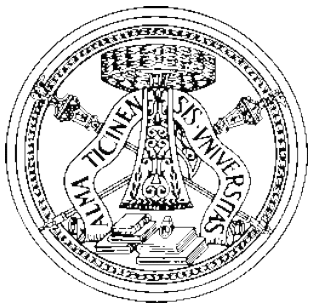
UNIVERSITÀ DEGLI STUDI DI PAVIA

Department of Industrial and Information Engineering



A Discrete Approach to Reeb Graph Computation and Surface Mesh Segmentation: Theory and Algorithm

Advisors
Prof. Marco Piastra
Prof. Virginio Cantoni

Ph.D. dissertation by
Laura Brandolini

Dottorato di Ricerca in Ingegneria Elettronica, Informatica ed Elettrica
XXIV Ciclo (2008–2011)

Contents

1 Introduction **3**

2 Theory **7**

2.1 Reeb graph in the smooth domain 7

2.1.1 Betti Numbers 8

2.1.2 Manifold 8

2.1.3 Morse theory in the smooth settings 15

2.1.4 Reeb graph 18

2.2 Discrete domain 22

2.2.1 Simplicial complexes 23

2.2.2 Triangulated manifolds 25

2.2.3 Morse theory in the discrete setting 27

2.2.4 Reeb graphs of a PL-Morse function 28

2.2.5 Contour strip Reeb graphs 33

2.2.6 Simplified Reeb graphs 47

2.3 Segmentation 57

2.3.1 Segmenting with SRG 59

3 Related Techniques **61**

3.1 Reeb Graphs 61

3.1.1 Level Set Diagrams 62

3.1.2 Extended Reeb graphs 63
3.1.3 Sweep algorithm for extracting Reeb graphs of 2-manifold 64
3.1.4 On-line computation of Reeb graphs 66
3.1.5 Enhanced topological skeletons 67
3.1.6 Reeb graphs based on shape diameter function . . . 69
3.1.7 Dynamic graphs 69
3.1.8 Reeb graphs built on critical loops 70
3.1.9 Other approaches 72
3.2 Mesh Segmentation 72

4 The DRGSS algorithm 77
4.1 Computing the SRG and the segmentation 78
4.1.1 The main algorithm 79
4.1.2 Advancing Contours 80
4.1.3 Merge and Split of contours 83
4.1.4 Segmentation 88
4.1.5 Constructing the Reeb Graph 90
4.1.6 Removing folds: saddle-maximum cancellation . . . 92
4.1.7 Computational complexity 95
4.2 Scalar function 95
4.3 Implementation 96
4.3.1 Implementation of the algorithm 97

5 Experimental evidence 101
5.1 Test description 101
5.1.1 Number of loops of the SRG 102
5.1.2 Scalar function 102
5.1.3 Shape Genus 107
5.1.4 Mesh density 108
5.2 Results 108

5.2.1 The role of multiplicity 109
5.2.2 Robustness to different mesh densities 110
5.2.3 The random function 110
5.2.4 Intrinsic function variants 111

6 SRG for human striatum **117**
6.1 Stating the problem 117
6.1.1 Automatic inter-subject mesh registration 119
6.1.2 Automatic mesh decomposition 119
6.1.3 Inter-group striatal shapes comparison 119
6.2 Striatum shape processing 120
6.2.1 The dataset 120
6.2.2 Computing SRG 120
6.3 SRG-based Registration of Striatal Meshes 122
6.4 Results 123
6.4.1 SRG-based Registration of Striatal Meshes: a quantitative assessment 123
6.4.2 SRG-based Surface Decomposition: a qualitative assessment 125
6.4.3 Inter-group comparison 127
6.4.4 Stability of the SRG to Mesh Resolution 127
6.5 Conclusions 128

7 Conclusions **131**

Bibliography **135**

Acknowledgments

I want to thank all those people who guided, supported and helped me in these years.

Particularly I want to thank my two Ph.D. tutors, Marco Piastra who taught me the art of discovery and research and always encouraged me, and Virginio Cantoni who has always been a mentor in these years.

I want also to express my gratitude toward Marisa Alicanti and Flavio Ferlini, who gave me the chance to combine my work at the server farm of the University of Pavia and my work as a Ph.D. student.

A great thanks goes to my colleagues at the server farm of the University of Pavia: Daniela Barbieri, Dante Spizzi, Dario Lanterna, Massimiliano Pini, Maurizio Quoex, Nicola Corea and Stefano Tavazzani. They encouraged and sustained me every day. Thanks to Francesco Marchesi, for his valuable suggestions about the English language.

I want to thank Antonietta Pepe, and Jussi Tohka of the Tampere University of Technology for the inspiring work that we did together.

Now a really special thanks goes to my husband Dino and to my sons Luca and Matteo: they accepted, sustained and always encouraged a wife and a mother who was often engaged with her laptop.

A great thanks to all my family that made this work possible, helping me in everything I needed, particularly to my mother and my father that have been busy grandparents during vacations.

This work is also dedicated to my grandmother Carmen, that left us this summer. Your strength and your love are always with us.

Chapter 1

Introduction

Automated shape analysis is a discipline that has seen a lot of contributions in the last two decades. In this large sector, an interesting research field is the one that studies the methods and techniques to give a mesh a graph-like representation of its shape.
As described in Chap. 2, in recent years, a lot of graph-like or skeleton-like descriptors have been proposed with different aims. Two main purposes of such a representation are:

- *augmenting shape representations* (e.g. to detect thinnings and thickenings in a shape)
- *compact shape representations* (e.g. to perform shape matching).

Reeb graphs take place in this landscape both as compact shape descriptors and as descriptors that augment a shape representation. They play a leading role in different fields of computer graphics: shape matching and encoding (SKK02; SSGD03), mesh deformation (TVD08),(SY07), 3D search (HSKK01), mesh compression (BMS00), medical imaging (SKSI95; WXS06; SLK+08; PBP+12) and several other fields.

Reeb graphs describe the topology of the level sets of a function defined on a $n-$manifold. In this work we will focus on triangulated $2-$manifolds embedded in R^3.

Reeb graphs for 2-manifolds enclose important shape properties such as connectivity, genus and, when embedded in R^3, length, width and direction

in a faithful fashion.

Assuming to have a 2D shape embedded in R^3 and a scalar function f (with some regularity properties - see Chap. 2) defined on its surface, then, intuitively, the Reeb graph describes the connectivity relation between the level lines of the function f. Morse theory (Mil64) lays the foundations for the formal definition of Reeb graph, as it will be described in Chap. 2.

Unlike skeletons, which are sensitive to little surface perturbations (BMMP03; SLK+08), Reeb graphs of topologically equivalent shapes maintain their fundamental topological properties. In particular, given a closed, orientable, connected, triangulated 2−manifold and a *general* function f (a function that assigns a different value at each vertex), it has been proved that the number of loops of the Reeb graph is always equal to the genus of the surface (CMEH+03).

This work illustrates a new robust method for constructing discrete Reeb graphs for triangulated surfaces that can work with any predefined *general* function, including a random function.

The Discrete Reeb Graph and Surface Segmentation (DRGSS) algorithm here presented is a *sweep algorithm* that constructs in a single swept both the *simplified* Reeb graph (SRG) and the segmentation of the input triangulated surface. The produced surface segmentation is topologically *correct*, in a sense that will be described in Chapter 2.
The *sweep* process is made with contours, made up of edges and vertices, that are initialized at each minimum of the function f and are then evolved in the direction of ascending values of f. Split and merge events occur for contours each time a saddle is met, until they have reached a maximum.
In one of its main elements of novelty, with respect to the existing approaches, the DRGSS algorithm maintains a correct 1-skeleton description of the topology of contours by allowing their edges and vertices to have *multiplicity* greater than 1. This leads to the building of a Reeb graph that is guaranteed to be correct.

The algorithm deals successfully even with surfaces having higher *genus* (experiments have been made with surfaces with a genus up to 22) and being coarse, that is, with a relatively low density of triangles.

At the best of the information available, this is the only algorithm that constructs simultaneously the Reeb graph and the corresponding segmenta-

tion that has been reportedly validated with the random function. Indeed (PSBM07) builds the Reeb graph on the random function but does not compute a segmentation.
As another element of novelty, the DRGSS algorithm produces a segmentation that adapts to mesh density. Indeed, in DRGSS algorithm it is possible for a face to be *multiple*, i.e. to have one or more vertices shared between more segments. Vertices's *multiple membership* allows segments to be always connected and topologically correct, as it will be shown in Chapter 4.

The DRGSS algorithm has been tested with different types of scalar functions f: height, intrinsic and random function. In all of these tests, the acceptance condition, for each mesh and each function, is that the number of loops of the resulting Reeb graph must be equal to the genus of the mesh. Experimental results are described in detail in Chapter 5: they show that the DRGSS algorithm is effective with real-world data and therefore suitable for practical applications.

The DRGSS algorithm has been applied, with very good results, to medical imaging (see Chap. 6). We used the simplified Reeb graph (SRG) as a shape descriptor for the human striatum (a part of the brain), to perform mesh registration in a simplified way. In this application mesh Reeb graph nodes are used in place of mesh vertices, to register meshes to each others. We show also the possibility of using SRG for striatal shape decomposition and inter-group shape comparison.

The following chapters of this dissertation are organized as described below. Chapter 2 provides a background on theoretical definitions, Chapter 3 explores the related works on the subjects of Reeb graphs extraction and segmentation. The DRGSS algorithm is described in detail in Chapter 4. Chapter 5 presents the experimental results. Chapter 6 describes an application of the algorithm in the medical imaging field. In Chapter 7 the conclusions and possible developments of this work are discussed.

Chapter 2

Theory

Contents

2.1 Reeb graph in the smooth domain 7

2.2 Discrete domain 22

2.3 Segmentation 57

This chapter will introduce the theory of the Reeb graphs in the smooth and in the discrete domain. In detail: section 2.1 introduces the Reeb graph theory in the smooth domain, section 2.2 presents the discrete domain (Sec. 2.2.1, 2.2.2) and illustrates both the discrete counterpart of the Morse theory (see Sec. 2.2.3) and a definition of the Reeb graph theory completely justified in the discrete domain (see Sec. 2.2.4). Section 2.2.5 introduces the *Contour Strip Reeb graphs* and section 2.2.6 presents the *Simplified Reeb graphs*, that are the object of this work. A brief introduction of the segmentation problem is also given in section 2.3.

2.1 Reeb graph in the smooth domain

To introduce the concept of Reeb graph in the smooth domain we need a set of definitions from the wide world of computational topology.

2.1.1 Betti Numbers

Betti numbers can be viewed as the count of the number of topological features in meshes, cell complexes, or topological spaces (BW12). The term Betti number was coined by Henri Poincaré after Enrico Betti. Betti numbers are an answer to the questions: *how many connected components*, *how many tunnels*, and *how many holes* are there in a topological space?(DE93). As an example, the first Betti number of a space counts the maximum number of cuts that can be made without dividing the space into two components (Wik12a). In (DE93; DE95) an algorithm has been proposed to compute Betti numbers for simplicial complexes.

Here we will give only the informal definitions of Betti numbers, leaving the formal ones to specific books like (War83; Hen94; Zom05) and (EH10).

Informally, the k^{th} Betti number refers to the number of unconnected k-dimensional surfaces. We are interested in Betti numbers because, if two spaces are homotopy equivalent, then all their Betti numbers are equal (Car09).

The first few Betti numbers can be informally defined as follows:

- β_0: number of connected components of a topological space
- β_1: number of two-dimensional or "circular" holes, or the number of cut that can be made without generating a new connected component
- β_2: number of three-dimensional holes or "voids"

For the spaces we are interested in[1] the sequence of Betti numbers is 0 from some points onwards. Indeed $\beta_k = 0$ for $k > d$, being d the dimension of the given space. We are interested in compact 2-manifolds embedded in R^3, so the Betti numbers we are interested in are β_0 and β_1.

2.1.2 Manifold

Here following the definitions necessary to introduce manifolds are presented. We will deal in particular with 2-manifold, even if some definitions are given in a more general sense.

[1] but also for mostly finite-dimensional spaces such as compact manifolds, finite simplicial complexes, ...

2.1.2.1 Sets

The *set* is a basic object in mathematics and its definition was given at first by Cantor in the late nineteenth century. Giving a formal definition for a set is not trivial, so, following (Rob10) here the axiomatic approach is taken: a set is a collection of objects that is well-defined so that one can say if an object *is* or *is not* included in the set. Objects belonging to the set are its *elements* or *members*. If x is an element of the set S we write $x \in S$. We can describe the set enumerating its elements $(white, yellow, red, blue)$ or using the *set-builder* notation in which we state a property valid for its elements: $S = \{x|P(x)\}$ (i.e. the set S is made of all the x for which the property $P(x)$ is true).
The only empty set is $\emptyset = \{x|x \neq x\}$.

Definition 1 (Subset). *A set A is a subset of a set B if every element of A is also an element of B. We say $A \subseteq B \Leftrightarrow (\forall x)[x \in A \Rightarrow x \in B]$.*

From this definition it follows that for any set A:

- $A \subseteq A$
- $\emptyset \subseteq A$

Given three sets A, B, C if $A \subseteq B$ and $B \subseteq C$ then $A \subseteq C$.

Definition 2 (Equality of sets). *Two sets are equal if they are identical:*

$$A = B \Leftrightarrow \{(A \subseteq B) \wedge (B \subseteq A)\}$$

Definition 3 (Power set). *If A is a set then 2^A is the collection of all subsets of A:*

$$2^A = \{B|B \subseteq A\}$$

Definition 4 (Intersection). *The intersection $A \cap B$ of sets A and B is the set consisting of all the elements belonging both to A and B.*
We write :

$$A \cap B = \{z|z \in A \text{ and } z \in B\}$$

Definition 5 (Union). *The union $A \cup B$ of sets A and B is the set consisting of all the elements belonging to A or to B.*
We write:

$$A \cup B = \{z | z \in A \text{ or } z \in B\}$$

Definition 6 (Complement). *The complement of a set $A \subset B$ is the set consisting of all the elements of B that are not elements of A.*
We write:

$$\overline{A} = \{z | z \in B \text{ and } z \notin A\}$$

To define the complement, it is important to define also the set B, in respect of which the complement is made. Different super-sets B mean different complements $\overline{A}$.

Definition 7 (Cartesian product). *The* cartesian product *or* direct product *of two sets A and B is the set consisting of all the ordered 2-tuples (a, b) having $a \in A$ and $b \in B$.*

$$A \times B = \{(a, b) | a \in A \text{ and } b \in B\}$$

Definition 8 (Finite union of sets). *We define the finite union of the sets $A_1, A_2, ..., A_i, .., A_n$ as the set of elements a that belong at least to one of the sets $A_1, ..., A_i, ..., A_n$:*

$$\bigcup_{i=1}^{n} A_i = \{a | \exists i \in [1, .., n] | a \in A_i\}$$

Definition 9 (Infinite union of sets). *We define the infinite union of the sets $A_1, A_2, ..., A_i, ..$ as the set of elements a that belong at least to one the possible sets A_i:*

$$\bigcup_{i=1}^{\infty} A_i = \{a | \exists i \in N | a \in A_i\}$$

Definition 10 (Finite intersection of sets). *We define the finite intersection of the sets $A_1, ..., A_i, ..., A_n$ as the set of elements a that belong to all the sets $A_1, ..., A_i, ..., A_n$:*

$$\bigcap_{i=1}^{n} A_i = \{a | \forall i \in [1, ..., n] | a \in A_i\}$$

Definition 11 (Infinite intersection of sets). *We define the infinite intersection of the sets $A_1, A_2, ..., A_i, ..$ as the set of elements a that belong to all the possible sets A_i:*

$$\bigcap_{i=1}^{\infty} A_i = \{a | \forall i \in N | a \in A_i\}$$

2.1.2.2 Functions

Definition 12 (Relation). *A relation f from a set X to a set Y is a rule that pairs each element x of X with one or more elements y of Y.*

Definition 13 (Function). *A function f from a set X to a set Y is a rule that pairs each element x of X with exactly one element y of Y. We write:*

$$y = f(x) \text{ or } f : X \to Y$$

X is called the function domain *and Y is called the* function codomain. *We can say that f* maps *X into Y.*

Definition 14 (Composite function). *If ϕ and ψ are two functions such that $\phi : X \to Y$ and $\psi : Y \to Z$ then $\psi(\phi(x)) = z$ is the composite function mapping X into Z. We write:*

$$\psi \circ \phi$$

Definition 15 (Injective, surjective and bijective functions). *A function from a set A to a set B is* injective *if each element in B has at most one element of A mapped into it. It is* surjective *if each element in B has at least one element of A mapped into it. A function is* bijective *if it is both* injective *and* surjective.

2.1.2.3 Topological spaces

Intuitively, *topology* is about the connectivity of a space. Following (Ede01), (Zom05) and (EH10) we will give a more formal definition.

Definition 16 (Topology). *A topology on a set $\mathbb{X}$ is a collection $T \subset 2^{\mathbb{X}}$ such that:*

1. *If $S_1, S_2 \in T$, then $S_1 \cap S_2 \in T$.*
2. *If $\{S_J | j \in J\} \subseteq T$, then $\cup_{j \in J} S_J \in T$.*
3. *$\emptyset, \mathbb{X} \in T$.*

Definition 17 (Topological spaces). *A* topological space *is $(\mathbb{X}, T)$ where $\mathbb{X}$ is a set and T is a topology.*

Definition 18 (Homeomorphism). *Given two topological spaces $\mathbb{X}$ and $\mathbb{Y}$, a function $f : \mathbb{X} \to \mathbb{Y}$ is an* homeomorphism *if it is continuous, bijective and has a continuous inverse.*

Definition 19 (Homeomorphic spaces). *Two topological spaces $\mathbb{X}$ and $\mathbb{Y}$ are* homeomorphic *if and only if exists an* homeomorphism *between the two. We write $\mathbb{X} \approx \mathbb{Y}$ meaning that $\mathbb{X}$ and $\mathbb{Y}$ are of the* same topological type.

2.1.2.4 Smooth manifold

Definition 20 (Chart). *A chart at $x \in \mathbb{X}$ is a function $\varphi : U \to \mathbb{R}^d$, where $U \subseteq \mathbb{X}$ is an open set containing x and φ is an homeomorphism onto an open subset of $\mathbb{R}^d$.*

Definition 21 (Atlas). *An atlas for a topological space $\mathbb{M}$ is a collection $\{(U_\alpha, \varphi_\alpha)\}$ of charts on $\mathbb{M}$ such that $\bigcup U_\alpha = \mathbb{M}$.*

Definition 22 (Hausdorff). *A topological space $\mathbb{M}$ is* Hausdorff *if, for every point $m, n \in \mathbb{M}$ such that $m \neq n$, there are neighborhoods O and P of m, n respectively, such that $O \cap P = \emptyset$.*

Definition 23 (Separable). *A topological space $\mathbb{X}$ is separable if it has a countable basis of neighborhoods.*

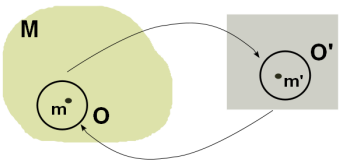


Figure 2.1 – A manifold of dimension n is a topological space that near each point resembles n-dimensional Euclidean space

Informally, a manifold of dimension n is a topological space that near each point resembles n-dimensional Euclidean space. More precisely, each point of an n-dimensional manifold has a neighborhood that is homeomorphic to the Euclidean space of dimension n.

Based on Def. 22 and 23 it is possible to give a formal definition of a manifold in the following terms:

Definition 24 (Manifold). *A separable Hausdorff space $\mathbb{M}$ is d-manifold if there is a d-dimensional chart at every point $m \in \mathbb{M}$ that has a neighborhood homeomorphic to $\mathbb{R}^d$.*

A 2-manifold is commonly called a *surface*.

Definition 25 (Half space). *The space $\mathbb{H}^d \subset \mathbb{R}^d$ such that $\mathbb{H}^d := \{h = (h_1, ...h_n) | h_i \geq 0\}$ is called the half space of $\mathbb{R}^d$.*

Definition 26 (Manifold with boundary). *A topological space M is a* d*-manifold with boundary if every element $m \in M$ has an open neighbourhood N homeomorphic either to $\mathbb{R}^d$ or to $\mathbb{H}^d$, the half space.*

Definition 27 (Manifold boundary). *The boundary ∂M of a* d*-manifold M with boundary is a* (d-1)*-manifold without boundary.*

Definition 28 (Closed Manifold). *A* d*-manifold M without boundary is called a* closed *manifold.*

Generally, a 2-manifold (without boundary) is a topological space M whose points all lie in open disks.

We are interested in *smooth*, *compact*, *orientable*, manifolds and so we need some more definitions.

Definition 29 (Covering). *A covering of* $\mathbb{Y} \subseteq \mathbb{X}$ *is a family* $\{C_j | j \in J\}$ *in* $2^{\mathbb{X}}$*, such that* $\mathbb{Y} \subseteq \bigcup_{j \in J} C_j$*.*

Definition 30 (open covering). *An* open covering *is a covering consisting of* open sets*.*

Definition 31 (subcovering). *A* subcovering *of the covering defined in Def. 29 is* $\{C_k | k \in K\}$ *with* $K \subseteq J$*.*

Definition 32 (Compact covering). *If every open covering of a covering* $\mathbb{Y} \subseteq \mathbb{X}$ *has a* finite *subcovering then the covering* $\mathbb{Y} \subseteq \mathbb{X}$ *is compact.*

Definition 33 (Compact Manifold). *A* d*-manifold* $\mathbb{M}$ *is compact if every open covering* $\mathbb{C}$ *of* $\mathbb{M}$ *contains a finite sub-collection that is also a covering of* $\mathbb{M}$*.*

We can then intuitively state that a manifold is compact if it can be covered with a finite number of charts.

The first to classify the closed surface into two categories, *orientable* and *non-orientable*, was Klein (Kle27). He used the instrument of *indicatrix*: a small oriented circle placed on a surface, then transported around an arbitrary closed curve. If it exists a curve which brings the *indicatrix* back with its orientation reversed, then the surface is *non-orientable* (see Fig. 2.2).

Definition 34 (Orientable manifold). *A manifold is* orientable *if it exists an atlas so that each pair of coordinate systems is consistently oriented (i.e. the determinant of their Jacobian is positive whenever defined).*

Let's now take $U, V \subseteq \mathbb{R}^d$.

