Computer Animation*, pages 151–160, San Diego, USA.

Baran, I. and Popović, J. (2007). Automatic rigging and animation of 3d characters. *ACM Transactions on Graphics (SIGGRAPH proceedings)*, 26(3):72.

Bendels, G. and Klein, R. (2003). Mesh forging: Editing of 3d-meshes using implicitly defined occluders. In *Symposium on Geometry Processing*, pages 207–217, Aachen, Germany.

Bloomenthal, J. (2002). Medial-based vertex deformation. In *Symposium on Computer Animation*, pages 147–151, San Antonio, USA.

de Goes, F., Goldenstein, S., and Velho, L. (2008). A hierarchical segmentation of articulated bodies. *Computer Graphics Forum (Symposium on Geometry Processing proceedings)*, 27(5):1349–1356.

Forstmann, S., Ohya, J., Krohn-Grimberghe, A., and McDougall, R. (2007). Deformation styles for spline-based skeletal animation. In *Symposium on Computer Animation*, pages 141–150, San Diego, USA.

Grimm, C. (2004). Parameterization using manifolds. *International Journal of Shape Modeling*, 10(1):51–80.

Grimm, C. and Hughes, J. (1995). Modeling surfaces of arbitrary topology. In *SIGGRAPH*, pages 359–367, Los Angeles, USA.

Grimm, C. and Zorin, D. (2005). Surface modeling and parameterization with manifolds. In *SIGGRAPH Course Notes*, Los Angeles, USA.

Katz, S. and Tal, A. (2003). Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics (SIGGRAPH proceedings)*, 22(3):954–961.

Kavan, L., Collins, S., Zara, J., and O'Sullivan, C. (2007). Skinning with dual quaternions. In *Symposium on Interactive 3D Graphics and Games*, pages 39–46, Seattle, USA.

Merry, B., Marais, P., and Gain, J. (2006). Animation space: a truly linear framework for character animation. *ACM Transactions on Graphics*, 25(4):1400–1423.

Mohr, A., Tokheim, L., and Gleicher, M. (2003). Direct manipulation of interactive character skins. In *Symposium on Interactive 3D Graphics and Games*, pages 27–30, Monterey, USA.

Navau, J. C. and Garcia, N. P. (2000). Modeling surfaces from meshes of arbitrary topology. *Computer Aided Geometric Design*, 17(1):643–671.

Praun, E., Finkelstein, A., and Hoppe, H. (2000). Lapped textures. In *SIGGRAPH*, pages 465–470, New Orleans, USA.

Rohmer, D., Hahmann, S., and Cani, M. (2008). Local volume preservation for skinned characters. *Computer Graphics Forum (Pacific Graphics proceedings)*, 27(7).

Wang, R., Pulli, K., and Popović, J. (2007). Real-time enveloping with rotational regression. *ACM Transactions on Graphics (SIGGRAPH proceedings)*, 26(3):73.

Weber, O., Sorkine, O., Lipman, Y., and Gotsman, C. (2007). Context-aware skeletal shape deformation. *Computer Graphics Forum (Eurographics proceedings)*, 26(3):265–274.

Ying, L. and Zorin, D. (2004). A simple manifold-based construction of surfaces of arbitrary smoothness. *ACM Transactions on Graphics (SIGGRAPH proceedings)*, 23(3):271–275.

$\therefore$

A.7 SEGMENTATION OF TEMPORAL MESH SEQUENCES INTO RIGIDLY MOVING COMPONENTS

Romain Arcila, Cédric Cagniart, Franck Hétroy, Edmond Boyer, Florent Dupont

Graphical Models 75 (1), Elsevier, 2013.

Segmentation of temporal mesh sequences into rigidly moving components

Romain Arcila[a,b], Cédric Cagniart[c,a], Franck Hétroy[a,*], Edmond Boyer[a], Florent Dupont[b]

[a] *Laboratoire Jean Kuntzmann, Inria & Grenoble University, France*
[b] *LIRIS, CNRS & Université de Lyon, France*
[c] *Computer Aided Medical Procedures & Augmented Reality (CAMPAR), Technische Universität München, Germany*

Abstract

In this paper is considered the segmentation of meshes into rigid components given temporal sequences of deforming meshes. We propose a fully automatic approach that identifies model parts that consistently move rigidly over time. This approach can handle meshes independently reconstructed at each time instant. It allows therefore for sequences of meshes with varying connectivities as well as varying topology. It incrementally adapts, merges and splits segments along a sequence based on the coherence of motion information within each segment. In order to provide tools for the evaluation of the approach, we also introduce new criteria to quantify a mesh segmentation. Results on both synthetic and real data as well as comparisons are provided in the paper.

Keywords: mesh sequence, segmentation, topology, mesh matching, rigid part

1. Introduction

Temporal sequences of deforming meshes, also called mesh animations [1, 43], are widely used to represent 3D shapes evolving through time. They can be created from a single static mesh, which is deformed using standard animation techniques such as skeletal subspace deformation [25] or cloth simulation methods [15]. They can also be generated from multiple video cameras [38, 43]. In this case, meshes are usually independently estimated at each frame using 2D visual cues such as silhouettes or photometric information.

These deforming mesh sequences can be edited [21, 8], compressed [24], or used for deformation transfer [39, 23]. When the shape represents an articulated body, such as a human or animal character, identifying its rigid, or almost rigid, parts offers useful understanding for most of these applications. To recover the shape kinematic structure, an animation skeleton can be extracted from the deforming mesh sequence [1]. Another strategy is to segment the meshes into components that move rigidly over the sequence [22, 19, 44, 29]. In both cases, motion information is required in order to cluster mesh elements into regions with rigid motions. Most existing approaches assume that surface registration is available for that purpose and consider as the input a single mesh that deforms over time. In contrast, we do not make any assumptions on the input mesh sequences and we propose to match meshes and recover their rigid parts simultaneously. Consequently, our method applies to any kind of deforming mesh sequence including inconsistent mesh sequences such as provided by multi-camera systems.

*Corresponding author.
Email addresses: `Franck.Hetroy@grenoble-inp.fr` (Franck Hétroy), `Edmond.Boyer@inria.fr` (Edmond Boyer), `Florent.Dupont@liris.cnrs.fr` (Florent Dupont)

1.1. Classification of mesh sequences

In order to distinguish between mesh sequences with or without temporal coherence, i.e. with or without a one-to-one correspondence between vertices of successive meshes, we first introduce the following definitions.

Definition 1.1 (Temporally coherent mesh sequence (TCMS), temporally incoherent mesh sequence (TIMS))**.** *Let $MS = \{M^i = (V^i, E^i, F^i), i = 1 \dots f\}$ be a mesh sequence: V^i is the set of vertices of the ith mesh M^i of the sequence, E^i its set of edges and F^i its set of faces. If the connectivity is constant over the whole sequence, that is to say if there is an isomorphism between any E^i and $E^j, 1 \leq i, j \leq f$, then MS is called a* temporally coherent mesh sequence (TCMS)*. Otherwise, MS is called a* temporally incoherent mesh sequence (TIMS)*.*

Note that the definition of TCMS not only implies that the number of vertices remains constant through time, but also that there is a one-to-one correspondence between faces of any two meshes. As a consequence, topological changes (genus and number of connected components) are not possible in a TCMS.

Figure 1 shows an example of a TCMS and an example of a TIMS.

1.2. Classification of mesh sequence segmentations

In contrast to single mesh segmentation that consists in grouping mesh vertices into spatial regions the segmentation of a mesh sequence can have various interpretations with respect to time and space. We propose here three different definitions. Let us first recall a formal definition of a static mesh segmentation.

Definition 1.2 (Segmentation of a static mesh [33])**.** *Let $M = (V, E, F)$ be a 3D surface mesh. A* segmentation *Σ of M is the set of sub-meshes $\Sigma = \{M_1, \dots, M_k\}$ induced by a partition of either V or E or F into k disjoint sub-sets.*

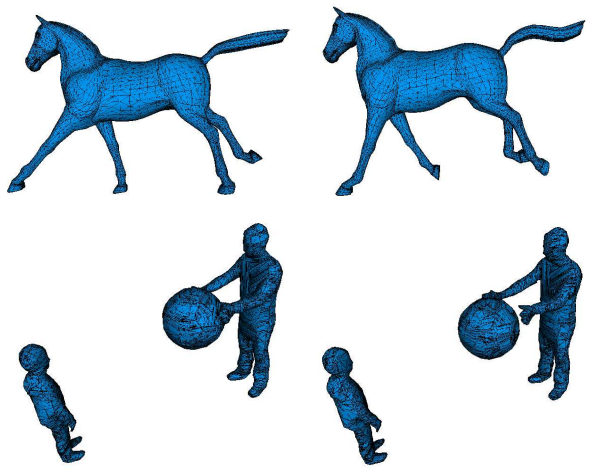


Figure 1: First row: two consecutive frames of a TCMS. Second row: two consecutive frames of a TIMS (in particular, notice the change in topology).

Definition 1.2 can be generalized in various ways to mesh sequences. For instance, the sequence itself can be partitionned into sub-sequences:

Definition 1.3 (Temporal segmentation)**.** *Let* $MS = \{M^i, i = 1 \dots f\}$ *be a mesh sequence. A* temporal segmentation Σ_t *of* MS *is a set of sub-sequences* $\Sigma_t = \{MS_1, \dots, MS_k\}$ *such that* $\forall j \in [1, k], MS_j = \{M^{i_j}, \dots, M^{i_{j+1}-1}\}$ *with* $i_1 = 1 < i_2 < \dots < i_{k+1} = f + 1$.

Possible applications of a temporal segmentation of a TIMS are mesh sequence decomposition into sub-sequences without topological changes or motion-based mesh sequence decomposition, as could be done for instance with the methods of Yamasaki and Aizawa [45] or Tung and Matsuyama [40].

In this paper, we are interested by geometric segmentations, that is to say the spatial segmentation of each mesh of the input sequence. We propose two different definitions.

Definition 1.4 (Coherent segmentation, variable segmentation)**.** *Let* $MS = \{M^i, i = 1 \dots f\}$ *be a mesh sequence. A* coherent segmentation Σ_c *of* MS *is a set of segmentations* $\Sigma^i = \{M^i_1, \dots, M^i_{k_i}\}$ *of each mesh* M^i *of* MS*, such that:*

- *the number* k *of sub-meshes is the same for all segmentations:* $\forall i, j \in [1, f], k_i = k_j$*;*
- *there is a one-to-one correspondence between sub-meshes of any two meshes;*
- *the connectivity of the segmentations, that is to say the neighborhood relationships between sub-meshes, is preserved over the sequence.*

A variable segmentation Σ_v *of* MS *is a set of segmentations* $\Sigma^i = \{M^i_1, \dots, M^i_{k_i}\}$ *of each mesh* M^i *of* MS *which is not a coherent segmentation.*

Note that our definition of a variable segmentation is very general. Intermediate mesh sequence segmentation definitions can be thought of, such as a sequence of successive coherent segmentations which would differ only for a few sub-meshes.

A coherent segmentation of a mesh sequence can be thought as a segmentation of some mesh of the sequence (for instance, the first one) which is mapped to the other meshes. Coherent segmentations are usually desired for shape analysis and understanding, when the overall structure of the shape is preserved during the deformation. However, variable segmentations can be helpful to display different information at each time step. For instance, they can be used to detect when changes in motion occur (see Figure 9 (a,b,c) for an example), which is useful e.g. for animation compression or event detection with a CCTV system. In this paper, we propose a variable segmentation algorithm which recovers the decomposition of the motion over the sequence. For instance, two neighboring parts of the shape with different rigid motions are first put into different sub-meshes. They are later merged when they start sharing the same motion. Our algorithm can also create a coherent segmentation, which distinguishes between parts with different motion for at least a few meshes.

Please see the accompanying video for examples of coherent and variable segmentations.

1.3. Contributions

We propose an algorithm to compute a variable segmentation of a mesh sequence into components that move rigidly over time (section 3). This algorithm can also create a coherent segmentation of the mesh sequence. It applies to any types of mesh sequences though it was originally designed for the most general case of temporally incoherent mesh sequences, with possibly topology changes that occur over time. In contrast to existing approaches, it does not require any prior knowledge as input. Another contribution lies in the design of error metrics to assess the results of existing mesh sequence segmentation techniques (section 5).

2. Related work

Solutions have been proposed to decompose a static mesh into meaningful regions for motion (e.g., invariant under isometric deformations), e.g. [10, 3, 14, 17, 20, 31, 37, 12]. However and since our concern is the recovery of the rigid, or almost rigid, parts of a moving 3D shape, we focus in the following on approaches that consider deforming mesh sequence as input.

2.1. Segmentation of temporally coherent mesh sequences

Several methods have been proposed to compute motion-based coherent segmentation of temporally coherent mesh sequences. Among them, [23, 1, 19, 44, 32, 29] segment a TCMS into rigid components. In particular, de Aguiar [1] proposes a spectral approach which relies on the fact that the distance between two points is invariant under rigid transformation. In this paper, a spectral decomposition is also used (see Section 3.3.3). However, the invariant proposed by de Aguiar et al. cannot be used since mesh sequences without explicit temporal coherence are considered.

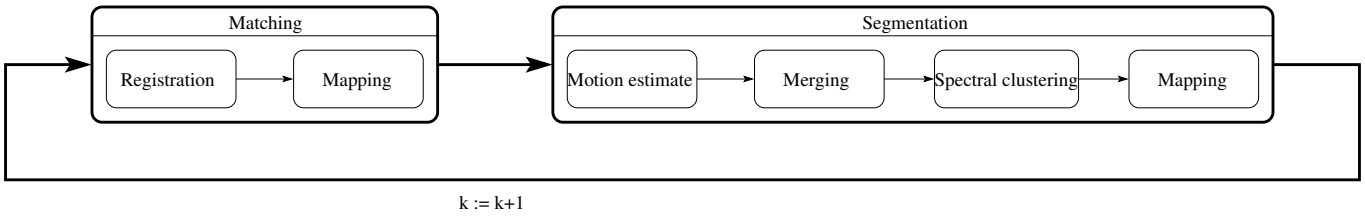


Figure 2: Overall pipeline of our algorithm, at iteration $k, 1 \leq k < f$. As input we have meshes M^k, together with an initial segmentation estimate Σ^k_{est}, and M^{k+1}. As output we get a segmentation Σ^k of M^k and an initial segmentation estimate Σ^{k+1}_{est} of M^{k+1}.

2.2. Segmentation of temporally incoherent mesh sequences

To solve the problem for temporally incoherent mesh sequences, a first strategy is to convert them to TCMS [43, 7]. While providing rich information for segmentation over time sequences, this usually requires a reference model that introduces an additional step in the acquisition pipeline, hence increasing the noise level. Moreover, the reference model usually strongly constrains shape evolution to a limited domain and does not allow for topology changes.

Only a few methods directly work on TIMS. Lee et al. [22] propose a segmentation method for TIMS using an additional skeleton as input. Franco and Boyer [13] propose to track and recover motion over a TIMS at the same time, hence creating a coherent segmentation, but the number of sub-meshes must be known. Varanasi and Boyer [42] segment a few meshes of a TIMS into convex parts, then register these regions to create a coherent segmentation. Their approach does not take into account the shape topology, thus the produced segmentation does not change with the topology. Tung and Matsuyama [41] handle topology changes, however their segmentation uses a learning step from training input sequences. In our work, we do not consider any *a priori* knowledge about the desired segmentation. In a previous work [2] we proposed a framework to segment a TIMS into rigid parts. As for the other works, our approach was only able to create coherent segmentations. In particular, it did not handle topology changes.

Another interesting work is Cuzzolin et al.'s method [11] that computes protrusion segmentation on point cloud sequences. This method is based on the detection of shape extremities, such as hands or legs. Our objective is different, it is to decompose it into rigidly moving parts.

3. Mesh sequence segmentation

In this section we describe our main contribution, that is a segmentation algorithm of a mesh sequence into rigidly moving components. Our algorithm takes as input a TIMS. This mesh sequence can include topology changes (genus and/or number of connected components of the meshes). It can produce either a variable or a coherent segmentation, depending on the user's choice.

3.1. Overview

We propose an iterative scheme that clusters vertices into rigid segments along a TIMS using motion information between successive meshes. For each mesh, rigid segments can be refined by separating parts that present inconsistent motions or otherwise merged when neighboring segments present similar motion. Motion information are estimated by matching meshes at successive instants. The main features of our algorithm are:

- it is fully automatic and does not require prior knowledge on the observed shape;
- it handles arbitrary shape evolutions, including changes in topology;
- it only requires a few meshes in memory at a time. Thus, segmentation can be computed on the fly and long sequences composed of meshes with a high number of vertices can be handled, see e.g. Figure 9.

The algorithm alternates between two stages at iteration $k, 1 \leq k < f$ (see Figure 2, f is the number of meshes in the sequence):

1. matching between 2 consecutive meshes M^k and M^{k+1} and computation of displacement vectors within a time window;
2. segmentation of M^k and mapping to M^{k+1}.

Matching and segmentation algorithms are described in sections 3.2 and 3.3, respectively. This algorithm produces a variable segmentation. In case a coherent segmentation is needed, a post-processing stage is added (Section 3.4).

Four parameters can be tuned to drive the segmentation:

- the *minimum segment size* prevents the creation of too small segments. It is set to 4% of the total number of vertices of the current mesh in all our experiments. We noticed that this number is sufficient to avoid the creation of small segments around articulations, that are usually not rigid;
- the *maximum subdivision* of a segment prevents a segment to be split into too many small segments, when the motion becomes highly non rigid. It is set to 8 segments in all of our experiments;
- the *eigengap* value is used to determine the allowed motion variation within a segment. It thus affects the refinement of the segmentation (see Section 3.3.3 and Figures 10 and 12);
- the *merge threshold* is used to decide whether two segments represent the same motion and need to be merged (see Section 3.3.2).

The notations used throughout the rest of the paper are the following:

- f: the number of meshes in the sequence;
- M^k: the kth mesh of the sequence (can be composed of several connected components);
- M'^k: the kth mesh M^k registered to M^{k+1};
- $nv(M^k)$: the number of vertices in M^k;
- $v_i^{(k)}$: the vertex with index i in M^k;
- $\mathrm{Ng}(v_i^{(k)})$: the 1-ring neighbors of vertex $v_i^{(k)}$.

Note that k is always used as the index for a mesh, and i and j as the indices for vertices in a mesh.

3.2. Mesh matching

The objective of this stage is, given meshes M^k and $M^{k+1}, k \in [1, f-1]$, to provide a mapping from vertices $v_i^{(k)}$ to vertices $v_j^{(k+1)}$, and a possibly different mapping from vertices $v_j^{(k+1)}$ to vertices $v_i^{(k)}$. This mapping is further used to propagate segment labels over the sequence. We proceed iteratively according to the following successive steps (see Figure 3): first, meshes M^k and M^{k+1} are registered (vertices $v_i^{(k)}$ are moved to new locations $v_i'^{(k)}$ close to M^{k+1}), then displacement vectors and vertex correspondences are estimated. The following subsections detail these steps.

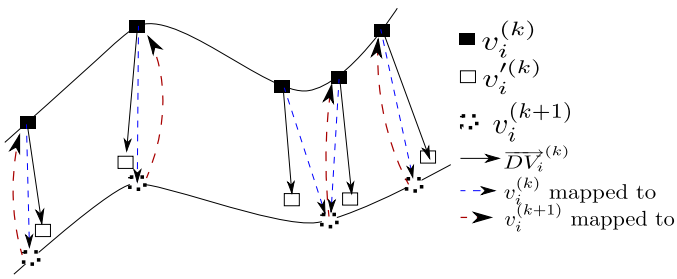


Figure 3: Matching process. Mesh M^k with vertices $v_i^{(k)}$ is first registered to mesh M^{k+1} with vertices $v_i^{(k+1)}$, inducing new vertices $v_i'^{(k)}$. Displacement vectors $DV_i^{(k)}$ are defined thanks to this registration. Finally, mappings from M^k to M^{k+1} and from M^{k+1} to M^k are computed.

3.2.1. Mesh registration

The matching stage of our approach aims at establishing a dense cross parametrization between pairs of successive meshes of the sequence. Among the many available algorithms for this task, we chose to favor generality by casting the problem as the registration of two sets of points and normals. This means that we exclusively use geometric cues to align the two meshes, even when photometric information is available like in the case of meshes reconstructed from multi-camera systems. Thus, our approach also handles the case of software generated mesh sequences.

We implemented the method of Cagniart et al. [7] that iteratively deforms the mesh M^k to fit the mesh M^{k+1}. This approach decouples the dimensionality of the deformation from the complexity of the input geometry by arbitrarily dividing the surface into elements called patches. Each of these patches is associated to a rigid frame that encodes for a local deformation with respect to the reference pose M^k. The optimization procedure is inspired by ICP as it iteratively re-estimates point correspondences between the deformed mesh and the target point set and then minimizes the distance between the two point sets while penalizing non rigid deformations of a patch with respect to its neighbors. Running this algorithm in a coarse-to-fine manner by varying the radii of the patches has proven in our experiments to robustly converge, and to be faster than using a single patch-subdivision level.

3.2.2. Mappings and displacement vectors computation

By using the previous stage, we get the registered mesh M'^k of the mesh M^k on mesh M^{k+1}. The *displacement vector* of each vertex $v_i^{(k)}$ in $M^k, 0 \leqslant i < nv(M^k)$ is then defined as:

$$DV_i^{(k)} = v_i'^{(k)} - v_i^{(k)},$$

with $v_i'^{(k)}$ the corresponding vertex in M'^k. To create a mapping from M^k to M^{k+1}, the closest vertex in M^{k+1} is found for each vertex $v_i'^{(k)}$ in M'^k using Euclidean distance. A mapping from M^{k+1} to M^k is also created by finding for each vertex in M^{k+1} the closest vertex in M'^k. Both mappings are necessary for the subsequent stage of our algorithm (see Sections 3.3.1 and 3.3.4). Note that mesh M^{k+1} is not registered to mesh M^k to compute the second mapping. Apart from saving computation time, this reduces inconsistencies between the two mappings: in most (though not all) cases, if $v_i^{(k)}$ is mapped to $v_i^{(k+1)}$, then $v_i^{(k+1)}$ is mapped to $v_i^{(k)}$. Also, note that these mappings are defined on the vertex sets. Hence, topology changes are not handled here. This is done in the next stage.

Using Euclidean distance instead of geodesic one may lead to occasional mismatchs. However, error hardly accumulates thanks to our handling of topology changes, see Section 3.3.4.

3.3. Mesh segmentation

In this part the goal is to create a segmentation Σ^k of the mesh M^k into rigidly moving components. The displacement vectors over a small time window computed during the previous stage are used, as well as (if $k \neq 1$) the segmentation Σ^{k-1} of M^{k-1} mapped to M^k thanks to the bi-directional mapping between meshes M^{k-1} and M^k. This provides an initial segmentation estimate Σ^k_{est} of M^k. For $k = 1$, the initial estimate is the trivial segmentation of M^1 into a single segment, containing all vertices $v_i^{(1)}$ of M^1.

We proceed in four successive steps. First, the motion of each vertex $v_i^{(k)}$ of M^k is estimated using the displacements vectors (Section 3.3.1). Then, unless a coherent segmentation is required, neighboring segments in Σ^k_{est} that present similar motions are merged (Section 3.3.2). Then a spectral clustering approach is used to refine the segmentation. This yields the segmentation Σ^k of the vertices of M^k (Section 3.3.3). Finally, Σ^k is mapped onto M^{k+1}, to create the initial estimate Σ^{k+1}_{est} of Σ^{k+1} (Section 3.3.4).

Our segmentation algorithm produces, by construction, connected segments since the atomic operations over segments are: merging neighboring segments (see Section 3.3.2) and splitting a segment into connected sub-segments (see Section 3.3.3).

3.3.1. Motion estimate

To estimate the motion of each vertex $v_i^{(k)}$ of M^k, the rigid transformation which maps $v_i^{(k)}$ together with its one-ring neighborhood $\mathrm{Ng}(v_i^{(k)})$ onto M'^k is computed, using Horn's method [16]. This method estimates a 4×4 matrix representing the best rigid transformation between 2 point clouds. A transformation matrix $T_i^{(k)}$ is therefore associated to each vertex $v_i^{(k)}$. With such a method however, computed estimates are noise sensitive, and slow motion is hardly detected. This is due to the fact that only the two meshes M^k and M'^k are used. In order to improve robustness of motion estimates, we propose to work on a time window. Motion is estimated from M^l to M'^k, M^l being the mesh where the segment has been created, either by splitting (see Section 3.3.3) or merging (see Section 3.3.2) of previous segments, or at the beginning of the process ($l = 1$). l may be different for different vertices $v_i^{(k)}$ of M^k. Vertex $v_j^{(l)}$ of M^l from which motion is estimated is defined using the previously computed bi-directional mapping. This method allows to detect slow motion (see Figure 4), and is less sensitive to noise and matching errors. Notice that different parts of the mesh may move with different speeds, this is not a problem as long as they belong to different segments, since the size of the time window is segment-dependent.

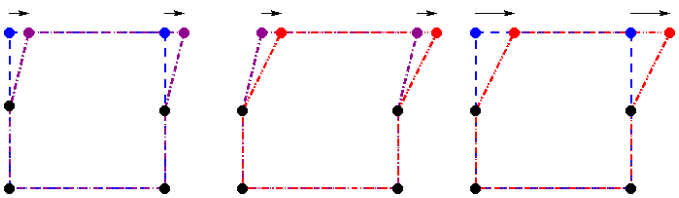


Figure 4: Three successive meshes M^k (blue), M^{k+1} (purple) and M^{k+2} (red). Black dots correspond to vertices with null motion. Motion (black arrows) between M^k and M^{k+1}, then between M^{k+1} and M^{k+2}, is too slow to be detected by our subsequent stage (Section 3.3.3). Using a larger time window $[k, k+2]$ allows to detect this motion.

3.3.2. Merging

In the case of a variable segmentation, neighboring segments with similar motions before are merged at each time step refining the current segmentation. To this aim, the rigid transformation $T^{(k)}(S)$ of any segment S is estimated over all its vertices, using Horn's method [16], as the rigid transformation $T_i^{(k)}$ of any vertex $v_i^{(k)}$ and its 1-ring neighborhood has been estimated. A greedy algorithm is then used:

- starting with the segment S with the minimal residual error, this segment is merged with all neighboring segments S' such that $\|\log(T^{(k)}(S)^{-1}T^{(k)}(S'))\| < T_{merge}$. T_{merge} is a user-defined threshold distance between the transformations of neighboring segments (see Section 3.1). The choice of this logarithm-based distance between transformations is explained in next section;
- the residual error for the new segment $S \cup \bigcup S'$ is computed;
- we iterate, merging the next segment with the minimal residual error with its neighbors.

We stop when no merging is possible anymore. Note that this algorithm allows to handle topology changes such as merging of connected components.

The residual error for a segment S corresponds to the mean distance, for all points $v_i^{(k)}$ of this segment, between the point $v_i^{(k+1)}$ and the location of $v_i^{(k)}$ after the computed rigid transformation $T^{(k)}(S)$ is applied:

$$ResidualError(S) = \frac{\sum_{v_i^{(k)} \in S} \|v_i^{(k+1)} - T^{(k)}(S) * v_i^{(k)}\|}{card(S)} \quad (1)$$

In our implementation, the choice of the threshold value T_{merge} is left to the user. According to our experiments, it needs a few trials to find a suitable value. Choosing a high value merges most of the segments, while choosing a low value generates many clusters. The following values have been chosen for the displayed results in Sections 4 and 5: 0.03 for the *Balloon* and the *Horse* sequences (Figures 9 and 11), 0.05 for the *Dancer* sequence (Figure 9) and 0.2 for the *Cat* sequence (Figure 13).

During the next step the current segmentation is refined. In order to prevent successive and useless merge and split of the same segments, we actually apply motion-based spectral clustering on detected pairs of segments to be merged *before* merging them. If the clustering results in some pairs splitting, then these pairs are not merged.

3.3.3. Motion-based spectral clustering

Spectral clustering is a popular and effective technique to robustly partition a graph according to some criterion [28]. It has been successfully applied to static meshes (see e.g. [26, 27, 34, 36]), using the mesh vertices as the graph nodes and the mesh edges as the graph edges. The graph should be weighted with respect to the partition criterion. More precisely, edge weights represent *similarity* between their endpoints. In our case, these weights are related to the motion of neighboring vertices. This is in contrast to [1] where Euclidean distances between vertices are considered. In fact Euclidean distances can be preserved by non rigud transformation. Related to our approach is Brox and Malik's motion-based segmentation algorithm for videos [6].

Edge weights. To compute the weights $W^{(k)}$ of the graph edges, the following expression is used [30]:

$$w_{i,j}^{(k)} = \begin{cases} \frac{1}{\|\log(T_i^{(k)^{-1}} T_j^{(k)})\|^2} & \text{if } i \neq j, \\ 0 & \text{if } i = j. \end{cases} \quad (2)$$

As demonstrated in [30], this distance is mathematically founded since it corresponds to distances on the special Euclidean group of rigid transformations $SE(3)$.

Spectral clustering algorithm. Using the weighted adjacency matrix $W^{(k)}$, the normalized Laplacian matrix $L_{rw}^{(k)}$ is built as follows. Then the well-known Shi and Malik's normalized spectral clustering algorithm [35] is used to segment the graph.

$$D_{ii}^{(k)} = \sum_{j \in \mathrm{Ng}(v_i^{(k)})} w_{ij^{(k)}}. \tag{3}$$

$$L^{(k)} = D^{(k)} - W^{(k)}. \tag{4}$$

$$L_{rw}^{(k)} = D^{(k)-1} L^{(k)} = I^{(k)} - D^{(k)-1} W^{(k)}. \tag{5}$$

Shi and Malik compute the first K eigenvectors $u_1, \ldots, u_K$ of $L_{rw}^{(k)}$ and store them as columns of a matrix U. The rows $y_i, i = 1 \ldots n$, of U are then clustered using the classical K-means algorithm. Clusters for the input graph correspond to clusters of the rows y_i: points i such that y_i belong to the same cluster are said to belong to the same segment of the graph.

This method assumes the number K of clusters to be known. K is computed using the classical eigengap method: let $\lambda_1, \lambda_2, \ldots, \lambda_K, \ldots$ be the eigenvalues of $L_{rw}^{(k)}$ ordered by increasing value, the smaller K such that $\lambda_K - \lambda_{K-1} > eigengap$ is chosen. In our implementation, the eigengap value's choice is left to the user. In our experiments, a few trials (less than 5) were necessary to set this parameter. Two parameters are also used to prevent the creation of small segments in non-rigid areas (see Section 3.1): a minimum segment size and a maximum subdivision of a segment. According to our experiments, results are not very sensitive to the choice of these three parameters; the same values have been used for most of our experiments (see Section 4).

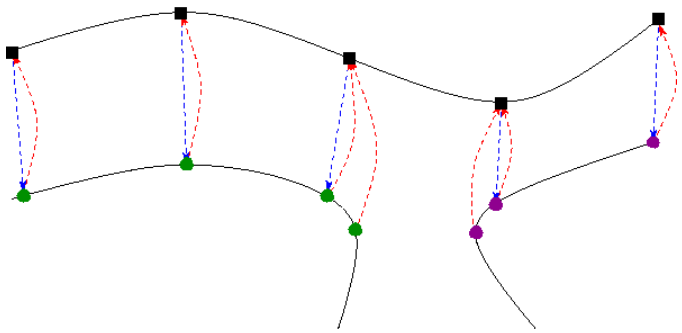


Figure 5: Splitting process. Blue and red arrows indicate the bi-directional mapping. The current segment (black squares) is split in two (green and magenta dots, respectively), since the three leftmost vertices and the two rightmost vertices are mapped to two different connected components.

3.3.4. Mapping to M^{k+1}

The segmentation is computed at each time step on the current mesh M^k. Labels are then mapped onto the mesh M^{k+1} using the bi-directional mapping defined in Section 3.2.2. Segments are first transferred using the mapping from M^k to M^{k+1}. Then for all unmatched vertices in M^{k+1}, the mapping from M^{k+1} to M^k is used. Segments which are mapped on different connected components are split, see Figure 5. This allows us to naturally handle topology changes. This segmentation of M^{k+1} serves as an initial estimate for the computation of Σ^{k+1}.

Note that segment splitting and merging allows to robustly handle mismatching, see Figure 6. In case a vertex $v_i^{(k)}$ is wrongly matched to a vertex $v_j^{(k+1)}$, the corresponding segment is split in two. The new segment containing $v_j^{(k+1)}$ is then likely to be merged with a neighboring segment with similar motion.

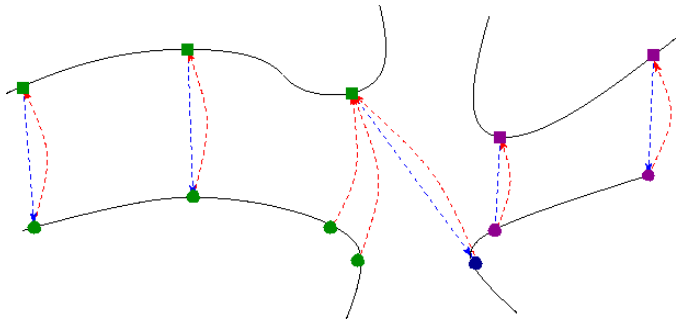


Figure 6: Segment splitting and merging allows to robustly handle mismatching. In case a vertex (rightmost green square) of M^k is mismatched to a vertex (dark blue dot) of M^{k+1}, a new segment is created. This segment is then likely to be merged with the neighboring segment (magenta dots), since they present similar motions.

3.4. Coherent segmentation

The algorithm can be modified to generate a coherent segmentation instead of a variable segmentation. This coherent segmentation clusters neighboring vertices that share similar rigid motion over the whole sequence. In other words, as long

as their motion differs over at least one small time window, two neighboring vertices do not belong to the same segment.

Creating a coherent segmentation is then straightforward. We only need:

- not to merge segments (step described in Section 3.3.2 is not applied);
- to map the segmentation Σ^f of the last mesh M^f back to the whole sequence.

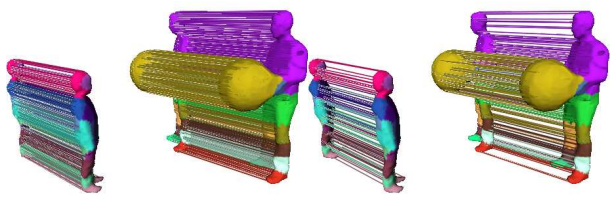
To this purpose, the bi-directional mapping described in Section 3.2.2 is simply applied in reverse order, from M^f to M^1. For each pair of successive meshes (M^k, M^{k+1}) we first use the mapping from M^{k+1} to M^k, then for all vertices of M^k which are not assigned to a segment, the mapping from M^k to M^{k+1} is used.

4. Results

In this section we show and discuss visual results of our algorithm. A quantitative evaluation of these results is discussed in the next section. We first examine matching results, then segmentation results on difficult cases (temporally incoherent mesh sequences with topological changes, acquired from real data). We also show that our results on temporally coherent mesh sequences are visually similar to state-of-the-art approaches.

4.1. Matching results

The vertex matching computation is an important step since our segmentation algorithm relies on it (see Figure 2). Figure 7 shows the result of vertex matching between two successive meshes of a TIMS. Computation time is about 30 seconds for two meshes with approximately 7000 vertices each. This outperforms the matching method proposed in [2] which takes about 13 minutes to complete computation with the same data, for a similar result. Note that outliers in the matching are not explicitly taken into account in the segmentation, however their influence is limited by the threshold on the minimum segment size (see Section 3.1) that tends to force them to merge with neighboring segments.



(a) (b)

Figure 7: Result of vertex matching on real data captured from video cameras. (a) Full display. (b) Partial display.

Figure 8 shows a matching result between two consecutive frames of a sequence where the vertex density differs drastically. Even if vertex-to-vertex matching is less accurate than vertex-to-face matching (that is to say, matching every vertex of M^k to the closest point of M^{k+1}, which can lie on an edge or inside a face), in our experiments it has proved to be sufficient for our purpose. Meanwhile, its computation is much faster.

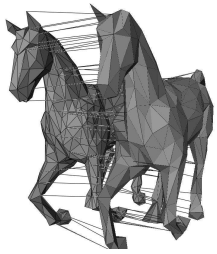


Figure 8: Result of vertex matching between two consecutive frames of a sequence with varying vertex density.

4.2. Segmentations of TIMS with topology changes

Figure 9 shows variable segmentations computed on two *Balloon* and *Dancer* sequences. Figure 10 shows coherent segmentations computed on the *Balloon* sequence. By construction, coherent segmentations contain more segments than variable segmentations since no merging operation occurs. Parameters for both variable and coherent segmentations of the *Balloon* sequence have the same values, except for the eigengap threshold that is slightly lower in the variable segmentation case (0.40 vs. 0.48 for result shown on Figure 10 (a)). According to our experiments, suitable parameter values for a given sequence are found in a few trials. The computation time of one mesh segmentation of the *Dancer* sequence is approximately 3 minutes with a (not optimized) Matlab implementation. Additional results appear in the accompanying video. Our algorithm does not require the whole sequence in memory at a given time step k, but only previous meshes which share at least one segment with the current segmentation, in addition to the next mesh (namely, meshes from M^l to M^{k+1}, see Section 3.3.1). Thus, it can handle long sequences with a high number of vertices, such as the *Balloon* sequence which contains 300 meshes with approximately 15,000 vertices each.

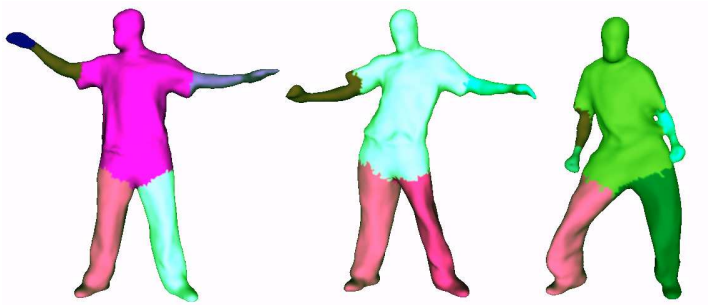
Timings are given in the following table. The algorithm was implemented using Matlab on a laptop with a one-core 2.13 GHz processor.

Segmentation	Total computation time
Fig. 9 (a–c)	43 min 14
Fig. 9 (d–g)	76 min 48
Fig. 12 (a)	29 min 07
Fig. 12 (b)	25 min 57
Fig. 13 (a)	3 min 46

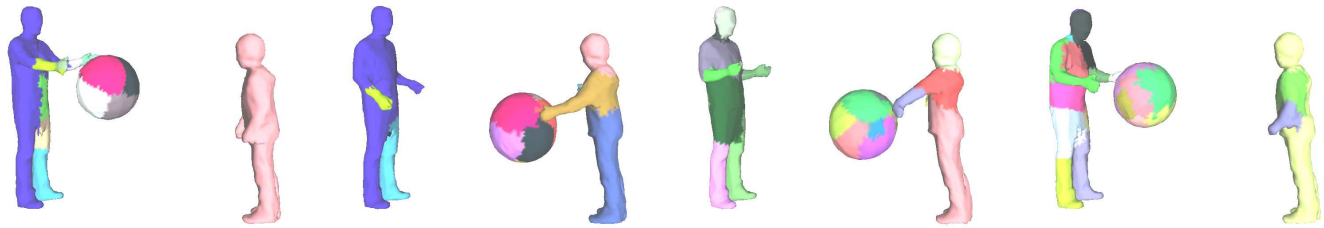
4.3. Segmentation of TCMS

Although our approach is designed for general cases, it can also handle TCMS and obtains visually similar results to previous TCMS-dedicated methods, as shown in Figure 11.

Figure 12 illustrates the influence of the eigengap threshold: the higher the eigengap value, the coarser the segmentation.

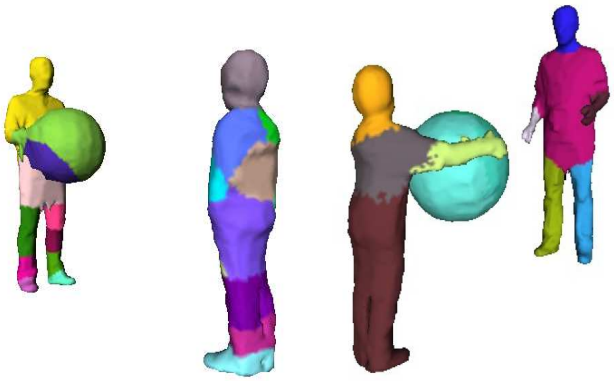


(a) (b) (c)



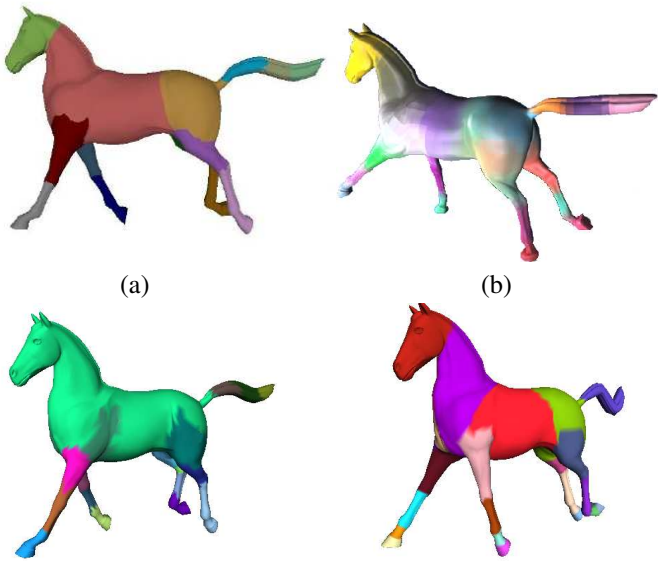
(d) (e) (f) (g)

Figure 9: (a,b,c) Variable segmentation generated by our algorithm on the *Dancer* sequence [38]. First meshes are decomposed into 6 segments, then the right arm and right hand segments merge since they move the same way. Finally, this segment is split again. Note that topology changes can be handled (in the last meshes, the left arm is connected to the body). (d,e,f,g): Variable segmentation of a sequence with 15,000 vertices per mesh and topology changes. The balloon is over-segmented because its motion is highly non rigid.



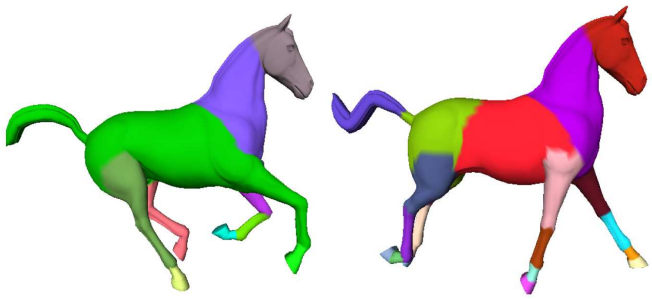
(a) (b)

Figure 10: Coherent segmentation results on the *Balloon* sequence, obtained with two different eigengap values: (a) 0.48, (b) 0.8. Segments cluster neighboring vertices that share the same motion over the whole sequence.



(a) (b) (c) (d)

Figure 11: Segmentation results on a TCMS. (a) [23]. (b) [1]. (c) [2]. (d) Our method.



(a) (b)

Figure 12: Segmentation of the *Horse* sequence [39] with two different eigengap values. (a) *eigengap* = 0.7. (b) *eigengap* = 0.5.

5. Evaluation

A quantitative and objective comparison of segmentation methods is an ill-posed problem since there is no common definition of what an optimal segmentation should be in the general case. Segmentation evaluation has been recently addressed in the static case using ground truth (i.e. segmentations defined by humans) [5, 9]. In the mesh sequence case, none of the previously cited articles in Section 2 proposes an evaluation of the obtained segmentations. We thus propose the following framework to evaluate a mesh sequence segmentation method.

5.1. Optimal segmentation

The *optimal* segmentation of a mesh sequence, be it a TCMS or a TIMS, into rigid components can be guessed when the motion and/or the kinematic structure is known. This is, for instance, the case with skeleton-based mesh animations, as created in the computer graphics industry. In this case, each mesh vertex of the sequence is attached to at least one (usually, no more than 4) *joints* of the animation skeleton, with given weights called *skinning weights*. These joints are organized in a hierarchy, which is represented by the "bones" of the skeleton that are, therefore, directed. For our evaluation, we attach each vertex to only one joint among the related joints, the furthest in the hierarchy from the root joint. If this joint is not unique, the one with the greatest skinning weight is kept. Each joint has its own motion, but several joints can move together in a rigid manner. For a given mesh, cluster joints of the animation skeleton can therefore be clustered into *joint sets*, each joint set representing a different motion. We now define as an *optimal segment* the set of vertices related to joints in the same joint set. Since the motion of each joint is known, we exactly know, for each mesh of the animation, what are the optimal segments.

This definition can be applied in the general case of TIMS, provided that each vertex of each mesh can be attached to a joint. However, we only tested it in the more convenient case of a TCMS.

5.2. Error metrics

We propose the following three metrics in order to evaluate a given segmentation with respect to the previously defined optimal segmentation:

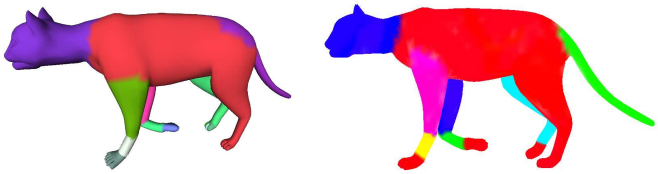
- *Assignment Error* (AE): for a given mesh, the ratio of vertices which are not assigned to the correct segment. This includes the case of segments which are not created, or which are wrongly created;
- *Global Assignment Error* (GAE): the mean AE among all meshes of the sequence;
- *Vertex Assignment Confidence* (VAC): for a given vertex of a TCMS, the ratio of meshes in which the vertex is assigned to the correct segment.

AE and GAE give a quantitative evaluation of a mesh segmentation and the mesh sequence segmentation, respectively, with respect to the optimal segmentation. VAC can help to locate wrongly segmented areas.

Note that more sophisticated evaluation metrics exist to compare two static mesh segmentations [9]. We define AE as a simple ratio for sake of simplicity, but other metrics can also be used to define global assignment errors.

5.3. Evaluation results

We tested our algorithm on a walking cat skeleton-based animation (see Figure 13 and the accompanying video). We get a variable segmentation with a AE up to 17%, in the worst case. Wrongly assigned vertices correspond to the cat skin around joints and to a wrong subdivision in cat paw, i.e. in the less rigid areas.



(a) (b)

Figure 13: Result on a skeleton-based synthetic animation. (a) Computed variable segmentation. (b) Optimal variable segmentation, for the same mesh of the sequence.

In the case of coherent segmentations, and if matching issues are not taken into account, then the AE is the same for all meshes. Therefore, the GAE is equal to the AE of any mesh. For the cat sequence, the GAE is also 17%. The VAC can be 0% or 100%, and is only relevant as a relative criterion to compare vertices and find ill-segmented areas. On the cat sequence vertices in rigid areas (paws, tail, body) are often always assigned to the correct segment; their confidence is equal to 1. In contrast, some vertices around joints can be assigned to the same neighboring segment in all meshes; their confidence drops to 0, see Figure 14. We also computed these metrics for the method described in [2], using the same cat sequence. The GAE reaches 42%, while the VAC can also be 0% or 100%.

6. Conclusion

In this paper we addressed the problem of 3D mesh sequence segmentation into rigidly moving components. We have proposed a classification of mesh sequence segmentations, together

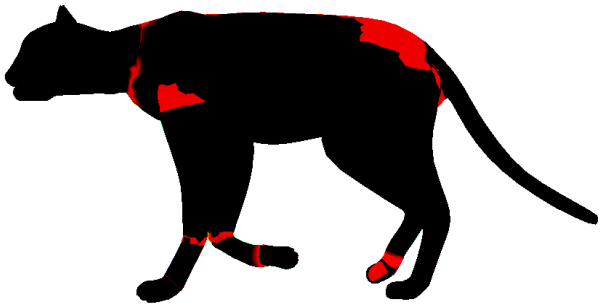


Figure 14: Vertex Assignment Confidence results. Vertices for which VAC is 0 are colored in red, while vertices with confidence equal to 1 are in black.

with a segmentation method that takes as input a mesh sequence, even when no explicit temporal coherence is available, and possibly with topology changes. This method produces either a coherent or a variable segmentation into rigid components depending on the user's choice. It uses a few parameters which can be set in a few trials, according to our experiments. We have also proposed a framework for quantitative evaluations of rigid segmentation methods.

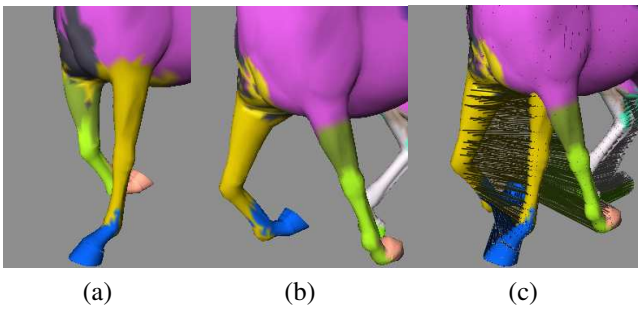
6.1. Current limitations

We are currently aware of three limitations in the proposed algorithm:

- our method clearly depends on the quality of the matching process. Important errors in matching computation may lead to wrong results;
- segmentation can slightly drift: this is due to the fact that only 2 meshes are considered when matching;
- segments which are wrongly subdivided are transferred to the following meshes, meaning that errors on an early mesh in the sequence can affect the whole segmentation. Such errors are generally due to errors in the matching process. This issue is less critical on variable segmentations than on coherent segmentations, since segments are merged later.

Figure 15 shows an example of these limitations. In this example, the entire left front leg of the horse at frame k was intentionally mismatched to the right front leg at frame $k+1$, and vice-versa. Resulting erroneous segmentation at frame $k+1$ is then propagated to the following frames, since no merging with the neighboring segment occurs. Fortunately, this problem seldom happens. As shown in our quantitative evaluations, using the matching process described in Section 3.2, vertices that are wrongly assigned to a segment are located near articulations. Vertices in rigid regions are generally correctly clustered.

Despite these limitations, our method has shown as good results as current state-of-the-art methods on temporally coherent mesh sequences (see Figure 11), although it has been designed for the more difficult case of mesh sequences without temporal coherence.



(a) (b) (c)

Figure 15: Matching error and resulting coherent segmentation. (a,b) Two consecutive frames of the Horse sequence. (c) Matching between these two frames.

6.2. Future work

Our method can be improved in various ways. As explained above, it would be interesting to improve the vertex assignments around articulations. Adding prior knowledge about the geometry of desired segments (e.g. cylindrical shape, or symmetry information) would be helpful to enhance the robustness of the method. It would also be useful to reduce the number of parameters. Our algorithm handles topology changes, but our solution is not semantically satisfactory in case a new connected component (e.g., the shade of the balloon in the *Balloon* sequence) appears, since it is first attached to an existing segment before being split from it.