Definition 35 (Smooth functions). *A function* $f : U \to \mathbb{R}$ *is smooth (or* C^∞*) if it has partial derivatives of all orders and types.*

Definition 36 (C^∞ maps). *A function* $f : U \to \mathbb{R}^a$ *is a* C^∞ *maps if all its components* $a^i \circ f : U \to \mathbb{R}$ *are* C^∞ *.*

Definition 37 (C^∞*-related* charts). *Two charts* $f_1 : U \to \mathbb{R}^d$ *and* $f_2 : V \to \mathbb{R}^a$ *are* C^∞-related *charts if* $a = d$ *and either* $U \cap V = \emptyset$ *or* $f_1 \circ f_2^{-1}$ *and* $f_2 \circ f_1^{-1}$ *are* C^∞ *maps.*

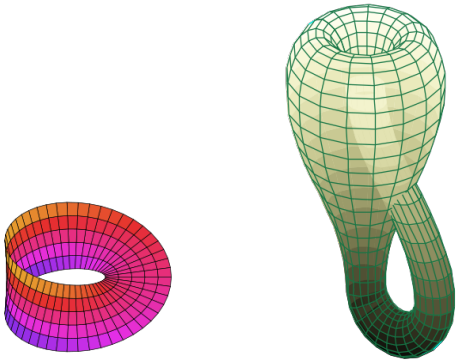


Figure 2.2 – Two examples of non-orientable 2-manifold: the Moebius strip and the Klein bottle.(Wik12d; Wik12c)

Definition 38 (C^∞*atlas*). *An* atlas *is a C^∞ atlas if every pair of its charts are* C^∞-related *and a new chart is admissible in this atlas only if it is* C^∞-related *to every pre-existing chart in the atlas.*

Definition 39 (Smooth manifold or C^∞ manifold). *A topological manifold with all the admissible charts of a* C^∞atlas *is a C^∞ manifold.*

2.1.3 Morse theory in the smooth settings

Here we assume that M is a smooth, compact, 2-manifold without boundary. We give this topological space a *metric*, embedding it in $\mathbb{R}^3$, that is $M \in \mathbb{R}^3$. The idea behind Morse theory is to analyse such a manifold studying the evolution of the level lines of a real-valued scalar function f defined on M, having only *non-degenerate critical points* in the sense defined below. Here following we define more formally what this implies in terms of the properties of the function f. In this exposition the main sources are (EH10) and (Mil63), and the fundamental ideas have been first introduced in (Mor31).

Let's consider a *smooth* scalar function f defined on a smooth, compact, closed, 2-manifold M:

$$f : M \to \mathbb{R}$$

Definition 40 (Critical point). *A point $p \in M$ is* critical *for f if:*

$$\frac{\partial f}{\partial u}(p) = \frac{\partial f}{\partial v}(p) = 0$$

where (u, v) is a local coordinate system defined on M.

Definition 41 (Hessian). *The* Hessian *of f at the point p is the matrix of second derivatives:*

$$H_f(p) := \begin{bmatrix} \frac{\partial^2 f}{\partial u^2}(p) & \frac{\partial^2 f}{\partial u \partial v}(p) \\ \frac{\partial^2 f}{\partial v \partial u}(p) & \frac{\partial^2 f}{\partial v^2}(p) \end{bmatrix}$$

Definition 42 (*Non-degenerate* critical point). *A critical point p is* non degenerate *if* $\det(H_f(p)) \neq 0$*, i.e. the matrix $H_f(p)$ is non-singular.*

Definition 43 (Morse function). *f is a* Morse function *if all its critical points are non-degenerate.*

The number of critical points of a Morse function is always *finite*.

An example of a non-Morse function in the smooth setting is the height of a torus, or of an eight shape, when they lay on the side parallel to their symmetry axis. In this case there will be an entire circle of maxima and an entire circle of minima, for the torus, and two entire circles of maxima and two entire circles of minima for the eight shape.
In this case the critical points are degenerate and non-isolated. It is worth noting that just a little perturbation could make this function a Morse one.

Definition 44 (*Simple* function). *f is* simple *if for any two critical points p_1 and p_2, $f(p_1) \neq f(p_2)$.*

2.1.3.1 Morse lemma

If f is a Morse function on M, for each critical point $p \in M$ there exists a local coordinate system (u, v) such that f takes the form:

$$f(u, v) = f(p) \pm u^2 \pm v^2 \tag{2.1}$$

This result is particularly powerful because it tells us that the neighborhood of a critical point of a Morse function cannot be more complicated than the ones shown in Fig. 2.3.

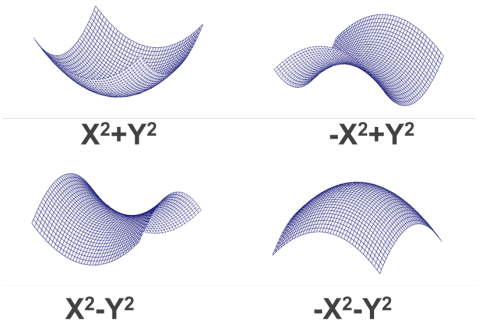


Figure 2.3 – The Morse Lemma tells us that the neighborhood of a critical point of a Morse function cannot be more complicated than the ones shown in this figure.

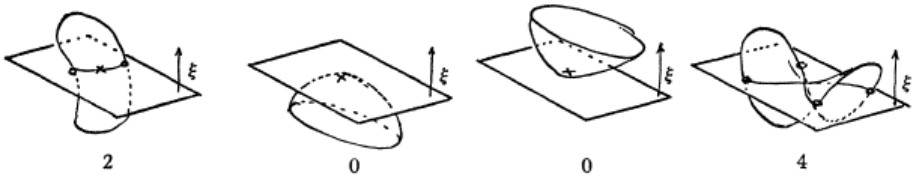


Figure 2.4 – Small circles around a point on M (from (Ban70))

2.1.3.2 Critical Point Theorem

In order to introduce the *Critical Point Theorem* we give here the Banchoff's (Ban70) definition of *index* of a critical point p[2]:

Definition 45 (Index of a critical point). $i(p) = 1$ *if* p *is critical and the two eigenvalues of* $H(p)$ *have the same sign (i.e. for* minima *and* maxima*);*
$i(p) = -1$ *if* p *is critical and the two eigenvalues of* $H(p)$ *have different signs (i.e. for* saddles*).*
$i(p) = 0$ *if* p *is* regular.

[2]In some texts the *index* of a critical point p is defined as the number of negative eigenvalues of $H(p)$, which equals the number of minuses in Eq. (2.1): a *minimum* has index 0, a *saddle* has index 1, a *maximum* has index 2.

We can also give a more *geometric* definition of the index of a critical point, imagining to define a *small circle* around each critical point:

$$i(p) = 1 - \frac{1}{2}s_c$$

where s_c is the number of intersections between the plane normal to the considered function f and the *small circle* around the critical point. The intuitive meaning of index is shown in Fig. 2.4.

Theorem 1 (Critical point theorem). *The* Euler-Poincaré characteristic *of M, $\chi(M)$, is:*

$$\chi(M) = \sum_p i(p) \tag{2.2}$$

where the sum is extended over all critical points of f in M.

Regular points do not contribute to Eq. (2.2): if x is a *regular* point, the plane normal to f passing by x will meet the *small circle* around x in exactly 2 points, making its index equal to zero.

On the other hand, a saddle having index $p < -1$ is called a *degenerate* saddle, and its *multiplicity* is equal to $-p + 1$.

Theorem 1 can also be reformulated as:

$$\chi(M) = n_1 - n_{-1} \tag{2.3}$$

where n_1 and n_{-1} are the number of critical points with the given index, indeed vertices with index 0 do not contribute in Eq. (2.2).

Based on this, Eq. (2.2) can also be rewritten as:

$$\chi(M) = m - s + M$$

where m is the number of minima, s is the number of saddles (counted with their *multiplicity*) and M is the number of maxima.

2.1.4 Reeb graph

Definition 46 (Level set). *Given a Morse function f defined on a smooth, closed, compact 2-manifold M, the* level set *of such a function f at $l \in \mathbb{R}$*

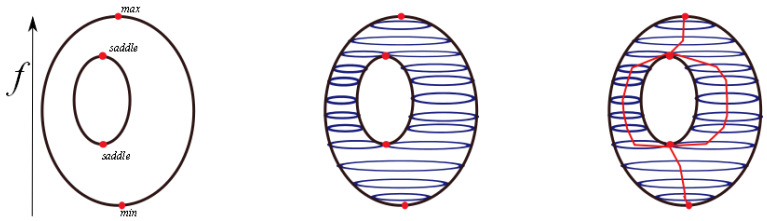


Figure 2.5 – The Reeb graph of a torus built on the height function $f = z$.

is the pre-image $f^{-1}(l)$.

$$f^{-1}(l) = \{x \in M | f(x) = l\} \tag{2.4}$$

The level sets of f form a *partition* of the manifold M.

Definition 47 (Critical and regular values of f). *The value $l \in \mathbb{R}$ of f is a* critical value *if it exists a critical point p such that $l = f(p)$. All non-critical values are deemed* regular.

If f is *simple* (see Def. 44), the correspondence between *critical values* and *critical points* is one-to-one.

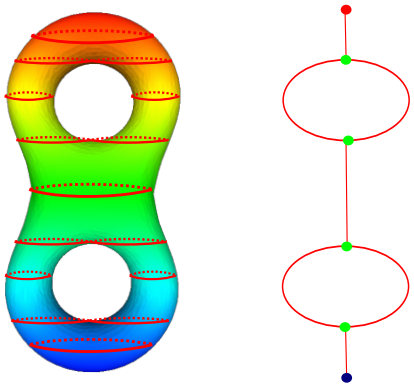
For any regular value l, $f^{-1}(l)$ is a smooth, closed 1-manifold, since at all *regular* points $p \in f^{-1}(l)$ the gradient of f is non-null. At critical points, the level set either degenerates into a point (maxima and minima) or it becomes self-intersecting (saddles) (see Fig. 2.6).

Definition 48 (Contour in the smooth setting). *Each connected component γ of a* level set *is called a* contour.

$$\gamma \subseteq f^{-1}(l)$$

Definition 49 (Reeb graph). *Given a* simple *Morse function f defined on a smooth, closed, compact 2-manifold M, the* Reeb graph $R_f(M)$ *(Ree46) is the* quotient space *defined by the equivalence relation:*

$$(x, f(x)) \sim (y, f(y)) \Leftrightarrow f(x) = f(y) = l \text{ and } \exists \gamma \subseteq f^{-1}(l), x, y \in \gamma$$



(a) (b)

Figure 2.6 – The evolution of the connected components of the level lines of the height function $f = z$ on the smooth surface of the bi-torus (a), the corresponding Reeb graph (b).

In other words, $R_f(M)$ is the space of all *contours* defined by f on M and is equipped with the *quotient topology* defined by a map $\gamma_f : M \to R_f(M)$:

$$\gamma_f(x) := \text{ The contour } \gamma \subseteq f^{-1}(f(x)) \text{ that contains } x$$

Any *open set* in M corresponds by definition to an open set in $R_f(M)$ via γ_f.

The *nodes* in the Reeb graph $R_f(M)$ of a *simple* Morse function f correspond to the contours containing one critical point p. The rest of the contours correspond to a point of the *arcs* of $R_f(M)$.

2.1.4.1 Loop lemma for Reeb graphs on 2-manifolds

Definition 50 (Loop in a graph). *A* loop *in a graph is a minimal cycle.*

The *genus* of a connected, orientable surface is an integer representing the maximum number of cuttings - along non-intersecting closed simple curves - we can do, without making the surface disconnected. It is equal to the number of handles on it (Wik12b).

Definition 51 (Genus and Euler characteristic). *The* genus g *and the* Euler characteristic χ *of any closed, orientable 2-manifold* M *are bound together by:*

$$\chi(M) = 2 - 2g(M) \tag{2.5}$$

Lemma 2 (Loop lemma). *The Reeb graph* $R_f(M)$ *of a Morse function on a connected, closed and orientable 2-manifold of genus* g *has exactly* g *loops (CMEH+03).*

Proof. For any graph, it is true that:

$$\chi(R_f(M)) = \beta_0(R_f(M)) - \beta_1(R_f(M))$$

where β_0 is the number of connected components and β_1 is the number of cycles in $R_f(M)$. It is a special case of the *Euler-Poincaré Theorem* ((EH10)), valid for any topological space T:

$$\chi(T) = \sum_{i \geq 0} (-1)^i \beta_i(T)$$

From the definition of χ:

$$V(R_f(M)) - E(R_f(M)) = \beta_0(R_f(M)) - \beta_1(R_f(M))$$

If M is connected, then also $R_f(M)$ is connected: $\beta_0(R_f(M)) = 1$. Then, using Eq. 2.3

$$V(R_f(M)) - E(R_f(M)) = 1 - \beta_1(R_f(M))$$
$$\beta_1(R_f(M)) = 1 - (n_1 + n_{-1}) + E(R_f(M))$$

The number of edges in $R_f(M)$ is:

$$E(R_f(M)) = \frac{1}{2}(n_1 + 3n_{-1})$$

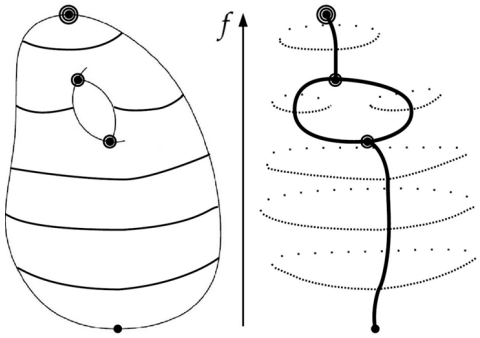


Figure 2.7 – Reeb graph for a smooth M (from (EH10))

Then:

$$\begin{aligned}\beta_1(R_f(M)) &= 1 - (n_1 + n_{-1}) + \frac{1}{2}(n_1 + 3n_{-1}) \\ &= 1 - \frac{1}{2}(n_1 - n_{-1}) \\ &= 1 - \frac{1}{2}\chi(M)\end{aligned}$$

The last step is justified by Thm. 1. Therefore, from Def. 51:

$$\begin{aligned}\beta_1(R_f(M)) &= 1 - \frac{1}{2}(2 - 2g(M)) \\ &= g(M)\end{aligned}$$

2.2 Discrete domain

We are now focused on triangulated 2-manifolds, intended as 2-simplicial complexes, as defined below. Firstly we will introduce some definitions in order to express the surface in terms of the triangulation of a manifold. Then we will illustrate how the discrete setting can show interesting properties, especially useful for our aims. Our main sources will be (Ban70), (Ede01), (Zom05), (EH10).

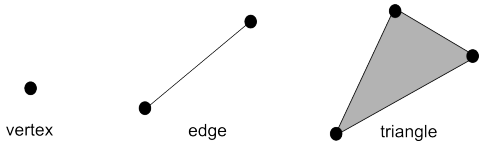


Figure 2.8 – k-simplices, with $k \in [0-2]$: a vertex is a 0-simplex, an edge is a 1-simplex, a triangle is a 2-simplex.

2.2.1 Simplicial complexes

Here following we will give some definitions, referring to embedded simplicial complexes. We will not discuss the theory of abstract simplicial complexes, leaving it to more specific texts (see e.g. (Zom05; EH10)).

Definition 52 (*0*-simplex). *A* 0*-simplex* p *is a point in* $\mathbb{R}^d$*. It is also called a* vertex.

Definition 53 (*k*-simplex). *A* k*-simplex* σ *is the* convex hull *of a set of* k+1 *affinely independent points* $S = \{p_0, ..., p_i, ..., p_{k+1}\}$ *in* $\mathbb{R}^d$*. Those points are called the* vertices *of the simplex.*

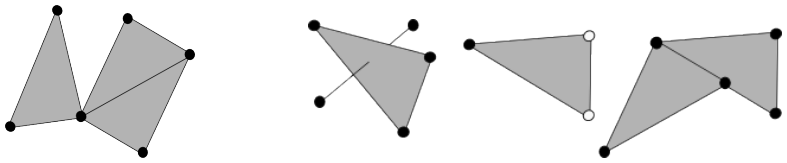
Definition 54 (Face of a simplex). *Given a* k*-simplex* σ*, we say that* τ*, defined by the set of points* $T \subseteq S$ *- being* S *the set of vertices of* σ *- is a* face *of* σ*, and we write* $\tau \preceq \sigma$*.*

Definition 55 (Number of faces of a simplex). *A* k*-simplex has* $\binom{k+1}{l+1}$ *faces of dimension* l *and* $\sum_{l=-1}^{k} \binom{k+1}{l+1} = 2^{k+1}$ *faces in total.* $2^{k+1} - 1$ *if we do not consider the empty set.*

Every simplex is a face of itself: $\sigma \preceq \sigma$. All the other faces different from σ are *proper* faces of σ. Note that a vertex is a *0*-simplex, having just itself as face, an edge is a *1*-simplex, having 2 vertices and an edge as faces, a triangle is a *2*-simplex, having 3 vertices, 3 edges and a triangle as face (see Fig 2.8).

Definition 56 (Simplicial complex). *K is a simplicial complex if it is made of a* finite *set of simplices such that:*

- $\forall \tau \preceq \sigma,\ \sigma \in K \rightarrow \tau \in K$
- $\forall \sigma_1, \sigma_2 \in K \rightarrow \tau = \sigma_1 \cap \sigma_2, \tau \preceq \sigma_1, \tau \preceq \sigma_2$



(a) *proper* (b) *improper*

Figure 2.9 – A *proper* simplicial complex (a), three examples of *improper* simplicial complexes (b).

A simplicial complex is thus a collection of faces of a finite number of simplices, any pair of which is either disjoint or meets at a common face.

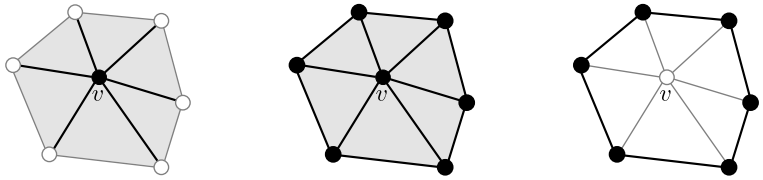
Definition 57 (Subcomplex of a simplicial complex). *A* subcomplex *L of K is a simplicial complex such that $L \subseteq K$.*

Definition 58 (j-skeleton). *The* j-skeleton *is a particular subcomplex made of all the simplices having dimension $d \leq j$.*

$$K^{(j)} = \{\sigma \in K | dim\sigma \leq j\}$$

The 0-skeleton is the subcomplex also called the *vertices set* and referred to as:

$$VertK = K^{(0)}$$



(a) St(v) (b) Cl(St(v)) (c) Lk(v)

Figure 2.10 – St(v) (a), Cl(St(v)) (b) and Lk(v) (c) of a vertex v.

Definition 59 (Closure). *A* closure *of a subset of simplices $S \subseteq K$ is the smallest subcomplex containing S.*

$$\mathrm{Cl}(S) = S \ \cup \ \{\tau : \exists \sigma \in S \mid \tau \preceq \sigma\}$$

Definition 60 (Star). *The* star *of a simplex τ is the set of all the simplices having τ as a face:*

$$\mathrm{St}(\tau) = \{\sigma \in K \mid \tau \preceq \sigma\}$$

Definition 61 (Link). *The* link *of a simplex τ is the set of all the faces of the simplices of the* star *of τ not having τ as a face:*

$$\mathrm{Lk}(\tau) = \{\sigma \in \mathrm{St}\tau | \sigma \cap \tau = \emptyset\}$$

We can also write:

$$\mathrm{Lk}(\tau) := \mathrm{Cl}(\mathrm{St}(\tau)) - \mathrm{St}(\tau)$$

2.2.2 Triangulated manifolds

From this point on we will use *simplicial complexes* for representing manifolds in the discrete setting, in a sense described by the following definitions.

Definition 62 (Underlying space). *The* underlying space $|K|$ *of a simplicial complex K is the finite union of all its simplices:*

$$|K| = \bigcup_{\sigma \in K} \sigma$$

Definition 63 (Triangulation). *The* triangulation *of a topological space $\mathbb{S}$ is a* simplicial complex K *such that its underlying space is homeomorphic to $\mathbb{S}$:*

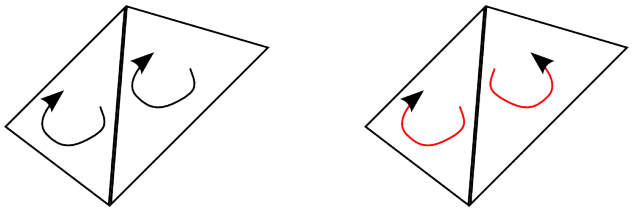
$$|K| \approx \mathbb{S}$$

We now introduce the concept of orientability in the discrete settings. It will emerge that orientability is a topological property that does not depend on the smoothness of the underlying manifold. Intuitively a mesh is orientable if all the faces in it have a consistent orientation (all clockwise or all counterclockwise).

Definition 64 (Orientation). *Let K be a simplicial complex. An orientation of a k-simplex $\sigma \in K$, $\sigma = \{v_0, v_1, ..., v_k\}, v_i \in K$, is an equivalence class of orderings of the vertices of σ, where*

$$(v_0, v_1, ..., v_k) \sim (v_{\tau(0)}, v_{\tau(1)}, ..., v_{\tau(k)})$$

are equivalent orderings if the parity of the permutation τ is even. We denote an oriented simplex, a simplex with an equivalence class of orderings, by $[\sigma]$.



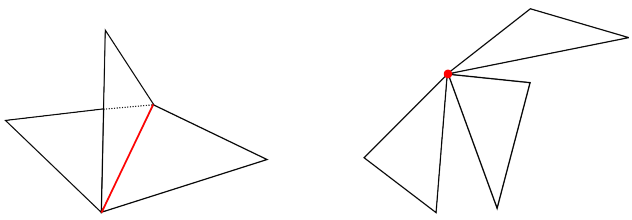
(a) *orientable* (b) *non-orientable*

Figure 2.11 – *Orientable* (a) and *non-orientable* (b) simplicial complexes.

Definition 65 (Orientability). *Two $k-$simplices sharing a* (k-1)*-face σ are consistently oriented if they induce different orientations on σ. A triangulated $d-$manifold is* orientable *if all $d-$simplices can be oriented consistently. Otherwise, the $d-$manifold is* non-orientable.

Definition 66 (Triangulated manifold with/without boundary). *A triangulated $2-$manifold is without boundary if for every edge it is incident to exactly two faces in the manifold. In a manifold with boundary the edges incident to only one face form the boundary of the manifold.*

It is also clear that on a closed triangulated 2-manifold the neighborhood of each vertex is homeomorphic to a disc.



(a) (b)

Figure 2.12 – In (a) in red, a non-manifold edge, and in (b), in red, a non-manifold vertex.

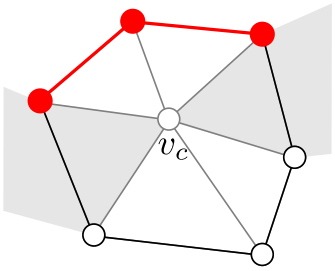


Figure 2.13 – $\mathrm{Lk}^+(v_c, f)$

2.2.3 Morse theory in the discrete setting

In order to introduce Morse theory in the discrete setting we have now to define what a *critical point* in the discrete setting is.

Let's take a function f defined only at the vertices of the triangulated surface X.

Definition 67 (The $Lk^+(v, f)$ of a vertex). *The $Lk^+(v, f)$ of a vertex v is the set of vertices in $v_i \in \mathrm{Lk}(v)$ having an higher value of the function f with respect to v, plus their connecting edges. Formally we define:*

$$\mathrm{Lk}^+(v, f) := \{v_i \in \mathrm{Lk}(v) | f(v_i) > f(v)\} \cup \{e_j \in \mathrm{Lk}(v) | f(v_j) > f(v),\ \forall v_j \preceq e_j\}$$

Fig. 2.13 shows an example of Lk^+ for a vertex v_c.

The $Lk^-(v)$ can be defined in a similar way:

Definition 68 (The $Lk^-(v, f)$ of a vertex). *The $Lk^-(v, f)$ of a vertex v is the set of vertices having a lower value of the function f with respect to v, plus their connecting edges.*

Definition 69 ($|Lk^+|$). *$|Lk^+|$ is the number of connected components of $Lk^+(v, f)$*

Definition 70 ($|Lk^-|$). *$|Lk^-|$ is the number of connected components of $Lk^-(v, f)$*

As shown in Fig. 2.14 we can count the number of connected components of the Lk^+ and the Lk^-: in this case they are respectively $|Lk^+| = 2$ and $|Lk^-| = 2$.