We hope our evaluation metrics would be helpful for further work in the domain. However, a more in-depth study of the three proposed criteria need to be performed to assess their usefulness. Finally, a user validation can also help to quantify segmentations produced by our algorithm.

Acknowledgments

The *Balloon* sequence is courtesy of Inria Grenoble [18]. The *Dancer* sequence is courtesy of University of Surrey [38]. The *Horse* sequence is courtesy of M.I.T. [39]. The *Cat* sequence is courtesy of Inria Grenoble [4]. This work has been partially funded by the french National Reseach Agency (ANR) through the MADRAS (ANR-07-MDCO-015) and MORPHO (ANR-10-BLAN-0206) projects.

References

[1] de Aguiar, E., Theobalt, C., Thrun, S., Seidel, H., 2008. Automatic conversion of mesh animations into skeleton-based animations. Computer Graphics Forum (Eurographics proceedings) 27.

[2] Arcila, R., Buddha, K., Hétroy, F., Denis, F., Dupont, F., 2010. A framework for motion-based mesh sequence segmentation, in: Proceedings of the International Conference on Computer Graphics, Visualization and Computer Vision (WSCG).

[3] Attene, M., Falcidieno, B., Spagnuolo, M., 2006. Hierarchical mesh segmentation based on fitting primitives. The Visual Computer 22.

[4] Aujay, G., Hétroy, F., Lazarus, F., Depraz, C., 2007. Harmonic skeleton for realistic character animation, in: Proceedings of the Symposium on Computer Animation (SCA).

[5] Benhabiles, H., Vandeborre, J., Lavoué, G., Daoudi, M., 2009. A framework for the objective evaluation of segmentation algorithms using a ground-truth of human segmented 3D-models, in: Proceedings of the IEEE International Conference on Shape Modeling and Applications (SMI).

[6] Brox, T., Malik, J., 2010. Object segmentation by long term analysis of point trajectories, in: Proceedings of the European Conference on Computer Vision (ECCV).

[7] Cagniart, C., Boyer, E., Ilic, S., 2010. Iterative deformable surface tracking in multi-view setups, in: Proceedings of the International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT).

[8] Cashman, T., Hormann, K., 2012. A continuous, editable representation for deforming mesh sequences with separate signals for time, pose and shape. Computer Graphics Forum (Eurographics proceedings) 31.

[9] Chen, X., Golovinskiy, A., Funkhouser, T., 2009. A benchmark for 3d mesh segmentation. ACM Transactions on Graphics (SIGGRAPH proceedings) 28.

[10] Cutzu, F., 2000. Computing 3d object parts from similarities among object views, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[11] Cuzzolin, F., Mateus, D., Knossow, D., Boyer, E., Horaud, R., 2008. Coherent laplacian 3-d protrusion segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[12] Fang, Y., Sun, M., Kim, M., Ramani, K., 2011. Heat mapping: a robust approach toward perceptually consistent mesh segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[13] Franco, J., Boyer, E., 2011. Learning temporally consistent rigidities, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[14] de Goes, F., Goldenstein, S., Velho, L., 2008. A hierarchical segmentation of articulated bodies. Computer Graphics Forum (Symposium on Geometry Processing proceedings) 27.

[15] Goldenthal, R., Harmon, D., Fattal, R., Bercovier, M., Grinspun, E., 2007. Efficient simulation of inextensible cloth. ACM Transactions on Graphics (SIGGRAPH proceedings) 26.

[16] Horn, B., 1987. Closed-form solution of absolute orientation using unit quaternions. J. Opt. Soc. Am. A 4.

[17] Huang, Q., Wicke, M., Adams, B., Guibas, L., 2009. Shape decomposition using modal analysis. Computer Graphics Forum (Eurographics proceedings) 28.

[18] Inria, . Balloon sequence. `http://4drepository.inrialpes.fr/`.

[19] Kalafatlar, E., Yemez, Y., 2010. 3d articulated shape segmentation using motion information, in: Proceedings of the International Conference on Pattern Recognition (ICPR).

[20] Kalogerakis, E., Hertzmann, A., Singh, K., 2010. Learning 3d mesh segmentation and labeling. ACM Transactions on Graphics (SIGGRAPH proceedings) 29.

[21] Kircher, S., Garland, M., 2006. Editing arbitrarily deforming surface animations. ACM Transactions on Graphics (SIGGRAPH proceedings) 25.

[22] Lee, N., T.Yamasaki, Aizawa, K., 2008. Hierarchical mesh decomposition and motion tracking for time-varying-meshes, in: Proceedings of the IEEE International Conference on Multimedia and Expo (ICME).

[23] Lee, T.Y., Wang, Y.S., Chen, T.G., 2006. Segmenting a deforming mesh into near-rigid components. The Visual Computer 22.

[24] Lengyel, J., 1999. Compression of time-dependent geometry, in: Proceedings of the Symposium on Interactive 3D graphics (I3D).

[25] Lewis, J., Cordner, M., Fong, N., 2000. Pose-space deformation: a unified approach to shape interpolation and skeleton-driven deformation, in: Proceedings of SIGGRAPH.

[26] Liu, R., Zhang, H., 2004. Segmentation of 3d meshes through spectral clustering, in: Proceedings of Pacific Graphics.

[27] Liu, R., Zhang, H., 2007. Mesh segmentation via spectral embedding and contour analysis. Computer Graphics Forum (Eurographics proceedings) 26.

[28] von Luxburg, U., 2007. A tutorial on spectral clustering. Statistics and Computing 17.

[29] Marras, S., Bronstein, M.M., Hormann, K., Scateni, R., Scopigno, R., 2012. Motion-based mesh segmentation using augmented silhouettes. Graphical Models .

[30] Murray, R., Sastry, S., Zexiang, L., 1994. A Mathematical Introduction to Robotic Manipulation. CRC Press, Inc.

[31] Reuter, M., 2010. Hierarchical shape segmentation and registration via topological features of laplace-beltrami eigenfunctions. International Journal of Computer Vision 89.

[32] Rosman, G., Bronstein, M.M., Bronstein, A.M., Wolf, A., Kimmel, R., 2011. Group-valued regularization framework for motion segmentation of dynamic non-rigid shapes, in: Scale Space and Variational Methods in Computer Vision.

[33] Shamir, A., 2008. A survey on mesh segmentation techniques. Computer Graphics Forum 27.

[34] Sharma, A., von Lavante, E., Horaud, R., 2010. Learning shape segmentation using constrained spectral clustering and probabilistic label transfer, in: Proceedings of the European Conference on Computer Vision (ECCV).

[35] Shi, J., Malik, J., 2000. Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 22.

[36] Sidi, O., van Kaick, O., Kleiman, Y., Zhang, H., Cohen-Or, D., 2011. Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. ACM Transactions on Graphics (SIGGRAPH Asia proceedings) 30.

[37] Skraba, P., Ovsjanikov, M., Chazal, F., Guibas, L., 2010. Persistence-based segmentation of deformable shapes, in: CVPR Workshop on Non-Rigid Shape Analysis and Deformable Image Alignment.

[38] Starck, J., Hilton, A., 2007. Surface capture for performance-based animation. IEEE Computer Graphics and Applications .

[39] Sumner, R., Popović, J., 2004. Deformation transfer for triangle meshes. ACM Transactions on Graphics (SIGGRAPH proceedings) 23.

[40] Tung, T., Matsuyama, T., 2009. Topology dictionary with markov model for 3d video content-based skimming and description, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[41] Tung, T., Matsuyama, T., 2010. 3d video performance segmentation, in: Proceedings of the IEEE International Conference on Image Processing (ICIP).

[42] Varanasi, K., Boyer, E., 2010. Temporally coherent segmentation of 3d reconstructions, in: Proceedings of the International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT).

[43] Vlasic, D., Baran, I., Matusik, W., Popović, J., 2008. Articulated mesh animation from multi-view silhouettes. ACM Transactions on Graphics (SIGGRAPH proceedings) 27.

[44] Wuhrer, S., Brunton, A., 2010. Segmenting animated objects into near-rigid components. The Visual Computer .

[45] Yamasaki, T., Aizawa, K., 2007. Motion segmentation and retrieval for 3d video based on modified shape distribution. EURASIP Journal on Applied Signal Processing .

$\therefore$

A.8 AUTOMATIC LOCALIZATION AND QUANTIFICATION OF INTRACRANIAL ANEURYSMS

Sahar Hassan, Franck Hétroy, François Faure, Olivier Palombi

Lecture Notes in Computer Science 6854, Springer, 2011. Presented at the International Conference on Computer Analysis of Images and Patterns (CAIP), 2011.

Automatic localization and quantification of intracranial aneurysms

Sahar Hassan[1,2], Franck Hétroy[1,2], François Faure[1,2], and Olivier Palombi[1,2,3]

[1] Université de Grenoble & CNRS, Laboratoire Jean Kuntzmann, Grenoble, France
[2] INRIA Grenoble - Rhône-Alpes, Grenoble, France
[3] Grenoble University Hospital, France

Abstract. We discuss in this paper the problem of localizing and quantifying intracranial aneurysms. Assuming that the segmentation of medical images is done, and that a 3D representation of the vascular tree is available, we present a new automatic algorithm to extract vessels centerlines. Aneurysms are then automatically detected by studying variations of vessels diameters. Once an aneurysm is detected, we give measures that are important to decide its treatment. The name of the aneurysm-carrying vessel is computed using an inexact graph matching technique. The proposed approach is evaluated on segmented real images issued from Magnetic Resonance Angiography (MRA) and CT scan.

1 Introduction

Aneurysms are dilatations in the wall of a blood vessel, leading to little pockets. Aneurysms can be saccular, fusiform or dissecting, see Fig. 1. In this article we are interested in saccular aneurysms which are connected to the vessel by a narrowed zone called the *neck*. If not treated, an aneurysm may burst causing a stroke and in most cases the death of the patient.

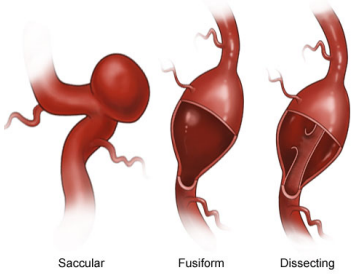


Fig. 1: Aneurysm types[4].

The decision of treating an aneurysm or just observing it is made according to its risk of rupture. When the treatment is needed, two possible ways exist: either embolization using a platinum coil, or clipping. A lot of studies and statistical surveys have been done in order to know what factors affect the rupture of an aneurysm [1–3], and thus help in making the best decision about the treatment. According to these studies the most important factors are: size, shape, neck, and location of the aneurysm.

A lot of work has been done in the domain of intracranial aneurysms, most of which is about segmenting the vascular tree and giving the user a 3D view of the aneurysm. This segmentation can be statistical [4], or it can be based on the tubular shape of vessels [5–8]. In [9], a morphological characterization of the aneurysm is given in order to predict the rupture rate, and thus decide if there should be a treatment.

In this paper, we suppose that the segmentation is done, and we go further. The set of voxels representing the cerebrovascular tree goes through several processes including: extraction of vessels' centerlines, detection of aneurysms, quantification and

[4] `http://nyp.org/health/neuro-cerbaneu.html`

localization of the detected aneurysms. An approach based on Dijkstra's algorithm [10] is proposed to get thin, connected and centered centerlines. These centerlines are then used to study the evolution of the diameters and automatically detect aneurysms. Blood vessels have a cylindrical shape and thus their diameters are almost steady, whereas those of aneurysms change considerably. Relevant measures of found aneurysms and their location are then given using a partial graph matching technique. To our knowledge, this is the first time these steps are performed together to detect, quantify and localize intracranial aneurysms.

2 Methods

2.1 Centerlines extraction

Extraction of blood vessels centerlines can be done either while segmenting these blood vessels [7, 8, 11, 12], or after segmenting blood vessels from medical images as in our case. Various methods for centerline extraction are proposed for different uses. Some categories of these methods are presented in [13] along with the usually desired properties of centerlines. Since we want to use the centerlines to study the evolution of blood vessels diameters, these centerlines should be: **1. connected**: the centerlines we are looking for should be 26-connected, **2. thin**: a centerline is thin if each voxel of the centerline has only two of its neighbors in the centerline, except for the extremities which have one neighbor in the centerline, **3. centered**: the centerlines should be centered within the vascular tree, and **4. connections between branches**: should be as perpendicular as possible, see Fig. 2. Finally, the algorithm should be efficient since it is a step out of four in the processing chain, besides cerebral vascular trees are complex.

Main centerline using Dijkstra's algorithm with Euclidian distance (a), the wanted centered centerline (b).

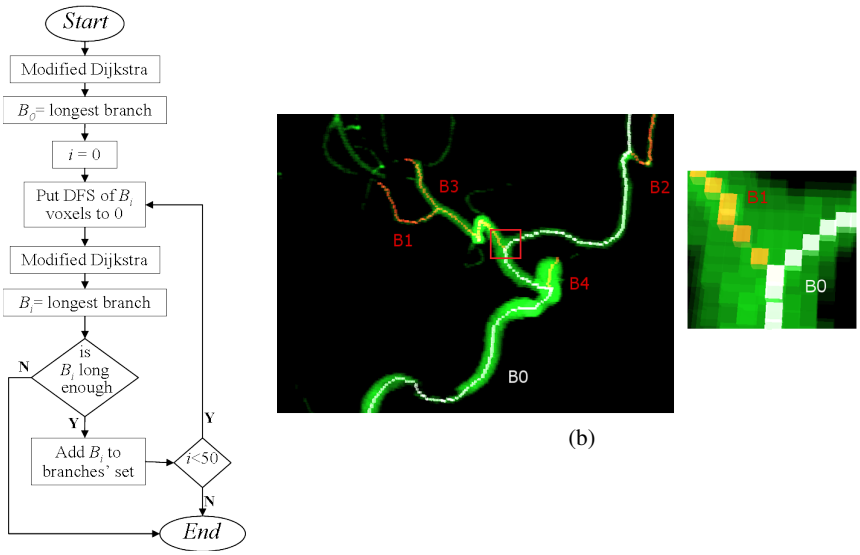
Connection between branches: (a) the connection is not perpendicular, (b) the wanted perpendicular connection.

Fig. 2: Important features of the desired centerlines.

In the following, we call *skeleton* the set of centerlines. The longest centerline is called the *main centerline*, while the others are called *branches*. The main centerline and each branch have a diameter which is the mean diameter of the corresponding blood vessels.

To fulfil our requirements, we propose a centerline extraction method that falls in the **distance-based methods** category. The main idea of these methods is to construct a shortest distance tree (SDT) [10]. After the construction of such a tree, we get a graph. Nodes of the graph are the voxels of the object. The voxels will be connected (a connection between two voxels corresponds to an edge in the graph) in a way to minimize the distance to a source voxel S, hereafter called Distance From Source (DFS). The main centerline is then extracted by tracing from E, the voxel with maximum DFS, back to the source S, and thus is connected and thin by construction. The use of a heap for

the priority queue makes the complexity of these methods of $O(NlogN)$ where N is the number of voxels and thus computationally efficient. However, using the Euclidian metric as the distance to minimize leads to a centerline that cuts the corners, see Fig. 2. Several variations of this algorithm were proposed to solve the "cutting corners" problem and get centered centerlines [14–16]. The common idea is to use another distance function while constructing the tree to privilege voxels near the center of the object.



(a) (b)

Fig. 3: Our method. (a) Flowchart. 50 is a sufficient number to extract all significant branches in all our experiments. (b) Result of the method on a real dataset with a zoom on the branching.

Our method is illustrated on Fig. 3. **First**, the source voxel is chosen automatically, to be sure that it is an extremity of a vessel. We construct a SDT taking an arbitrary voxel as a source, the end voxel (furthest one of the arbitrary source) is necessarily an extremity and is used as the source voxel for our algorithm.
We use the following distance function instead of using the Euclidian distance:

$$d(v1, v2) = \frac{dist(v1, v2)}{1 + (DFB[v1] + DFB[v2])}$$

with: $dist(v1, v2)$ the Euclidian distance between $v1, v2$, $DFB[v_i]$ v_i's distance from the boundary, i.e. Euclidian distance between v_i and the closest surface voxel (a surface voxel is a voxel with at least one of its 26-neighbors missing in the voxel set).

The division by the distance from boundary (depth) privileges the voxels that are far from the boundary, and thus enforce the centeredness. At the same time, we keep using the Euclidian distance to find the *end voxel* at each iteration, and thus extract branches in a descending length order. Each branch B_i is connected to a *father* branch that is not necessarily B_{i-1}. Another important advantage of our algorithm is junctions between branches. Putting DFS of voxels of extracted branches to zero, makes each branch join its *father* in a perpendicular way (see Fig. 3). We emphasize on this point

because variations of branches' diameters play a major role in aneurysm detection and quantification, see Section 2.2.

The complexity of our algorithm is $O(KNlogN)$ where K is the number of extracted branches, and N is the number of voxels. One drawback of this method is that the set of branches is not homotopic to the object. This method gives by construction a tree-like structure with no loops.

2.2 Automatic detection of the aneurysm

One key characteristic that differentiates a saccular aneurysm from a normal vessel, is that the normal vessel -which has a cylindrical shape- has an almost steady diameter, whereas the aneurysm -which has an irregular shape- has a diameter that changes considerably.

In order to model the appearance of a vessel, we define a set of points (x, y). Each point corresponds to a voxel v of the branch, where:

• x, represents the distance between the voxel v and the origin of the branch j.

• y, represents the approximate diameter of the branch at v.

To calculate y, we compute the real plane P passing through the center of voxel v and perpendicular to the branch, see Fig. 4. P cuts the vessel or aneurysm surface on voxels $v_i, 1 \leq i \leq k$. Let y_i be the distance between v_i and v, y is defined as the average value of $y_i : y = \frac{\sum_{i=1}^{k} y_i}{k}$. Thanks to the centeredness of centerlines, and perpendicular connections between branches, y represents a reliable measure of the diameter changes.

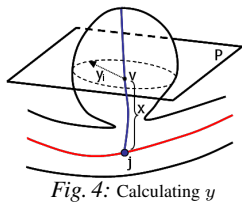


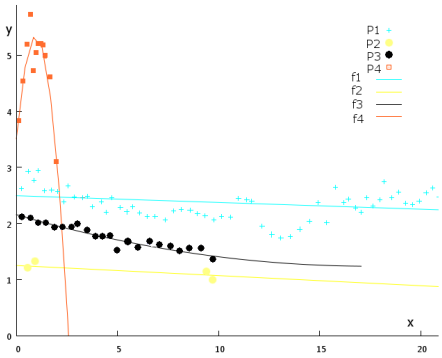
Fig. 4: Calculating y

Then, we use the least-squares method to find the quadratic function ($y = a + bx + cx^2$) that best matches our set of points. A more complex function could be used, but this one is sufficient to discriminate between a diameter variation which is linear and a one that is not. Since normal vessels have a cylindrical shape, their diameter is almost steady and thus the value of c is very small. So, by thresholding on c, we decide if the corresponding branch is in an aneurysm. The threshold we use has been found after a ROC analysis, and is 0.2. The threshold is not null because a branch can traverse several blood vessels (see branches in Fig. 3), which makes the associated diameter change. However, this change remains insignificant in comparison with the one caused by an aneurysm.

During the extraction of branches, the above test is made on each branch B_i to decide if it is an aneurysm or not. Branches that are in aneurysms are saved in a list to be treated later for quantification.

2.3 Aneurysm quantification

The construction of a shortest distance tree creates an oriented graph. The nodes of the graph are the voxels. The oriented edges link these voxels together to minimize their distance from the source voxel. Voxels of an aneurysm are the voxels that can be reached from voxels of the *aneurysm branch* by descending the graph. Since the *aneurysm branch* is connected to the *father* branch, which is inside the holding vessel, some of its voxels are inside the holding vessel, see Fig. 6-(a). In order to get rid of

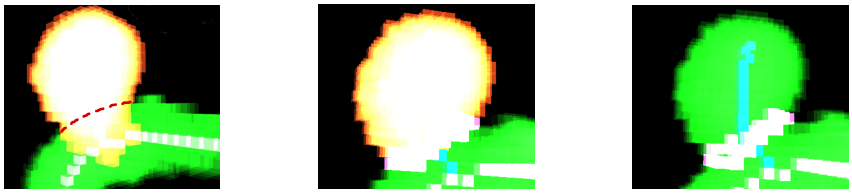


(a)

Branch	a	b	c
B_1	2.494	-0.012	0.000
B_2	1.250	-0.018	0.000
B_3	2.156	-0.105	0.003
B_4	3.509	3.831	-2.006

(b)

Fig. 5: Diameters variations for branches of the real dataset shown in Fig. 3: (a) The quadratic functions, note that they closely match straight lines for vessels, which is not the case for the one of the aneurysm (B_4). (b) Table1 shows values of a,b and c for each branch.



(a)In yellow, voxels linked to those of the aneurysmal branch.

(b) The voxels of the aneurysm.

(c) The neck of the aneurysm.

Fig. 6: Compute aneurysm's neck.

these voxels, we only add voxels if their distance from the branch of the holding vessel is greater than its radius, see Fig. 6-(b). The aneurysm's neck is the surface voxels of the aneurysm that have at least one neighbor that is not in the aneurysm, see Fig. 6-(c),(d).

Following a discussion with a surgeon, we found out that the following measures of the aneurysm are relevant to help the treatment decision:

• Size of the aneurysm: number of aneurysmal voxels.

• Maximum vertical diameter of the aneurysm ($Diam1$): to find this diameter, we look for the surface voxel which is the furthest from the origin j of the aneurysmal branch. $Diam1$ is the distance between this voxel and j.

• Maximum horizontal diameter of the aneurysm ($Diam2$): we look for the voxel m of the aneurysmal branch with maximum DFB, then $Diam2 = 2 \times DFB[m]$.

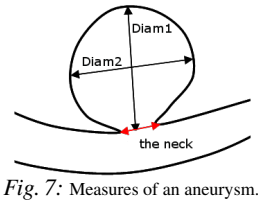
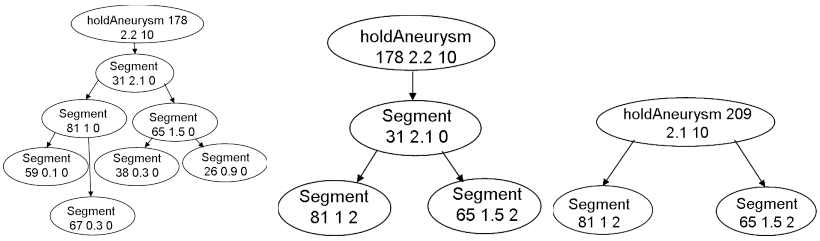


Fig. 7: Measures of an aneurysm.

2.4 Localization of the aneurysm

Regarding the method we use to extract centerlines, the result is a set of branches where each branch B_i (except B_0) has a father branch. On the same time, the branches do not correspond to blood vessels, a branch can be within several blood vessels. To get a graph that represents the resulting tree, we deal with segments. A segment is made of the voxels of a branch between its extremity and a junction, or between two successive junctions. We choose the widest segment (aneurysms excluded) as root, because it corresponds to the carotid (widest blood vessel), and we construct a graph. In Fig. 8-(a), we see the graph corresponding to the dataset of Fig. 5-(a).



(a)Initial graph (b)Graph without small branch (c)Final graph

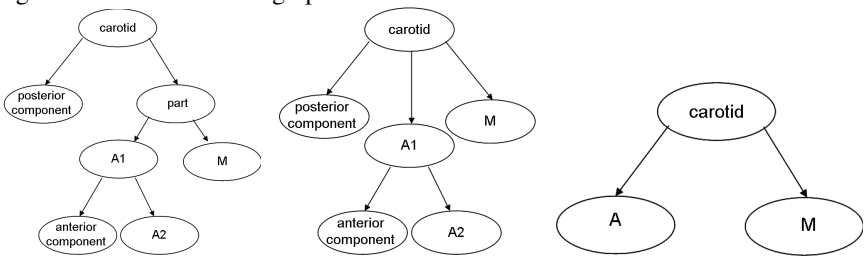
Fig. 8: Graphs for the dataset of Figure5-(a)

Graph matching is a well known problem, and graphs can be with or without attributes for both nodes and edges. If we consider our graph of segments without any attributes, the matching process will be mainly a topological one, meaning that if a node has two child nodes, it may be matched with any node with two children in the reference graph. To get a more accurate matching, we choose to use a graph with attributes.

As can be seen in Fig. 8-(a), we associate to each node of the graph three attributes: length, diameter of the segment, and number of children. The first two attributes are used to give an idea about the importance of the segment. Segments with small diameters or short lengths are considered very patient specific and unimportant. The corresponding nodes are then deleted from the graph (Fig. 8-(b)). We can describe this deletion step as a simplification of the graph. To keep a trace of the deleted nodes, we use the third attribute "number of children". Each time we decide to delete a node, we increase the number of children of its parent by one. Finally, we give the root of our graph a big number of children (10), to be sure that the root will be matched with the carotid.

Only the third attribute (number of children), is then used in the matching step. It helps to differentiate between vessels that are known to have a lot of bifurcations (vessel M) and those who have less bifurcations (vessel A), and both issued from the same parent (carotid), see Fig. 9.

Since the anatomy of the cerebral vascular tree is known, especially regarding the main vessels, we use a reference graph. In practice, not all vessels are segmented from acquired images, so several reference graphs with different resolutions are needed. Fig. 9 shows the reference graphs we use.



(a) (b) (c)

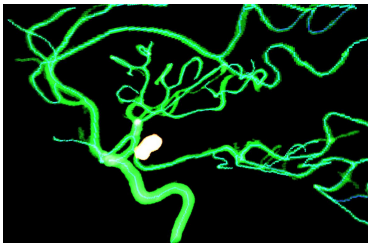
Fig. 9: Reference graphs

The localization of the aneurysm is then reduced to an inexact graph matching problem. We use the VF algorithm [17] to solve this problem. We try first to match our simplified graph with the most detailed reference graph 9-(a), then with 9-(b), and finally with 9-(c). In practice, more reference graphs can be used if needed.

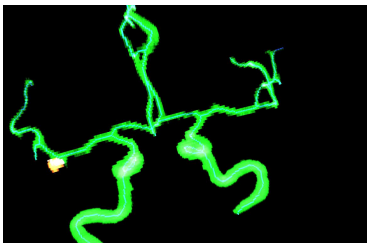
3 Results

We validated our approach on a set of twenty patients, using both MRA and CT imaging techniques for five and fifteen patients respectively. The set contained five males and fifteen females, the patients' ages varied from 33 to 78 years with an average of 51.68.

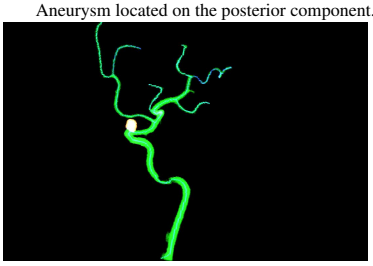
After segmentation, our method is applied on one connected component (either chosen by the user, or the largest one if no choice is made). The results reported no error of typeI (false negative) and two errors of typeII (false positive). Results of quantifications were compared to those provided by experts (experts provided quantifications for only 10 cases). We use the following formule to calculate the error of a measurement: $E = 100 \times \frac{\|provided - calculated\|}{provided}$. For $Diam1$, the error varied from 0.8 to 48 with an average of 11.7, for $Diam2$, it varied from 1.7 to 17.1 with an average of 8.25.



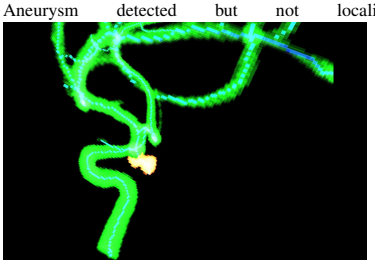
Aneurysm located on the posterior component.



Aneurysm detected but not localized.



Aneurysm located on the carotid.



Aneurysm located on the carotid.

Fig. 10: Some examples of aneurysms detected by our method.

Since our technique of localization does not consider cases where the whole cerebrovascular tree is present, the localisation was possible in ten cases and the localizations were distributed as follows: six aneurysms were localized on the carotid, two on vessel $A1$ and two on the posterior component. Fig. 10 shows some examples of the detected aneurysms. Calculation time on a Pentium(R) 4 CPU 3.00 GHz varied from 4.5 to 145.23 seconds with an average of 29.97. In practice, this time is almost linearly connected to the number of voxels.

4 Conclusion and future work

In this paper, we have presented a complete solution to automatically localize and quantify intracranial saccular aneurysms. First, we use a new distance-based method to find centerlines of the vascular tree. The centerlines are connected, thin (by construction), and centered, due to our modification of Dijkstra's algorithm. Moreover, since the distance map is calculated relative to a source voxel, the presented approach is invariant to rigid transformations. Then, aneurysms are automatically detected and quantified. Finally, the aneurysm is localized by graph-subgraph matching between a graph repre-

senting the centerlines and a reference graph.
When applying our method to 3D medical images, it proved to be fast and robust since the quality of the results is independent of small segmentation artifacts.

References

1. Ujiie, H., Tachibana, H., Hiramatsu, O., Hazel, A., Matsumoto, T., Ogasawara, Y., Nakajima, H., Hori, T., Takakura, K., Kajiya, F.: Effectes of size and shape (aspect ratio) on the heomdynamics of saccular aneurysms: a possible index for surgical treatment of intracranial aneurysms. Neurosurgery **45** (1999) 119–130
2. Weir, B.: Unruptured intracranial aneurysms: a review. J Neurosurgery **96** (2002) 3–42
3. Ecker, R., Hopkins, L.: Natural history of unruptured intracranial aneurysms. Neurosurg Focus **17**(5) (2004)
4. Wilson, D.L., Noble, J.A.: Segmentation of cerebral vessels and aneurysms from mr angiography data. In: IPMI '97: Proceedings of the 15th International Conference on Information Processing in Medical Imaging, London, UK, Springer-Verlag (1997) 423–428
5. Aylward, S., Pizer, S., Eberly, D., Bullitt, E.: Intensity ridge and widths for tubular object segmentation and description. Mathematical Methods in Biomedical Image Analysis, IEEE Workshop on **0** (1996) 0131
6. Frangi, A.F., Niessen, W.J., Vincken, K.L., Viergever, M.A.: Multiscale vessel enhancement filtering. Medical Image Computing and Computer-Assisted Interventation, MICCAI 98 (1998)
7. Wink, O., Niessen, W., Viergever, M.: Multiscale vessel tracking. Medical Image Analysis **23**(1) (January 2004) 130–133
8. Descoteaux, M., Collins, D.L., Siddiqi, K.: A geometric flow for segmenting vasculature in proton-density weighted mri. Medical Image Analysis **12**(4) (2008) 497 – 513
9. Millán, R.D., Dempere-Marco, L., Pozo, J., Cebral, J., Frangi, A.: Morphological characterization of intracranial aneurysms using 3-d moment invariants. Medical Imaging, IEEE Transactions on **26**(9) (2007) 1270–1282
10. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1** (1959) 269–271
11. Aylward, S.R., Bullitt, E.: Initialization, noise, singularities, and scale in height ridge traversal for tubular object centerline extraction. IEEE Transactions on Medical Imaging **21**(2) (2002) 61–75
12. Deschamps, T., Cohen, L.: Fast extraction of minimal paths in 3D images and applications to virtual endoscopy. Medical Image Analysis **5**(4) (2001)
13. Cornea, N., Silver, D., Min, P.: Curve-skeleton properties, applications, and algorithms. IEEE Transactions on Visualization and Computer Graphics **13**(3) (2007) 530–548
14. Bitter, I., Sato, M., Bender, M., McDonnell, K.T., Kaufman, A., Wan, M.: CEASAR: a smooth, accurate and robust centerline extraction algorithm. In: VIS '00: Proceedings of the conference on Visualization '00, Los Alamitos, CA, USA, IEEE Computer Society Press (2000) 45–52
15. Bitter, I., Kaufman, A.E., Sato, M.: Penalized-distance volumetric skeleton algorithm. IEEE Transactions on Visualization and Computer Graphics **7**(3) (2001) 195–206
16. Wan, M., Liang, Z., Ke, Q., Hong, L., Bitter, I., Kaufman, A.E.: Automatic centerline extraction for virtual colonoscopy. IEEE Trans. Med. Imaging **21** (2002) 1450–1460
17. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: An improved algorithm for matching large graphs. 3rd IAPRTC15 Workshop on Graph-based representations in Pattern Recognition (2001) 149–159

$\therefore$

A.9 SEGMENTATION OF TREE SEEDLING POINT CLOUDS INTO ELEMENTARY UNITS

Franck Hétroy-Wheeler, Eric Casella, Dobrina Boltcheva
International Journal of Remote Sensing, Taylor & Francis, 2016.

To appear in the *International Journal of Remote Sensing*
Vol. 00, No. 00, Month 20XX, 1–27

Segmentation of tree seedling point clouds into elementary units

Franck Hétroy-Wheeler[a*], Eric Casella[b*] and Dobrina Boltcheva[c]

[a] *Univ. Grenoble Alpes & Inria, Laboratoire Jean Kuntzmann, 655 avenue de l'Europe, F-38334 Saint Ismier cedex, France*
[b] *Centre for Sustainable Forestry and Climate Change, Forest research, Farnham, Surrey, GU10 4LH, United Kingdom*
[c] *Univ. Lorraine & Inria, LORIA, Nancy, France*

(*Received 00 Month 20XX; accepted 00 Month 20XX*)

This paper describes a new semi-automatic method to cluster TLS data into meaningful sets of points to extract plant components. The approach is designed for small plants with distinguishable branches and leaves, such as tree seedlings. It first creates a graph by connecting each point to its most relevant neighbours, then embeds the graph into a spectral space, and finally segments the embedding into clusters of points. The process can then be iterated on each cluster separately. The main idea underlying the approach is that the spectral embedding of the graph aligns the points along the shape's principal directions. A quantitative evaluation of the segmentation accuracy, as well as of leaf area estimates, is provided on a poplar seedling mock-up. It shows that the segmentation is robust with false positive and false negative rates around 1%. Qualitative results on four contrasting plant species with three different scan resolution levels each are also shown.

Keywords: terrestrial laser scanning; tLiDAR; 3D point cloud; segmentation; spectral clustering; tree seedling

1. Introduction

Functional-structural plant models describe a plant as a collection of interconnected elementary units (internode, petiole, leaf-blade, see Godin et al. (1999)). Their goal is to help biologists understand the relationships between the plant structure and the biological and physical mechanisms underlying the plant growth (Godin and Sinoquet (2005)). These models require an *in situ* validation on real plants, that can be done by measuring the three-dimensional (3D) characteristics of vegetation. Similarly the growing field of plant phenomics, which is concerned with the discovery and analysis of complex plant traits (Furbank and Tester (2011); International Plant Phenotyping Network (2016)), requires the measurement of individual quantitative parameters such as leaf characteristics.

Destructive measurements have long been used but are time consuming and expensive. As a consequence, various kinds of sensors are being investigated

*Corresponding authors. Emails: Franck.Hetroy@grenoble-inp.fr; Eric.Casella@forestry.gsi.gov.uk

for non-destructive and non-invasive plant metrology. For example, the use of different imaging techniques has been proposed for plant phenotyping, see Li et al. (2014) for a review. The most popular imaging techniques are based on single-lens cameras (Quan et al. (2006); Paproki et al. (2012)), time-of-flight cameras (Chéné et al. (2012); Alenyà et al. (2013); Chaivivatrakul et al. (2014); Xia et al. (2015)) or multi-view stereo imaging systems (Golbach et al. (2015); Lou et al. (2015); Rose et al. (2015)). All these methods allow one to reconstruct and measure single leaves, although some of them require manual interaction (Quan et al. (2006); Golbach et al. (2015); Rose et al. (2015)) or prior knowledge of the plant (Chéné et al. (2012); Chaivivatrakul et al. (2014)).

Terrestrial laser scanning (TLS), a remote sensing technique, has become an increasingly popular technique to measure vegetation from grass to forest plant species, see e.g. Dassot et al. (2011); Lin (2015) for recent reviews. Compared to imaging techniques, TLS provides direct accurate 3D measurements. It has also proved to be more robust to diverse environments, in particular to changing lighting conditions (Li et al. (2014); Lin (2015)). Thus, LiDAR seems more adapted to greenhouse and field conditions (Tilly et al. (2012); Lin (2015)).

TLS generates unstructured sets of points where its laser beam is incident and reflected. Thus it gives a raw sketch of the spatial distribution of plant elements in 3D, but it lacks explicit and essential information on their shape and connectivity. The points need to be clustered into geometrically meaningful sets for further analysis and dendrometric measurements. For example, leaf-blade points need to be separated from petiole and internode points to assess leaf areas.

In this paper, we segment TLS data of small plants or tree seedling scans into their elementary units: internodes, petioles and leaf-blades. Our method only considers the 3D positions of the points (no intensity value or normal estimate is required). As a consequence, it can be applied to sets of 3D points generated by other techniques than TLS such as time-of-flight cameras. We focus on accurate segmentation so that individual elementary unit characteristics such as leaf area are estimated as accurately as possible.

1.1. *Related work*

Segmentation of 3D data is critical for many applications in science. Research has considered the segmentation of point clouds into basic geometric primitives (planes, cylinders, spheres, etc.) for various purposes, such as building or city modelling (see Haala and Kada (2010) for a survey), reverse engineering of mechanical objects (e.g. Bey et al. (2011); Li et al. (2011)), or background subtraction, see Nguyen and Le (2013) for a recent overview. These approaches are designed for man-made objects which can be almost completely decomposed into uniform geometric shapes. Yet since stems and leaves are not exactly cylindrical and planar shapes, their efficiency to robustly segment plants is questionable.

The problem we are interested in, that is the segmentation of plants into their elementary units, has been partially tackled in the literature. Recovering the branching structure of a plant or a leaf-on tree is a specific issue that has been addressed in order to estimate various wood parameters by e.g., Bayer et al. (2013); Belton et al. (2013). Some authors additionally propose to reconstruct

the foliage, using some heuristics to position them, for example to create visually pleasing virtual 3D models of trees from TLS point clouds (e.g., Xu et al. (2007); Livny et al. (2010)) or to derive global characteristics such as total leaf and wood areas (Côté et al. (2009)). The number of leaves as well as their individual location and shape are plausible but do not correspond to the actual tree.

Other works segment a plant into two clusters only, one for the stems and one for the leaves, mostly by classifying points according to local geometric features (Belton et al. (2013); Paulus et al. (2013, 2014); Wahabzada et al. (2015)). Other approaches use geometric distance information (Tao et al. (2015)) or intensity information (Douglas et al. (2015)). Deriving a full segmentation of the plant from such a classification is possible in some cases, using prior knowledge about the plant or its organs (Paulus et al. (2014)). However this is not straightforward in general, especially when leaves are almost overlapping.

Geometric segmentation of a plant into elementary units has been proposed using either plant-specific prior knowledge (e.g., Kaminuma et al. (2004)) or a tedious interactive procedure (Dornbusch et al. (2007); Hosoi et al. (2011); Paulus et al. (2014)). In contrast, our approach only requires a minimum user interaction, and no prior knowledge. It is thus applicable to any species.

Yin et al. (2015) have recently proposed a destructive approach to accurately segment and reconstruct a pot plant. Their approach requires laser scanning the whole plant, manually cutting the plant leaves and laser scanning each leaf individually. This is time-consuming and obviously does not allow for tracking changes in the plant traits over time.

A approach analogous to that used here has recently been published by Lou et al. (2015), although they work on point cloud data generated using a multi-view stereo imaging system. The methodology is similar: a graph is first constructed to build neighbourhood relationships between the plant's points, then a spectral clustering approach is performed on this graph. However, we used more advanced graph construction and clustering techniques, as detailed in Sections 2.1.2 and 2.1.4.

1.2. *Approach*

The main contributions of this paper are:

(1) a semi-automatic approach to accurately segment a TLS 3D point cloud of a tree seedling into meaningful clusters of points (Section 2.1). More specifically, each cluster gathers points of an elementary unit of the tree seedling.The approach is global in the sense that leaves are not segmented first from branches, and robust to non uniform density in the point cloud;
(2) the assessment of the method robustness by quantitatively evaluating clustering results and individual leaf-blade areas on computer-generated scans from a plant mock-up (Sections 3.2.1 and 3.2.2).

Qualitative validation on real plant scans and parameter sensitivity analysis are also given (Sections 3.1 and 3.2.3, respectively).

2. Materials and methods

The segmentation algorithm underlying the approach is first explained in detail in Section 2.1. Reference data used in the experiments is then described in Section 2.2. The method used for statistical analysis of leaf area estimates is explained in Section 2.3.

2.1. *Point cloud segmentation algorithm*

In this section, the algorithm that analyses data collected from the TLS is described. The input point cloud data is merely a 3D location of points with no additional information. The algorithm is designed to cluster points into subsets corresponding to the plant elementary units. This algorithm is a three-stage process (Figure 1).

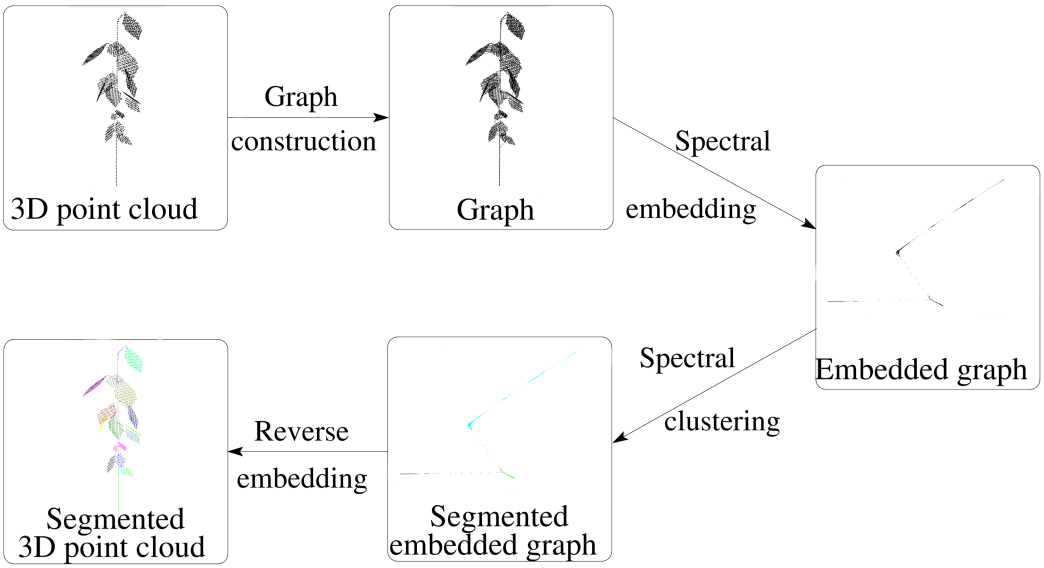


Figure 1. Pipeline of the segmentation method.

The first stage, called Graph Construction, finds neighbouring points for each point of the raw TLS data (see Section 2.1.2 for details). The second stage, called Spectral Embedding, finds the major intrinsic plant directions, i.e. the main directions of each elementary unit (see Section 2.1.3). This allows us to define the distances between neighbouring points according to the intrinsic plant directions, rather than the usual Euclidean distance. For example, the distance between two points sampled on a leaf-blade with an ellipsoid shape corresponds to the distance between their projections on the leaf's midrib (ellipsoid's main axis). Thus, two points on both sides of the ellipsoid's main axis but with similar projections will appear close to each other. As a consequence, this stage of the algorithm transforms the raw TLS data into a cloud of points aligned along principal plant axes (see Figure 2). Finally, the third stage, called Spectral Clustering, uses the computed neighbouring relationships to decompose the shape into subsets of points according to the principal plant axes. All points in a subset are given the same label (in our experiments, a colour), and points in different subsets have different labels. During this stage, each elementary unit is thus split from the one it originates. For example, a leaf-blade is separated from its petiole (see Section 2.1.4). Since each

point in the embedding space corresponds to a point in the Euclidean space, the segmentation of the input TLS data is automatically found by giving to each point the label of its associated point in the embedding space (reverse embedding).

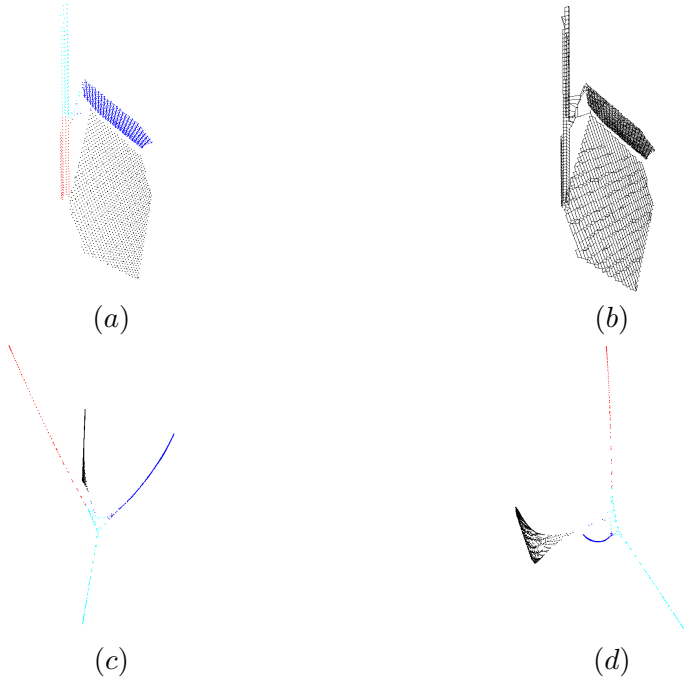


Figure 2. Example of the spectral embedding process. (*a*) Input scan. (*b*) Computed graph. (*c*,*d*) Point cloud embedded into a 3-D spectral space (two different views; see also the accompanying video). Colours are set to roughly indicate the elementary units.

2.1.1. Parameters

The algorithm uses three parameters, one for each stage:

(1) the minimum angle a between two neighbours of any point in the point cloud, for the graph construction;
(2) an estimate d of the number of intrinsic directions in the plant, for the spectral embedding;
(3) the number c of desired subsets of points (elementary units), for the clustering stage.

2.1.2. Graph construction

The first stage of the method aims at recovering neighbouring information between points. This is a difficult task since the only information available is the 3-D location of the points.

Usual methods create neighbouring relationships, called edges, between any point p and either all points which lie within a sphere of radius ε centred at p, or the k nearest points (Figure 3 (a,b)). These methods are known as the ε-Neighbourhood and the k-Nearest-Neighbours methods, respectively (Yang (2005); von Luxburg (2007)). ε and k are user-chosen parameters. ε-Neighbourhood is for example used by Belton et al. (2013), while the k-Nearest-Neighbours method is used by Côté et al. (2009); Lou et al. (2015). These methods are convenient so

long as the density of the point cloud is uniform, which is not the case for our TLS data. For non-uniform samples, many redundant edges may be created or relevant ones may be missed and the main problem is to find the right value for the parameters. This problem is shown on Figure 3 (a), where the ε-neighbourhood of a blue point is depicted for two different values of ε (in green and in red and green, respectively). Similarly, the k-nearest-neighbours, for $k = 2$ (in green) and $k = 5$ (in red and green), are shown on Figure 3 (b). If ε or k is low, the corresponding methods may miss relevant edges, such as the one between the blue point and the upper red point. If ε or k is high, they may create redundant edges such as the ones between the blue point and the left and right red points.

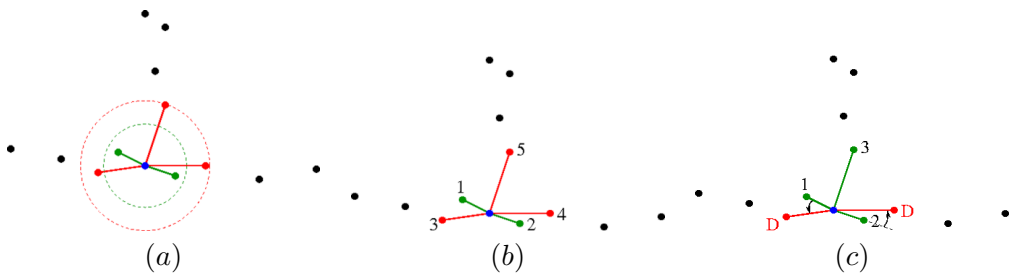


Figure 3. Examples of graph construction for the methods: (a) ε-neighbourhood of a point in blue, for two different values of ε. (b) k-nearest-neighbours, for $k = 2$ and $k = 5$. (c) Proposed method, for $a = 45°$.

Note that it is critical for the next stage of our approach, both in terms of memory usage and computation time, to avoid redundant neighbouring relationships since during this stage we work with an adjacency matrix computed from the neighbouring graph. Therefore, the lower the number of neighbours for a given point the sparser the matrix thus the faster the computation. This is why we have developed a specific algorithm which is summarised in Algorithm 1 and Figure 3 (c).

This algorithm starts by selecting a number k of candidate neighbours for every point $p \in P$. In practice, we choose $k = 0.1\%$ of the total number of points. Then, in order to select the neighbours within the set of candidates, it uses one parameter which is the minimum angle a at p allowed between two edges with endpoint p (see Figure 3 (c)). If many candidates lie in the same direction, only one (the closest) is selected as a neighbour of p. This prevents the creation of redundant, almost parallel, edges. On the contrary, this algorithm having found the closest point in a given direction will go on to look for other points farther away but in a distinctly different direction. Thus, this method captures all relevant edges and is robust to non-uniform density within the point cloud. Figure 3 (c) shows the result of the method for $a = 45°$. This method discards the two red points D since the corresponding edges are within a small angle of existing edges (in green) but it does capture point 3 which is a neighbour in a clearly different direction.

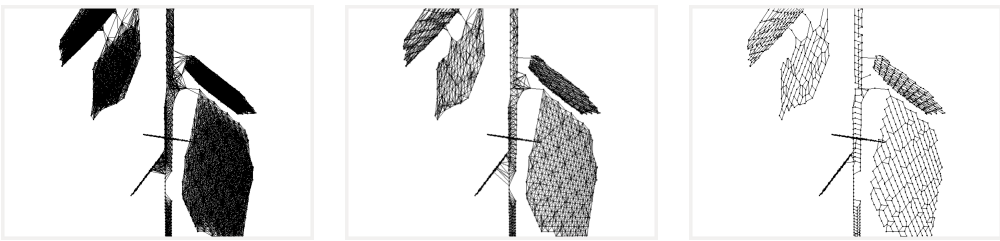
The graph construction runs this algorithm for every point p in the point cloud. The resulting neighbouring graph is thus the union of the selected edges $E(p)$ for every $p \in P$. Note that we do not look for mutual nearest neighbours: if q is computed as a valid neighbour of p but p is not considered as a valid neighbour of q, we still connect these two points. Our experimental results have shown that choosing the angle parameter $a = 90°$ is a good compromise in practice (see Section 3.2.3). This allows us to search for neighbours in the 3 cardinal directions around a point in 3D, which has been shown to be sufficient for building

Data: Point cloud P, a point $p \in P$, a user-chosen angle parameter a (in radian)
Result: Set $E(p)$ of the edges of the graph with endpoint p
$E(p) := \emptyset$;
Compute the k nearest neighbours of p in P, and put them in a priority queue Q ordered by increasing distance to p;
for $p' \in Q$ **do**
  if $\exists e \in E(p)$ *such that* $angle(pp', e) < a$ **then**
    Discard p';
  end
  else
    Put the edge pp' in $E(p)$;
  end
end

Algorithm 1: Building the neighbouring edges of a single point p in the cloud

a connected graph with as few as possible redundant edges.