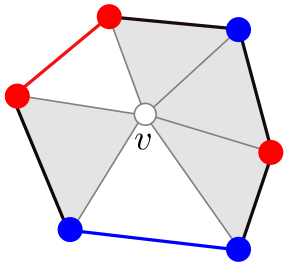


Figure 2.14 – This vertex has $|Lk^+| = 2$ and $|Lk^-| = 2$.

Definition 71 (Critical point). *A point x is defined* critical *for f if*

$$|Lk^+| = |Lk^-| > 1$$

A *regular* vertex has $|Lk^+| = |Lk^-| = 1$.

A *saddle* vertex has $|Lk^+| = |Lk^-| = 1+m$, where m is called the *multiplicity* of the saddle (see Fig. 2.14).

Following this definition *maxima* and *minima* are a kind of critical points such that:

- a *maximum* vertex has $|Lk^+| = 0$, $|Lk^-| = 1$.
- a *minimum* vertex has $|Lk^-| = 0$, $|Lk^+| = 1$.

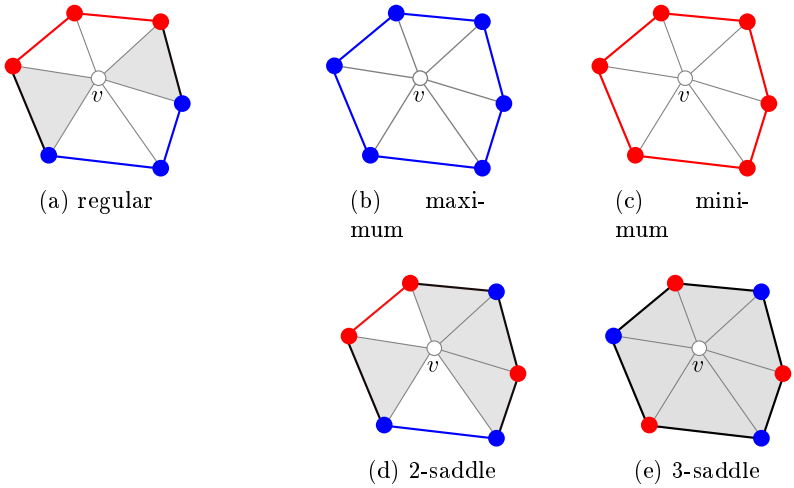
Definition 72 (Degenerate critical point). *A point x is defined a* degenerate critical point *for f if*

$$|Lk^+| = |Lk^-| = 1 + m \text{ and } m > 1$$

A *degenerate* critical point is thus a critical point with *multiplicity* greater than one. In Fig. 2.15 *regular* and *critical* (both *degenerate* and *non-degenerate*) vertices are shown.

2.2.4 Reeb graphs of a PL-Morse function

In this section we will give the definition of a piecewise-linear Morse function defined on a triangulated manifold X, and of its Reeb graph.



(a) regular (b) maximum (c) minimum (d) 2-saddle (e) 3-saddle

Figure 2.15 – A *regular* vertex (a), and four *critical* vertices (b), (c), (d) and (e).

Given a real-valued function f defined on the vertices of a triangulated surface X, we consider its extension via linear interpolation f^*. This function f^* is defined on the whole surface of X.

Definition 73 (Piecewise linear function f^*). *The function $f^* : X \to \mathbb{R}$ is the extension via linear interpolation of the real valued function $f : V \to \mathbb{R}$:*

$$f^*(p) := \sum_{v_i} \lambda_i f(v_i), \ \forall p \in \sigma \in X \tag{2.6}$$

where λ_i are the barycentric coordinates *of p in the k-simplex σ.*

Definition 74 (PL Morse function). *A function $f^* : X \to \mathbb{R}$ is a PL Morse function ifs all its* critical *points x are* non degenerate.

We now introduce the concept of the *level sets* of a PL-function f^* over a triangulated surface.

Definition 75 (Level set of f^*). *The* level set *$LS_{f^*}(l)$ of f^* at l is $f^{*-1}(l)$, the set of all the points x of the surface X having $f^*(x) = l$ (see Fig. 2.16).*

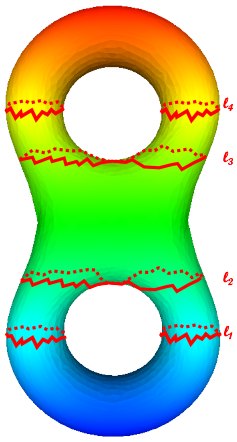


Figure 2.16 – The level sets of a function f^* for four different values of $l = f^*(x)$.

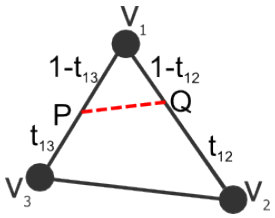


Figure 2.17 – The level set of a function f^* defined by barycentric interpolation of the values on the vertices is a piecewise straight line.

Theorem 3. *The level sets of a function, defined by barycentric interpolation of the values on the vertices, is a straight line.*

Proof. In Fig. 2.17 it is possible to see the representation of f^* in barycentric coordinates on a given face. Given the hypothesis that:

$$\begin{aligned} t_{12}f(v_1) + (1 - t_{12})f(v_2) &= l \\ t_{13}f(v_1) + (1 - t_{13})f(v_3) &= l \end{aligned} \tag{2.7}$$

The expression of P and Q with barycentric coordinates are:

$$\begin{aligned} P &= (t_{13}, 0, 1 - t_{13}) \\ Q &= (t_{12}, 1 - t_{12}, 0) \end{aligned}$$

The equation of the line passing through P, Q is (Wei12):

$$\begin{vmatrix} t_{12} & (1-t_{12}) & 0 \\ t_{13} & 0 & (1-t_{13}) \\ \lambda_1 & \lambda_2 & \lambda_3 \end{vmatrix} = 0$$

that is

$$\lambda_1(1-t_{12})(1-t_{13}) - \lambda_2 t_{12}(1-t_{13}) - \lambda_3 t_{13}(1-t_{12}) = 0$$

that can be simplified in:

$$\lambda_1 - \lambda_2 \frac{t_{12}}{(1-t_{12})} - \lambda_3 \frac{t_{13}}{(1-t_{13})} = 0 \tag{2.8}$$

but for Eq. (2.7):

$$t_{12} = \frac{l - f(v_2)}{f(v_1) - f(v_2)}$$
$$t_{13} = \frac{l - f(v_3)}{f(v_1) - f(v_3)}$$

and thus:

$$\frac{t_{12}}{(1-t_{12})} = \frac{l - f(v_2)}{f(v_1) - l}$$
$$\frac{t_{13}}{(1-t_{13})} = \frac{l - f(v_3)}{f(v_1) - l}$$

Thus replacing these terms in Eq. (2.8):

$$\lambda_1 - \lambda_2 \frac{l - f(v_2)}{f(v_1) - l} - \lambda_3 \frac{l - f(v_3)}{f(v_1) - l} = 0$$

$$\lambda_1(l - f(v_1)) + \lambda_2(l - f(v_2)) + \lambda_3(l - f(v_3)) = 0$$

but recalling that $\lambda_1 + \lambda_2 + \lambda_3 = 1$ it gives us:

$$\lambda_1 f(v_1) + \lambda_2 f(v_2) + \lambda_3 f(v_3) = l$$

$LS_{f^*}(l)$ is a 1-manifold (see (EHZ03)), namely a *line* made up of a finite number of connected components, unless $l = f(v)$ and v is a critical point. Indeed, another way of proving Th. 3 is showing that a direction of steepest

ascent is uniquely defined and transversal at all points of such $LS_{f^*}(l)$. Due to the generality of f, each level line can contain at most one vertex of X. When l is equal to the value of f of either a *maximum* or a *minimum*, one contour in the corresponding level set degenerates into a point. When l is equal to the value of f on a saddle v, one contour in the level set becomes the union of two or more closed lines intersecting in v (see Fig. 2.20). In this case l is deemed *critical* because there exists a critical vertex $v \in X$ such that $f(v) = l$. l is deemed *regular* otherwise.

Definition 76 (Contour γ). *Each connected component of the level set $LS_{f^*}(l)$ is called a contour γ.*

f^* has $[f_{min}, f_{max}]$ as its image, where f_{min} and f_{max} are the minimum and maximum values of f over the set V, respectively.

The space formed by the contours in f^{*-1} for all values $l \in [f_{min}, f_{max}]$ can be equipped with a topology along the lines as described in (EH10). This allows establishing a one-to-one relation between contours and the points of a *graph*. More precisely, all non-degenerate contours (that thus contain one closed line only) correspond to a point in an *arc* of the graph while all degenerate contours, either composed by a single point or by multiple closed lines, correspond to a *node* in the graph.

When f^* is a Morse function, defined on a triangulated manifold X, the definition of the Reeb graph $R_{f^*}(X)$ goes as in the smooth case.

Definition 77 (Reeb graph of PL-Morse function). *Given a* piecewise linear simple *Morse function f^* defined on a smooth, closed, compact 2-manifold X, the* Reeb graph $R_{f^*}(X)$ *is the* quotient space *defined by the equivalence relation:*

$$(x, f^*(x)) \sim (y, f^*(y)) \Leftrightarrow f^*(x) = f^*(y) = l \text{ and } \exists \gamma \subseteq f^{*-1}(l), x, y \in \gamma$$

being x and y any two points on the surface X.

In other words, $R_{f^*}(X)$ is the space of all *contours* defined by f^* on X and is equipped with the *quotient topology* defined by a map $\gamma_{f^*} : X \to R_{f^*}(X)$:

$$\gamma_{f^*}(x) := \text{ The contour } \gamma \subseteq f^{*-1}(f^*(x)) \text{ that contains } x$$

When f is not bound to be Morse, then the graph obtained is called *Extended Reeb graph* (ERG) (BFS00) because its definition derives directly

from that of *Reeb graph* for smooth surfaces (Ree46), but it is *extended* because one of the basic tenets of the original definition is that the scalar function f defined on a smooth, compact 2-manifold must also be a *Morse* function, which implies having only *simple* saddles, i.e. points where only two closed lines in the level set intersect. It follows that, in the original Reeb graph, all nodes corresponding to saddles have connectivity 3. In the realm of triangulated surfaces and with scalar functions which are initially defined on vertices only, the Morse condition is no longer necessary, and this implies that the nodes of an ERG corresponding to a saddle can have connectivity greater than 3.

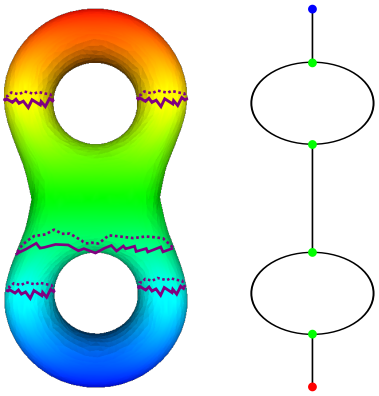


Figure 2.18 – An example of *Reeb graph* corresponding to the connected components of all the level sets f^{*-1}.

2.2.5 Contour strip Reeb graphs

We are now going to introduce a discrete theory for Reeb graphs, founded on works of (Ban70) and (EH10). As in the case of ERG, we will not need the function f to have only *non-degenerate* saddles: when f^* is *not* a Morse function, i.e. when there are *degenerate saddles* for f (see Fig. 2.15e), the definition of $R_{f^*}(X)$ still holds but there might be nodes in the graph with connectivity higher than 3, corresponding to *degenerate saddles*.

We now introduce the Contour Strip Reeb graphs. We need to give or recall some definitions. The definition of the neighborhood of a vertex $v \in X$ is the *link* of the vertex itself: $\mathrm{Lk}(v)$ (see Def. 61 and Fig. 2.10c).

Definition 78 (Cross simplex). *Given a function* $f : X \to \mathbb{R}$ *and a value* l*, a simplex* $\sigma \in X$ *is a* cross-simplex *for a value* $l \in \mathbb{R}$ *if either:*
- σ *is a vertex and* $f(\sigma) = l$
- $\exists v_i, v_j \prec \sigma, f(v_i) < l < f(v_j)$.

As shown by Fig. 2.19, a cross simplex could be a vertex, an edge or a triangle: a cross-triangle for a regular value l is such that the level line passes through it. The same is true for edges, while a vertex v can only belong to a cross simplex if $l = f(v)$.

Definition 79 (The *index* of a vertex ((Ban70))). *The* index *of* v *in the discrete setting is:*

$$i(v) = 1 - \frac{1}{2} t_c(v) \tag{2.9}$$

where $t_c(v)$ *is the number of cross-triangles in* $\mathrm{St}(v)$.

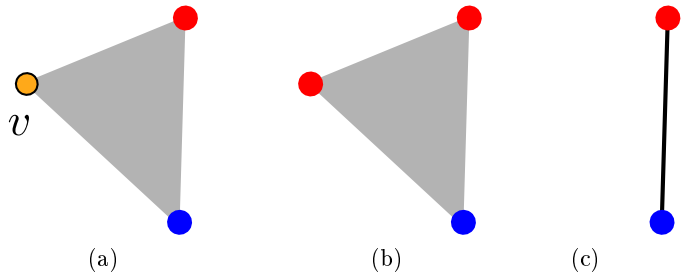
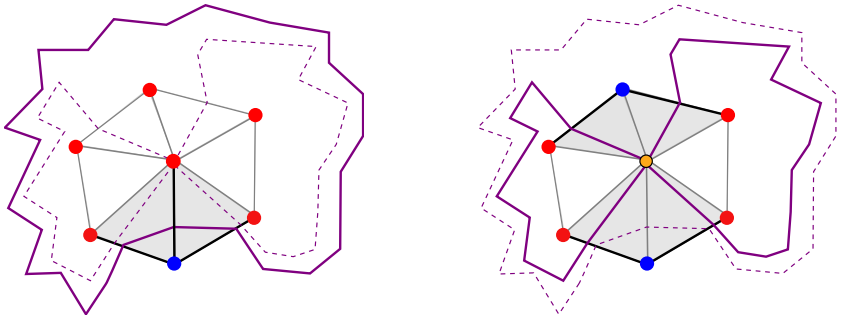


Figure 2.19 – In (a) the value l coincides with $f(v)$, and the triangle in gray is a cross-face. In (b) the triangle is also a cross-face but the value l is between the value of the blue and the red vertices. A cross-edge (c).

As it can be seen in the Fig. 2.15 each *regular point* has exactly two crossing triangles, so its *index* is 0. If v is a *minimum* or a *maximum* it does not have crossing triangles, so its *index* is 1. In the case of a *saddle* there will be so many couples of crossing triangles as $m + 1$, being m the *multiplicity* of the saddle. Consequently the index of the saddle will be $-m$. For the reasons shown in Fig. 2.20, the number of closed lines f^* intersecting at a saddle v is equal to $1/2$ the number of cross-faces in the $\mathrm{St}(v)$.

As described in (Ban70), Def. 79 induces a few considerations that are specific to the discrete setting. Considering an height functions defined by



(a) (b)

Figure 2.20 – In (a) the level set f^{*-1} (solid line) crosses the *star* of v. In (b), the level set f^{*-1} passes through v, revealing that it is in fact a *2-saddle*. The number of closed lines intersecting at a saddle is equal to 1/2 the number of cross-triangles in its star.

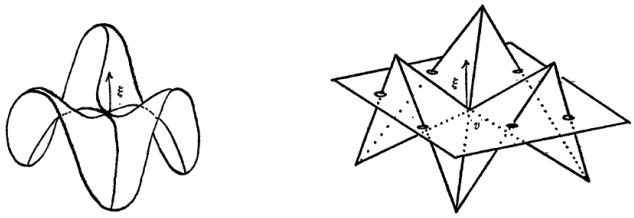


Figure 2.21 – Degenerate saddles in the smooth domain (left) and in the discrete domain (right) (from (Ban70))

a vector ξ on a closed, smooth 2-manifold (see Fig. 2.21), the directions of ξ producing degenerate critical points are a subset of $\mathbb{S}^2$ with zero measure, which means it always exists an infintesimal perturbation than turns a non-Morse height function into a Morse function. We say that Morse functions are a *dense set* in the continuous setting.
On a triangulated surface on the contrary, any direction ξ generating a degenerate point is included in a subset of $\mathbb{S}^2$ having non-zero area (see Fig. 2.21). This means that perfectly defective triangulated surfaces can exist in the discrete settings.

Nevertheless every *general* function f on X can be turned into a Morse function by altering the triangulated surface with a local remeshing, as shown in Fig. 2.22, with the iterative procedure of *saddle unfolding* (EHZ03). This technique unfolds a k-saddle into two new ones, having *multiplicity* i, j

with $i, j < k$ and $i + j = k$. This approach is adopted by many authors ((CMEH+03), (BHEP04), (TVD09) among the others).

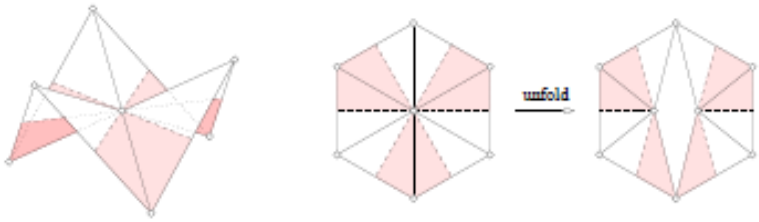
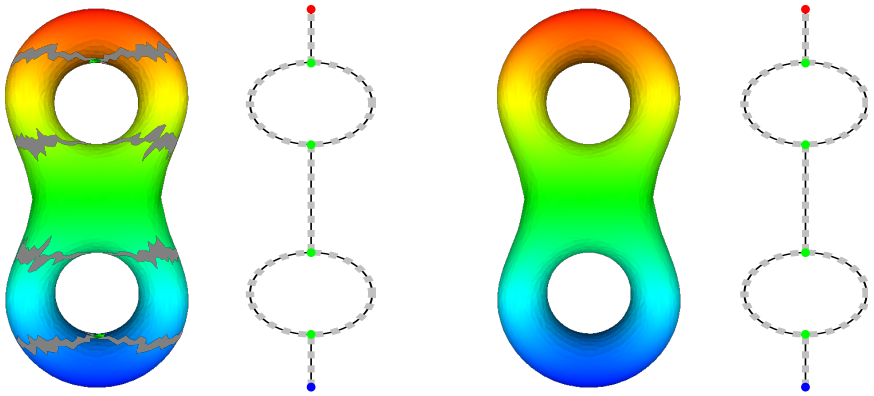


Figure 2.22 – A degenerate saddle unfolded into two simple saddles (EH10).



(a) CSTRG (b) SRG

Figure 2.23 – In the CSTRG (a) each node corresponds to the contour strip of a vertex (see text), while arcs correspond to the contour strips 'in between'. In the SRG (b) there are nodes for each critical point of f and for each segment, while the arcs represent adjacency.

2.2.5.1 Critical Point Theorem

We will now introduce the *Critical Point Theorem* in the discrete settings (Ban70) showing that f need not be a Morse function for the *Critical Point Theorem* to hold, it is enough it is *general*.

Definition 80 (The *Euler characteristic* of a triangulated surface).

The Euler characteristic *of any triangulated surface is:*

$$\chi(X) := V(X) - E(X) + T(X) \tag{2.10}$$

where $V(X)$, $E(X)$ and $T(X)$ are the sets of vertices, edges and triangles in X, respectively.

We need also the following lemma:

Lemma 4. *For a closed triangulated surface:*

$$3T(X) = 2E(X) \tag{2.11}$$

Indeed any triangle has three edges as faces and each edge is shared by two triangles.

Theorem 5. *For any general function $f : V(X) \to \mathbb{R}$ the following is true:*

$$\chi(X) = \sum_{v \in V(X)} i(v) \tag{2.12}$$

where $V(X)$ is the set of vertices of X and $i(v)$ is the index of vertex v (in the sense of Def. 79).

Proof. Adding Lemma 4 and Definition 80 we obtain:

$$\chi(X) = V(X) - \frac{1}{2}T(X) \tag{2.13}$$

The proof of the Theorem 5 then proceeds as follows:

$$\begin{aligned}
\chi(X) &= \sum_{v \in V(X)} i(v) \\
\chi(X) &= \sum_{v \in V(X)} (1 - \frac{1}{2}t_c(v)) \\
\chi(X) &= V(x) - \sum_{v \in V(X)} \frac{1}{2}t_c(v) \\
\chi(X) &= V(X) - \frac{1}{2}T(X)
\end{aligned}$$

The last step is justified by the fact that, since f is *general*, each triangle

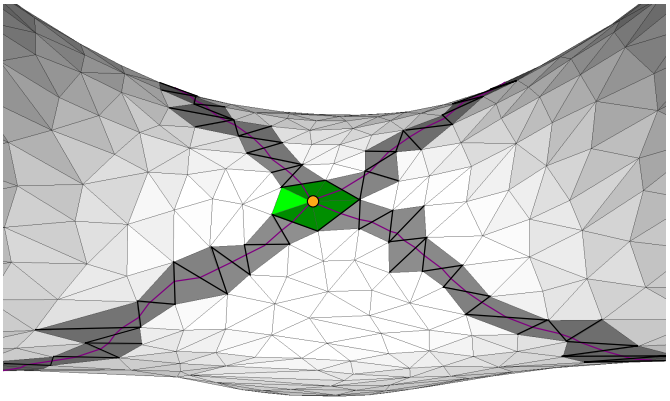


Figure 2.24 – A critical vertex (in orange), its *level strip* for f (edges in black and triangles in dark gray), its *level set* for f^* (the line in violet)

in X crosses the "plane" for exactly one of its vertices: the one in between in terms of values of f.

Theorem 5 can also be reformulated as:

$$\chi(X) = n_1 - \sum_{i>0} i \cdot n_{-i} \tag{2.14}$$

where n_1 and n_{-i} are the numbers of critical points with the index 1 and $-i$ respectively.

It is worth highlighting that f generality is a sufficient condition for the Theorem 5 to hold.