Figure 4 shows an example of a graph construction. In this example, the petioles were very sparsely scanned compared to the leaf-blades and the main branch. For the ε-neighbourhood and the k-nearest-neighbours methods, the minimum value of the parameter was chosen such that the resulting graph was connected. Notice how both methods, contrary to ours, create numerous redundant edges on the main branch.



(a) (b) (c)

Figure 4. Example of graph reconstruction by (a) the ε-neighbourhood method (with $\varepsilon = 0.006m$); (b) the k-nearest-neighbours method (with $k = 8$) and (c) the method used in this study ($a = 90°$).

Other methods, which guarantee connectedness of the graph, are proposed by Yang (2005). However, their computational complexity (at least $O(n^2)$, where n is the number of points) may become prohibitive in the context of this study. The approach proposed in Algorithm 1 reaches a $O(n \log n)$ complexity with appropriate data structures, i.e. a kd-tree for the k-nearest-neighbours searches and heaps for the priority queues.

2.1.3. Spectral embedding

In the second stage, the major intrinsic directions of the shape are recovered and the weights of the edges modified accordingly. This is done using a technique called dimension reduction or spectral embedding. Indeed, embedding a (discrete) shape into a low-dimensional spectral space is known to help recover its intrinsic features (see e.g. Reuter et al. (2006)). In this work, we build on the Laplacian Eigenmaps framework of Belkin and Niyogi (2003), the main differences being

the graph construction approach described above and the choice of the distance between neighbouring points. This framework is now described.

Let A be the adjacency matrix of the graph constructed in the previous stage. Points are numbered from 1 to n, A is a $n \times n$ matrix such that $A(i,j)$ is equal to the weight of the edge connecting points i and j. $A(i,j) = 0$ if there is no edge between these points. The Euclidean distance between i and j is used as a weight. Let W be the diagonal valency matrix of the graph. $W(i,i)$ is equal to the sum of the weights of edges with endpoint i. The matrix $L = W - A$ is called the Laplacian matrix of the graph. The spectral embedding of the graph into a d-dimensional space is given by the d eigenvectors $V_1, \ldots, V_d$ of L associated with the first d non zero eigenvalues (in increasing order). Namely, the embedding coordinates of point number i are given by row i of the matrix whose columns are vectors $V_1, \ldots, V_d$ (von Luxburg (2007)).

It is known that the eigenvectors associated to the lowest non zero eigenvalues of L give the main "intrinsic" (curved) directions of the graph (Lévy (2006)). This property has previously been used for shape compression (Karni and Gotsman (2000)), progressive reconstruction (Lévy (2006)) and deformation (Dey et al. (2012)) purposes. The Laplacian spectral embedding is also known as the eigen skeleton of the input graph (Dey et al. (2012)). Using this property makes sense in the context of this study, since a plant is a strongly anisotropic shape; the natural directions of the plant follow the directions of each stem, branch, petiole and the main directions of each leaf-blade. It is therefore expected that the spectral embedding of the graph aligns points into a curve, or at least a strongly anisotropic shape, that samples each elementary unit of the plant, as shown on Figure 2. It is easier to segment the spectral embedding of the graph into subsets of points than the TLS data, since it does not depend on the particular shape of the leaves. Moreover, geometrical noise accumulated during the acquisition process is implicitly altered by the spectral embedding.

Note that computing the eigen-decomposition depends on the number of edges in the graph. The lower number of neighbours a point has, the sparser the matrix is, thus the faster the computation is. This is why the algorithm described in Section 2.1.2 is used rather than the standard ε-neighbourhood or k-nearest-neighbours methods. Figure 2 shows the 3-D embedding of a simple plant model with two leaves, thus having three main directions. Notice how the plant is nearly collapsed to a set of curves.

2.1.4. Spectral clustering

This stage clusters points into sets corresponding to the plant's elementary units (internodes, petioles and leaf-blades). To this aim, the point cloud is segmented according to its spectral embedding; the objective is thus to cluster together points of an elongated curve in the embedded shape. Since the embedded point cloud is almost a set of elongated curves (see Figure 2), segmentation techniques for stems such as e.g. Paulus et al. (2014) could be applied. However, they would not benefit from the point neighbourhood information retrieved from stage one of the approach (Section 2.1.2).

The usual clustering technique, applied in spectral space, is known as K-means clustering (von Luxburg (2007)). It is used for example by Lou et al. (2015).

K-means clustering randomly selects K initial "means" among the points, with K being a user-defined parameter. Each point is assigned to the nearest mean. Then, for each cluster of points, the closest point to the centroid (centre of mass) of the cluster is computed and selected as the new mean. The process is iterated until convergence to stable mean positions is reached, which is generally fast. This technique is well adapted to isotropic data, i.e, a point cloud without any principal direction. This is obviously not the case in this study where the graph is embedded in spectral space almost as a set of elongated curves. More general approaches such as expectation maximisation could be used, but as K-means clustering they do not naturally benefit from the neighbourhood information (graph edges). Note that Lou et al. (2015) merge neighbouring clusters with similar normals, but this may lead to undersegmentation since different elementary units (e.g., two leaves) may have similar normals.

A new clustering method, more adapted to elongated shapes, is therefore proposed. This method is described in Algorithm 2 and Figure 5. The idea is to compute the main directions of the graph (in spectral space), as sets of edge-connected points which are called the *segments*. As many segments as the desired number c of clusters are computed. Finally, each point of the graph is labelled according to its closest segment. Note that c should be odd, by construction. In case the desired number of clusters is even, we recommend to segment in $c+1$ clusters and merge two of them.

Data: Graph $G = (V, E)$ (in spectral space), desired number c of clusters
Result: Segmentation of V into disjoint sets $\{Cluster[1], \ldots, Cluster[c]\}$
$Source$:= farthest point to a random point of G;
$i := 1$;
$Segment[i]$:= ComputeShortestPaths($Source$,G);
while $i < c$ **do**
  $Segment[i+1]$:= ComputeShortestPaths($Segment[1..i]$,G);
  p := point of $Segment[1..i]$ connected to $Segment[i+1]$;
  j := number of the segment to which belongs p;
  Remove successive points of $Segment[j]$ from p to one of its end and add them to $Segment[i+2]$;
  $i+=2$;
end
ComputeShortestPaths($Segment[1..c]$,G);
for $p \in V$ **do**
  p' := closest point of $Segment[1..c]$ from p;
  j := number of the segment to which belongs p';
  Add p to $Cluster[j]$;
end

Algorithm 2: Proposed graph segmentation method (applied in spectral space).

2.1.5. Algorithmic details

Edges of the computed graph are weighted by a distance between their two endpoints called the commute-time distance, which represents the expected time for a random walk on the graph to travel from one point to the other and then return (Qiu and Hancock (2007)). In our plant segmentation context, this is a

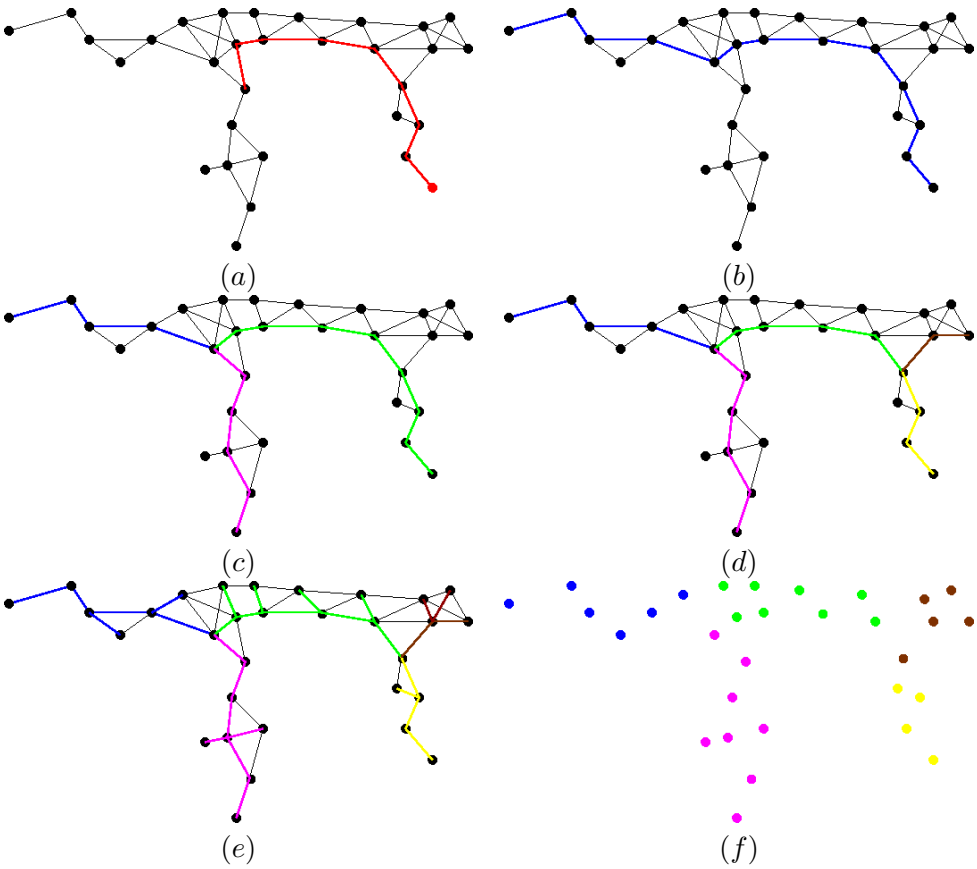


Figure 5. The segmentation process. (a) Input graph (in spectral space) and selected source point (in red, together with the path from the initial random point). (b,c,d) Computation of successive segments. (e) Shortest paths from each remaining point to the segments. (f) Computed clusters.

more meaningful distance than the Euclidean distance. For example, points on two different leaf-blades connected by a few edges (see Figure 6) may have a short Euclidean distance and a large commute-time distance in the graph. Since we want such points to belong to different clusters, we want their distance to be large. Moreover, commute-time distance has been proved to be robust against noise for clustering purposes (Qiu and Hancock (2007), Sec. 5.1).

The commute-time distance is similar to the Euclidean distance in spectral space, except each coordinate is divided by the corresponding eigenvalue. More precisely, the commute-time distance between points i and j is given by $\sqrt{\sum_k \frac{(i(k)-j(k))^2}{e(k)}}$, with $i(k)$ and $j(k)$ the k-th coordinates in spectral space of i and j, respectively (that is to say, the i-th and j-th coordinates of the k-th eigenvector V_k of the Laplacian matrix of the graph, as explained above), and $e(k)$ the k-th eigenvalue of the Laplacian matrix of the graph (Qiu and Hancock (2007), Sec. 2.3).

Finding the main segment of a weighted graph is a typical issue in medial structure axis and skeleton-related problems. The main segment of a graph can be computed successfully by using a one-source shortest path algorithm from an endpoint of the graph, e.g. the Dijkstra's algorithm (Dijkstra (1959)). This endpoint can be found as the farthest point to some random point (Lazarus and Verroust (1999)) and computed once again using a one-source shortest path algorithm. Other seg-

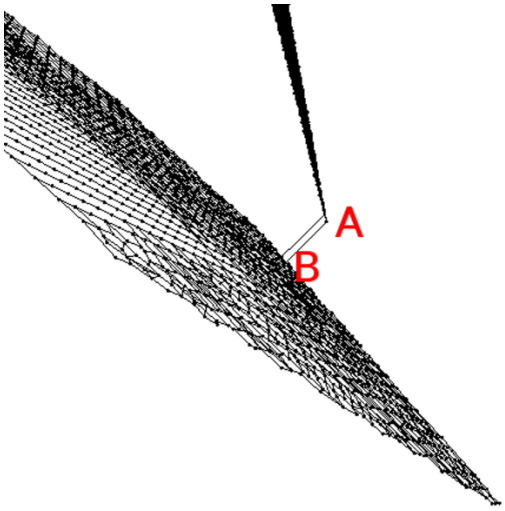


Figure 6. Points A and B are on two different leaf-blades of the poplar mock-up with high TLS resolution level (see Section 2.2.1). Their commute-time distance in the graph is large while their Euclidean distance is small.

ments are then computed the same way by taking all points of already computed segments as source points as in Hassan et al. (2011). As a result, each point of the graph is linked to its closest point on the segments and its distance to this point is computed. See Figure 5 for an example.

2.1.6. Asymptotic computational complexity

As explained above, Algorithm 2 uses a one-source shortest path algorithm $(c-1)/2+3$ times. Then, a Disjoint Set data structure (Cormen et al. (2009)) is used to cluster and label the points according to their closest point on the segments. The computational complexity of Dijkstra's algorithm, using a heap data structure, is $O(m+n\log n)$, where n is the number of points in the graph and m is the number of edges. The complexity of cluster creation within a Disjoint Set framework and using relevant heuristics is $O(n\log n)$ (Cormen et al. (2009)). The computational complexity of Algorithm 2 is thus $O(c(m+n\log n))$.

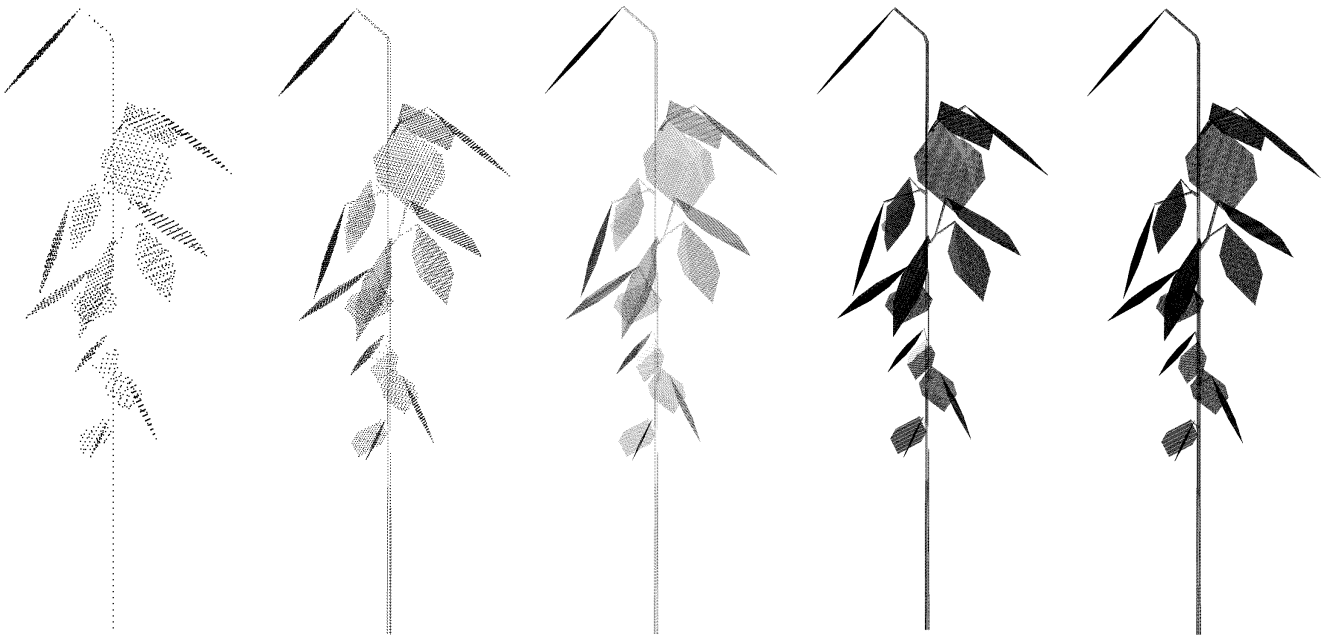
2.2. *Reference data*

Reference point clouds were obtained at various resolution levels, from two different ways. First, point clouds were generated from a virtual poplar seedling mock-up through a computer simulation of TLS (Section 2.2.1). Second, points clouds were acquired from four real plants using a Leica Geosystems HDS-6100 TLS device (Section 2.2.2).

2.2.1. Point cloud computations from a poplar mock-up

The 3D structure of a one-year-old single-stem seedling of poplar clone Trichobel (*Populus trichocarpa* Torr. & Gray x *P. trichocarpa*) was generated by the 3D Coppice Poplar Canopy Architecture model (3D CPCA) developed by Casella and Sinoquet (2003) (Figure 7, Table 1). The model is based on a multi-scale decomposition of a plant structure into components (axis and growth unit) described as a collection of metamers, themselves defined as a collection of elementary units (nodes plus internodes, petioles and leaf-blades) (Godin et al.

(1999)). For this study, axes (stem and branches) were divided as a sequence of conical frustums (a sequence of internode units), petioles were represented as cylinders and leaf-blades were regarded as planar objects. Each elementary unit was scaled to the appropriate geometric dimensions (e.g. height, base and top radius for a conical frustum) although a leaf-blade prototype was created and represented as a polygon with a set of 4 contiguous triangles to fit the leaf-blade shape and the allometric relationships between the leaf-blade area, the leaf-midrib length and the leaf-blade width. Each unit was them rotated and translated according to its orientation and location in the scene. Each unit was scaled so that no discontinuity between elements was possible, and there were no contact between laminae. Empirical functions and random deviation used in this study for the reconstruction of the plant architecture were as in Casella and Sinoquet (2003). The resulting poplar mock-up consisted of 17 leaves, 17 petioles and 24 internodes.



(L) (M) (H) (U) (U)

Figure 7. 3-D point cloud images of the poplar seedling mock-up used in this study for the low (L), medium (M), high (H) and ultra high (U) TLS resolution levels (Table 2). The last image shows the point cloud generated without simulation of the occlusion: all hits from the laser source to objects were recorded.

Seedling	Height (m)	Nb. of leaves	Total leaf area ($10^{-2}m^2$)
Poplar mock-up	0.462	17	2.617
Birch	0.650	7	1.296
Horse chestnut	0.607	9	16.567
Sweet chestnut	0.465	19	4.530
Red oak	0.547	10	5.553

Table 1. Structure parameters of the tree seedlings.

This mock-up could then be scanned from any point of view, after having placed a virtual TLS in the scene. Point clouds were computed for three TLS positions around the mock-up and for four scanner resolution levels i.e. by simulating the characteristics and settings of a Leica Geosystems HDS-6100 TLS device (Table 2)

used in this study for point cloud acquisitions from real plants (see next Section). The positions of the virtual TLS in the scene were computed for a distance of 3 meters from the base of the stem to the laser source, an elevation angle of 25° and an azimuth angle of 0, 120 or 240°. For each TLS resolution level, a point cloud was generated using a simple hit/not hit determination algorithm coded from a set of ray/objects (i.e. /cylinder, /cone and /triangle) intersection equations (see Haines (1989)) by determining either the closest or all hits from the laser source along any simulated ray trajectory within the scene. Every recorded hit $(x_{hit}, y_{hit}, z_{hit})$ was then referred to both its related object in the scene (e.g. leaf-blade #) and the position of the laser source $(x_{TLS}, y_{TLS}, z_{TLS})$. Point clouds were generated both with and without simulation of the occlusion, i.e. recording either only the closest or all hits from the laser source to objects.

Single shot phase-shift technology with single return signal				
Wavelength (nm)	$650 - 690$			
Range (m)	$0.3 - 50$ at 18% albedo			
Spot size at exit (m)	0.003			
Beam divergence (°)	0.0126			
Pre-set scanner resolution levels	Low (L)	Medium (M)	High (H)	Ultra high (U)
Angular sampling resolution (°)	0.072	0.036	0.018	0.009
Maximum point spacing at $3m$ (m)	0.0038	0.0019	0.0009	0.0005

Table 2. Characteristics and settings of the Leica HDS-6100 terrestrial laser scanner used in this study for point cloud acquisitions.

2.2.2. Point cloud acquisitions from real tree seedlings

TLS point clouds were acquired from tree seedlings of silver birch (*Betula pendula* Roth), horse chestnut (*Aesculus hippocastanum* L.), sweet chestnut (*Castanea sativa* Mill.) and red oak (*Quercus rubra* L.) (Table 1) using a Leica Geosystems HDS-6100 TLS device (Table 2). Each seedling was scanned inside a large area warehouse from three TLS positions around the plant (two for the Horse chestnut seedling) and for three scanner resolution levels. These species were chosen in order to get seedlings with varying structural and leaf geometrical complexities.

TLS point clouds of trees are usually noisy because of various interferences during the acquisition process, see e.g. Hebert and Krotkov (1992). Each point cloud was thus filtered in order to remove most of the outliers, using the statistical outlier removal filter of the Point Cloud Library (Rusu and Cousins (2011)). For each point p, its k nearest neighbours were first retrieved, and the mean distance d of these points to p was computed. If d is outside an interval defined by the mean and the standard deviation of the mean distances to all points, then p is detected as an outlier and removed. We have set k so that to remove as many outliers as possible, without removing relevant points. $k = 20$ has been taken for the horse chestnut, $k = 30$ for the sweet chestnut, and $k = 50$ for the silver birch and the red oak, for all resolution levels. Between 0.2% (red oak) and 11% (silver birch) of the points have been removed by this filtering. Table 3 shows the number of point in each point cloud after filtering. Each resolution level contains about 4 times more points than the previous one.

Seedling	Low (L)	Medium (M)	High (H)	Ultra high (U)
Poplar mock-up	2452	10244	40560	162704
Birch	3230	14412	60517	-
Horse chestnut	6691	12615	122022	-
Sweet chestnut	9761	38111	155186	-
Red oak	12667	50997	187054	-

Table 3. Number of points for each point cloud (after filtering).

2.3. *Statistical analysis of leaf area estimates on the poplar mock-up*

Leaf area (LA) has been estimated for each labelled leaf-blade, by projecting its points into the least-square fitting plane, computing the Delaunay triangulation of the projected points (Edelsbrunner (2001)), projecting the points back to their original positions and summing the areas of the Delaunay triangles. This was done for the leaf clusters as labelled in the input data, both without and with occlusions, as well as for the clusters computed with the presented algorithm. The quality of the method was then determined by two parameters, the root mean square error ($RMSE$) and bias (b), defined as:

$$RMSE = \sqrt{\frac{\sum_{k=1}^{n} (\hat{y}_{pk} - y_{ak})^2}{n}}$$

$$b = \sum_{k=1}^{n} \frac{(y_{sk} - y_{ak})}{n}$$

where n is the number of observations and $\hat{y}_{pk}$ is the predicted average value from the regression line between the simulated y_{sk} and the actual y_{ak} values for the kth observation.

3. Results and discussion

The algorithm has been implemented in C++ and Matlab. Because of the generally complex structure of a plant, perfect clusters may not be created in a single run. In practice, the algorithm is first run with a low number of desired clusters (less than the actual number of metamers), then each cluster is segmented by running this algorithm again. A simple graphical user interface has also been implemented, which allows merging clusters by selecting a point in each cluster. The overall approach is thus semi-automatic.

Results of the segmentation process on the poplar mock-up and on the four tree seedlings are shown in Section 3.1. A quantitative validation is provided in Section 3.2. It includes an evaluation of the segmentation accuracy, a statistical analysis at the leaf area scale, and a parameter sensitivity analysis of the algorithm.

3.1. *Qualitative results*

The method has been tested on the five different tree seedlings (Tables 1 and 3), for the low, medium and high resolution levels (Table 2), as well as the ultra high resolution level for the poplar mock-up.

3.1.1. First segmentation

Results of the first run of the algorithm are shown on Figure 8 for the poplar mock-up (ultra high resolution), the sweet chestnut (high resolution) and the red oak (medium resolution) seedlings. When a small number c of clusters is set, the algorithm usually segments the point cloud into connected subsets of elementary units, even when the point cloud is very noisy (e.g., the red oak). The higher value for c, the higher probability that a elementary unit (usually, a leaf) is segmented by the algorithm into several clusters (see Figure 9). We elaborate on the choice of c in Section 3.2.3.

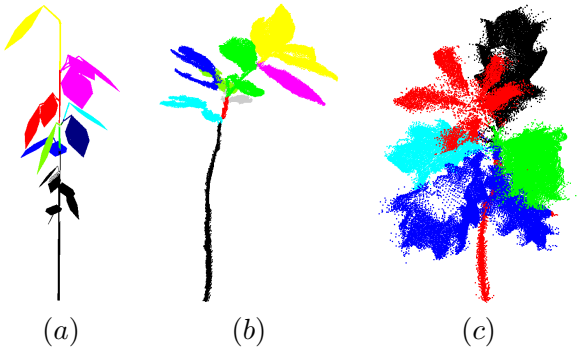


Figure 8. Segmentation results after the first run for (a) the poplar mock-up with occlusions, (b) the sweet chestnut and (c) the red oak, with $c = 11$ (a), $c = 9$ (b) and $c = 5$ (c), respectively.

Once the initial point cloud has been segmented, the user can select any given cluster through the graphical interface and re-run the algorithm on this cluster. This is done interactively; no botanical knowledge is used in our approach and the user decides which subsets of points to segment and when to stop this process.

3.1.2. Final segmentations

Qualitative final segmentation results on all scans are shown on Figures 10, 11, 12 and 13. The accompanying videos also show the segmentation results for the high resolution level point cloud of each of these five seedlings.