2.2.5.2 Loop lemma for Reeb graphs on a triangulated surface

Lemma 6 (Loop lemma). *The Reeb graph $R_{f^*}(X)$ of a general function f on a connected, closed and orientable triangulated 2-manifold X of genus g has exactly g loops (CMEH$^+$03).*

Proof. As a direct consequence of the critical point theorem, the loop lemma too does not require f to be a Morse function. Given that $R_{f^*}(X)$ is a graph, clearly, this is valid:

$$\chi(R_{f^*}(X)) = \beta_0(R_{f^*}(X)) - \beta_1(R_{f^*}(X))$$

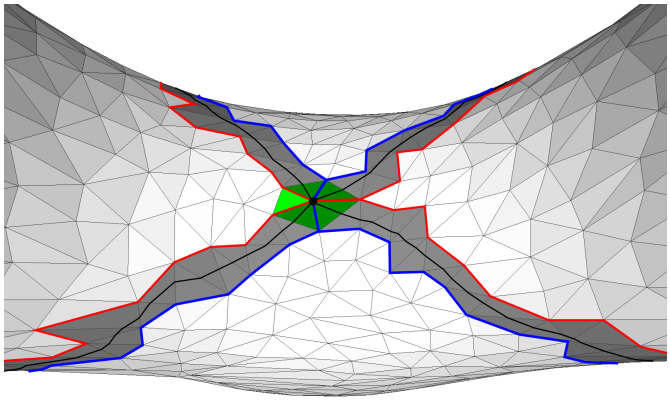


Figure 2.25 – A critical vertex (in black), its *level strip* for f (in gray), its *level set* for f^* (the line in black), its *upper level set* (in red) and *lower level set* (in blue)

Where β_0 is the Betti number (see Sec. 2.1.1) representing the number of connected components of the graph of X, that, for a connected manifold, is always 1. β_1 is the first Betti number, which is equal to the number of loops in the graph (see again sec. 2.1.1). Therefore, with the previous assumption that X is connected:

$$\begin{aligned} V(R_{f^*}(X)) - E(R_{f^*}(X)) &= \beta_0(R_{f^*}(X)) - \beta_1(R_{f^*}(X)) \\ V(R_{f^*}(X)) - E(R_{f^*}(X)) &= 1 - \beta_1(R_{f^*}(X)) \\ \beta_1(R_{f^*}(X)) &= 1 - (n_1 + \sum_{i>0} n_{-i}) + E(R_{f^*}(X)) \end{aligned}$$

But, counting the number of edges departing from a graph node, and dividing by 2 for internal nodes (for which each edge is adjacent to 2 nodes):

$$E(R_{f^*}(X)) = \frac{1}{2}(n_1 + \sum_{i>0}(2+i) \cdot n_{-i})$$

Then:

$$\beta_1(R_{f^*}(X)) = 1 - \frac{1}{2}(n_1 - \sum_{i>0} i \cdot n_{-i})$$

but, due to Theorem 5:

$$\begin{aligned}\beta_1(R_{f^*}(X)) &= 1 - \frac{1}{2}(n_1 - \sum_{i>0} i \cdot n_{-i}) \\ &= 1 - \frac{1}{2}\chi(X) \\ &= g(X)\end{aligned}$$

The number of loops of the Reeb graph $\beta_1(R_{f^*}(X))$ is equal to the genus of the mesh $g(X)$.

2.2.5.3 Level strip

Definition 81 (Level strip of α). *The level strip at α is:*

$$\begin{aligned}LS(\alpha) :=& \{v \in X : f(v) = \alpha\} \cup \\ & \{\sigma \in X : \forall v_i, v_j | v_i \preceq \sigma \text{ and } v_j \preceq \sigma | f(v_i) < \alpha < f(v_j)\} \cup \\ & \{\tau \in X : \forall v_i, v_j, v_k | v_i, v_j, v_k \preceq \tau | f(v_i) < f(v_k) < \alpha < f(v_j)\}\end{aligned}$$

The $LS(\alpha)$ is the set of all the cross faces and the cross edges at α and at most the vertex v such that $f(v) = \alpha$.
Among the level strips of α we can define those passing from a vertex v:

Definition 82 (Level strip of v). *We can define the level strip of v as a sub-case of $LS(\alpha)$ where $f(v) = \alpha$:*

$$LS(v) := LS(\alpha) \mid \alpha = f(v)$$

We can see an example of level strip in Figure 2.26.

Two cross-faces in $LS(\alpha)$ are said to be connected if they share either an edge or a vertex which is also in $LS(\alpha)$. With this definition in mind, we can introduce the following.

Lemma 7 (Homotopy between $LS(\alpha)$ and $f^{*-1}(\alpha)$). *For each $v \in V(X)$, the union of the interior of the simplices in the level strip $LS(v)$ has the same* homotopy type *of the level set $f^{*-1}(f(v))$. In addition, for any other value α 'in between' (i.e. for which there is no v such that $\alpha = f(v)$), the level strip $LS(\alpha)$ has the same* homotopy type *of $f^{*-1}(\alpha)$. (see Fig. 2.27)*

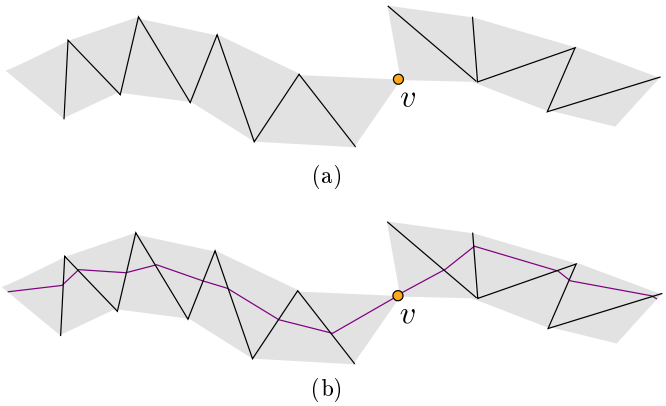


Figure 2.26 – Level strips contain cross-faces, cross-edges and at most one cross-vertex (a). The level set $f^{*-1}(f(v))$ is entirely contained in the level strip $LS(v)$ and has the same homotopy type (b).

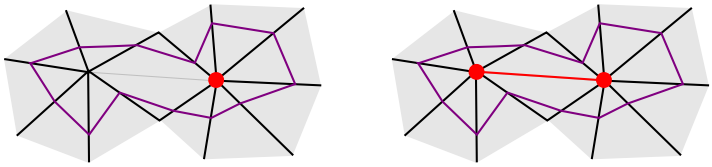


Figure 2.27 – A ring-like contour f^{*-1} (solid line) that runs around a maximum (a). The level strip does not include the edge in gray in the middle and hence has the homotopy type of the level set f^{*-1}, i.e. a ring, whereas the ULS (in red) in (b) does not.

Proof. This can be proved by parts, by *deformation retraction.*

Definition 83 (Deformation retraction). *Given two topological spaces* X, Y, *such that* $\mathrm{Y} \subseteq \mathrm{X}$, *a deformation retraction is a map :*

$$h : \mathrm{X} \times [0,1] \to \mathrm{X}$$

such that:

- h *is continuous*
- $\forall x \in \mathrm{X},\ h(x,0) = x$ (*i.e.* $h(.,0) = \mathrm{Id}_x$)
- $\forall x \in \mathrm{X},\ h(x,1) \in \mathrm{Y}$

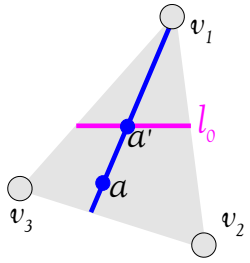


Figure 2.28 – The interior of a triangular face has the same homotopy type of the interior of the segment of a level set that crosses it.

- $\forall y \in \mathrm{Y},\ \forall t \in [0,1],\ h(y,t) = y$

We consider the level strip as union of simplices (faces, edges and vertices) and we consider, for each of them, their interior. We build a deformation retraction that transforms each of them in the corresponding piece of f^{*-1} that crosses them.

In the case of the only vertex v in the contour strip $LS(v)$ it is trivial to see that its interior is deformed retracted in the vertex v itself. It is also easy to show that the interior of each cross edge in the contour strip can be deformed retracted in the point of f^{*-1} that crosses it. In the case of the interior of a face we need an additional result:

Lemma 8. *The interior of a triangular face has the same homotopy type of the interior of the segment of a level set that crosses it.*

Proof. Also in this case we prove it by constructing a deformation retraction. Each point a is projected along the radius that joins a and v_1 (see Fig. 2.28), where v_1 is the vertex shared by the two cross-edges. The generic equation of a level set in barycentric coordinates is:

$$\lambda_1 f(v_1) + \lambda_2 f(v_2) + \lambda_3 f(v_3) = l$$

The point a has barycentric coordinates (a_1, a_2, a_3).

The line that runs through v_1 and a is also called the radial line through a, and its equation can be found solving this determinant (Wei12):

$$\begin{vmatrix} 1 & 0 & 0 \\ a_1 & a_2 & a_3 \\ \lambda_1 & \lambda_2 & \lambda_3 \end{vmatrix} = 0$$

which gives:

$$a_2\lambda_3 - a_3\lambda_2 = 0 \quad \textit{and thus } a_2\lambda_3 = a_3\lambda_2$$

The intersection between the radial line through a and the generic level set at l is:

$$\begin{cases} \lambda_1 f(v_1) + \lambda_2 f(v_2) + \lambda_3 f(v_3) = l \\ a_2\lambda_3 = a_3\lambda_2 \\ \lambda_1 + \lambda_2 + \lambda_3 = 1 \end{cases}$$

The solution of this system is:

$$\lambda_1 = \frac{a_2 l + a_3 l - a_2 f(v_2) - a_3 f(v_3)}{a_2 f(v_1) - a_2 f(v_2) + a_3 f(v_1) - a_3 f(v_3)}$$

$$\lambda_2 = \frac{a_2 (f(v_1) - l)}{a_2 f(v_1) - a_2 f(v_2) + a_3 f(v_1) - a_3 f(v_3)}$$

$$\lambda_3 = \frac{a_3 (f(v_1) - l)}{a_2 f(v_1) - a_2 f(v_2) + a_3 f(v_1) - a_3 f(v_3)}$$

The equation of f^* at a is:

$$a_1 f(v_1) + a_2 f(v_2) + a_3 f(v_3) = l_a$$

The map $h : \mathrm{X} \times [0,1] \to \mathrm{X}$ (where X is the interior of a triangle) projects each point $a \in \mathrm{X}$ to the intersection between the radial line through a and a level set at level $l = l_a - t(l_a - l_0)$ where l_0 is the value of the target level set.

- clearly h is continuous
- $\forall a \in \mathrm{X},\ h(a,0) = a$ (that is the intersection between the radial line through a and the level set l_a)
- $\forall a \in \mathrm{X},\ h(a,1) \in \mathrm{Y}$ (where Y is the interior of the level set in X)
- $\forall a \in \mathrm{Y},\ \forall t \in [0,1],\ h(a,t) = a$

Thus the union of the interior of the simplices in the level strip $LS(v)$ has the same *homotopy type* of the level set $f^{*-1}(f(v))$.

It is now natural to define the *contour strip*:

Definition 84 (Contour strip of v). *$CS(v)$, the contour strip of v, is the connected component of $LS(v)$ containing v.*

Considering any two vertices v_1 and v_2 that are either regular or saddle, their contour strips are said to be *adjacent* if they overlap, in the sense that they share at least one cross-face, and there is no other overlapping contour strip for a vertex v_3 such that $f(v_3)$ lies between $f(v_1)$ and $f(v_2)$. If vertex v is either a minimum or a maximum, the above definition also applies by assuming that any other contour strip overlaps the (degenerate) contour strip of v if it overlaps $\mathrm{St}(v)$.

2.2.5.4 CSTRG

In order to define the adjacency between nodes of the *contour strip Reeb graph* ($CSTRG$) we are going to introduce, we need two more definitions:

Definition 85 (Upper strip of v). *The upper strip of v is:*

$$UpS(v) := CS(v) \cup \mathrm{St}^{+}(v) - v$$

and also:

Definition 86 (Lower strip of v). *The lower strip of v is:*

$$LowS(v) := CS(v) \cup \mathrm{St}^{-}(v) - v$$

where $\mathrm{St}^{+}(v)$ and $\mathrm{St}^{-}(v)$ are respectively the set of cross simplices in $\mathrm{St}(v)$ for $\alpha = f(v) + \epsilon$ and $\alpha = f(v) - \epsilon$. We can now define the *contour strip Reeb graph*:

Definition 87 (Contour strip Reeb graph $CSTRG$). *The contour strip Reeb graph of triangulated surface X is a graph whose nodes n_v correspond to the $CS(v)$, $\forall v \in V(X)$ and whose arcs $arc_{i,j}$ represent an adjacency relation between the nodes n_{v_i}, n_{v_j}. In particular, two nodes n_{v_i}, n_{v_j} are adjacent on the graph if $f(v_i) < f(v_j)$ and*

$$\exists CS \mid CS \subseteq UpS(v_i) \quad and \quad CS \subseteq LowS(v_j) \tag{2.15}$$

In particular when v_i, v_j are both regular, or one regular and one maximum/minimum then $UpS(v_i) = LowS(v_j)$, on the other hand, when at

least one of them is a saddle point then the relation of Eq. (2.15) holds.

More important, the $CSTRG$ represents all the possible, distinct contour strips of X: every node n_v represents a $CS(v)$, every arc between two nodes $arc_{(n_{v_i}, n_{v_j})}$ represents the adjacency relation between $CS(v_i)$ and $CS(v_j)$, relation that is guaranteed through the $UpS(v_i)$ and the $LowS(v_j)$.

It is worth to note that $UpS(v)$ and $LowS(v)$ have been obtained with a geometric operation (see Def. 85 and Def. 86). Nevertheless their definition is also justified in terms of the level lines of f^*, indeed:

$$UpS(v_i) \subseteq LS(f^*(v_i) + \epsilon)$$

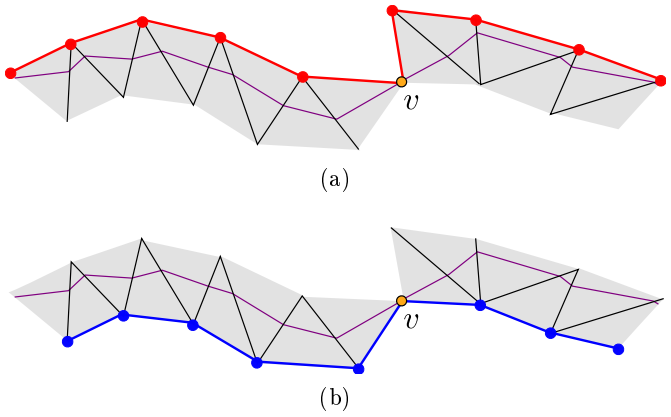


Figure 2.29 – The ULS (*upper level set*) (a) and the LLS (*lower level set*) (b) for the same level strip in Fig. 2.26. Vertex v is included in both.

As we have seen, the level strips guarantee the construction of a graph (the CSTRG) that is homeomorphic to the ERG, and furthermore they are a computational simpler alternative to the level lines of f^*. But our aim is to find a simpler representation of the level lines of f^*, that uses only the 1-skeleton of the X: we want to use existing lines (vertices and edges in the mesh) without tracing new ones. One idea could be to use the upper (lower) bound of the level strip. But this could lead to some theoretical problems, as we will show in the next section.

2.2.5.5 Upper and lower level sets

The *upper* (resp. *lower*) *level set* (ULS, LLS) for a vertex v is the upper (resp. lower) boundary of the closure of $LS(v)$ in X. More precisely, the ULS for v contains all the vertices v_i in the closure of the level strip such that $f(v_i) \geq f(v)$ plus all edges between the vertices that also belong to the closure of the level strip. Here following we give the formal definitions.

Definition 88 (Upper level set). *The* upper level set *of f at α is:*

$$ULS(\alpha) := \{\sigma \in Cl(LS(\alpha)) : \forall v \preceq \sigma, f(v) \geq \alpha\}$$

The dual definition for the LLS is obvious.

Definition 89 (Lower level set). *The* lower level set *of f at α is:*

$$LLS(\alpha) := \{\sigma \in Cl(LS(\alpha)) : \forall v \preceq \sigma, f(v) \leq \alpha\}$$

Clearly v belongs to both $ULS(v)$ and $LLS(v)$.
In an effort to ease computations, defining a graph using either the ULS or the LLS in the place of the level sets $f^{*-1}(f(v))$ or even of the level strips $LS(v)$ may seem attractive, since it entails dealing with edges and vertices only. However, as shown by the examples in Fig. 2.30, in general neither the ULS nor LLS have the same homotopy type of $f^{*-1}(f(v))$.

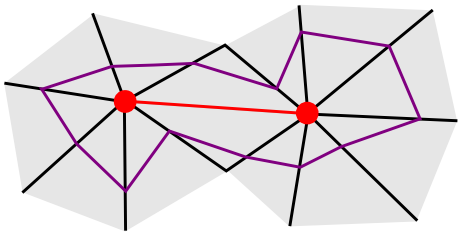


Figure 2.30 – *ULL* (in red) does not have the same homotopy type of $f^{*-1}(f)$ (in purple).

If we build a graph made up of nodes and arcs, where nodes are the connected components of either ULS (or LLS) and the arcs are the corresponding adjacency relations, we do not obtain a graph homeomorphic to the Reeb graph of the f^*. It is possible to see by counterexamples (e.g. in Fig. 2.31), that there exist cases for which the graph constructed on the

ULS (or LLS) is not homeomorphic to the CSTRG and also violates the Loop Lemma.

This problem, in our opinion (and in our experiments), affects the algorithm described in (TVD08) which has, on the other hand, largely inspired the work on the algorithm presented in this thesis.

2.2.6 Simplified Reeb graphs

The definition of ULS of LLS for a vertex v can be enhanced in order to regain the equivalence with the level sets $f^{*-1}(f(v))$ and, through this, with the CSTRG. We now introduce the concept of *multiplicity* of an edge and a vertex in the *upper* or *lower* level set and we show how this construct guarantees the homeomorphism between this *augmented upper* or *lower* level set and the the level line of f^*.

2.2.6.1 Multiplicity for augmented upper and lower level set

Vertices and edges in the ULS (resp. LLS) must be assigned a *multiplicity*, in order to gain the homeomorphism with $f^{*-1}(f(v))$.

The multiplicity of an edge in the $ULS(\alpha)$ is equal to the number of faces that are adjacent to the edge and also belong to the level strip $LS(\alpha)$.

Definition 90 (Multiplicity of edges in the $ULS(\alpha)$). *An edge $e \in ULS(\alpha)$ has* multiplicity 2 *in $ULS(\alpha)$ if:*

$$\exists t_i, t_j \in LS(\alpha) \text{ and } e \prec t_i, e \prec t_j$$

Any other edge $e \in ULS(\alpha)$ has multiplicity 1.

The definition of multiplicity in $LLS(\alpha)$ is dual:

Definition 91 (Multiplicity of edges in the $LLS(\alpha)$). *An edge $e \in LLS(\alpha)$ has* multiplicity 2 *in $LLS(\alpha)$ if:*

$$\exists t_i, t_j \in LS(\alpha) \text{ and } e \prec t_i, e \prec t_j$$

Any other edge $e \in LLS(\alpha)$ has multiplicity 1.

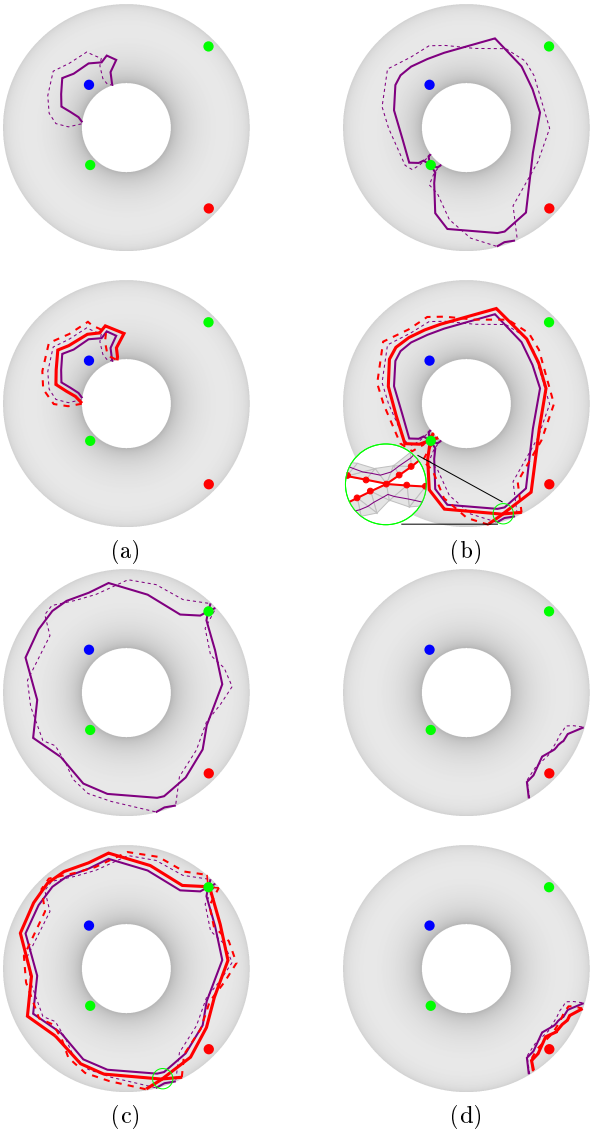


Figure 2.31 – The figure represents a toroidal surface over which a general function f (not shown) is defined; it has a minimum, a maximum and two saddles. A sequence of four level sets f^{*-1} is considered, for increasing values of α. ULS are shown in red. In step (a) f^{*-1} has a unique connected component, which *splits* in two in step (b). The two connected components *merge* in (c), hence in step (d) f^{*-1} has again one connected component. In contrast, in step (b) the ULS has two singularities: one at the saddle and another one, in the lower part of the torus, which does *not* correspond to a critical point of f. Due to the latter, the ULS remains connected even after meeting the first saddle. In step (c) the ULS meets the second saddle but no *merge* occurs, since the connected component is unique. Hence, in this example, the graph corresponding to the connected components of the ULS has no loops and violates the Loop Lemma.

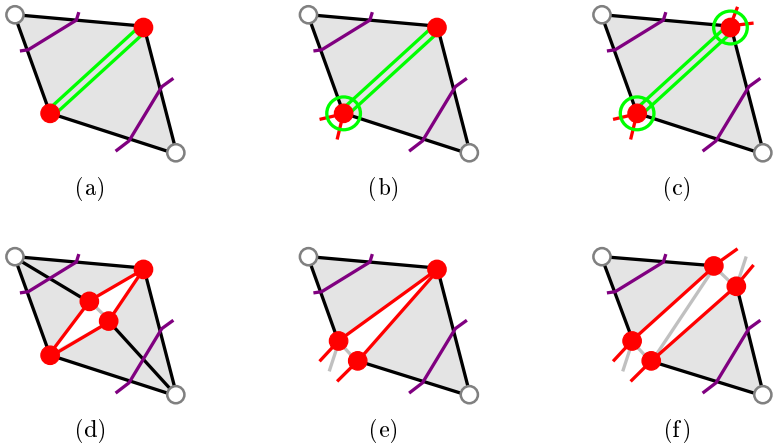


Figure 2.32 – A double presence edge in its three possible configurations: with two vertices with *multiplicity* $m = 1$ (a), with one vertex with *multiplicity* $m = 1$ and one vertex with *multiplicity* $m = 2$ (b), with two vertices with *multiplicity* $m = 2$ (c). The lower row shows their corresponding *implicit* unfolding.

The multiplicity of and edge cannot be higher than 2 because, in a triangulated manifold, an edge cannot be face of more than two triangles (see Def. 66 and Fig. 2.12).

In Figures 2.32a, 2.32b, 2.32c are shown all the possible configurations of multiplicity of an edge on a ULS.

By definition, the multiplicity of a vertex v_i in the $ULS(v)$ is equal to 1/2 the number of faces in its star $\mathrm{St}(v_i)$ that also belong to the level strip $LS(v)$. A vertex v which is a saddle for f has multiplicity 2 or greater in its $ULS(v)$, depending on how many contours of f^{*-1} intersect at it.

Definition 92 (Multiplicity of a vertex in the $ULS(\alpha)$). *The* multiplicity *of a vertex $v \in ULS(\alpha)$ is:*

$$m(v) = \frac{1}{2} \sum_{e \in ULS(\alpha), v \prec e} m(e)$$

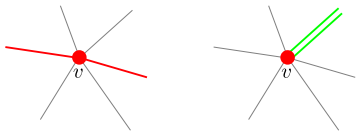
where $m(e)$ is the multiplicity *of the edge in $ULS(\alpha)$.*

The same holds for $LLS(\alpha)$:

Definition 93 (Multiplicity of a vertex in the $LLS(\alpha)$). *The* multiplicity *of a vertex $v \in LLS(\alpha)$ is:*

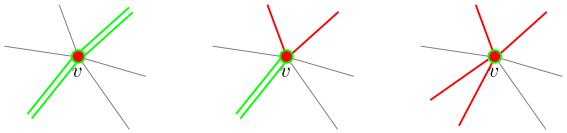
$$m(v) = \frac{1}{2} \sum_{e \in LLS(\alpha), v \prec e} m(e)$$

where $m(e)$ is the multiplicity *of the edge in $LLS(\alpha)$.*



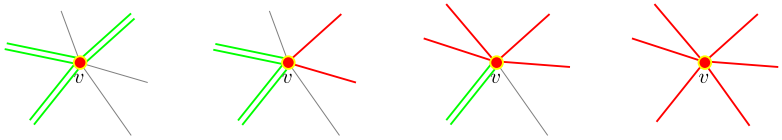
(a) (b)

Figure 2.33 – The catalog of all possible configurations of one vertex with multiplicity 1 in a contour.



(a) (b) (c)

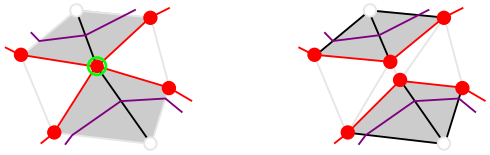
Figure 2.34 – The catalog of all possible configurations of one vertex with multiplicity 2 in a contour.



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

Figure 2.35 – The catalog of all possible configurations of one vertex with multiplicity 3 in a contour.

Vertices can have a multiplicity greater than two, even if there is an higher limit fixed from the topology of the link of the vertex itself: in any case



(a) (b)

Figure 2.36 – A double presence vertex (a) and its *implicit* unfolding (b).

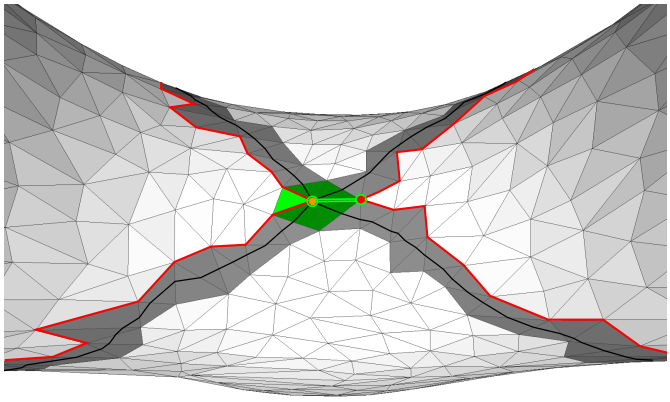


Figure 2.37 – A critical vertex (in orange), its *level set* for f^* (the line in black), its *augmented* upper level set (in red) with two vertices and one edge that have multiplicity $m = 2$ (in green). The vertex with $m = 2$ on the right is not critical because it is not the minimum in its ULS.

the multiplicity of a vertex in the upper/lower level set is limited by the number of vertices in its link:

$$1 < m < \frac{|Lk|}{2} \tag{2.16}$$

In Figures 2.33, 2.34, 2.35 are shown the configurations of multiplicity m of a vertex in the ULS, for $1 \leq m \leq 3$.

Having a multiplicity greater than one in ULS (resp. LLS) is not a sufficient condition for a vertex to be a saddle: indeed in general, both $ULS(v)$ and $LLS(v)$ include multiple vertices that are non-critical (see Fig. 2.37).

For a vertex to be a saddle a stronger condition must be true, as following described.

Lemma 9. *Vertex v is a* saddle *iff $m(v) = 2$ in both $ULS(v)$ and $LLS(v)$.*

Proof. If v is a saddle it must have four cross-triangles in its star. Each of them has both an upper and a lower edge, although it is possible that some edges are shared (i.e. with multiplicity 2).

The lemma can be extended to degenerate saddles, having index -2 and more.

In other words, a vertex is a saddle if it has multiplicity $m > 1$ in a ULS (for any vertex) and it has the minimum value of f among all the vertices in the same connected component. Both $ULS(v)$ and $LLS(v)$ may well contain other vertices with multiplicity $m > 1$ but they are not minimal, in $ULS(v)$, or maximal, in $LLS(v)$, as shown in Fig. 2.37.

Inspired by (EHZ03), each multiple edge and multiple non-critical vertex can be *unfolded* by an (implicit) local remeshing of the kind shown in Fig. 2.32 and 2.36. It is *implicit*, however, because in the algorithm we will propose in chapter 4 such local remeshing never takes place. The algorithm behaves 'as if' the triangulation X was unfolded accordingly. From this point on, we assume by default the *augmented* definitions of ULS and LLS, i.e. those including multiplicity values.

2.2.6.2 Homotopy equivalence: f^{*-1} and the *augmented* level set

Now we introduce some definitions in order to show that the ULS and LLS *augmented* with multiplicity re-establish the topologically equivalence with $f^{*-1}(f(v))$. We will use notions from (Coh73), (RWS11). Consider a simplicial complex K:

Definition 94 (Free face). *$\tau \in K$ is a free face iif there exists $\sigma \in K$ such that $\tau \preceq \sigma$ and σ is unique, i.e. there is no other $\sigma' \in K$ such that $\tau \preceq \sigma'$.*

Definition 95 (Free pair). *The pair (τ, σ) of Def. 94 is called a free pair.*

Definition 96 (Elementary collapse). *An elementary collapse of K is a subcomplex K' that can be obtained from K by removing one free pair.*

Definition 97 (Elementary expansion). *An elementary expansion of K is a subcomplex K' that can be reduced to K by removing one free pair.*

Definition 98 (Collapse). *A collapse of K is a subcomplex K' that can be obtained from K via a sequence of elementary collapses.*

Definition 99 (Expansion). *An expansion of K is a subcomplex K' that can be reduced to K via a sequence of elementary expansions.*

Definition 100 (Simple-homotopy equivalence). *Two simplicial complexes K and M are simple-homotopy equivalent if M can be obtained from K via a sequence of collapses and expansions.*

Theorem 10. *If K and M are simple-homotopy equivalent they are also homotopy equivalent.*

Note that the vice-versa is not necessarily true. With this definition in mind we can prove the following.

Theorem 11. *A connected component of a level set f^{*-1} is homotopy equivalent to the corresponding component of the ULS (resp. LLS) if the latter contains at least one edge.*

Proof. Consider the connected component of f^{*-1} as a 1-simplicial complex and construct an essential simplicial complex (i.e. a strip of triangles) having the connected component of f^{*-1} and the corresponding unfolded component of the ULS (resp. LLS) as boundaries. The strip of triangles is simply homotopy equivalent to both, hence they are homotopy equivalent.

The ULS (resp. LLS) must contain at least one edge, otherwise the counterexample shown in Fig. 2.38 becomes possible.

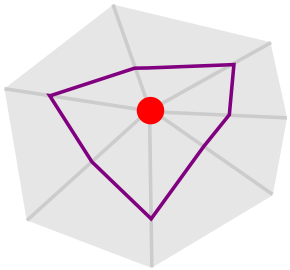


Figure 2.38 – The strip of triangles is simple-homotopy equivalent to the point ULS but not to the ring (f^{*-1}).

2.2.6.3 Upper and lower contour

We now introduce the definition of *upper* (resp. *lower*) *contour* of v, as the connected component of the *augmented* ULS (resp. LLS) that contains v.

Definition 101 (Upper Contour). *An* upper contour $\gamma^+(v) \subseteq ULS(v)$ *is a maximal subset of connected simplices* $\sigma \in ULS(v)$ *that includes* v.

By Def. 88 v is the vertex having the minimum value of f in all $ULS(v)$.

Definition 102 (Lower contour). *A* lower contour $\gamma^-(v) \subseteq LLS(v)$ *is a maximal subset of connected simplices* $\sigma \in LLS(v)$ *that includes* v.

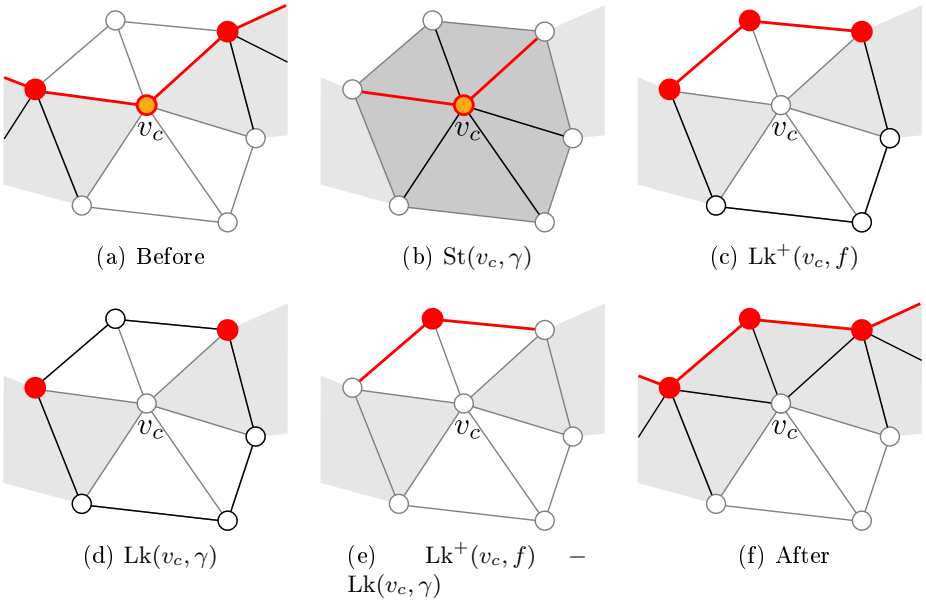
Symmetrically, by Def. 89 v is the vertex having the maximum value of f in all $LLS(v)$.

Upper and lower contours are the connected component of ULS (resp. LLS) and thus they are in the same way augmented with multiplicity.

Corollary 12. *The* upper contour $\gamma^+(v)$ *(resp.* lower contour $\gamma^-(v)$*) is homotopy equivalent to the connected component of* f^{*-1} *containing* v*.*

Proof. If $\gamma^+(v)$ does not contain an edge then v is a maximum and $\gamma^+(v)$ and f^{*-1} are equivalent. If $\gamma^+(v)$ contains an edge then Theorem 11 holds. The same is true for $\gamma^-(v)$: if $\gamma^-(v)$ do not contains an edge then v is a minimum and $\gamma^-(v)$ and f^{*-1} are equivalent. If $\gamma^-(v)$ contains an edge then Theorem 11 holds.

In the following definitions we will use only the ULS and γ^+ for sake of clarity. The same theory holds for the LLS and γ^-. For the same reasons γ^+ will be simply indicated as γ. To introduce the adjacency between contours we will present how the basic step of contour evolution is done (see Fig. 2.39). We thus need some further definitions.



(a) Before (b) $\mathrm{St}(v_c, \gamma)$ (c) $\mathrm{Lk}^+(v_c, f)$

(d) $\mathrm{Lk}(v_c, \gamma)$ (e) $\mathrm{Lk}^+(v_c, f) - \mathrm{Lk}(v_c, \gamma)$ (f) After

Figure 2.39 – The basic step of contour evolution for a *regular* vertex v_c: starting from the initial contour (a), the figure shows the details of the contour update. The *contour strip* (in gray) is shown for convenience only, since it is not explicitly computed.

Definition 103 ($\mathrm{St}(v)$ **on the contour).** *The* $\mathrm{St}(v, \gamma(v))$ *is the intersec-*

tion between the contour $\gamma(v)$ *and the* $\mathrm{St}(v)$*:*

$$\mathrm{St}(v, \gamma(v)) := \mathrm{St}(v) \cap \gamma(v)$$

Fig. 2.39b shows (in red) an example of $\mathrm{St}(v_c, \gamma)$ for the vertex v_c.

Definition 104 ($\mathrm{Lk}(v)$ **on the contour).** *The* $\mathrm{Lk}(v, \gamma(v))$ *is the intersection between the contour* $\gamma(v)$ *and the* $\mathrm{Lk}(v)$*:*

$$\mathrm{Lk}(v, \gamma(v)) := \mathrm{Lk}(v) \cap \gamma(v)$$

Fig. 2.39d shows (in red) an example of $\mathrm{Lk}(v_c, \gamma)$ for the vertex v_c.

Definition 105 (Updated Contour). *An contour* $\gamma(v)$ *is updated in* $\gamma_{upd}(v)$*:*

$$\gamma_{upd}(v) = \gamma(v) - \mathrm{St}(v, \gamma(v)) + (\mathrm{Lk}^+(v, f) - \mathrm{Lk}(v, \gamma(v))) \tag{2.17}$$

Fig. 2.39f shows the contour γ_{upd} once it has been updated for the vertex v_c.

Definition 106 (Adjacency between contours). *Two contours* $\gamma(v_i)$, $\gamma(v_j)$ *are adjacent if* $f(v_i) < f(v_j)$ *and there exists a contour* γ *such that:*

$$\gamma \subseteq \gamma_{upd}(v_i) \text{ and } \gamma \subseteq \gamma(v_j) \tag{2.18}$$

The reason for which Def. 106 is not simply:

$$\gamma_{upd}(v_i) = \gamma(v_j)$$

is that it can happen that the topology of γ changes if the considered vertex is a critical one. The definition of adjacency will have a central role in the algorithm presented in Chap. 4.

2.2.6.4 SRG

Equipped with the previous adjacency relation, the resulting graph will be a graph with a node for each mesh vertex: nodes corresponding to regular

values will have connectivity 2, nodes corresponding to critical values will have connectivity 1 in case of maxima and minima, or > 2 in case of saddles.

Sweeping the mesh from minima (resp. maxima) there will be sequences of regular vertices (corresponding to node with connectivity 2 on the graph), separated by critical vertices.

Definition 107 (Simplified Reeb graph). *The* simplified *Reeb graph (SRG) is a graph that has one node for each critical point and one node for each segment, whose boundaries are the contours passing by the critical points. Each segment-node is adjacent to the nodes corresponding to the two critical points whose contours barrier the segment.*

Since f is general there cannot be two critical points at the same height, and thus each segment will be bordered by the contours of *exactly* 2 critical points (one lower and one upper).

From a topological perspective the SRG is a simplification of the CSTRG, in which all the nodes corresponding to regular values, between two critical nodes, are collapsed in a single node, representing the *mesh segment* swept by the corresponding contour. It is thus easy to see that the SRG is homemorphic to the ERG, indeed it respects the *Definition 3.1.1* of (Bia04) and obeys the Loop Lemma, for every choice of a *general* function f.

2.3 Segmentation

Mesh surface segmentation represents an important area of interest in several scientific fields, like medical applications, geographical maps rendering, and more in general, in computer graphic (for compression, simplification, morphing, ...). Skeletonization and segmentation tasks are often complementary, one being the driver to the other. Mesh segmentation is about partitioning mesh elements (vertices, edges and faces) in disjoint sets. Segmentation tasks can be divided into two different types (Sha08), (AKM$^+$06):

- *part-type* segmentation, whose aim is to partition a surface in its meaningful parts; the methods that use this type of segmentation are focused on identifying relevant shape features. Part segmentation is strictly related on how human beings perceive objects: as a collection of parts.

- *surface-type* or *patch-type* segmentation, whose aim is to subdivide a surface into patches under a certain criterion (curvature, patch area, ...). These methods are applied in the case of e.g. texture mapping, charts painting, mesh simplification, radiosity, ray tracing...

We give now a more formal definition of the segmentation of a mesh M.

Definition 108 (Mesh segmentation Σ). *Let M be a $2-d$ mesh embedded in R^3, and S the set of mesh elements (V, E or T). A segmentation Σ of M is the set of sub-meshes $\Sigma = \{M_0, ..., M_{k-1}\}$ induced by a partition of S into k disjoint subsets.*

In certain cases, e.g. when partitioning vertices or edges, it could happen that an element (e.g. a face) could be cross between two different segments, in this case it must be attached to one of the two segments under a certain criterion.

Segmenting a surface mesh in patches is not easy: under the condition that the patches are convex, (CDST97) proved that the problem is NP-complete.
(Sha04) defines mesh segmentation as an optimization problem with the following definition:

Definition 109 (Mesh segmentation as an optimization problem). *Given a mesh M and the set of elements $S \in \{V, E, T\}$, the objective is to find a disjoint partitioning of S into $S_0, ..., S_{k-1}$ such that the criterion function $J = J(S_0, ..., S_{k-1})$ is minimized (or maximized) under a set of constraints C.*

The objective is thus to find an approximate solution that optimizes some parameters, being bound to certain constraints. (Sha08) groups the constraints considered to guide the segmentation process into these families:

- *cardinality constraints*: these constraints are about the number of parts of a partition. They could be absolute or relative bounds (i.e. the minimum number of elements in each part, for example to eliminate too small and not-significant partitions).
- *topological constraints*: these are bounds that make a partition to be connected or homeomorphic to a disk.

- *geometric constraints* these constraints are about the maximal/minimal area of a partition, its convexity, its maximal diameter, etc...

2.3.1 Segmenting with SRG

Aside from the construction of a compact shape representation in the form of a graph, the theoretical notions presented let us define a suitable *segmentation* of X, namely a partition of the surface into segments which are themselves organized in a way that reflects the topology of X (BGSF08).

An effective segmentation of this sort can be defined by taking the contours in $f^{*-1}(f(v))$ for each saddle vertex v as boundaries for segments (BMS00), (PSF09). From this, yet another definition of a graph can be derived. We define the *Simplified Reeb Graph* (SRG) as the graph obtained by associating each segment to a node, plus one node corresponding to each critical point of f. Arcs correspond to adjacency relations: each node associated to a segment is adjacent to the two nodes corresponding to the two critical points v_i whose contours $f^{*-1}(f(v_i))$ border the segment itself.

By definition, each contour of $f^{*-1}(f(v_i))$ corresponding to a saddle vertex passes through the vertex itself and then runs across the triangular faces in the contour strip. This means that the remeshing of X becomes necessary whenever an accurate representation of segments is required. For many applications, however, an approximate segmentation by which each non-critical vertex is univocally associated to a segment can be enough, making the remeshing operation unnecessary.

To make this possible, however, each face of X must be *simple* for f, i.e. that only one segment boundary should pass across it. Often in practice, a few faces can be *multiple*, in the sense that they are crossed by multiple segment boundaries. Ignoring this could cause problems, as the resulting approximated segments could be disconnected. Intuitively, this problem becomes more evident when the density of vertices in X is lower, the genus of X is higher and/or f is less 'regular'. In Sec. 5.2 we will discuss some experimental evidences for this.

As we will see, the algorithm presented can detect *multiple faces* (see Fig. 4.7b) hence mark those parts of X for which a local remeshing operation is truly necessary, if the simplicity condition above is to be ensured.

Anticipating (Sha08) classification, that will be described in Sec. 3.2, the

method here proposed is an *implicit* method, in that the produced segmentation is a by-product of the *SRG* computation.

Seeing the segmentation task as a *constrained* problem, the constraint that we cannot ignore is that the segmentation must be topologically correct. If we imagine to have a segmentation, and to build a dual graph, having one node for each segment, and one arc for each adjacency relation between segments, the resulting graph must have a number of loops equal to the genus of the surface for the segmentation to be correct.

Chapter 3

Related Techniques

Contents

3.1 Reeb Graphs 61

3.2 Mesh Segmentation 72

In this chapter the main contributions in the subject of Reeb graph and segmentation will be presented.

3.1 Reeb Graphs

Reeb graphs have been extensively applied in recent years in different fields of computer science. Among the main applications of Reeb graphs we recall surface coding (SKK91), modelling interaction between objects (SKSI95), surface compression (BMS00), shape matching (HSKK01), human body segmentation (XSW03; WXS06), mesh deformation (TVD06), mechanical design (PSBM07), (TGSP09) and time analysis of mixed fluids data (COH^+12).

The introduction of Reeb graphs in computer graphics is due to (SKK91) and the first algorithm to automatically compute Reeb graphs is described in (SK91). This algorithm automatically constructs the Reeb graph of a 2D manifold surface embedded in 3D using surface contours, a weight function and an a priori knowledge of the number of holes of the object.

In the following sections we will describe some of the most interesting techniques existing in literature for Reeb graph construction, even if this should not be considered a complete exploration of all the contributions in this matter. A detailed work on the subject of Reeb graphs for shape analysis can be found in (BGSF08).

3.1.1 Level Set Diagrams

In their work (LV99) describe an algorithm that constructs *level set diagrams* (LSD) for 0-genus polyhedrons P using geodesic distance from a *source point*. The *level set diagram* is an embedded graph that encodes both the topology and the geometry of a shape.

This work has several interesting aspects. One of these is the method to compute a *mesh diameter*, in other words the identification of the two *diameter vertices* of a mesh: the couple of most distant vertices in the whole mesh. The algorithm starts choosing a random vertex r on the mesh and then, with the Dijkstra algorithm (Dij59) it computes the farthest vertex from r. The procedure is then iterated, computing s, the farthest vertex on the mesh w.r.t. r. This latter vertex s in then used as the *source point* of a geodesic distance function f computed on mesh vertices with the Dijkstra algorithm.

Other interesting concepts are those of *level sets* and contour strip, that will be taken by (CMEH+03) and then by (TVD08). *Level sets* of f at l (i.e. $x \in P | f(x) = l$) define a set of cross-faces (w.r.t l) that form disjoint cycles, for any value l distinct from the values that the function assumes at the vertices of the polyhedron.

The level set realization for the value l is a polygonal curve $C(l)$, with possible multiple components, that intersects each cross edge $e_{i,j}$ at a point $p_{i,j}$ defined as:

$$p_{i,j} = \frac{l - f(v_i)}{f(v_j) - f(v_i)} v_i + \frac{f(v_j) - l}{f(v_j) - f(v_i)} v_j$$

Ideally the level sets would be one for each vertex on the mesh, but the authors choose to consider a limited number of level sets, dividing the function range $[0, f_{max}]$ in a user-defined number of intervals. They analyse the topology of each considered level set, based on the index of the vertex for

which the level set is computed: they use a slight variation of the index of a point of a scalar function f defined over a polyhedral surface defined in (Ban70) and (Hen94):

$$Ind_f(v) = 1 - \frac{Sgc_f(v)}{2}$$

where $Sgc_f(v)$ is the number of sign changes of $f(v_i) - f(v)$, being v_i the vertices in the link of the vertex v, considered in a counter clockwise order.

Changes in topology of the level set can happen only at critical vertices, those vertices having an index $\neq 0$.

The *LSD* is obtained computing the *centroid* of each polygonal curve $C(l_i)\ \forall l_i \in [0, f_{max}]$. The author choose this solution – instead of computing the centroid over the curve vertices – in order to filter the artefacts caused by non-uniform meshes. The *LSD* algorithm provides a curve for each branch of the *LSD* but does not tell how those branches must be linked together. Centroids become discontinuous each time a contour is split into several contours, for this reason virtual links are added between branches.