These results show that overall, despite large occlusions in real scans (see e.g. Figures 12 (d) and 13 (b)), the method correctly segments the point cloud into sets of individual leaf-blades, petioles and stem sections. Internodes can be detected when both ends are delimited by petioles and/or incident stems, otherwise they are merged. The method is insensitive to the leaf anatomy. It behaves correctly for both simple, small (e.g. sweet chestnut) and complex, large (e.g. red oak) leaves, as well as for both planar and curved leaves. However, a compound leaf is segmented into its leaflets, as shown for the horse chestnut, as each leaflet corresponds to a different intrinsic direction.

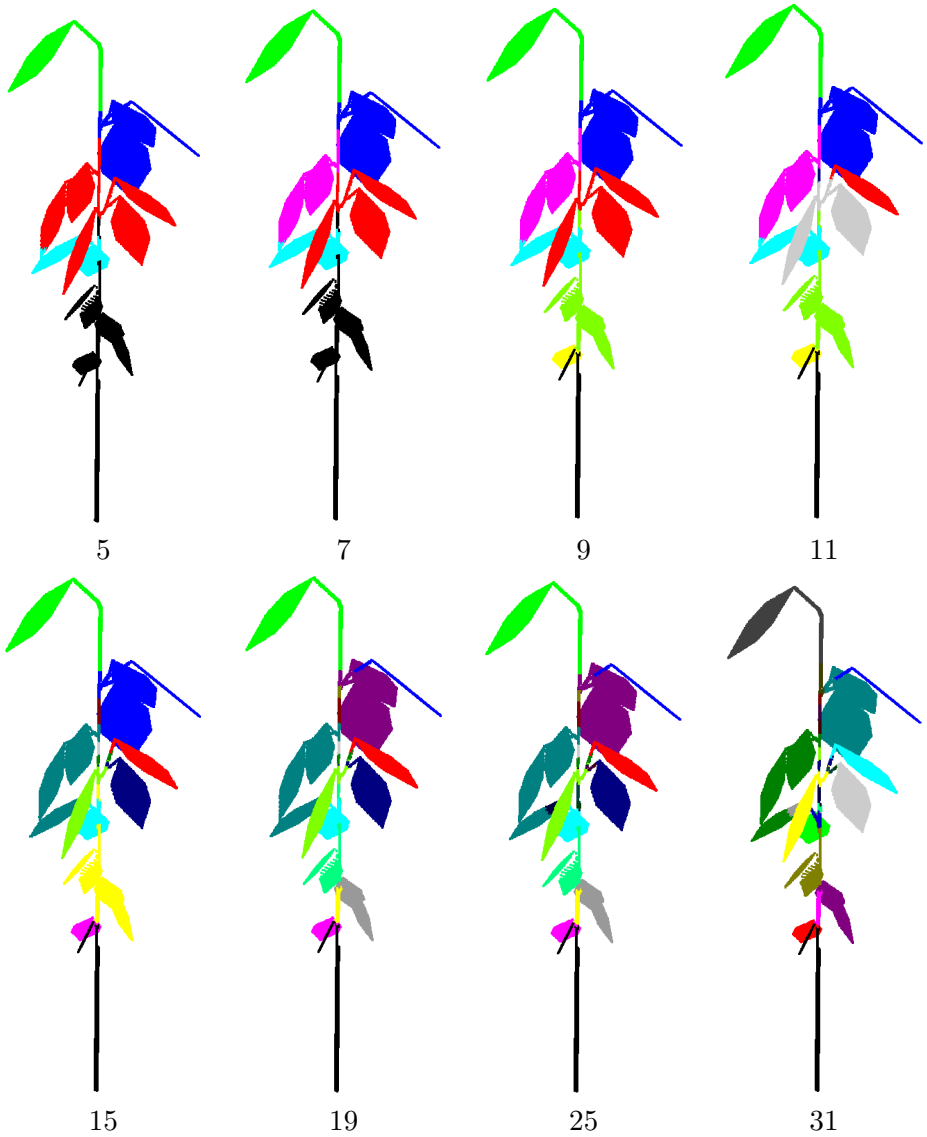


Figure 9. Result of the first iteration of the algorithm for various numbers c of clusters.

Figures 10 and 11 show that the resolution level does not have a strong influence on the segmentation, as will be demonstrated in Section 3.2. The algorithm is also robust to non uniform density within a point cloud, as shown for example on Figure 12 (a). Finally, the method is insensitive to the noise level. Even when points are spread over the boundaries of a unit (leaf or stem), they are included into the correct cluster (see Figures 12 (a) and 13 (c)). This is also shown by the following experiment.

3.1.3. Robustness to acquisition noise

In order to test the robustness of the approach, a raw scan of the sweet chestnut (high resolution level) from a single viewpoint has also been segmented. 72754 points belong to this unfiltered point cloud. Results are shown on Figure 14, to be compared with Figures 11 (H) and 12 (c). Points are correctly assigned to their corresponding cluster, except on ambiguous areas (for example between two neighbouring leaves).

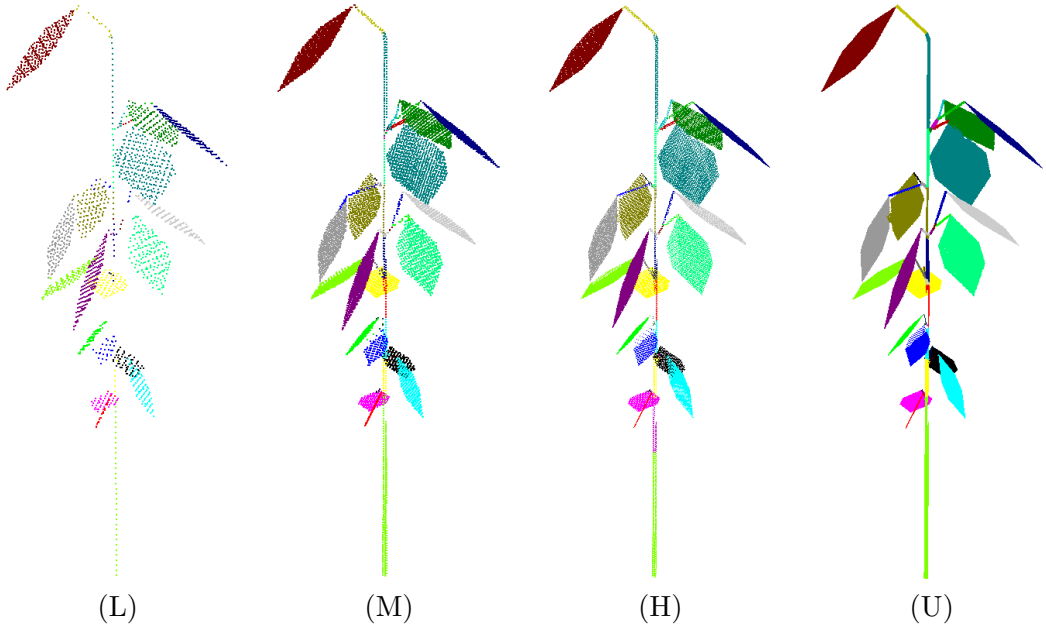


Figure 10. Segmentation results for the poplar mock-up with occlusions. The letter indicates the resolution level. On each point cloud, all points with the same colour belong to the same cluster.

3.2. *Evaluation*

3.2.1. Segmentation accuracy

The number of points correctly assigned to each elementary unit has been retrieved on the poplar mock-up. We call *false positive* (FP) for a given cluster a point that is labelled as part of this cluster by the segmentation algorithm, while it does not belong to this cluster in the input poplar mock-up point cloud. A point is a cluster *false negative* (FN) if it is not labelled as part of this cluster, while it actually belongs to it. A cluster's false positive rate (FPR) is the ratio of false positives over the actual number of points in the cluster. We define false negative rates (FNR) the same way.

It is worth mentioning that all points were labelled by the algorithm. This is because the constructed graph contains all points of the point cloud, and the spectral segmentation algorithm browses the whole graph. Figure 15 (a) shows that a large majority of leaf points are assigned to the correct leaf cluster, the worst case being cluster 5 (the bottom red leaf on Figure 10) in the ultra high resolution level point cloud with 5.77% of points assigned to another cluster. As shown on Figure 15 (b), leaf false positive rates are similar to false negative rates. The maximum is reached for cluster 2 in the ultra high resolution point cloud, which correspond to the bottom dark blue leaf on Figure 10, with 5.84% of false positives. False positives and negatives usually occur near the junction of a leaf to its petiole. The segmentation is not always accurate for the petioles and the internodes. This is explained by the fact that several sets of internodes and/or petioles are not fully segmented into elementary units, thus points of different internodes are assigned to the same cluster. This is for example the case of clusters 35 to 39, which correspond to internodes of the poplar's basis stem. Since no geometrical feature enables to split the stem into its internodes, and since the algorithm does not use any botanical knowledge, points are not segmented into internode clusters and remain in one global cluster, in green on Figure 10.

Figure 11. Segmentation results for point clouds of four contrasting plant seedlings with three different TLS resolution levels each. From top to bottom: birch, horse chestnut, sweet chestnut and red oak. On each point cloud, all points with the same colour belong to the same cluster.

Results are summarised in Table 4, in which we have computed means and standard deviations of the number of points over leaf, petiole and internode clusters, respectively. It shows that for leaves, false positive and negative rates remain below 3.4%. However, since it is difficult to unravel some petioles or internodes to their adjacent units from a pure geometrical point of view, points of neighbouring petiole or internode clusters are often pooled together. As a result, many petiole or internode clusters have no point assigned, leading to huge false positive and negative rates. Table 4 also shows that the resolution level has little impact on the segmentation accuracy, although results are slightly better for low resolution point clouds than for high resolution ones.

3.2.2. Statistical analysis of the leaf area estimates

Results of the leaf area estimates for the poplar mock-up are shown in Table 5. They show that the resolution level has a stronger influence on leaf area estimates than our segmentation method. Our estimates are always close to the estimates computed for the correct clusters.

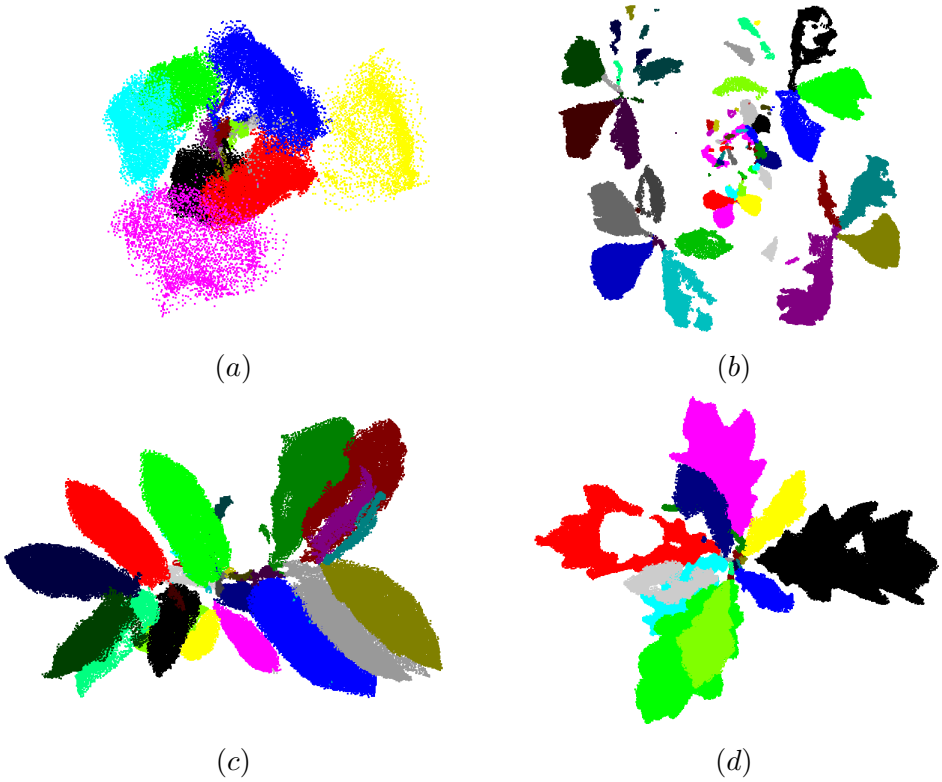


Figure 12. Segmentation results (top view). (*a*) Birch. (*b*) Horse chestnut. (*c*) Sweet chestnut. (*d*) Red oak. All are high TLS resolution level point clouds.

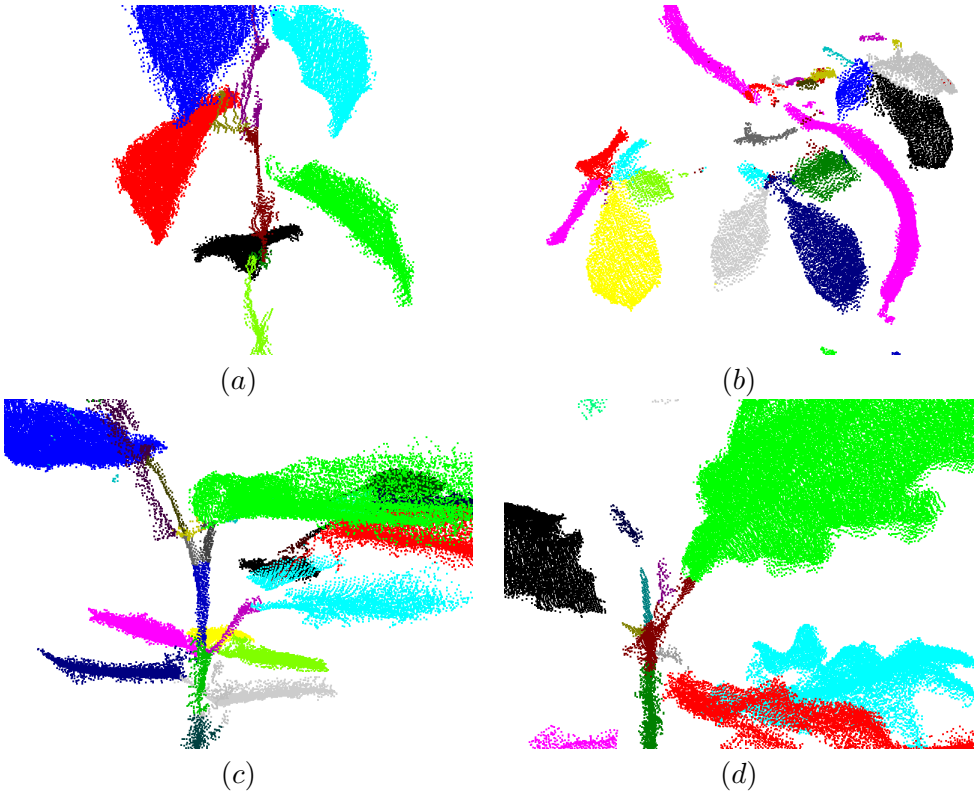
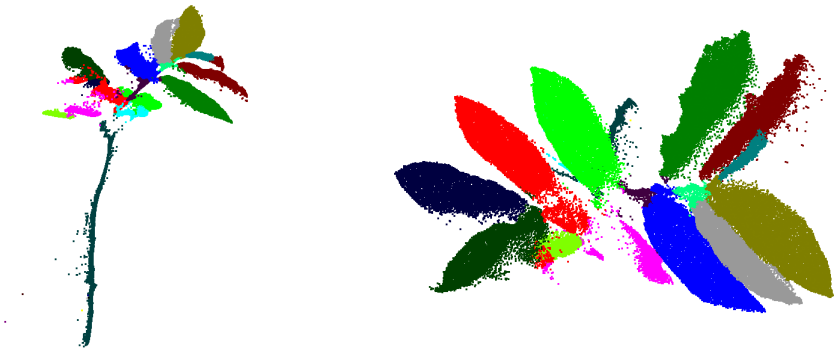
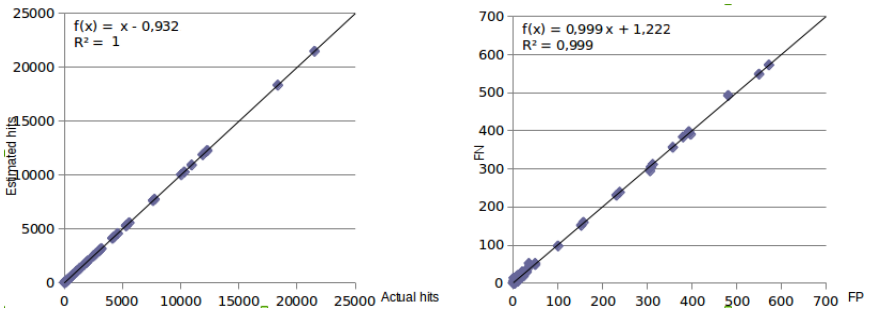


Figure 13. Segmentation results (close ups). (*a*) Birch. (*b*) Horse chestnut. (*c*) Sweet chestnut. (*d*) Red oak. All are high TLS resolution level point clouds.



(a) (b)

Figure 14. Segmentation results on a raw, unfiltered scan from a single viewpoint of the sweet chestnut seedling. (a) Front view. (b) Top view. High resolution level point cloud.



(a) (b)

Figure 15. Correlations between (a) the number of estimated points vs. the number of actual points and (b) the number of false negatives vs. the number of false positives, for all leaf clusters at all resolution levels.

Resolution	Nb. of points	Leaves		Petioles		Internodes	
		Mean	SD	Mean	SD	Mean	SD
Low (L)	Actual	136	85	3	3	5	4
	Est.	136	84	3	4	5	10
	FP	1	1	1	3	3	7
	FN	1	1	1	1	3	3
Medium (M)	Actual	548	346	13	12	39	26
	Est.	547	347	14	12	40	83
	FP	3	2	3	4	24	62
	FN	4	4	2	2	22	25
High (H)	Actual	2193	1382	50	49	133	81
	Est.	2190	1380	57	53	137	223
	FP	21	11	11	16	74	174
	FN	24	13	4	5	70	97
Ultra high (U)	Actual	8773	5528	196	192	562	338
	Est.	8773	5527	200	195	571	1086
	FP	295	163	13	14	285	805
	FN	295	163	10	13	276	381

Table 4. Number of points and false positives (FP) and negatives (FN) for the poplar mock-up. SD stands for *standard deviation*.

Resolution	Occlusions	Segmentation	Slope	Intercept ($10^{-4}m^2$)	RMSE ($10^{-4}m^2$)	Bias ($10^{-4}m^2$)	Mean LA (± SD) ($10^{-4}m^2$)
Low (L)	No	Actual	0.99	−1.17	0.28	−1.34	14.06 (7.7)
	Yes	Actual	1	−1.63	0.38	−1.64	13.75 (7.7)
	Yes	Computed	1	−1.19	0.90	−1.25	14.15 (7.8)
Medium (M)	No	Actual	1	−0.33	0.10	−0.39	15.01 (7.7)
	Yes	Actual	1	−0.53	0.14	−0.52	14.87 (7.7)
	Yes	Computed	1.01	−0.64	0.37	−0.43	14.96 (7.9)
High (H)	No	Actual	1	−0.09	0.02	−0.13	15.27 (7.7)
	Yes	Actual	1	−0.21	0.08	−0.18	15.21 (7.8)
	Yes	Computed	1	−0.20	0.15	−0.17	15.22 (7.8)
Ultra high (U)	No	Actual	1	−0.04	0.01	−0.04	15.35 (7.7)
	Yes	Actual	1	−0.09	0.04	−0.06	15.33 (7.8)
	Yes	Computed	1.03	−0.37	0.55	+0.12	15.51 (8.0)

Table 5. Statistical analysis of leaf area estimates on the poplar mock-up, for all resolution levels. Mean LA (± SD) control = 15.39 (7.7) $10^{-4}m^2$. SD stands for *standard deviation*.

Sampling underestimates leaf areas because our area estimation method creates a piecewise linear surface which boundary is defined by points labelled as belonging of the leaf. Since these points are actually fully inside the leaf and not on its boundary, and since leaves of the poplar mock-up are approximated by convex flat surfaces, the computed surface is smaller than the actual one. The higher resolution, the smaller underestimate, since the boundary points for the Delaunay triangulation are closer to the actual leaf boundary. In case of occlusions some points may be missing in a leaf cluster, leading to a smaller surface estimate, thus again an underestimate of the leaf area.

It can also be noticed on Table 5 that our approach tends to slightly overestimate leaf areas with respect to the estimate of the actual segmentation. This is mainly due to the fact that a false positive point may easily add a large area to the estimate, since the Delaunay triangulation will create big triangles between this point and other points in the cluster. This is a counterbalancing effect to the underestimates of the sampling and the resolution.

3.2.3. Sensitivity analysis

We now detail some experiments on the sensitivity of the method to the three parameters. The algorithm has been run on the poplar mock-up with different values for all three parameters, see Tables 6, 7 and 8. We have computed the false positive and false negative rates for each set of parameters, as well as the variation of the estimated total leaf area (−1% means that the estimated total leaf area is 1% lower than the actual leaf area, which is $0.02617m^2$).

a (°)	30	45	60	75	90
Edges	204195	136617	100446	81855	69618
Computation time (s)	376	371	361	367	361
Leaf FPR	0.91%	0.94%	0.89%	0.90%	0.94%
Leaf FNR	0.98%	0.96%	0.92%	0.97%	1.10%
Signed TLA error	−0.57%	−0.65%	1.03%	−0.84%	−1.15%

Table 6. Influence of parameter a on the poplar mock-up (H), with $d = 10$ and $c = 11$. TLA stands for *total leaf area*.

According to the experiments made (Table 6), the total leaf false positive (FPR) and negative (FNR) rates and signed leaf area errors only vary by 0.05%, 0.12%

d	5	10	15	30
Computation time (s)	272	361	439	711
Leaf FPR	0.93%	0.94%	0.87%	0.94%
Leaf FNR	0.95%	1.10%	0.95%	0.93%
Signed TLA error	−0.23%	−1.15%	−1.22%	−0.19%

Table 7. Influence of parameter d for the poplar mock-up (H), with $a = 90°$ and $c = 11$.

c	5	7	9	11	15	19	25	31
Nb. of overseg. leaves	0	0	1	1	1	1	2	2
Computation time (s)	329	340	351	361	384	409	441	478

Table 8. Influence of parameter c for the poplar mock-up (H), with $a = 90°$ and $d = 10$.

and 2.18%, respectively, with respect to the angle a. As explained in Section 2.1.6, computation time should be affected by the number of edges in the graph, which in turn depends on the value chosen for the angle parameter a. However, as shown in Table 6, although the number of edges exponentially decreases with the angle (see also Figure 16), the total computation time does not vary much with a. In particular, the computation time for stage 3 is always $91s$. This contradicts the theoretical computational complexity analysis (Section 2.1.6). We explain this counter-intuitive result by the fact that in practice, since our data is a set of elongated shapes in the embedding space, Dijkstra's algorithm does not update the shortest paths much and many edges of the graph are not used. Its complexity in practice is thus close to $O(n \log n)$ rather than $O(m + n \log n)$.

As a conclusion, the method is rather insensitive to parameter a. However, in the case the graph is to be stored in a file, we advise to choose a value of $a = 90°$ to reduce its size (see Figure 16). According to our experiments (not shown here), a value of a greater than 90° may lead to a disconnected graph.

The total computation time linearly increases with respect to the number d of intrinsic directions (Table 7). This parameter does not affect much the total false positive and negative rates, which only varies by 0.07% and 0.17%, respectively, and the total leaf area error, which only varies by 1.03%. Therefore, it is not necessary to set a high number of intrinsic directions. Our experiments indicate that $d = 5$ or $d = 10$ are good guesses in most of the cases.

Our experiments (Table 8, Figure 9) show that choosing a large number c of clusters may lead to over-segmentations of leaves. On the poplar mock-up, the bottom leaf is segmented in two different clusters from $c = 9$ (not visible on Figure 9 since this leaf is side-view), and this is also the case for a second leaf from $c = 25$. To overcome this problem, we suggest to first set a small value for c. According to our experiments, $c \sim 25\%$ of the total final number of clusters is generally a good guess. If some elementary units are nonetheless over-segmented, we provide a graphical interface to easily select and merge the corresponding clusters. For the examples shown on Figure 8, c was set to $23\%, 27\%$ and 26% of the final number of clusters, respectively ($c = 11, 9$ and 5 for $48, 33$ and 19 final clusters).

Computation time linearly increases with respect to the number c of desired clusters, as shown in Table 8. This is consistent with the previously explained computational complexity analysis (Section 2.1.6). Note that indicated computation times are for the first iteration only.

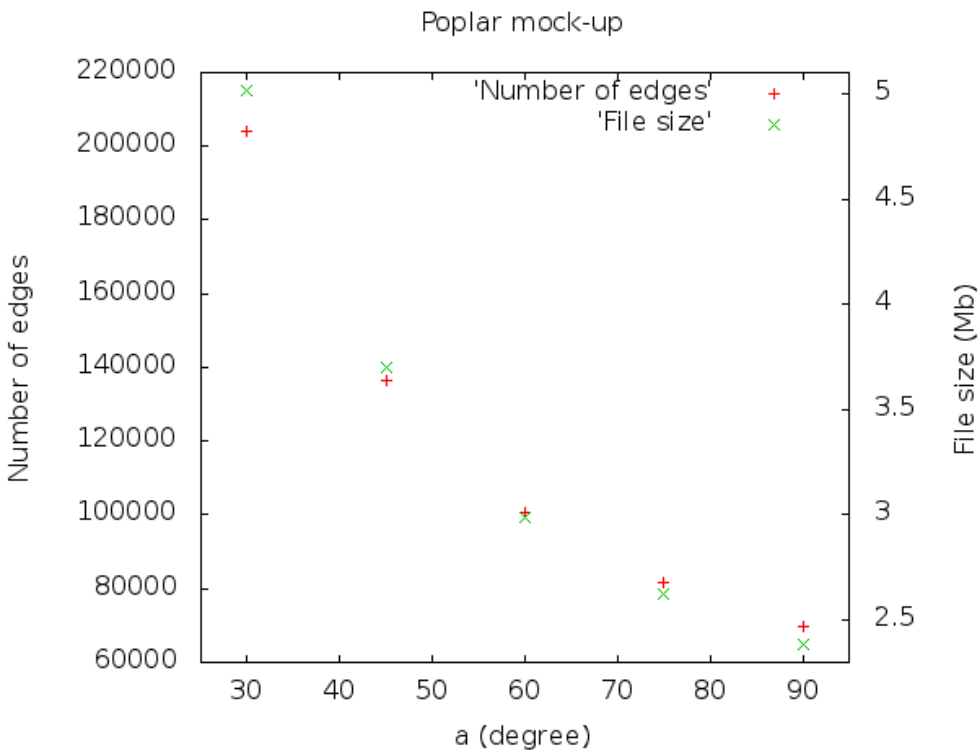


Figure 16. Number of edges in the graph and file size with respect to the chosen angle a.