The *LSD* algorithm is limited to 0-genus surfaces, and the only topological event that is considered is the *split*, i.e. when a connected component of the level set divides in two. This algorithm does not compute any kind of segmentation. This method has been extended to surfaces with genus higher than 0 in (HA03), and with a technique to find seed points based on a multi-scale curvature evaluation in (MP02b)

3.1.2 Extended Reeb graphs

In (BFS00) *Extended Reeb graphs* are introduced as conceptual models for surface representation applied to terrain models. They are *extended* because, in the original definition of Reeb graph (Ree46) f is required to be Morse. In (BFS00) the original definition is *extended* to elevation functions, also not being Morse or *general*. Initially the triangulated meshes (with boundaries) are virtually closed with a virtual minimum. The algorithm then discovers and reorders (by function values) the critical areas (those containing a non degenerate maximum, a minimum, a saddle but also the flat areas containing non isolated critical vertices), then for each of them an *area of influence* is computed, and finally resulting Reeb graph

nodes are linked by arcs following the adjacency relations between areas.

In (BMS00) an approach based on *Extended Reeb graphs* is applied. The co-domain of the function f is partitioned in $c+1$ intervals, being c the number of non degenerate critical points. Two vertices are equivalent, in the Reeb sense, if the function computed in each of the two vertices belongs to the same interval and if the two points belong to the same connected component of the counter-image of the partition. All the equivalent vertices, in the Reeb sense, can be collapsed in one node corresponding to the whole partition. Arcs of the graph represent equivalence relations. An embedding is also provided, positioning each node in the centroid of the corresponding region.

3.1.3 Sweep algorithm for extracting Reeb graphs of 2-manifold

(CMEH$^+$03) is a fundamental contribution, in that it proves a certain number of results concerning the number of loops of a Reeb graph of a *simple* Morse function f, defined on a smooth, compact, connected, 2-manifold.

3.1.3.1 Topological invariants

Lemma 13 (Loops of a RG of a closed orientable 2-manifold). *The Reeb graph of a Morse function over a connected orientable 2-manifold of genus g without boundaries has a number of loops equal to g.*

For this proof the authors do not make use of the Morse property of the f function, implying that Lemma 13 holds for more *general* but not for arbitrary continuous functions.

Lemma 14 (Loops of a RG of an *orientable* 2-manifold with h boundaries). *The Reeb graph of a Morse function over a connected orientable 2-manifold of genus g with $h \geq 1$ boundary components has a number of loops between g and $2g + h - 1$.*

Lemma 15 (Loops of a RG of a closed, *non-orientable* 2-manifold). *The Reeb graph of a Morse function over a connected non-orientable 2-manifold of genus g without boundary has a number of loops between 0 and $\frac{g}{2}$.*

Lemma 16 (Loops of a RG of a *non-orientable* 2-manifold with h boundaries). *The Reeb graph of a Morse function over a connected non-orientable 2-manifold of genus g with $h \geq 1$ boundary components has a number of loops between 0 and $2g + h - 1$.*

3.1.3.2 The sweep algorithm

The authors then present a *sweep algorithm*, based on the results of the previous lemma, for a compact, triangulated manifold and for a PL Morse function f.
Vertices of the triangulation are processed in *order* of ascending values of the Morse function f, that should be thus sorted globally. The function f, valued at mesh vertices, is also supposed to be *general*. f is extended, via linear interpolation, to the whole mesh surface as f^*. Level sets of f^* are represented by a set of *contour strips*, i.e. piecewise linear paths or cycles. In case of a manifold without boundaries the *contour strips* will be made of cycles only. In the *contour strip* every triangle contains a *line segment* (made of all the points in the triangle that are part of the pre-image $f^{*-1}(\alpha)$) in the path, and any two contiguous triangles meet in an edge that contains a shared endpoint of two *line segments* in the path. This shared endpoint is not a vertex of the triangulation, but is a point on an edge, except when it exists a vertex x such that $f^*(x) = \alpha$.

For each possible value of the function f^* there will be a finite number of contour strips (or contour paths in the case of meshes with boundaries).

The authors propose the following *artifact* in order to represent the contour: each contour (or path) is stored as a list of the edges of the triangulation corresponding to descending paths from contour vertices. The edges carry the vertices of the contours, and the triangles between them carry the edges of the contours.

Each time a critical point is met, a new node is inserted in the Reeb graph and also arcs are updated consequently. Each time a *minimum* is met, a new list (cyclic or linear) of ascending edges is created. Each time a *regular* vertex is met, its descending edges are replaced by its ascending ones. In this case all edges belong to a single list (cyclic or linear). Each time a saddle is met, two cases can arise: two lists merge in a new one or a list splits into two new ones. In this case it is also necessary to find the lists from which the descending edges are removed and to which the ascending

edges are added. In the case a *degenerate* saddle is met, the authors apply the *saddle unfolding* procedure, that transforms a saddle with multiplicity m into m simple saddles. The implementation of the cyclic and linear list is made with *balanced search tree* (see (CLRS09)).
Each time a *maximum* is met, the corresponding list is closed (because there are no more ascending edges for this vertex).

This approach is particularly interesting both because of its efficiency and its capability of dealing with *degenerate* critical points. On the other hand, no *embedding* is given for the graph, nor a segmentation is proposed, even if, for the sweeping nature of this approach, one can easily imagine to integrate both these aspects. The handling of the degenerate saddles is made through a local *remeshing*. Finally a *global* ordering of the function f is required.

3.1.4 On-line computation of Reeb graphs

In (PSBM07) an interesting method for constructing the Reeb graph of very large meshes is described. The method computes also an embedding of the Reeb graph using a number of nodes that depends on the range of the mapping function and on a user defined parameter.
The algorithm has been tested with different functions, showing significant performances with large meshes.

It is an on-line algorithm with an iterative approach that requires taking into account all the simplicial elements of the mesh (vertices, edges and triangles) during the computation: at each step, a new simplicial element is considered and the Reeb graph is incrementally updated, until all simplicial elements have been considered. The only requirement on the input mesh is that all vertices of a triangle must appear before the triangle itself, because vertices and triangles are processed in order of arrival. Initially the Reeb graph is empty. Each time a new vertex is read a corresponding new node in the Reeb graph is created, and each time a new triangle is read, a corresponding new arc is created for each of its edges that has not been already processed. Each time the interior of a triangle is read it could connect disjoint contours already processed and the Reeb graph is consequently updated. The method is based on a tightly coupled data structure between mesh vertices and edges and Reeb graph arcs and nodes. Each node in the graph maintains a pointer to its original vertex, each arc maintains a

reference to all the cross edges for the function value represented by that arc and each mesh edge points back to the higher valued (with respect to f) arc in the graph. The algorithm takes also advantage of the possible mesh manifoldness, removing redundant arcs or nodes in the graphs: e.g. in case of a 2-manifold mesh, considering an edge whose two adjacent triangles has already been processed, it can be removed from the graph reference. This method does not require the input mesh to be manifold, and can be applied to structure of whatever dimension, not only 2 or 3 simplicial complexes. This method does not require the input function to be Morse. The authors show results with different kinds of mapping functions: x, y, z coordinates, a random function, and some eigenvectors of the Laplacian matrix of the input mesh. Furthermore they simplify the obtained Reeb graph using the *extended persistence algorithm* (AEHW06), which generalizes the notion of persistence to loops. Output graphs are stored in a coarser-to-fine format: this method is particularly useful for searching subtle defects on a mesh, because defects disappear at a certain level of simplification, and thus they can be easily identified. The algorithm is also available as VTK implementation (see (vis10)). At our best knowledge this is the fastest and most accurate Reeb graph algorithm available. It does not compute a mesh partitioning.

3.1.5 Enhanced topological skeletons

In (TVD08) an interesting approach is presented in terms of an algorithm for mesh segmentation and Reeb graphs extraction and embedding, resulting in an *enhanced topological skeleton*. This is a sweep method that, differently from (CMEH$^+$03), uses 1-skeleton to represent the level sets of f. This approach is based on three steps: feature points extraction, PL general Morse function computation, Reeb graph extraction.

Feature points are extracted as mesh most prominent points. The algorithm finds the two most distant vertices on the mesh applying the "Source point location" algorithm of (LV99), i.e. the *extrema vertices*.

Then it computes two distance functions δ_1 and δ_2, one for each *extrema vertex*. It finds, for each of the two functions δ_1 and δ_2, their local extrema (maxima and minima). These two sets of local extrema are then merged in a soft way: two extrema belonging to each of the two functions are merged in a

common feature point if they overlap or if they are not more far away than a predefined tolerance ϵ from each other, otherwise they are simply discarded. Two feature points cannot be closer than the same defined tolerance ϵ and thus their number varies in function of this parameter.

The function f is computed as the complement to 1 of the *normalized* (with respect to mesh diameter) geodesic distance from the nearest feature point. f is valued 1 (the maximum value) in the feature points and tends to 0 in the center of the mesh. Since f could present a great number of *minima-saddles* configuration, and could be not *simple*, in the sense described in the chapter 2, the authors suggest a perturbation of f, to guarantee both generality and *minimum-saddle* cancellation. Perturbation requires one mesh swept from the global minimum in order to sort the f values and is performed as: $f = \frac{i}{n}$ where i is the position of the ith value of f. Finally f is made PL Morse via the procedure of *m-saddle unfolding* described in (CMEH+03).

The Reeb graph equivalence classes computation is started at the mapping function minimum, that, after the perturbation step, is unique. The authors adopt an approximated representation of level sets using chains of vertices and edges. This representation is what we call ULS (see Chap. 2, Def. 88). *Contours* are the connected components of the ULS. They sweep the mesh starting from the f minimum. At each step, a candidate vertex v_c is selected, as the minimum in all contours. The corresponding contour is updated, adding $\mathrm{Lk}(v_c) - v_{visited}$, i.e. the set of vertices in the contour that have not already been visited and having an higher value of the mapping function. Variation in the number of the contours lead to *split*, or *merge* events. At the end of the sweep n equivalence classes (in the Reeb definition sense) have been computed, one for each of the n mesh vertices. The sweep procedure keeps track of the creation (at f minimum), split and merge and termination (at f maxima) of the discrete contours, and it appends the correspondent equivalence class to the relative Reeb graph branch. A list of contiguous equivalence classes form a Reeb graph edge. Each edge is connected to two nodes represented by critical points, minimum (starting node) and maxima (termination nodes). A dual, geometrically-embedded representation is then implemented, replacing each Reeb graph edge with the centroid of the corresponding equivalence classes. Edges linking these nodes represent adjacency relations.

In our experience this technique has some unsolved issues (as shown in

Sec. 2.2.5.5) that could lead to critical contour configurations, especially in case of coarse, real-world meshes. Indeed, discrete contours could become non 1-manifold, also at vertices different from the critical ones (see Fig. 2.37), generating self intersections. As shown in Fig. 2.31 this could lead to missing split and merge events detection, and thus to a graph that has not the same topological properties of the RG.

Furthermore, minimum-saddle cancellation procedure in our experience didn't solve extra critical vertices problems, and moreover these problems are more frequent on coarse meshes. Indeed examples provided in (TVD08) are limited to genus 2 mesh, with a great number of vertices.

The split process could happen with high frequency in case of coarse meshes with higher genus, and the corresponding segments could be disconnected. The algorithm needs a global lookup on two heaps, one for the sweeping contours and the other for the visited vertices. This could affect performances in the case of meshes with a great number of vertices.

3.1.6 Reeb graphs based on shape diameter function

In (SSCO08) a method for both mesh segmentation and skeletonization based on *shape diameter function* (SDF) is given. The SDF is computed at each vertex 'sending' several rays inside a cone centered at the vertex, and selecting the one normal to the intersecting surface. The SDF is a measure of the diameter of the shape's volume in the neighborhood of each point on the surface and it is used as a scalar function defined on the mesh surface. Its isocontours are used for segmentation. Skeletons are built choosing a set of points over the mesh as samples, and projecting for each of them, in the inward normal direction, a skeleton point at half the distance of the SDF in that point. Iterating and filtering, a skeleton like structure is obtained.

SDF is robust with respect to rigid body transformation, an also versus deformations that do not alter mesh volume, but may have limitations with non-cylindrical parts of objects.

3.1.7 Dynamic graphs

(DN09) propose an approach for computing Reeb graphs of 3d-manifold with the use of dynamic graphs. Their algorithm uses a sweep approach,

that tracks the evolution of isosurface components while visiting each vertex with a subsequent higher value. Events occur when an isosurface, passing through a vertex, changes the number of connected components. The Reeb graph is updated accordingly during the sweep. The algorithm maintains a *spanning tree* and a *spanning cotree* of the given manifold and generates a *tree-cotree* partition of the manifold itself.

Dynamic graphs work on 3-manifold but do not compute neither a mesh segmentation nor a graph embedding. Furthermore this approach requires a global sorting of vertices values in order to start the computation. Finally the mapping function is required to be Morse.

As in (HSKK01) this work uses AGD (*average geodesic distance*) calculated from a small set of evenly-spaced vertices (*base vertices*) and this choice may possibly lead to inaccurate results with meshes having non uniform sampling.

3.1.8 Reeb graphs built on critical loops

(PSF09) propose an approach for building the Reeb graph of a mesh under a *general* function f (extended by linear interpolation over the whole mesh surface as f^*) using its critical points and their isocontours. To filter the topological noise of the function f (intended as a function with a great number of critical points, close to each other and with close f-values and with possible multiple saddles) the use of two possible methods is proposed: persistency-based simplification as described in (BHEP04) or salient critical points identification as described in (LLKR07).

First of all, the critical points of f are extracted and classified as maxima, minima, and saddles and one isocontour is computed for each of them.

Isocontours $f^{*-1}(f(s))$, for each saddle s, cut the mesh into distinct parts.

Each isocontour self-intersects at its saddle s, generating so many *critical loops* β_k as $m+1$ with m being the multiplicity of the saddle s.

For each saddle s an isocontour is computed starting at s and connecting the interpolated values - on the edges - of f^* of the connected triangles having $f^*(x) = \alpha$, where $x \in e$ and e is a mesh edge so that its extrema a and b have f value so that $f(a) < \alpha < f(b)$.

The process of isocontouring implies a remeshing in the neighborhoods of

the isocontour itself, with a duplication of all the vertices and edges along the loop β_k.

Each contour β_k identifies the set of remeshed triangles that have one edge of β_k as an edge. It divides the mesh in different shells, at the same time pairing triangles belonging to the two different cells with adjacency relations.

An *adjacency graph* A is iteratively built, having a node for each saddle, shell, or critical loop. A geometric embedding is also provided positioning shell and critical loop nodes in the centroids of the corresponding set of mesh vertices. Adjacency graph arcs correspond to adjacency relations between the corresponding mesh shells: each shell has a node in its barycentre and this node is adjacent to the two saddles whose critical loops border the shell. Each saddle is adjacent to all its critical loops.

In the final step, the adjacency graph A is converted into the Reeb graph: the resulting graph has one node for each boundary connected component, one for each critical point (saddles, maxima and minima) and arcs encode adjacency relations between the resulting components. If all the critical loops of two different saddles border the same shell, then the two saddles will be connected with an arc; if two critical loops of two different saddles border the same shell then their barycentre will be connected with an arc, furthermore both of them will be connected to their own saddle. Finally if two critical loops of a saddle and one critical loop of another saddle border the same shell, then there will be an arc between the first saddle and the barycentre of the critical loop of the second one.

As the authors state, this approach is particularly suited for densely sampled large meshes with small genus and really smooth functions (with a small number of critical points). It is worth noting that this method requires a noise filtering procedure at the start and a remeshing procedure during the computation of each isocontour. Finally the segmentation must be necessarily computed with a distinct additional step that sweeps the whole mesh with an adjacency-adjacency relation. The procedure starts at one of the remeshed faces belonging to a particular shell, following then adjacency-adjacency relations, and keeping saddle critical loops as a *not to be crossed* limit, until all mesh faces has been visited.

3.1.9 Other approaches

3.1.9.1 Reeb graphs from LS-graphs

Recently (DN12b) proposed an output-sensitive approach to compute Reeb graphs of d-manifold and non-manifold. Also in this case the Reeb graph is built by first computing the level sets at critical points, and then extracting Reeb graph arcs by traversing the mesh, starting from function minima and reaching previously computed level sets. Their approach is particularly efficient with meshes with a low number of critical points, but the embedding proposed for the graph may arise some considerations, e.g. in terms of symmetry. Also in this case the segmentation can be derived as a further step, and here it is partially manual.

3.1.9.2 Reeb graph as union of contour trees

Recently some contributions has been proposed (TGSP09; DN12a) for computing the Reeb graph of meshes as the union of contour trees (CSA00). These approaches are particularly suited for huge meshes. In particular the (TGSP09) is specifically designed for volumetric meshes embedded in R^3, while (DN12a) can handle manifold and non manifold in any dimension. See (DN12a) for further details on the differences between the two approaches.

(HWW10) and afterwards (DN12a) show time comparison between (PSBM07; TGSP09; HWW10; DN12b; DN12a), explaining specificities and limitations of each of the described methods.

3.2 Mesh Segmentation

Mesh partitioning is considered to be still an open problem (Sha08). There is not a unique solution, but there exist different solutions, each tailored on a different problem. A recent work (LVB+12) proposed the evaluation of different algorithms with respect to a unified benchmark database, the SHREC database (BVLD09). This tool has been proposed in 2009 and since then it stimulates and supports discussion and confrontation about the segmentation problem.
In a previous work (AKM+06) proposed seven criteria to evaluate different segmentation algorithms:

- *Type of segmentation*: part-type, surface-type.
- Extracting the *correct* segments: this is not an objective parameter and establishing the correctness of a boundary is strongly dependent on application and can often be done looking at the images.
- *Boundaries*: even if this parameter is not stricly objective, it is possible to define requirements as smoothness, length and location along concave features.
- *Hierarchical / multi-scale segmentation*: to fit possible different user requirements.
- *Sensitivity to pose*: whether a change in the pose of the shape can affect its segmentation.
- Sensitivity to *noise* and *tesselation*: robustness with respect to noise and mesh sampling.
- *Asymptotic complexity*: this can give an idea of the running time in the worst case.
- *Parameters*: whether an algorithm requires user controlled parameters, or previous knowledge of the mesh.

(Sha08) identifies the following categories of segmentation methods:

- **Region growing**: this method starts at a seed and expands the segment until a certain condition is reached. This kind of technique is appliyed with some variations in (CDST97; KT96; MPS+04).
  This method has also a variation, called *multiple source region growing* in which more than one seed is chosen at start.
  It is applied by (SCOGL02; ZH04) among others.
- **Hierarchical clustering**: in this method possible couples of elements are tested for being partitioned in the same cluster. Initially each element has its own cluster, then clusters are merged, applying a cost function each time a merge is done. This approach is adopted in (GWH01; AFS06). In this latter work a set of primitives is used (planes, spheres and cylinders) in order to segment the mesh.

- **Iterative clustering**: differently from *region growing* and *hierarchical clustering*, in *iterative clustering* the number of resulting patches is one of the input parameters. The optimization algorithm tries to converge to a best solution measuring at each step the *best representative* for a partition. This technique is applied with some variances by (STK02; Kob05). In the first algorithm one of the main decisions the procedure has to take is about whether two faces belong to the same patch or not. A cost function, based on the distance from each two faces is evaluated and, at each step, faces are exchanged from one patch to another till when all the faces in a given patch result to be connected. Resulting patches could be or disc-like or cylinder-like. In the second algorithm patches type are extended to planes, spheres, cylinders and rolling ball blend patches.

- **Spectral analysis**: given a graph G, we define its adjacency matrix A as:

$$A_{ij} = \begin{cases} 1, & \text{if } i,j \text{ are neighbors.} \\ 0, & \text{otherwise.} \end{cases}$$

and the diagonal matrix D which is made of d_i, the degree of vertex i, then the Laplacian of G is defined as the matrix $L = D - A$.

$$L_{ij} = \begin{cases} 1, & \text{if } i = j. \\ -\frac{1}{d_i}, & \text{if } i,j \text{ are neighbors.} \\ 0, & \text{otherwise.} \end{cases}$$

(KG00) uses this technique for compressing meshes, adding a preprocessing step in which the mesh is segmented in parts.
In (LZ04) spectral clustering is applied to 3D mesh segmentation, introducing the concept of an *affinity matrix* that encodes whether two faces belong to the same cluster, based on their distance:

$$W(i,j) = \frac{1}{e^{\frac{\delta(i,j)}{2\sigma^2}}}$$

In a succeeding work (LZ07) propose a mesh segmentation algorithm based on spectral analysis via recursive bisection of the original mesh. In this case they distinguish between *structural segmentability* and

geometrical segmentability, introducing two different affinity matrix operators for the spectral projection done at each step.

- **Implicit methods**: (Sha08) defines these kinds of approach as the ones that create a mesh partition as a by-product of other techniques, like boundary or skeleton extraction.
  Some of these approaches are based on constructing boundaries, like (LLS+05). In this work the authors propose to scissor the mesh with contours in correspondence of features, identified with *minima rule* and *part salience theory*. This method is robust in practice, but feature extraction could produce overhead in case of huge meshes. In this case the authors propose the use of a simplified version of the mesh for feature extraction. (PSF09) identify contours that are the boundaries of the segmentation in correspondence of the critical points of a PL-function defined over the mesh.

  (KLT05) propose a coarse to fine approach to segmentation, based on *multi-dimensional scaling*, feature points detection and core component extraction. Feature points are defined as witnesses of prominent components, and as the points more distant to all other points in the mesh. Once both feature points and core component are identified, the coarse mesh is segmented and finally the cuts and the segments are refined in the original high-resolution mesh.

  In (LKA06) is described an iterative algorithm that computes both the mesh skeleton and the segmentation. The heart of the recursive procedure goes as follows: a skeleton of the mesh is computed using a *principal axis*, then the quality of the skeleton is evaluated measuring the *convexity* of the corresponding component. If the test is passed the skeleton is returned, otherwise the mesh is further segmented with *approximated convex decomposition*.

  (SSCO08) propose an algorithm for mesh decomposition and skeletonization using a shape diameter function (SDF), i.e. a scalar function defined on mesh faces (see also Sec. 3.1.6). This method consists of a pre-processing step (for SDF calculation, normalizing and smoothing), then of a step in which isovalues of the SDF create iso-contours for a first soft partitioning of mesh faces and finally a step in which actual partitioning is found using k-way graph-cut.

  In (BDBP09) a method for mesh segmentation based on Reeb graph

is presented. Reeb graph is built over the *average geodesic distance* AGD mapped over the mesh surface. Formally the AGD is, at each vertex, the average geodesic distance of the vertex itself to all the other points on the model surface. But the authors suggest a simplification, computing the AGD from a small number of evenly spaced vertices (*base vertices*) of the surface. The values of the function f are divided in intervals. This algorithm takes in consideration, at each step, the number of connected components associated with each interval. Contiguous intervals with the same number of connected components are fused together. The process is iterated until all the different intervals have a different number of connected components. A further step is required in order to adjust segmentation boundaries with respect to deep surface concavities.

Chapter 4

The DRGSS algorithm

Contents

4.1 Computing the SRG and the segmentation 78

4.2 Scalar function 95

4.3 Implementation 96

In this chapter the theory illustrated in chapter 2 will be applied to introduce the *discrete Reeb graph and segmentation algorithm* (DRGSS), an algorithm that builds both the Reeb graph (in this case a *simplified Reeb graph* - SRG) and the corresponding segmentation.

Here it will be shown how to implement an upper level set (ULS), augmented with edges and vertices multiplicity and how to build an SRG based on the evolution of the *augmented ULS*. This guarantees that the resulting graph will be homeomorphic to the CSTRG and thus to the Reeb graph of f^* (BFS00).

The focus of this work is not on the particular function f received as input, it is enough for it to be *general*.

4.1 Computing the SRG and the segmentation

Algorithm 4.1.1: REEBSEGMENTATION(T, f)

```
main
 Γ ← ∅
 Σ ← ∅
 V_minima ← minima(T, f)
 for each v_min ∈ V_minima
   do { comment: γ is a multiset of vertices and edges
        σ_min ← new σ({v_min})
        γ ← new γ({v_min})
        σ(γ) ← new σ(∅)
        Γ ← Γ ∪ {γ}
        Σ ← Σ ∪ {σ_min} ∪ {σ(γ)}
        adjacency(Σ) ← adjacency(Σ) ∪ {(σ_min, σ(γ))}

 while (Γ ≠ ∅)
   do { v_c ← arg min_{v∈Γ} f(v)
        Γ_c ← {γ ∈ Γ | v_c ∈ γ}
        if (isMULTIPLE(v_c, Γ_c))
          then MERGESPLIT(v_c, Γ_c)
          else { ADVANCECONTOUR(v_c, Γ_c(1))
                 if not (isEmpty(γ))
                   then σ(γ) ← σ(γ) ∪ {v_c}
                   else { Γ ← Γ − {γ}
                          σ_max ← new σ({v_c})
                          Σ ← Σ ∪ {σ_max}
                          adjacency(Σ) ← adjacency(Σ) ∪ {(σ(γ), σ_max)}

 output (Σ)

procedure isMULTIPLE(v_c, Γ_c)
 if (|Γ_c| > 1) or (∃γ ∈ Γ_c | multiplicity(v_c, γ) > 1)
   then return (true)
   else return (false)

procedure ADVANCECONTOUR(v_c, γ)
 γ ← γ − St(v_c, γ)
 γ ← γ ∪ (Lk+(v_c, f) − Lk(v_c, γ))
```

The algorithm that will be described is modular in that it receives as input a mesh in PLY format and it returns as output the resulting SRG and its

corresponding segmentation.

The proposed algorithm uses the f function valued at mesh vertices, when available, having as only requirement on the function its generality. The algorithm can also compute an f function, if this is not given as input.

4.1.1 The main algorithm

The main algorithm (see Algorithm 4.1.1) takes as input a triangulated mesh X and a *general* function f. We choose, for simplicity, to present the DRGSS algorithm for an *upward* sweep of T, i.e in the direction of ascending values of f. The definition of a dual algorithm, that goes in the opposite direction, follows immediately.

Initially the set of the evolving contours Γ and of the segments Σ are both empty. The local minima V_{minima} of the function f are found and a segment is created for each of them and stored in Σ.

In the same iteration, for each minimum of the function f, a *contour* is created and stored in Γ. Each contour is assigned an empty segment, that is marked adjacent to the segment in Σ containing the corresponding minimum.

The main iteration is repeated until all the evolving contours have vanished. At each step the vertex v_c, having the lowest value of the function f in all contours, is selected as the *candidate vertex*; if v_c has *multiplicity* $m > 1$ in Γ (see Sec. 4.1.2.1), the MERGESPLIT operation is performed (see Sec. 4.1.3), otherwise the contour containing v_c is updated and v_c is added in the segment of its contour (see Sec. 4.1.2).

The contour γ just updated, will be empty in case v_c is a maximum. In this case indeed v_c does not have neighbors with an higher value of f, and it is also, by construction, the minimum of its contour: any contour thus vanishes at maxima. If it would be the case, γ is removed from the set of the *evolving contours* and a new segment σ_{max} is created containing the maximum and this segment is set to be adjacent to $\sigma(\gamma)$, the segment of the contour γ that has vanished.

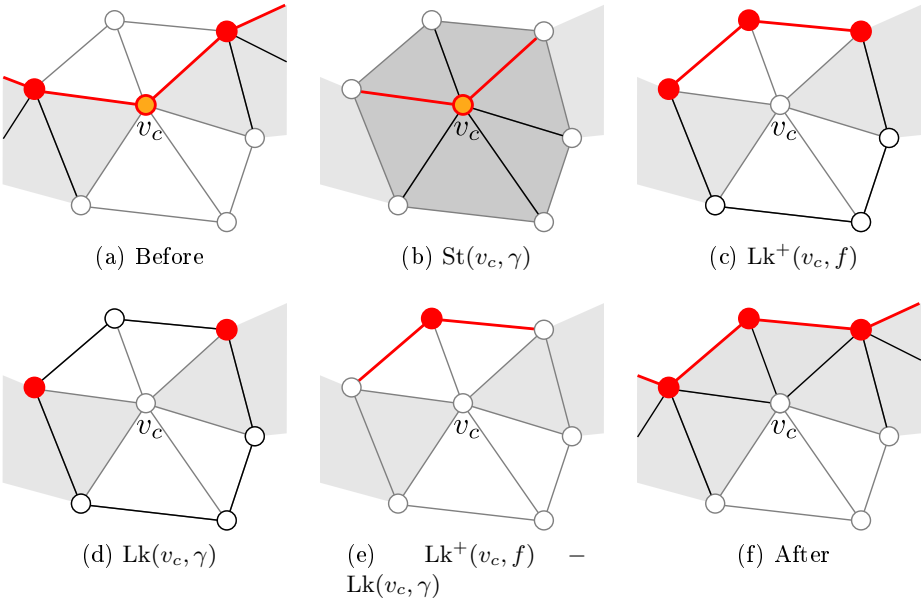
The main loop ends when Γ is empty, that is when all the evolving contours have met a maximum (see Sec. 4.1.5).

At the end Σ will contain all the mesh segments and their adjacencies:

the *SRG* will have a node for each segment and an arc for each adjacency relation.

4.1.2 Advancing Contours

In order to describe how the contour evolution is done we need to recall some definitions from Chap. 2: $\mathrm{St}(v_c, \gamma)$ (see Def. 103) is the St of v_c on the contour γ and it is shown (in red) in Fig. 4.1b, $\mathrm{Lk}(v_c, \gamma)$ (see Def. 104) is the Lk of v_c on the contour γ and it is shown (in red) in Fig. 4.1d.



(a) Before (b) $\mathrm{St}(v_c, \gamma)$ (c) $\mathrm{Lk}^+(v_c, f)$

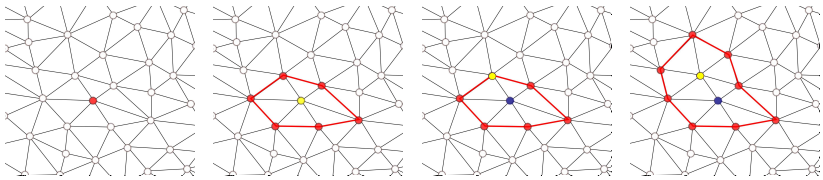
(d) $\mathrm{Lk}(v_c, \gamma)$ (e) $\mathrm{Lk}^+(v_c, f) - \mathrm{Lk}(v_c, \gamma)$ (f) After

Figure 4.1 – This figure is the same of Fig. 2.39, it has been repeated for convenience only. It shows the contour update for a *regular* vertex v_c: starting from the initial contour (a), the figure shows the details of the computation.

Finally we recall the definition of $Lk^+(v_c, f)$: it is the Lk^+ of the vertex v_c on the mesh (see Def. 67 in Chap. 2). An example of $\mathrm{Lk}^+(v_c, f)$ is shown (in red) in Fig. 4.1c.

Given these definitions we can now introduce the method by which contours are updated. Recalling Def. 105 in Chap. 2:

$$\gamma_{upd}(v) = \gamma(v_c) - \mathrm{St}(v_c, \gamma(v_c)) + (\mathrm{Lk}^+(v_c, f) - \mathrm{Lk}(v_c, \gamma(v_c)))$$



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

Figure 4.2 – Contour evolution starts at minima (a), at each step a candidate vertex is selected (orange) (b,c) as the one with the lowest value of f in all contours. All contours evolve in the direction of ascending values of the f function (d).

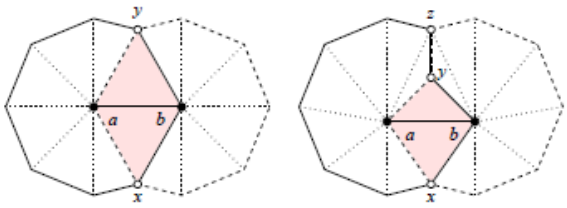


Figure 4.3 – Possible configurations of Lk between two vertices (EH10).

We can see that contour evolution involves only the immediate neighborhood of the *candidate vertex* v_c: the $\mathrm{St}(v_c, \gamma)$ is replaced by $\mathrm{Lk}^+(v_c, f) - \mathrm{Lk}(v_c, \gamma)$.

Fig. 4.1 shows an example of this basic step. Fig. 4.1a shows the initial contour, with the candidate vertex v_c in orange. Fig. 4.1b shows $\mathrm{St}(v_c, \gamma)$ that is removed from the contour, Fig. 4.1c and 4.1d show respectively $\mathrm{Lk}^+(v_c, f)$ and $\mathrm{Lk}(v_c, \gamma)$ whose difference (shown in Fig. 4.1e) is added to the contour. Finally Fig. 4.1f shows the updated contour.

The vertex v_c is also inserted in the segment $\sigma(\gamma)$ after the contour evolution. Figure 4.2 describes the first stages of the evolution of a contour after having been initialized, at the beginning of the algorithm, for a specific *minimum*.

Fig. 4.3 shows a special case in a triangulation. This is the case in which a mesh contains a *tetrahedron* (Ede01). Formally this configuration happens

when, given two neighboors vertices a and b:

$$\mathrm{Lk}(a) \cap \mathrm{Lk}(b) \neq \mathrm{Lk}(a) \cap \mathrm{Lk}(b) \cap \mathrm{Lk}(\overline{ab})$$

where $\mathrm{Lk}(\overline{ab})$ is the Lk of the edge ab. When these configurations are found, particular care must be exerted in contour management.

Thanks to the properties explained in sections 2.2.5 and 2.2.6, the contour evolution operation does not require looking up in a collection of visited vertices, as it happens in other evolving contour methods (TVD08). It uses only the immediate neighborhood of the candidate vertex, or, only in case it is a saddle, the ULS and the contour strip, as we will see in Sec. 4.1.3.1.

It can happen that a candidate v_c is a local *maximum*: in this case the operation $\gamma \leftarrow \gamma \cup (\mathrm{Lk}^+(v_c, f) - \mathrm{Lk}(v_c, \gamma))$ in ADVANCECONTOUR is resulting in an *empty* contour γ. Consequently γ is simply removed from the evolving contours.

4.1.2.1 Contours as Multisets

As shown in Sec. 2.2.6.1, a key point of our algorithm, is the introduction of an *augmented* representation of the ULS, in which edges and vertices have a multiplicity $m \geq 1$. More precisely, each *upper contour* is a *multiset* (Knu98) that contains vertices and edges, each associated to a value of multiplicity describing their *presence* in the *upper contour*.

Multiplicity greater than 1 arises during contour evolution (see Fig. 4.10a), when the vertices and the edges that have been added to the contour are already present either in the contour itself or in other contours (see Fig. 4.10c).

If this would be the case, in the ADVANCECONTOUR procedure the values of multiplicity for vertices and edges are updated accordingly.

Multiplicity has a fundamental role in detecting contour events: if the candidate vertex v_c has $m > 1$ it will certainly be a saddle (see Sec. 2.2.6.1), because v_c is also the minimum of $\gamma(v_c)$ w.r.t. the values of f. Only in this case the algorithm will walk the contour γ_{tmp} in order to identify contour connected components. On the other side, because of multiplicity of vertices and edges in the contour, identifying contour connected components must be done with particular care (see Algorithm 4.1.3).

4.1.3 Merge and Split of contours

As described in Algorithm 4.1.1, split and merge events detection is a local task that depends only on the multiplicity of the candidate v_c in γ, or in the other evolving contours.

Algorithm 4.1.2: MERGESPLIT(v_c, Γ_c)

$\gamma_{tmp} \leftarrow$ new $\gamma(\emptyset)$
$\sigma_{saddle} \leftarrow$ new $\sigma(\{v_c\})$
$\Sigma \leftarrow \Sigma \cup \sigma_{saddle}$
for each $\gamma_c \in \Gamma_c$
do $\begin{cases} \gamma_{tmp} \leftarrow \gamma_{tmp} \cup \gamma_c \\ adjacency(\Sigma) \leftarrow adjacency(\Sigma) \cup \{(\sigma(\gamma_c), \sigma_{saddle})\} \end{cases}$
ADVANCECONTOUR(v_c, γ_{tmp})
$\Gamma_n \leftarrow$ CONNECTEDCOMPONENTS(γ_{tmp})
for each $\gamma_n \in \Gamma_n$
do $\begin{cases} \Gamma \leftarrow \Gamma \cup \{\gamma_n\} \\ \sigma(\gamma_n) \leftarrow \text{new } \sigma(\{v \in \gamma_n\}) \\ \Sigma \leftarrow \Sigma \cup \sigma(\gamma_n) \\ adjacency(\Sigma) \leftarrow adjacency(\Sigma) \cup \{(\sigma_{saddle}, \sigma(\gamma_n))\} \end{cases}$

In the ISMULTIPLE procedure in Algorithm 4.1.1 the multiplicity of the candidate vertex is checked: if the candidate vertex has multiplicity $m > 1$ in Γ then it certainly is a saddle, indeed it is, by construction, also the minimum in its contour γ.

The Algorithm 4.1.2 can be thought as made of three sub-parts:

- the first one merges all the contours that share v_c as a candidate. Whenever the candidate vertex v_c is simultaneously present in more than one contour (i.e. $|\Gamma_c| > 1$) then a *merge event* occurs. Figure 4.6 describes the merging of two contours;
- the second part advances the (possibly) merged contour γ_{tmp} in the direction of ascending values of the function f;
- the third splits the resulting contour in its connected components. Indeed if v_c has a presence greater than one in its contour, then a

split event will surely occur.

When f is a Morse function, split and merge events cannot occur simultaneously. Furthermore, every split event will produce just two new contours and every merge event will merge just two contours into a single one.

However, since f is only required to be *general* it can happen that multiple merge or multiple split take place at a saddle, also simultaneously. In particular, this happens in the case of a *degenerate* saddle.

At first, the Algorithm 4.1.2 creates both a new temporary empty contour γ_{tmp}, and a new segment σ_{saddle} containing only the saddle v_c. This segment is stored in Σ with all the other segments. Then, in the case of a *merge* (i.e. when v_c belongs to more than one contour), contours sharing v_c are merged in γ_{tmp}, and each corresponding segment $\sigma(\gamma_c)$ is set to be adjacent to the segment σ_{saddle} just created. Then γ_{tmp} is advanced, as described in Sec. 4.1.2.

Then the procedure identifies the connected components of the evolved contour (see Sec. 4.1.3.1). For each connected component a new contour and a new segment are initialized: the new segment is initialized with the vertices of the new contour ($\sigma(\gamma_n) \leftarrow$ new $\sigma(\{v \in \gamma_n\})$. These initialization guarantees segment proper connectivity also in case of multiple and frequent split or merge events (as it will be described in Sec. 4.1.4.1). The adjacency relations in Σ are also updated: σ_{saddle} is set to be adjacent to each of the $\sigma(\gamma_n)$ just created.

Overall, this means that the nodes in the SRG corresponding to a *degenerate* saddle will have connectivity greater than 3.

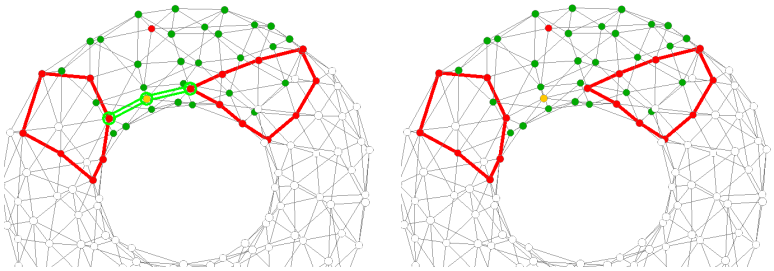
4.1.3.1 Identifying connected components

Algorithm 4.1.3: CONNECTEDCOMPONENTS(γ_{tmp})

main
$adjacents \leftarrow \text{Lk}(v_c, \gamma_{tmp})$
while $(adjacents.size > 0)$
do
$v_u \leftarrow adjacents[0]$
$\gamma_n \leftarrow$ new $\gamma(v_u)$
$\Gamma_n \leftarrow \{\gamma_n\}$
$v_l, v_{startLower} \leftarrow v_c$
$v_p, v_n \leftarrow (Lk(v_u, f) \cap Lk(v_l, f))$
$adjacents \leftarrow adjacents - \{v_p, v_n\}$
ADVANCEANDUPDATE($\gamma_n, v_n, v_p, v_l, v_u, adjacents, edge_{cur}$)
while (**not** ISCLOSEDCYCLE($v_u, v_n, v_l, v_p, v_{startLower}$))
do
INSERTEDGE($\gamma_n, edge_{cur}$)
REMOVEEDGEANDCLEANVERTICES($\gamma_{tmp}, edge_{cur}$)
$v_n \leftarrow$ OTHERFACEVERTEX(v_p, v_l, v_u)
ADVANCEANDUPDATE($\gamma_n, v_n, v_p, v_l, v_u, adjacents, edge_{cur}$)
INSERTEDGE($\gamma_n, edge_{cur}$)
REMOVEEDGEANDCLEANVERTICES($\gamma_{tmp}, edge_{cur}$)
output (Γ_n)

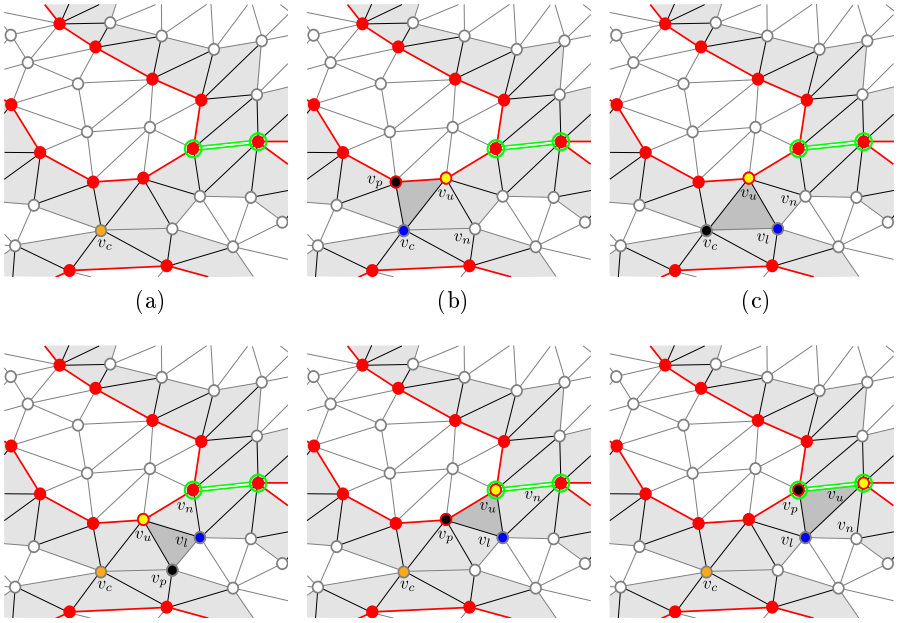
procedure ADVANCEANDUPDATE($\gamma_n, v_n, v_p, v_l, v_u, adjacents, edge_{cur}$)
if $(v_n \in \gamma_n)$
then $v_p \leftarrow v_u$; $v_u \leftarrow v_n$
else $v_p \leftarrow v_l$; $v_l \leftarrow v_n$
$edge_{cur} \leftarrow$ EDGEBETWEEN(v_p, v_u)
$adjacents \leftarrow adjacents - \{v_n\}$

Determining the connected subsets Γ_n of the splitting contour γ_{tmp} involves *walking* the contour across vertices and edges with possible multiplicity $m > 1$. Figure 4.4 shows a relatively simple case in which, after updating the contour for the candidate vertex v_c, the two subsets are clearly separated and easy to follow. The multiplicity of both edges and vertices can make the problem more complex. In particular, in coarse meshes with higher genus it is not infrequent to have contours that have more than one connected subset of vertices and edges with multiplicity $m > 1$.



(a) (b)

Figure 4.4 – Split event: the candidate vertex v_c (in orange) has a multiplicity $m > 1$ in its contour (a); the contour is split into two distinct ones (b).



(a) (b) (c)

(d) (e) (f)

Figure 4.5 – Identifying connected components in a contour after having advanced contour γ_{tmp} (a). Starting from $v_c = v_l$, the three vertices v_u, v_l and v_p identify the current face in the contour strip (in gray) and allow walking across vertices and edges with *multiplicity* greater than 1.

The situation is described in Figure 4.5, where the vertices and edges with double presence (in green) do not include the splitting point, i.e. the candidate vertex v_c. As shown in Figure 4.5a, the problem is that the procedure must walk through the subset with double presence by entering and exiting 'on the same side' of the contour, i.e. without crossing over.

While all other operations are done manipulating only the contour γ, (a connected component of the ULS), in order to find the connected components of a splitting contour also the *contour strip* is required, since the upper level set does not contain, per se, the required information.

The method adopted starts from the candidate vertex v_c, which at this point is, by construction, outside γ_{tmp} (Fig. 4.5b) and has a lower value with respect to each other vertex in γ_{tmp} and also belongs to $\mathrm{St}(v_c)$. The current face will be identified by a triplet of vertices v_u, v_l and v_p in which initially $v_l = v_c$ and v_u must be in γ_{tmp}. In particular v_u is initially one of the vertices of $\mathrm{Lk}(v_c, \gamma_{tmp})$. From this point on, the method is a 'walk' along the contour strip: at each step, the next face will be the unique one that shares the edge (v_u, v_l) and is opposite to v_p. The delicate part of the method is labelling the vertices of the next face: if the vertex v_n in the next face that is opposite to v_p belongs to the contour γ_{tmp} (Fig. 4.5d), then the new labelling is:

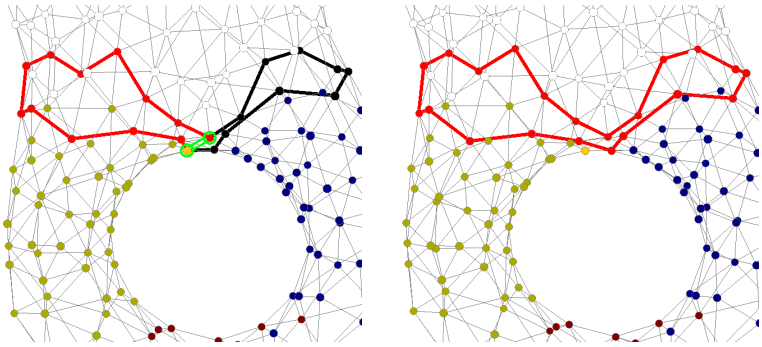
$$v_p = v_u,\ v_u = v_n,\ v_l = v_l.$$

Otherwise (Fig. 4.5c), i.e. if v_n does not belong to γ_{tmp}, the new labelling is:

$$v_p = v_l,\ v_l = v_n,\ v_u = v_u.$$

The 'walk' is completed, and one connected component has been identified, when it returns back to v_c, more precisely when v_l comes to coincide with v_c again. Another 'walk' is then performed starting from another, unexplored face in $\mathrm{St}(v_c)$ that also belongs to the contour strip, until there are no more such unexplored faces.