4. Discussion and conclusion

We presented here a semi-automatic method to segment a TLS point cloud of a small plant into its elementary units (internodes, petioles and leaf-blades). Qualitative results on four real tree seedlings show that such small scale plants are successfully split into leaf, petiole and stem components. The only two exceptions are compound leaves which are segmented in their leaflets, and adjacent internodes on a stem which may not be separated if no geometrical feature (bud, incident stem) is available. The method does not use any prior botanical knowledge, therefore can be applied in a wide variety of cases. Quantitative results on leaves show that the method is robust (around 1% labelling error) and leads to accurate leaf area estimates.

Only three parameters are used by the method. Only one of them, namely the desired number c of clusters, has an actual influence on the results. According to our tests on four different seedlings, results are insensitive to the branching structure and the leaf anatomy. Acquisition noise during the scanning process is also robustly handled, as shown on the red oak scans. The scan resolution also has little effect on the segmentation result, but has a strong influence on the leaf area computation.

Since no botanical knowledge is used by the algorithm, computed elementary units are not explicitly labelled as leaves, petioles or leaf-blades. This could be done in an additional step with a Principal Component Analysis as in Belton

et al. (2013) or feature based histograms as in Paulus et al. (2013).

The proposed method being semi-automatic, it is suited for small plants such as tree seedlings but may be time consuming for more complex structures. In order to enhance the quality of the segmentation with a large number c of clusters, thus to reduce the interaction time for large scale trees, two improvements are planned in the future. First, we are currently working on the correction of the acquisition noise during the scanning process, in order to reduce the number of points sparsely sampled between elementary units. Filtering the input scans in a pre-processing step, which has been done in this paper, is not a perfect solution since it removes points and thus leads to underestimates of the leaf areas. Second, we plan to enhance the graph construction process (first step of the algorithm), in order to decrease the number of edges between two non adjacent elementary units from a botanical point of view (e.g., two leaves, as in Figure 6). Then, the algorithm will be tested on more complex structures such as full-scale trees.

Acknowledgements

The authors would like to express their sincere gratitude to the Forestry Commission, the University of Grenoble Alpes and Inria for funding this work, as well as to Rémy Cumont for his participation in coding the segmentation algorithm and Dr Elisa Hétroy-Wheeler for proof-reading the paper.

Author contributions: E.C. and F.H.W. designed the research; F.H.W. designed and coded the segmentation algorithm; E.C. performed the real data acquisition and coded the hit-no hit algorithm for point cloud simulations; D.B. filtered the point clouds; F.H.W. and E.C. analysed the results and wrote the paper.

Funding

The Forestry Commission, the University of Grenoble Alpes (through an AGIR project) and Inria (through the Action de Recherche Collaborative PlantScan3D).

References

Alenyà, G., B. Dellen, S. Foix, and C. Torras (2013). Robotized plant probing: Leaf segmentation utilizing time-of-flight data. *IEEE Robotics and Automation Magazine 20*(3), 50–59.

Bayer, D., S. Seifert, and H. Pretzsch (2013). Structural crown properties of norway spruce (*Picea abies* [l.] karst.) and european beech (*Fagus sylvatica* [l.]) in mixed versus pure stands revealed by terrestrial laser scanning. *Trees 27*, 1035–1047.

Belkin, M. and P. Niyogi (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation 15*, 1373–1396.

Belton, D., S. Moncrieff, and J. Chapman (2013). Processing tree point clouds using gaussian mixture models. In *Proceedings of the ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 43–48.

Bey, A., R. Chaine, R. Marc, G. Thibault, and S. Akkouche (2011). Reconstruction of consistent 3d cad models from point cloud data using a priori cad model. In *Proceedings of the ISPRS Workshop on Laser Scanning*, pp. 289–294.

Casella, E. and H. Sinoquet (2003). A method for describing the canopy architecture of coppice poplar with allometric relationships. *Tree Physiology 23*, 1153–1169.

Chaivivatrakul, S., L. Tang, M. N. Dailey, and A. D. Nakarmi (2014). Automatic morphological trait characterization for corn plants via 3d holographic reconstruction. *Computers and Electronics in Agriculture 109*, 10–123.

Chéné, Y., D. Rousseau, P. Lucidarme, J. Bertheloot, V. Caffier, P. Morel, E. Belin, and F. Chapeau-Blondeau (2012). On the use of depth camera for 3d phenotyping of entire plants. *Computers and Electronics in Agriculture 82*, 122–127.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to algorithms. 3th ed.* MIT Press.

Côté, J.-F., J.-L. Widlowski, R. A. Fournier, and M. M. Verstraete (2009). The structural and radiative consistency of three-dimensional tree reconstructions from terrestrial lidar. *Remote Sensing of Environment 113*, 1067–1081.

Dassot, M., T. Constant, and M. Fournier (2011). The use of terrestrial lidar technology in forest science: application fields, benefits and challenges. *Annals of Forest Science 68*, 959–974.

Dey, T. K., P. Ranjan, and Y. Wang (2012). Eigen deformation of 3d models. *The Visual Computer 28*, 585–595.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269–271.

Dornbusch, T., P. Wernecke, and W. Diepenbrock (2007). A method to extract morphological traits of plant organs from 3d point clouds as a database for an architectural plant model. *Ecological Modelling 200*, 119–129.

Douglas, E. S., J. Martel, Z. Li, G. Howe, K. Hewawasam, R. A. Marshall, C. L. Schaaf, T. A. Cook, G. J. Newnham, A. Strahler, and S. Chakrabarti (2015). Finding leaves in the forest: the dual-wavelength echidna lidar. *IEEE Geoscience and Remote Sensing Letters 12*(4), 776–780.

Edelsbrunner, H. (2001). *Geometry and topology for mesh generation.* Cambridge University Press.

Furbank, R. T. and M. Tester (2011). Phenomics – technologies to relieve the phenotyping bottleneck. *Trends in Plant Science 16*(12), 635–644.

Godin, C., E. Costes, and H. Sinoquet (1999). A method for describing plant architecture which integrates topology and geometry. *Annals of Botany 84*, 343–357.

Godin, C. and H. Sinoquet (2005). Functionalstructural plant modelling. *New phytologist 166*(3), 705–708.

Golbach, F., G. Kootstra, S. Damjanovic, G. Otten, and R. van de Zedde (2015). Validation of plant part measurements using a 3d reconstruction method suitable for high-throughput seedling phenotyping. *Machine Vision and Applications*, 1–18.

Haala, N. and M. Kada (2010). An update on automatic 3d building reconstruction. *ISPRS Journal of Photogrammetry and Remote Sensing 65*, 570–580.

Haines, E. (1989). Essential ray tracing algorithms. In A. Glassner (Ed.), *An introduction to ray tracing*, pp. 33–77. Academic Press.

Hassan, S., F. Hétroy, F. Faure, and O. Palombi (2011). Automatic localization and quantification of intracranial aneurysms. *Computer Analysis of Images and Patterns. Lecture Notes in Computer Science 6854*, 554–562.

Hebert, M. and E. Krotkov (1992). 3-d measurements from imaging laser radars: how good are they? *Image and Vision Computing 10*(3), 170–178.

Hosoi, F., K. Nakabayashi, and K. Omasa (2011). 3-d modeling of tomato canopies using a high-resolution portable scanning lidar for extracting structural information. *Sensors 11*, 2166–2174.

International Plant Phenotyping Network (Accessed: 08-02-2016). http://www.plant-phenotyping.org/.

Kaminuma, E., N. Heida, Y. Tsumoto, N. Yamamoto, N. Goto, N. Okamoto, A. Konagaya, M. Matsui, and T. Toyoda (2004). Automatic quantification of morphological traits via three-dimensional measurement of arabidopsis. *The Plant Journal 38*, 358–365.

Karni, Z. and C. Gotsman (2000). Spectral compression of mesh geometry. In *Proceedings*

of the 27th annual conference on Computer graphics and interactive techniques (SIGGRAPH), pp. 279–286.

Lazarus, F. and A. Verroust (1999). Level set diagrams of polyhedral objects. In *Proceedings of the 5th ACM Symposium on Solid Modeling and Applications (SMA)*, pp. 130–140.

Lévy, B. (2006). Laplace-beltrami eigenfunctions: towards an algorithm that understands geometry. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications (SMI)*, pp. 13.

Li, L., Q. Zhang, and D. Huang (2014). A review of imaging techniques for plant phenotyping. *Sensors 14*, 20078–20111.

Li, Y., X. Wu, Y. Chrysathou, A. Sharf, D. Cohen-Or, and N. Mitra (2011). Globfit: consistently fitting primitives by discovering global relations. *ACM Transactions on Graphics 30*(4), 52.

Lin, Y. (2015). Lidar: An important tool for next-generation phenotyping technology of high potential for plant phenomics? *Computers and Electronics in Agriculture 119*, 61–73.

Livny, Y., F. Yan, M. Olson, B. Chen, H. Zhang, and J. El-Sana (2010). Automatic reconstruction of tree skeletal structures from point clouds. *ACM Transactions on Graphics 29*, 151.

Lou, L., Y. Liu, M. Shen, J. Han, F. Corke, and J. H. Doonan (2015). Estimation of branch angle from 3d point cloud of plants. In *Proceedings of the International Conference on 3D Vision (3DV)*, pp. 554–561.

Nguyen, A. and B. Le (2013). 3d point cloud segmentation: a survey. In *Proceedings of the 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, pp. 225–230.

Paproki, A., X. Sirault, S. Berry, R. Furbank, and J. Fripp (2012). A novel mesh processing based technique for 3d plant analysis. *BMC Plant Biology 12*(1), 63.

Paulus, S., J. Dupuis, A.-K. Mahlein, and H. Kuhlmann (2013). Surface feature based classification of plant organs from 3d laserscanned point clouds for plant phenotyping. *BMC Bioinformatics 14*, 238.

Paulus, S., J. Dupuis, S. Riedel, and H. Kuhlmann (2014). Automated analysis of barley organs using 3d laser scanning: an approach for high throughput phenotyping. *Sensors 14*, 12670–12686.

Paulus, S., H. Schumann, H. Kuhlmann, and J. Léon (2014). High-precision laser scanning system for capturing 3d plant architecture and analysing growth of cereal plants. *Biosystems Engineering 121*, 1–11.

Qiu, H. J. and E. R. Hancock (2007). Clustering and embedding using commute times. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 1873–1890.

Quan, L., P. Tan, G. Zeng, L. Yuan, J. Wang, and S. B. Kang (2006). Image-based plant modeling. *ACM Transactions on Graphics 25*(3), 599–604.

Reuter, M., F. E. Wolter, and N. Peinecke (2006). Laplace-beltrami spectra as 'shape-dna' of surfaces and solids. *Computer-Aided Design 38*, 342–366.

Rose, J. C., S. Paulus, and H. Kuhlmann (2015). Accuracy analysis of a multi-view stereo approach for phenotyping of tomato plants at the organ level. *Sensors 15*, 9651–9665.

Rusu, R. B. and S. Cousins (2011). 3d is here: Point cloud library (pcl). In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Tao, S., Q. Guo, S. Xu, Y. Su, Y. Li, and F. Wu (2015). A geometric method for wood-leaf separation using terrestrial and simulated lidar data. *Photogrammetric Engineering and Remote Sensing 81*(10), 767–776.

Tilly, N., D. Hoffmeister, H. Liang, Q. Cao, Y. Liu, V. Lenz-Wiedemann, Y. Miao, and G. Bareth (2012). Evaluation of terrestrial laser scanning for rice growth monitoring. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Science 39*, B7.

von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing 17*, 395–416.

Wahabzada, M., S. Paulus, K. Kersting, and A.-K. Mahlein (2015). Automated interpretation of 3d laserscanned point clouds for plant organ segmentation. *BMC Bioinformatics 16*, 248.

Xia, C., L. Wang, B.-K. Chung, and J.-M. Lee (2015). In situ 3d segmentation of individual plant leaves using a rgb-d camera for agricultural automation. *Sensors 15*, 20463–20479.

Xu, H., N. Gossett, and B. Chen (2007). Knowledge and heuristic based modeling of laser-scanned trees. *ACM Transactions on Graphics 26*(4), 19.

Yang, L. (2005). Building connected neighborhood graphs for isometric data embedding. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, pp. 722–728.

Yin, K., H. Huang, P. Long, A. Gaissinski, M. Gong, and A. Sharf (2015). Full 3d plant reconstruction via intrusive acquisition. *Computer Graphics Forum*.

APPENDIX B

List of publications

B.1 GEOMETRICAL, TOPOLOGICAL AND PERCEPTUAL ANALYSIS OF 3D MESHES

1. F. Hétroy, D. Attali. *Topological quadrangulations of closed triangulated surfaces using the Reeb graph*. In A. Braquelaire, J.-O. Lachaud and A. Vialard, editors, Lecture Notes in Computer Science, Vol. 2301, pp. 57–68, Springer, 2002. Proc. of Discrete Geometry for Computer Imagery (DGCI) 2002, Bordeaux, France, April 2002.

2. F. Hétroy, D. Attali. *Topological quadrangulations of closed triangulated surfaces using the Reeb graph*. Graphical Models, 65(1-3), pp. 131–148, Elsevier, 2003. doi:10.1016/S1524-0703(03)00005-5.

3. F. Hétroy, D. Attali. *Detection of constrictions on closed polyhedral surfaces*. Eurographics-IEEE TCVG Visualization Symposium, pp. 67–74, Grenoble, France, May 2003.

4. F. Hétroy, D. Attali. *From a closed piecewise geodesic to a constriction on a closed polyhedral surface*. Pacific Conference on Computer Graphics and Applications (Pacific Graphics) 2003, pp. 394–398, Canmore, Alberta, Canada, October 2003.

5. F. Hétroy. *Constriction computation using surface curvature*. Eurographics (short paper), Aug. 29, 2005, Dublin, Ireland, pp. 1–4.

6. F. Hétroy, S. Rey, C. Andujar, P. Brunet, A. Vinacua. *Mesh Repair with Topology Control*. Inria Research Report, RR-6535, 2008.

7. J.-C. Léon, L. de Floriani, F. Hétroy. *Classification of non-manifold singularities from transformations of 2-manifolds*. IEEE International Conference on Shape Modeling and Applications (SMI), Jun., 2009, Beijing, China.

8. D. Boltcheva, S. Merino Aceituno, J.-C. Léon, F. Hétroy. *Constructive Mayer-Vietoris Algorithm: Computing the Homology of Unions of Simplicial Complexes*. Inria Research Report, RR-7471, 2010.

9. F. Hétroy, S. Rey, C. Andujar, P. Brunet, A. Vinacua. *Mesh repair with user-friendly topology control*. Computer Aided Design, 43(1), pp. 101–113, Elsevier, 2011. doi:10.1016/j.cad.2010.09.012.

10. D. Boltcheva, D. Canino, S. Merino Aceituno, J.-C. Léon, L. De Floriani, F. Hétroy. *An iterative algorithm for homology computation on simplicial shapes*. Computer-Aided Design, 43(11), pp. 1457–1467, Elsevier, 2011. doi:10.1016/j.cad.2011.08.015.

11. L. Wang, F. Hétroy-Wheeler, E. Boyer. *A hierarchical approach for regular centroidal Voronoi tessellations*, Computer Graphics Forum, 35(1), pp. 152–165, Wiley-Blackwell, 2016. doi:10.1111/cgf.12716.

12. G. Nader, K. Wang, F. Hétroy-Wheeler, F. Dupont. *Just Noticeable Distortion profile for flat-shaded 3D mesh surfaces*, IEEE Transactions on Visualization and Computer Graphics, 2016. doi:10.1109/TVCG.2015.2507578.

B.2 Digital geometry processing for shapes in motion

1. G. Aujay, F. Hétroy, F. Lazarus. *Construction automatique d'un squelette pour l'animation de personnages*. 19èmes Journées de l'Association Française d'Informatique Graphique (AFIG), Nov. 22, 2006, Bordeaux, France.

2. G. Aujay, F. Hétroy, F. Lazarus, C. Depraz. *Harmonic skeleton for realistic character animation*. ACM-SIGGRAPH/Eurographics Symposium on Computer Animation, Aug. 3, 2007, San Diego, United States.

3. L. Skrba, L. Revéret, F. Hétroy, M.P. Cani, C. O'Sullivan. *Quadruped animation*. Eurographics State-of-the-Art Report, Apr., 2008, Hersonissos, Creete, Greece, pp. 1–17.

4. L. Lu, F. Hétroy, C. Gérot, B. Thibert. *Atlas-Based Character Skinning with Automatic Mesh Decomposition*. Inria Research Report, RR-6406, 2008.

5. L. Skrba, L. Revéret, F. Hétroy, M.P. Cani, C. O'Sullivan. *Animating Quadrupeds: Methods and Applications*. Computer Graphics Forum, 28(6), pp. 1541–1560, Blackwell Publishing, 2009. doi:10.1111/j.1467-8659.2008.01312.x.

6. F. Hétroy, C. Gérot, L. Lu, B. Thibert. *Simple flexible skinning based on manifold modeling*. International Conference on Computer Graphics Theory and Applications (GRAPP), Feb. 5, 2009, Lisbon, Portugal.

7. R. Arcila, F. Hétroy, F. Dupont. *Etat de l'art des méthodes de segmentation de séquences de maillages et proposition d'une classification*. COdage et REprésentation des Signaux Audiovisuels, CORESA'09, Mar., 2009, Toulouse, France.

8. J.-C. Léon, F. Hétroy. L. de Floriani. *Propriétés topologiques pour la modélisation géométrique de domaines d'études comportant des singularités non-variétés*. Congrès Français de Mécanique, Aug., 2009, Marseille, France.

9. R. Arcila, K. Buddha, F. Hétroy, F. Denis, F. Dupont. *A Framework for motion-based mesh sequence segmentation*. International Conference on Computer Graphics, Visualization and Computer Vision, WSCG, Feb., 2010, Plzen, Czech Republic.

10. R. Arcila, C. Cagniart, F. Hétroy, E. Boyer, F. Dupont. *Temporally coherent mesh sequence segmentations*. Inria Research Report, RR-7856, 2012.

11. F. Hétroy. *A discrete 3D+t Laplacian framework for mesh animation processing*. Inria Research Report, RR-8003, 2012.

12. R. Arcila, C. Cagniart, F. Hétroy, E. Boyer, F. Dupont. *Segmentation of temporal mesh sequences into rigidly moving components*, Graphical Models, 75(1), Elsevier, 2013. doi:10.1016/j.gmod.2012.10.004.

B.3 UNDERSTANDING DIGITAL SHAPES FROM THE LIFE SCIENCES

1. S. Hassan, F. Hétroy, O. Palombi. *Segmentation de maillage guidée par une ontologie*. 22èmes Journées de l'Association Française d'Informatique Graphique (AFIG), Nov. 24, 2009, Arles, France.

2. S. Hassan, F. Hétroy, O. Palombi. *Ontology-guided mesh segmentation*. FOCUS K3D Conference on Semantic 3D Media and Content, Feb. 11, 2010, Sophia Antipolis, France.

3. S. Hassan, F. Hétroy, F. Faure, O. Palombi. *Automatic localization and quantification of intracranial aneurysms*. Lecture Notes in Computer Science 6854, pp. 554–562, Springer. Proceedings of the 14th International Conference on Computer Analysis of Images and Patterns, Aug. 2011, Seville, Spain. doi:10.1007/978-3-642-23672-3_67.

4. D. Boltcheva, E. Casella, R. Cumont, F. Hétroy. *A spectral clustering approach of vegetation components for describing plant topology and geometry from terrestrial waveform LiDAR data*. 7th International Conference on Functional-Structural Plant Models, 2013 (poster).

5. F. Hétroy-Wheeler, E. Casella, D. Boltcheva. *Segmentation of tree seedling point clouds into elementary units*. International Journal of Remote Sensing, Taylor & Francis, 2016.

Bibliography

[ACH+13] Romain Arcila, Cédric Cagniart, Franck Hétroy, Edmond Boyer, and Florent Dupont, *Segmentation of temporal mesh sequences into rigidly moving components*, Graphical Models **75** (2013), no. 1, 10–22.

[ACK13] Marco Attene, Marcel Campen, and Leif Kobbelt, *Polygon mesh repairing: An application perspective*, ACM Computing Surveys **45** (2013), no. 2, 15:1–15:33.

[AFB15] Benjamin Allain, Jean-Sébastien Franco, and Edmond Boyer, *An efficient volumetric framework for shape tracking*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2015.

[All09] Pierre Alliez, *Variational approaches for digital geometry processing*, Habilitation à diriger des recherches, Université Nice Sophia Antipolis, 2009.

[ARAC14] Mathieu Andreux, Emanuele Rodola, Mathieu Aubry, and Daniel Cremers, *Anisotropic Laplace-Beltrami operators for shape analysis*, Computer Vision-ECCV 2014 Workshops, Springer, 2014, pp. 299–312.

[Arc11] Romain Arcila, *Mesh sequences: Classification and segmentation*, Phd thesis, Université Claude Bernard - Lyon I, 2011.

[ATC+08] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee, *Skeleton extraction by mesh contraction*, ACM Transactions on Graphics **27** (2008), no. 3, 44:1–44:10.

[AW11] Marc Alexa and Max Wardetzky, *Discrete Laplacians on general polygonal meshes*, ACM Transactions on Graphics **30** (2011), no. 4, 102:1–102:10.

[BB13] Vincent Barra and Silvia Biasotti, *3d shape retrieval using kernels on extended Reeb graphs*, Pattern Recognition **46** (2013), no. 11, 2985–2999.

[BBP+14] Sébastien Bauwens, Harm Bartholomeus, Alexandre Piboule, Kim Calders, and Philippe Lejeune, *Forest inventory with terrestrial LiDAR: what about hand-held mobile LiDAR?*, ForestSAT Conference, 2014.

[BGSF08] Silvia Biasotti, Daniela Giorgi, Michela Spagnuolo, and Bianca Falcidieno, *Reeb graphs for shape analysis and applications*, Theoretical Computer Science **392** (2008), no. 1-3, 5–22.

[BLMS14] Silvia Biasotti, Hamid Laga, Michela Mortara, and Michela Spagnuolo, *Reasoning about shape in complex datasets: Geometry, structure and semantics*, Eurographics Tutorials (Nicolas Holzschuch and Karol Myszkowski, eds.), The Eurographics Association, 2014.

[Blu67] Harry Blum, *A transformation for extracting new descriptors of shape*, Models for the Perception of Speech and Visual Form (Weiant Wathen-Dunn, ed.), MIT Press, Cambridge, 1967, pp. 362–380.

[BMSF06] Silvia Biasotti, Simone Marini, Michela Spagnuolo, and Bianca Falcidieno, *Sub-part correspondence by structural descriptors of 3d shapes*, Computer-Aided Design **38** (2006), no. 9, 1002–1019.

[BN03] Mikhail Belkin and Partha Niyogi, *Laplacian eigenmaps for dimensionality reduction and data representation*, Neural Computation **15** (2003), no. 6, 1373–1396.

[BP07] Ilya Baran and Jovan Popović, *Automatic rigging and animation of 3d characters*, ACM Transactions on Graphics **26** (2007), no. 3, 72.

[BS07] Alexander I. Bobenko and Boris A. Springborn, *A discrete Laplace-Beltrami operator for simplicial surfaces*, Discrete and Computational Geometry **38** (2007), no. 4, 740–756.

[BSW08] Mikhail Belkin, Jian Sun, and Yusu Wang, *Discrete Laplace operator on meshed surfaces*, Symposium on Computational Geometry (SoCG), ACM, 2008, pp. 278–287.

[BSW09] Mikhail Belkin, Jian Sun, and Yusu Wang, *Constructing Laplace operator from point clouds in* R^{d}, ACM-SIAM Symposium on Discrete Algorithms (SODA), 2009, pp. 1031–1040.

[BTS+14] Matthew Berger, Andrea Tagliasacchi, Lee Seversky, Pierre Alliez, Joshua Levine, Andrei Sharf, and Claudio Silva, *State of the art in surface reconstruction from point clouds*, Eurographics State of the Art Report, 2014, pp. 161–185.

[BVLD09] Halim Benhabiles, Jean-Phillipe Vandeborre, Guillaume Lavoué, and Mohamed Daoudi, *A framework for the objective evaluation of segmentation algorithms using a ground-truth of human segmented 3d-models*, IEEE International Conference on Shape Modeling and Applications (SMI), IEEE, 2009.

[BWDJ14] Derek Burrows, Chad Washington, Ralph Dacey, and Tao Ju, *Computer-assisted shape classification of middle cerebral artery aneurysms for surgical planning*, International Symposium on Biomedical Imaging (ISBI), IEEE, 2014, pp. 1311–1315.

[CBC+01] Jonathan C. Carr, Richard K. Beatson, Jon B. Cherrie, Tim J. Mitchell, Richard W. Fright, Bruce C. McCallum, and Tim R. Evans, *Reconstruction and representation of 3d objects with radial basis functions*, SIGGRAPH, ACM, 2001, pp. 67–76.

[CBI10] Cédric Cagniart, Edmond Boyer, and Slobodan Ilic, *Iterative deformable surface tracking in multi-view setups*, International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT), 2010.

[CdGDS13] Keenan Crane, Fernando de Goes, Mathieu Desbrun, and Peter Schröder, *Digital geometry processing with discrete exterior calculus*, SIGGRAPH Course Notes, ACM, 2013, pp. 7:1–7:126.

[CDMM13] Eric Casella, Mathias Disney, James Morison, and Helen McKay, *tLiDAR methodologies can overcome limitations in estimating forest canopy LAI from conventional hemispherical photograph analyses*, International Conference on Functional-Structural Plant Models (FSPM), 2013.

[CFSV04] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento, *A subgraph isomorphism algorithm for matching large graphs*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) **26** (2004), no. 10, 1367–1372.

[CGF09] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser, *A benchmark for 3d mesh segmentation*, ACM Transactions on Graphics **28** (2009), no. 3, 73.

[COT13] Gianluca Calcagni, Daniele Oriti, and Johannes Thürigen, *Laplacians on discrete and quantum geometries*, Classical and Quantum Gravity **30** (2013), no. 12, 125006.

[CS82] John H. Conway and Neil J. A. Sloane, *Voronoi regions of lattices, second moments of polytopes, and quantization*, IEEE Transactions on Information Theory **28** (1982), no. 2, 211–226.

[CSM07] Nicu D. Cornea, Deborah Silver, and Patrick Min, *Curve-skeleton properties, applications, and algorithms*, IEEE Transactions on Visualization and Computer Graphics **13** (2007), no. 3, 530–548.

[Dal93] Scott Daly, *Digital images and human vision*, MIT Press, 1993, pp. 179–206.

[DCF11] Mathieu Dassot, Thiéry Constant, and Meriem Fournier, *The use of terrestrial Lidar technology in forest science: Application fields, benefits and challenges*, Annals of Forest Science **68** (2011), no. 5, 959–974.

[DHLM05] Mathieu Desbrun, Anil N. Hirani, Melvin Leok, and Jerrold E. Marsden, *Discrete exterior calculus*, arXiv:math/0508341 (2005).

[Dij59] Edsger W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik **1** (1959), 269–271.

[DMMT+07] Fabien Dellas, Laurent Moccozet, Nadia Magnenat-Thalmann, Michela Mortara, Giuseppe Patanè, Michela Spagnuolo, and Bianca Falcidieno, *Knowledge-based extraction of control skeletons for animation*, IEEE International Conference on Shape Modeling and Applications (SMI), IEEE, 2007, pp. 51–60.

[DS06] Tamal K. Dey and Jian Sun, *Defining and computing curve-skeletons with medial geodesic function*, Eurographics Symposium on Geometry Processing (SGP), The Eurographics Association, 2006, pp. 143–152.

[EBV05] Jordi Esteve, Pere Brunet, and Alvar Vinacua, *Approximation of a variable density cloud of points by shrinking a discrete membrane*, Computer Graphics Forum **24** (2005), no. 4, 791–807.

[FB09] Jean-Sébastien Franco and Edmond Boyer, *Efficient polyhedral modeling from silhouettes*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) **31** (2009), no. 3, 414–427.

[FB11] Jean-Sébastien Franco and Edmond Boyer, *Learning temporally consistent rigidities*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2011, pp. 1241–1248.

[FDCO03] Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or, *Bilateral mesh denoising*, ACM Transactions on Graphics **22** (2003), no. 3, 950–953.

[FLA+05] Laure France, Julien Lenoir, Alexis Angelidis, Philippe Meseure, Marie-Paule Cani, François Faure, and Christophe Chaillou, *A layered model of a virtual human intestine for surgery simulation*, Medical Images Analysis **9** (2005), no. 2, 123–132.

[Fri10] Karl Friston, *The free-energy principle: a unified brain theory?*, Nature Reviews Neuroscience **11** (2010), no. 2, 127–138.

[GCS99] Christophe Godin, Evelyne Costes, and Hervé Sinoquet, *A method for describing plant architecture which integrates topology and geometry*, Annals of Botany **84** (1999), no. 3, 343–357.

[Ger79] Allen Gersho, *Asymptotically optimal block quantization*, IEEE Transactions on Information Theory **25** (1979), 373–380.

[GG13] Valeria Garro and Andrea Giachetti, *A tracking approach for the skeletonization of tubular parts of 3d shapes*, International Workshop on Vision, Modeling and Visualization (VMV), The Eurographics Association, 2013, pp. 73–80.

[Gli07] David Glickenstein, *A monotonicity property for weighted Delaunay triangulations.*, Discrete and Computational Geometry **38** (2007), no. 4, 651–664.

[GP10] Leo Grady and Jonathan R. Polimeni, *Discrete calculus: Applied analysis on graphs for computational science*, Springer, 2010.

[GTLH01] André Guéziec, Gabriel Taubin, Francis Lazarus, and William Horn, *Cutting and stitching: Converting sets of polygons to manifold surfaces*, IEEE Transactions on Visualization and Computer Graphics **7** (2001), no. 2, 136–151.

[H05] Franck Hétroy, *Constriction computation using surface curvature*, Eurographics (short paper) (J. Dingliana and F. Ganovelli, eds.), 2005, pp. 1–4.

[Has11] Sahar Hassan, *Integration of anatomic a priori knowledge into geometric models*, Phd thesis, Université de Grenoble, 2011.

[HBNI14] Chun-Hao Huang, Edmond Boyer, Nassir Navab, and Slobodan Ilic, *Human shape and pose tracking using keyframes*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2014, pp. 3446–3453.

[HF07] Annie Hui and Leila De Floriani, *A two-level topological decomposition for non-manifold simplicial shapes*, ACM Symposium on Solid and Physical Modeling (SPM), 2007, pp. 355–360.

[HHP10] Sahar Hassan, Franck Hétroy, and Olivier Palombi, *Ontology-guided mesh segmentation*, FOCUS K3D Conference on Semantic 3D Media and Content, FOCUS K3D, 2010, p. 5.

[Hir03] Anil Hirani, *Discrete exterior calculus*, Phd thesis, Caltech, 2003.

[HK92] Martial Hebert and Eric Krotkov, *3-d measurements from imaging laser radars: How good are they?*, International Journal of Image and Vision Computing **10** (1992), no. 3, 170–178.

[Hor87] Berthold Horn, *Closed-form solution of absolute orientation using unit quaternions*, Journal of the Optical Society of America A **4** (1987), no. 4, 629–642.

[HSKC12] Teemu Hakala, Juha Suomalainen, Sanna Kaasalainen, and Yuwei Chen, *Full waveform hyperspectral LiDAR for terrestrial laser scanning*, Optics Express **20** (2012), no. 7, 7119–7127.

[HSKK01] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii, *Topology matching for fully automatic similarity estimation of 3d shapes*, SIGGRAPH, ACM, 2001, pp. 203–212.

[HTRS10] Nils Hasler, Thorsten Thormählen, Bodo Rosenhahn, and Hans-Peter Seidel, *Learning skeletons for shape and pose*, ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D), ACM, 2010, pp. 23–30.

[HWCO+13] Hui Huang, Shihao Wu, Daniel Cohen-Or, Minglun Gong, Hao Zhang, Guiqing Li, and Baoquan Chen, *L1-medial skeleton of point cloud*, ACM Transactions on Graphics **32** (2013), 65:1–65:8.

[HXS09] Ying He, Xian Xiao, and Hock-Soon Seah, *Harmonic 1-form based skeleton extraction from examples*, Graphical Models **71** (2009), no. 2, 49–62.

[HZvK+15] Ruizhen Hu, Chenyang Zhu, Oliver van Kaick, Ligang Liu, Ariel Shamir, and Hao Zhang, *Interaction context (ICON): Towards a geometric functionality descriptor*, ACM Transactions on Graphics (2015).

[JXC+13] Wei Jiang, Kai Xu, Zhi-Quan Cheng, Ralph R. Martin, and Gang Dang, *Curve skeleton extraction by coupled graph contraction and surface clustering*, Graphical Models **75** (2013), no. 3, 137–148.

[KBAW11] Björn Krüger, Jan Baumann, Mohammad Abdallah, and Andreas Weber, *A study on perceptual similarity of human motions*, Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS) (Jan Bender, Kenny Erleben, and Eric Galin, eds.), The Eurographics Association, 2011, pp. 65–72.

[KBH06] Michael Kazhdan, Mathtew Bolitho, and Hugues Hoppe, *Poisson surface reconstruction*, Eurographics Symposium on Geometry Processing (SGP), The Eurographics Association, 2006, pp. 61–70.

[KBH12] Martin Klaudiny, Chris Budd, and Adrian Hilton, *Towards optimal non-rigid surface tracking*, European Conference on Computer Vision (ECCV), 2012, pp. 743–756.

[KCZO07] Ladislav Kavan, Steven Collins, Jiri Zara, and Carol O'Sullivan, *Skinning with dual quaternions*, Symposium on Interactive 3D Graphics and Games (I3D), 2007, pp. 39–46.

[KT03] Sagi Katz and Ayellet Tal, *Hierarchical mesh decomposition using fuzzy clustering and cuts*, ACM Transactions on Graphics **22** (2003), no. 3, 954–961.

[LÓ6] Bruno Lévy, *Laplace-Beltrami eigenfunctions: Towards an algorithm that "understands" geometry*, IEEE International Conference on Shape Modeling and Applications (SMI), IEEE, 2006, p. 13.

[LÓ8] _____, *Géométrie numérique*, Habilitation à diriger des recherches, Institut National Polytechnique de Lorraine - INPL, 2008.

[LABFL09] David Lesage, Elsa D. Angelini, Isabelle Bloch, and Gareth Funka-Lea, *A review of 3d vessel lumen segmentation techniques: Models, features and extraction schemes*, Medical Image Analysis **13** (2009), no. 6, 819–845.

[LB12] Antoine Letouzey and Edmond Boyer, *Progressive shape models*, IEEE Conference on Computer Vision and Patern Recognition (CVPR), IEEE, 2012, pp. 190–197.

[LC87] William E. Lorensen and Harvey E. Cline, *Marching Cubes: A high resolution 3d surface construction algorithm*, SIGGRAPH **21** (1987), no. 4, 163–169.

[LCF00] J. P. Lewis, Matt Cordner, and Nickson Fong, *Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation*, SIGGRAPH, ACM, 2000, pp. 165–172.

[LCLJ10] Lu Liu, Erin W. Chambers, David Letscher, and Tao Ju, *A simple and robust thinning algorithm on cell complexes*, Computer Graphics Forum **29** (2010), no. 7, 2253–2260.

[LcWcM+03] Pin-Chou Liu, Fu che Wu, Wan chun Ma, Rung huei Liang, and Ming Ouhyoung, *Automatic animation skeleton construction using repulsive force field*, Pacific Graphics, IEEE, 2003, pp. 409–413.

[LDFH09] Jean-Claude Léon, Leila De Floriani, and Franck Hétroy, *Classification of non-manifold singularities from transformations of 2-manifolds*, IEEE International Conference on Shape Modeling and Applications (SMI), IEEE, 2009, pp. 179–184.

[LF80] Gordon E. Legge and John M. Foley, *Contrast masking in human vision*, Journal of the Optical Society of America **70** (1980), no. 12, 1458–1471.

[LGS12] Marco Livesu, Fabio Guggeri, and Riccardo Scateni, *Reconstructing the curve-skeletons of 3d shapes using the visual hull*, IEEE Transactions on Visualization and Computer Graphics **18** (2012), no. 11, 1891–1901.

[LKA06] Jyh-Ming Lien, John Keyser, and Nancy M. Amato, *Simultaneous shape decomposition and skeletonization*, ACM Symposium on Solid and Physical Modeling (SPM), ACM, 2006, pp. 219–228.

[LL10] Bruno Lévy and Yang Liu, *Lp centroidal Voronoi tessellation and its applications*, ACM Transactions on Graphics **29** (2010), no. 4, 119:1–119:11.

[LPG12] Yang Liu, Balakrishnan Prabhakaran, and Xiaohu Guo, *Point-based manifold harmonics*, IEEE Transactions on Visualization and Computer Graphics **18** (2012), no. 10, 1693–1703.

[LSPW12] Lin Lu, Feng Sun, Hao Pan, and Wenping Wang, *Global optimization of centroidal Voronoi tessellation with Monte Carlo approach*, IEEE Transactions on Visualization and Computer Graphics **18** (2012), 1880–1890.

[LTBZ13] Zhenbao Liu, Sicong Tang, Shuhui Bu, and Hao Zhang, *New evaluation metrics for mesh segmentation*, Computers & Graphics **37** (2013), no. 6, 553–564.

[LV99] Francis Lazarus and Anne Verroust, *Level set diagrams of polyhedral objects*, ACM Symposium on Solid Modeling and Applications (SMA), ACM, 1999, pp. 130–140.

[LYO+10] Yotam Livny, Feilong Yan, Matt Olson, Baoquan Chen, Hao Zhang, and Jihad El-Sana, *Automatic reconstruction of tree skeletal structures from point clouds*, ACM Transactions on Graphics **29** (2010), no. 6.

[LZ09] Bruno Lévy and Hao Zhang, *Spectral mesh processing*, SIGGRAPH Asia Course Notes, ACM, 2009.

[LZ11] _____, *Elements of geometry processing*, SIGGRAPH Asia Course Notes, ACM, 2011, pp. 5:1–5:48.

[MDSB03] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H. Barr, *Discrete differential geometry operators for triangulated 2-manifolds*, Visualization and mathematics III, Springer, 2003, pp. 35–57.

[Meu92] G. Meurant, *A review on the inverse of symmetric tridiagonal and block tridiagonal matrices*, SIAM Journal on Matrix Analysis and Applications **13** (1992), no. 3.

[MLS94] Richard M. Murray, Zexiang Li, and S. Shankar Sastry, *A mathematical introduction to robotic manipulation*, CRC Press, 1994.

[Mun99] James R. Munkres, *Algebraic topology*, Prentice Hall, 1999.

[NCG13] Thibaut Le Naour, Nicolas Courty, and Sylvie Gibet, *Spatiotemporal coupling with the 3d+t motion laplacian*, Computer Animation and Virtual Worlds **24** (2013), 419–428.

[NWH13] Andreas Nüchter, Thomas Wiemann, and Hamid Reza Houshiar, *Large-scale 3d point cloud processing tutorial*, International Conference on Advanced Robotics, 2013.

[OBA+03] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel, *Multi-level partition of unity implicits*, ACM Transactions on Graphics **22** (2003), no. 3, 463–470.

[OVSP13] Ahlem Othmani, Lew-Fock-Chong Lew Yan Voon, Christophe Stolz, and Alexandre Piboule, *Single tree species classification from terrestrial laser scanning data for forest inventory*, Pattern Recognition Letters **34** (2013), 2144–2150.

[PP93] Ulrich Pinkall and Konrad Polthier, *Computing discrete minimal surfaces and their conjugates*, Experimental Mathematics **2** (1993), no. 1, 15–36.

[PPH+13] Fabiano Petronetto, Afonso Paiva, Elias S. Helou, D. E. Stewart, and Luis Gustavo Nonato, *Mesh-free discrete Laplace-Beltrami operator*, Computer Graphics Forum **32** (2013), no. 6, 214–226.

[PSB+12] Anthony Paproki, Xavier Sirault, Scott Berry, Robert Furbank, and Jurgen Fripp, *A novel mesh processing based technique for 3d plant analysis*, BMC Plant Biology **12** (2012), no. 1.

[QH07] Huaijun Qiu and Edwin R. Hancock, *Clustering and embedding using commute times*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) **29** (2007), no. 11, 1873–1890.

[RBG+09] Martin Reuter, Silvia Biasotti, Daniela Giorgi, Giuseppe Patanè, and Michela Spagnuolo, *Discrete Laplace-Beltrami operators for shape analysis and segmentation*, Computers & Graphics **33** (2009), no. 3, 381–390.

[RC99] Jarek Rossignac and David E. Cardoze, *Matchmaker: Manifold BReps for non-manifold r-sets*, Symposium on Solid Modeling and Applications (SMA), 1999, pp. 31–41.

[RC11] Radu Bogdan Rusu and Steve Cousins, *3d is here: Point Cloud Library (PCL)*, IEEE International Conference on Robotics and Automation (ICRA), 2011.

[Ree46] Georges Reeb, *Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique*, Comptes-Rendus de l'Académie des Sciences **222** (1946), 847–849.

[Rev14] Lionel Reveret, *Measurements and models for motion capture*, Habilitation à diriger des recherches, Institut Polytechnique de Grenoble - Grenoble INP, 2014.

[RKr+13] Pasi Raumonen, Mikko Kaasalainen, Markku Åkerblom, Sanna Kaasalainen, Harri Kaartinen, Mikko Vastaranta, Markus Holopainen, Mathias Disney, and Philip Lewis, *Fast automatic precision tree models from terrestrial laser scanner data*, Remote Sensing **5** (2013), no. 2, 491–520.

[rRKC15] Markku Åkerblom, Pasi Raumonen, Mikko Kaasalainen, and Eric Casella, *Analysis of geometric primitives in quantitative structure models of tree stems*, Remote Sensing **7** (2015), no. 4, 4581–4603.

[RWP06] Martin Reuter, Franz-Erich Wolter, and Niklas Peinecke, *Laplace-Beltrami spectra as 'shape-DNA' of surfaces and solids*, Computer-Aided Design **38** (2006), no. 4, 342–366.

[SA07] Olga Sorkine and Marc Alexa, *As-rigid-as-possible surface modeling*, Eurographics Symposium on Geometry Processing (SGP), The Eurographics Association, 2007, pp. 109–116.

[Sal06] Davod Khojasteh Salkuyeh, *Comments on "A note on a three-term recurrence for a triadiagonal matrix"*, Applied Mathematics and Computation **176** (2006).

[Ser94] Francis Sergeraert, *The computability problem in algebraic topology*, Advances in Mathematics **104** (1994), 139–155.

[SFR+12] Joël Schaerer, Aurora Fassi, Marco Riboldi, Pietro Cerveri, Guido Baroni, and David Sarrut, *Multi-dimensional respiratory motion tracking from markerless optical surface imaging based on deformable mesh registration*, Physics in Medicine and Biology **57** (2012), no. 2, 357.

[SH07] Jonathan Starck and Adrian Hilton, *Surface capture for performance-based animation*, IEEE Computer Graphics and Applications **27** (2007), no. 3, 21–31.

[Sha08] Ariel Shamir, *A survey on mesh segmentation techniques*, Computer Graphics Forum **27** (2008), no. 6, 1539–1556.

[SM00] Jianbo Shi and Jitandra Malik, *Normalized cuts and image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) **22** (2000), no. 8.

[Sor06] Olga Sorkine, *Differential representations for mesh processing*, Computer Graphics Forum **25** (2006), no. 4, 789–807.

[SP08] Kaleem Siddiqi and Stephen Pizer, *Medial representations: Mathematics, algorithms and applications*, 1st ed., Springer, 2008.

[SRH+09] Ljiljana Skrba, Lionel Reveret, Franck Hétroy, Marie-Paule Cani, and Carol O'Sullivan, *Animating quadrupeds: Methods and applications*, Computer Graphics Forum (2009), 1541–1560.

[SY07] Scott Schaefer and Can Yuksel, *Example-based skeleton extraction*, Eurographics Symposium on Geometry Processing (SGP), The Eurographics Association, 2007, pp. 153–162.

[Tag13] Andrea Tagliasacchi, *Skeletal representations and applications*, CoRR **abs/1301.6809** (2013).

[TAOZ12] Andrea Tagliasacchi, Ibraheem Alhashim, Matt Olson, and Hao Zhang, *Mean curvature skeletons*, Computer Graphics Forum **31** (2012), no. 5, 1735–1744.

[Tau95] Gabriel Taubin, *A signal processing approach to fair surface design*, SIGGRAPH, ACM, 1995, pp. 351–358.

[TCH13] Margara Tejera, Dan Casas, and Adrian Hilton, *Animation control of surface motion capture*, IEEE Transactions on Cybernetics **43** (2013), no. 6, 1532–1545.

[TM09] Tony Tung and Takashi Matsuyama, *Topology dictionary with Markov model for 3d video content-based skimming and description*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2009, pp. 469–476.

[TPT15] Panagiotis Theologou, Ioannis Pratikakis, and Theoharis Theoharis, *A comprehensive overview of methodologies and performance evaluation frameworks in 3d mesh segmentation*, Computer Vision and Image Understanding (2015).

[TS05] Tony Tung and Francis Schmitt, *The augmented multiresolution reeb graph approach for content-based retrieval of 3d shapes*, International Journal of Shape Modeling **11** (2005), no. 1, 91–120.

[TVD06] Julien Tierny, Jean-Philippe Vandeborre, and Mohamed Daoudi, *3d mesh skeleton extraction using topological and geometrical analyses*, Pacific Graphics, 2006.

[TVD09] _____, *Partial 3d shape retrieval by Reeb pattern unfolding*, Computer Graphics Forum **28** (2009), no. 1, 41–55.

[TZCO09] Andrea Tagliasacchi, Hao Zhang, and Daniel Cohen-Or, *Curve skeleton extraction from incomplete point cloud*, ACM Transactions on Graphics **28** (2009), no. 3, Article 71, 9 pages.

[vL07] Ulrike von Luxburg, *A tutorial on spectral clustering*, Statistics and Computing **17** (2007), no. 4, 395–416.

[VL08] Bruno Vallet and Bruno Lévy, *Spectral geometry processing with manifold harmonics*, Computer Graphics Forum **27** (2008), no. 2, 251–260.

[Wan95] Brian A. Wandell, *Foundations of vision*, Sinauer Associates, Inc., 1995.

[Wat97] Andrew B. Watson, *Image quality and entropy masking*, SPIE Conference on Human Vision, Visual Processing, and Digital Display VIII, vol. SPIE 3016, 1997, pp. 2–12.

[WMKG07] Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun, *Discrete Laplace operators: No free lunch*, Symposium on Geometry Processing (SGP), The Eurographics Association, 2007, pp. 33–37.

[WML+06] Fu-Che Wu, Wan-Chun Ma, Rung-Huei Liang, Bing-Yu Chen, and Ming Ouhyoung, *Domain connected graph: The skeleton of a closed 3d shape for animation*, The Visual Computer **22** (2006), no. 2, 117–135.

[WSK+15] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao, *3d ShapeNets: A deep representation for volumetric shape modeling*, IEEE Conference on Computer Vision and Patern Recognition (CVPR), 2015.

[XGC07] Hui Xu, Nathan Gossett, and Baoquan Chen, *Knowledge and heuristic-based modeling of laser-scanned trees*, ACM Transactions on Graphics **26** (2007), no. 4.

[XZY+07] Weiwei Xu, Kun Zhou, Yizhou Yu, Qifeng Tan, Qunsheng Peng, and Baining Guo, *Gradient domain editing of deforming mesh sequences*, ACM Transactions on Graphics **26** (2007), no. 3.

[YXF14] Long Yang, Chunxia Xiao, and Jun Fang, *Multi-scale geometric detail enhancement for time-varying surfaces*, Graphical Models **76** (2014), no. 5, 413–425.

[YZH+05] Yan Yang, Lei Zhu, Steven Haker, Allen Tannenbaum, and Don P. Giddens, *Harmonic skeleton guided evaluation of stenoses in human coronary arteries*, International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2005, pp. 490–497.

[ZST+10] Qian Zheng, Andrei Sharf, Andrea Tagliasacchi, Baoquan Chen, Hao Zhang, Alla Sheffer, and Daniel Cohen-Or, *Consensus skeleton for non-rigid space-time registration*, Computer Graphics Forum **29** (2010), no. 2, 635–644.

[ZvKD10] Hao Zhang, Oliver van Kaick, and Ramsay Dyer, *Spectral mesh processing*, Computer Graphics Forum **29** (2010), no. 6, 1865–1894.

Index

3D+t, 28

Aneurysm, 51
Animation skeleton, 32, 33

Betti number, 15
Boundary operator, 14

Centreline, 51
Centroidal Voronoi tessellation, 23
Chain, 14
Chain complex, 15
Closing, 12
Coherent segmentation, 43
Commute-time distance, 57
Cone of a morphism, 16
Constructive homology, 15
Contrast masking, 19
Contrast sensitivity function, 19
CW-complex, 36

Digital geometry processing, 2
Digital shape understanding, 2
Dijkstra's algorithm, 52
Dimension reduction, 56
Discrete exterior calculus, 36
Discrete membrane, 12

Generator, 15

Homology, 15
Homology group, 15

Incidence matrix, 15

Laplace operator, 35, 38
Laplacian Eigenmaps, 57
Laplacian matrix, 39
LiDAR, 55
Local contrast, 19

Manifold, 11
Manifold-Connected decomposition, 16
Mesh, 9
Mesh segmentation, 2
Mesh sequence segmentation, 43
Mesh singularity, 11
Mixed pixel, 59
Morphological operator, 12

Opening, 12

Reduction, 16
Reeb graph, 31, 32
Regularity, 24

Segmentation evaluation, 46
Semantics, 1
Simplex, 14
Simplicial complex, 14
Skeleton computation, 3, 30

Smith normal form, 15
Spatial frequency, 19
Spectral embedding, 57

Temporal segmentation, 43
Temporally coherent mesh sequence, 29, 36
Temporally incoherent mesh sequence, 29
Terrestrial laser scanning, 54
Topology, 12
Torsion coefficient, 15
Tree, 55

Variable segmentation, 43
Visual perception, 17