It has to be emphasized that these 'walks' over the connected components of the contour strip are performed in the algorithm only when a *saddle* vertex is met.



(a) (b)

Figure 4.6 – Merge event: the candidate vertex v_c (in orange) belongs to two distinct contours (a); the two distinct contours are merged into one (b).

4.1.4 Segmentation

As described in section 4.1.1, each contour has an associated segment, that contains the set of vertices visited by the contour. In detail, in the initialization step, the procedure creates a segment for each function *minima* and one empty segment associated to each contour. During contours evolution, each segment is progressively filled adding at each step the candidate vertex in its associated contour. When a saddle is met, a new segment is created containing the saddle, the corresponding contour is removed from Γ (the set of the evolving contours) and one or more new contours are created, with their associated segments. Clearly, segments are also created when the corresponding contour meets a maximum. The overall mesh partition is described by the collection of all the segments in Σ.

4.1.4.1 Multiple membership

When contours pass by a critical vertex, the corresponding *event* (split, merge, contour creation, contour end) is managed by the algorithm. In the same procedure also the *parent* contours are removed from the set of the evolving contours. As a consequence their associated segments are *closed*, in the sense that new vertices will no longer be added to these segments. This means also that segments are delimited by the level lines of the extended function f^* passing through a critical vertex. The resulting segmentation

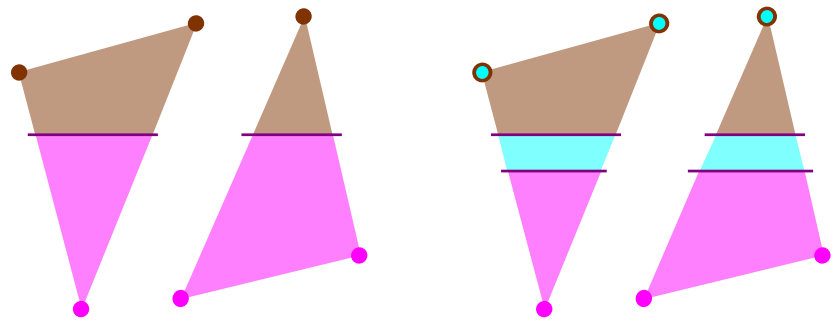


Figure 4.7 – A cross face between two segments (a), a cross face between three segments (b).

depends on the values of the function f but also on the *density* of the triangulation X, intended, informally, as the number of vertices per unit of surface. In general, in case of a dense triangulation each face is traversed by at most one segment boundary (see Fig. 4.7a), making the assignment of vertices to segments unique and non-ambiguous.

In the case of a *low density* triangulation, the boundaries delimiting the segments are very close to each other and it can happen that some faces are traversed by more than one segment boundary (see Fig. 4.7b). These kind of faces are *multiple* faces, in the sense that they are shared between more than two segments. In this case, if the unique assignment of vertices to segments is maintained, the resulting segmentation would often produce disconnected patches over the triangulated surface.

One possible solution to this problem is remeshing: *multiple* faces are unfolded creating new vertices that increase locally the mesh density and restore the connectedness of the corresponding segments.

We propose here a different solution: in Algorithm 4.1.2 we allow the vertices that are part of a *multiple* face to be assigned to all the segments that share that face. Let's consider a cross face for the values $f^*(v_{c_i})$ (where v_{c_i} are critical vertices and $i \in [0, n]$ and $n \geq 1$). Only those vertices in a multiple face having a function value higher than each of the $f^*(v_{c_i})$ are assigned to all the segments σ_i having one of the $f^*(v_{c_i})$ as lower boundary. In Algorithm 4.1.2 this is obtained at the creation of new segments, with the code:

$$\sigma(\gamma_n) \leftarrow \text{new } \sigma(\{v \in \gamma_n\})$$

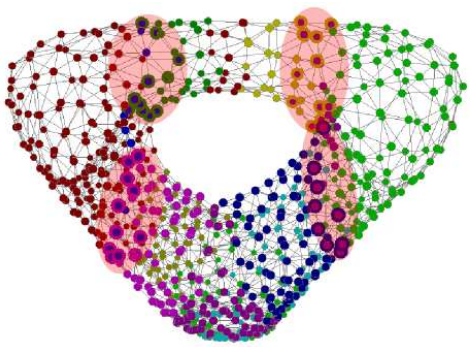


Figure 4.8 – Vertices with multiple membership are shown in the highlighted areas as having a different border color.

This means that all the vertices in the connected component γ_n are assigned to the corresponding segment. In this way each segment will be guaranteed to be *connected*, by construction.

In case a unique assignment of vertices to segments is required, the above line should be modified in:

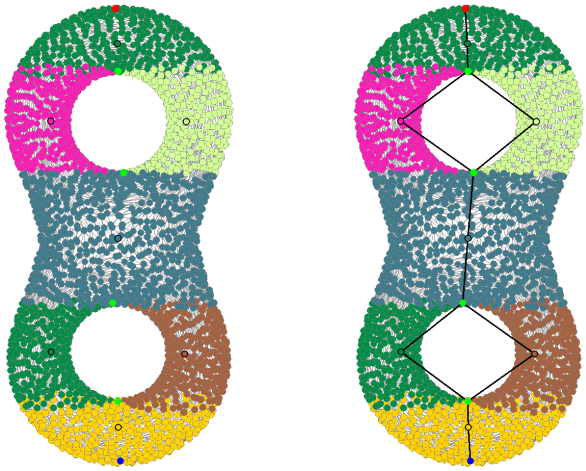
$$\sigma(\gamma_n) \leftarrow \text{new } \sigma(\{\emptyset\})$$

In this case each non-minimal vertex will have to 'wait' until it is selected as v_c in Algorithm 4.1.1 before being (uniquely) assigned to a segment. In contrast, in the method proposed, the initialization of a new segment with the vertices of γ_n guarantees the connectedness also making the multiple cross-faces easily identifiable.

Figure 4.8 shows an example of *multiple membership* of vertices in segments. Vertices with multiple membership are shown as having a border color that differs from the inside: the inside color is that of the first segment they have been assigned to and the border color is that of the second segment to which they have been assigned. Higher level of sharing (e.g. vertices shared between three or more segments) can occur in practise.

4.1.5 Constructing the Reeb Graph

In the output graph resulting from the application of the DRGSS algorithm, nodes correspond to the centroids of the identified segments and



(a) (b)

Figure 4.9 – The centroid of each segment is a node (a), arcs describe adjacency between segments (b).

arcs represent the adjacency relations between segments, as described in Algorithms 4.1.1, 4.1.2 and 4.1.3.

The graph is built incrementally, during contours evolution. In particular, every time the contour of a *regular* candidate vertex v is updated, the vertex v is stored in the corresponding segment. Every time a split or merge event occurs at a vertex v, the parent segments are declared adjacent to the segment containing the saddle v, and the same holds for the offspring segments. Figure 4.9 shows an example output graph.

The output graph is a *simplified Reeb graph* (SRG), in that each node that corresponds to the centroid of a segment represents the aggregation of a set of distinct contours, each corresponding to a *regular* vertex in the mesh. These *regular* nodes have connectivity 2. Each *critical* segment (a segment that contains a critical vertex alone), corresponds to a graph node too. In the case of a maximum or a minimum it will have connectivity 1, in the case of a saddle it will have connectivity equal to $m+1$, being m the saddle multiplicity.

Computing the number of loops To validate the properties of the Reeb graphs obtained, we used the procedure described in (SAA09) to compute

the *minimum cycle basis* and hence the *number of loops* in the obtained graph (see Algorithm 4.1.4).
The number of loops must be equal to the genus of the corresponding mesh, which can be computed with the Euler equation (see Def. 51 and 80 in Chap. 2):

$$v - e + t = 2 - 2g$$

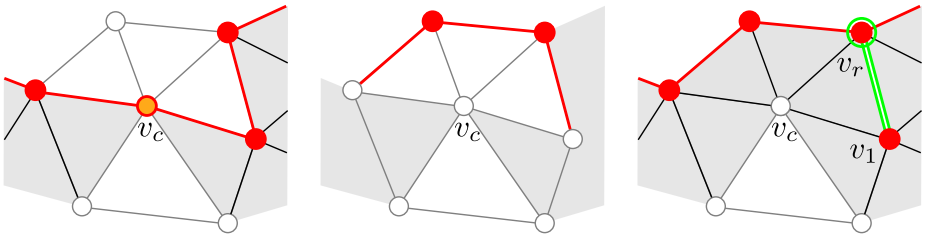
where v is the number of vertices in the mesh, e is the number of edges and t is the number of triangles.

Algorithm 4.1.4: FINDGRAPHNUMBEROFLOOPS(*graph*)

$edgeListSize \leftarrow graph.edgelist.size$
$V_{toBeVisited} \leftarrow \emptyset$
$V_{visited} \leftarrow \emptyset$
$E_{visited} \leftarrow \emptyset$
$cycleCount \leftarrow 0$
$v_c \leftarrow reebGraph.vertexlist[0]$
while $size(E_{visited}) \neq edgeListSize$
do
$V_{visited} \leftarrow V_{visited} \cup v_c$
for all $edge : \textbf{\textit{neighborhood}}(v_c)$
if $edge \notin E_{visited}$
$otherVertex \leftarrow \textbf{\textit{getOtherVertex}}(edge, v_c)$
$E_{visited} \leftarrow E_{visited} \cup edge$
if $((otherVertex \notin V_{visited})$ **and** $(otherVertex \notin V_{toBeVisited}))$
$V_{toBeVisited} \leftarrow V_{toBeVisited} \cup otherVertex$
else
$cycleCount \leftarrow cycleCount + 1;$
if $size(V_{toBeVisited}) \geq 0$
$v_c \leftarrow V_{toBeVisited}[0]$
$V_{toBeVisited} \leftarrow V_{toBeVisited} - v_c$
return $(cycleCount)$

4.1.6 Removing folds: saddle-maximum cancellation

Edges and vertices with multiplicity $m > 1$ appear during the basic contour evolution (see Fig. 4.10). We define a *fold* the set of simplices made up of a connected sequence of contour vertices and edges with $m = 2$ terminating in one vertex with $m = 1$. In Fig. 4.10c the fold is made up of the edge with $m = 2$ and its two face vertices: the one with $m = 2$ is the *fold root* vertex v_r (the vertex with $m = 2$ in which it arrives one fold edge with $m = 2$,



(a) Before advancing (b) $\mathrm{Lk}^+(v_c, f) - \mathrm{Lk}(v_c, \gamma)$ (c) After advancing

Figure 4.10 – In this case both a vertex and an edge with *multiplicity* 2 are produced while advancing the contour. The *contour strip* (in gray) is shown for convenience only, since it is not explicitly computed.

and from which other two edges with $m = 1$ depart), the other with $m = 1$ is the *fold terminating* vertex v_1.

If one of the vertices of the fold with $m = 2$ is the minimum in the contour (i.e. it is a saddle) a split will be generated, resulting in a *tiny* segment, containing a thin strip of vertices on a 'ridge' and in a graph with a *noisy* branch.

When a fold is met it would be an excellent opportunity to implement a *saddle-maximum cancellation*, indeed it is easy to proof that such a piece of contour cannot generate further *merge* events, changing the number of loops of the resulting graph (see Fig. 4.10c).

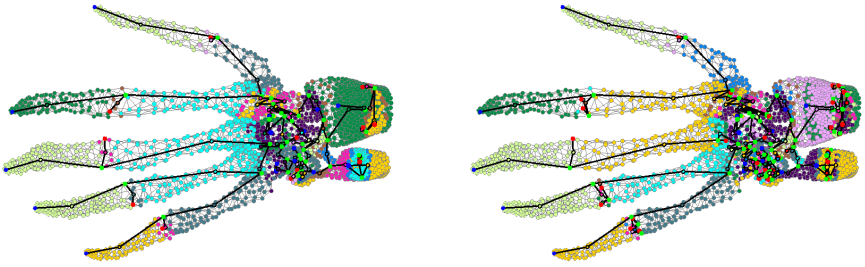
As an optional, optimization step in the DRGSS algorithm, we implement this *saddle-maximum cancellation* in that we reorder the value of f for the fold vertices, in ascending order, from the *fold root* to the *fold terminating* vertex.

The procedure takes as *maximum value*, the value of the *fold root*, and as *minimum value* :

$$f_{min} = f_{v_{neigh}} + \epsilon$$

where $f_{v_{neigh}}$ is the function value of the vertex having higher value of f between all the vertices in the 1-neighborhood of the fold vertices (the root excluded).

Then all the n fold vertices are labelled from $i = 1$ for the *fold root*, to $i = n$



(a) SRG has 144 nodes (b) SRG has 248 nodes

Figure 4.11 – The hand (genus 5) with the saddle-maximum cancellation has a SRG with 144 nodes (a); the one without the saddle-maximum cancellation has an SRG with 248 nodes (b). Both graphs have 5 loops.

for the *fold terminating* vertex, and they are assigned a new f value. Given:

$$\delta = (f_{root} - f_{min})/(n-1)$$

then each fold vertex v_i will be re-valued as:

$$f_{new}(v_i) = f_{root} - (i-1)\delta$$

with $i \in [1...n]$. It is easy to see that the *fold root* vertex maintains its original value. After re-assigning the values, the entire fold - up to v_r excluded - is removed from the contour γ.

This fold removing technique can be applied also to folds that are longer than the example in Fig. 4.10c, also containing multiple saddles and maxima.

This fold removing step is optional in the algorithm, and it can be excluded in the case the application is required to detect and not to filter also this kind of features.

As shown in Fig. 4.11 in case of a mesh with a relative high genus, with respect to the number of vertices, applying or not the saddle-maximum cancellation could lead to a completely different number of nodes in the SRG. Nevertheless, also in this case, both the graphs have the same number of loops (equal to the genus of the mesh).

4.1.7 Computational complexity

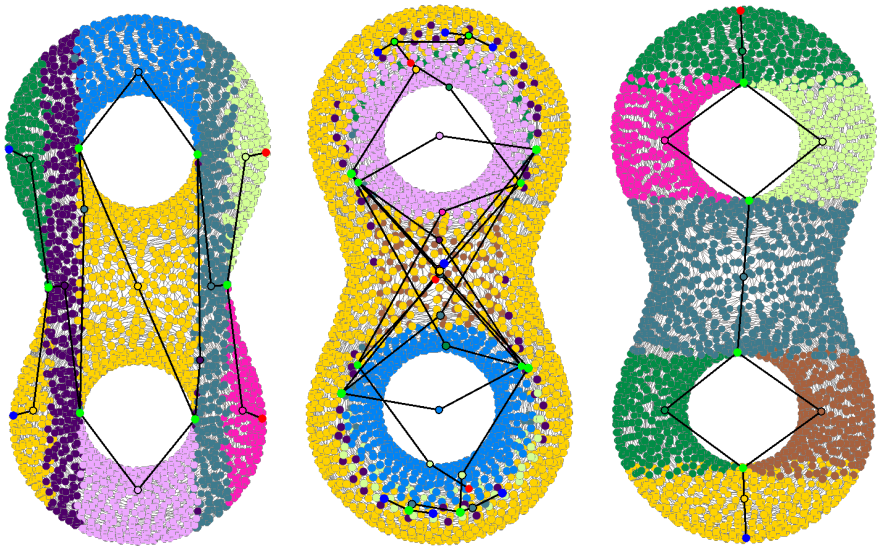
In the DRGSS algorithm, the 1-skeleton of X is represented by a list of vertices. Each vertex v has a set of attributes: the position in 3D space, a reference to the edges in $\mathrm{St}(v)$, the value of the scalar function in the vertex ($f(v)$). The time complexity of the overall algorithm is thus of $\mathcal{O}(n \log n)$, where n is the number of vertices in X. It is essentially the same of the algorithm presented in (CMEH+03) because per each vertex the algorithm performs at most a constant number of operations on contours, each represented by a list of vertices and edges. Using a *balanced search tree* (CLRS09) for such lists, as suggested in same work above, each operation can be performed in logarithmic time.

In this work we preferred representing contours with *hash tables*, in which each entry has a vertex v as its key and $\mathrm{St}(v, \gamma)$ (see Def. 103) as its value. In an hash table, the insert and delete operations are performed in constant time whereas the time required for the search operation depends on the *load factor*: it can take linear time in the worst case, when the load factor is large enough. However, in practice, an appropriate value of the load factor can make the search operation in hash tables to be performed in constant time (CLRS09), at the expense of some extra memory.

4.2 Scalar function

The DRGSS algorithm can receive in input any kind of *general* scalar function f. As it will be described in detail in Sec. 5.1.2, we tested the algorithm with different kinds of scalar functions: height functions, intrinsic functions and also random functions. As we have seen in Chap. 2, different f functions lead to different Reeb graphs (see Fig.4.12), although always compliant, under the hypothesis that f is general, with the Critical Point Theorem (see Thm. 5 in Chap. 2) and the Loop lemma (see Lemma 6 in Chap. 2).

However our aim was not to compare the functions in terms of their effectiveness in representing the shape. Contributions on this subject can be found in (NGH04), (LLT03), (BMMP03).

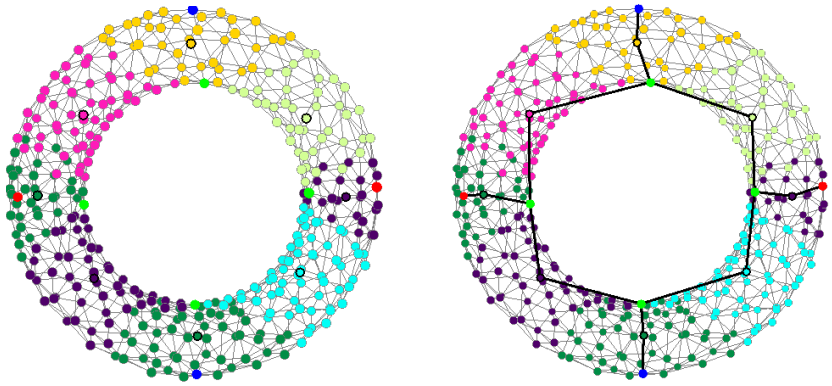


(a) $f = x$ (b) $f = y$ (c) $f = z$

Figure 4.12 – SRG and segmentation for the double torus with f growing normal to each of the three Cartesian axes: $f = x$ (a); $f = y$ (b); $f = z$ (c).

4.3 Implementation

In this section we analyse some aspects concerning the DRGSS implementation. The representation of the SRG together with the corresponding segmentation is the main output Σ of Algoritm 4.1.1. The SRG contains one node per each *critical* segment plus one node per each *regular* segment, with arcs representing the adjacency relations. The SRG is also *embedded*, in the sense that every SRG node is assigned a position in the ambient space of X: each node corresponding to a *critical* segment will share the same position of its critical vertex, while each node corresponding to a *regular* segment is assigned the centroid of the vertices belonging to the segment itself. Fig. 4.13 shows an example of SRG constructed and embedded in the way described.



(a) (b)

Figure 4.13 – Each node in the SRG is assigned a position: that of the corresponding critical vertex or the centroid of the corresponding segment (a). Arcs between nodes describe adjacency relations (b).

4.3.1 Implementation of the algorithm

We used a 100% pure Java implementation, developed inside the Eclipse SDK (Fou12) environment, using Java 1.5. We did not use any third-party Java libraries. The DRGSS algorithm has been built using some classes of the SOAM project (Pia12) and is composed of a DRGSS algorithm engine that can be called with some possible use cases and corresponding view models, as described below.

- The first is a Canvas repaint structure, triggered by an observer/observable pattern between the Algorithm (i.e. Controller) and its View, in order to monitor the segmentation and SRG construction. It is also possible to export the generated intrinsic f function, the produced segmentation and the SRG as separate PLY files (see Sec. 4.3.1.2).

- In the second use case the engine is called to segment and generate SRG, for a given mesh, for all the possible *intrinsic* functions generated choosing each time a different vertex as starting point (see Sec. 5.1.2.2), until all mesh vertices have been chosen.

- In the third use case an input file containing a list of meshes is entered as input. The engine returns as results the SRG computation and the

segmentation, for all the possible variants of the intrinsic function (see Sec. 5.1.2.2), for each of the input meshes.

- In another use case the engine accepts as input a PLY mesh having a *quality* property. This *quality* property is read in input as a given function value, that is used for mesh segmentation and SRG computation.

The DRGSS algorithm does not need mesh-dependent parameters. The only mesh parameter used is the one relative to the intrinsic function: ϵ (see Sec. 5.1.2.2). It is not dependent on the type of mesh and has been set to 0.05 as in (TVD08).

Furthermore it is possible to pass to the algorithm some parameters like: starting vertex index, time delay (for visualization purposes), PLY function export, casual starting vertex, reply function, and others visualization parameters. These parameters can be modified at runtime in the DRGSS user interface and can also be passed to the DRGSS algorithm as a property file. This can be particularly useful during the analysis of a mesh. This approach avoids the recompilation of the code.

The described implementation could be further optimized, as, in this assignment, our main interest was in the validation of the algorithm.

All the experiments have been run on commodity hardware.

4.3.1.1 The input PLY

The proposed algorithm takes advantage of the possible f function valued at mesh vertices and passed as quality attribute in the PLY file. In the case the PLY file does not contain quality values, the algorithm generates a function f, as described in Sec. 5.1.2.2.

generality In order to guarantee the f generality, in the implementation proposed, we did not explicitly perturbed f, but instead we introduced a *comparator*, using the *Vertex* object *hashcode*, that is unique, to disambiguating between vertices having the same value of f.

4.3.1.2 Producing Reeb graph and segmentation as PLY

As a result of the Reeb graph computation and mesh segmentation, two outputs are produced in *ply* format:

- *mesh-rg.ply*: the ply file containing the simplified Reeb graph. Reeb graph nodes are saved as vertices of the ply file, instead Reeb graph edges are stored as degenerate faces.
- *mesh-segmentation.ply*: the ply file containing the mesh vertices and triangles, with a *quality* property associated to the vertices, representing the label of the corresponding segment. Each vertex is associated to the first segment in which the vertex has been segmented. Multiple membership is not stored in this ply file, principally because existing viewers can't handle such kind of multiplicity, but if it would be the case, this will be a trivial extension.

Chapter 5

Experimental evidence

Contents

5.1 Test description 101

5.2 Results 108

To verify the correctness and effectiveness of the DRGSS algorithm we carried out extensive tests with a great number of meshes having different genus and density. We present here some of the most relevant results of our tests. Most of the meshes have been taken from on line repositories such as the AIM@Shape database (Fal04) and the SHREC12 database (LVB+12). They range from genus 0, both with great and small number of vertices, to genus 22 with over ten thousand vertices.

In Sec. 5.1 the proposed testing strategy is described and Sec. 5.2 illustrates and discusses the results of this work.

5.1 Test description

To validate the DRGSS algorithm we planned a test set considering a test space of multiple dimensions:

- different scalar functions,
- increasing complexity in terms of shape genus,

- different mesh densities, even for the same shape.

5.1.1 Number of loops of the SRG

An SRG is validated if it has a number of loops that is equal to the genus of the considered shape. To count the number of loops of the graph, we used the procedure described in (SAA09). We computed the *minimum cycle basis* and hence the number of loops in each of the obtained graphs. On the other hand, we compute shape genus as:

$$g = 1 - \frac{1}{2}(v - \frac{1}{3}e) \tag{5.1}$$

where v is the number of vertices in the mesh and e is the number of edges. Eq. 5.1 derives from the Euler equation (see Def. 51 and Lemma 4 in Chap. 2).

5.1.2 Scalar function

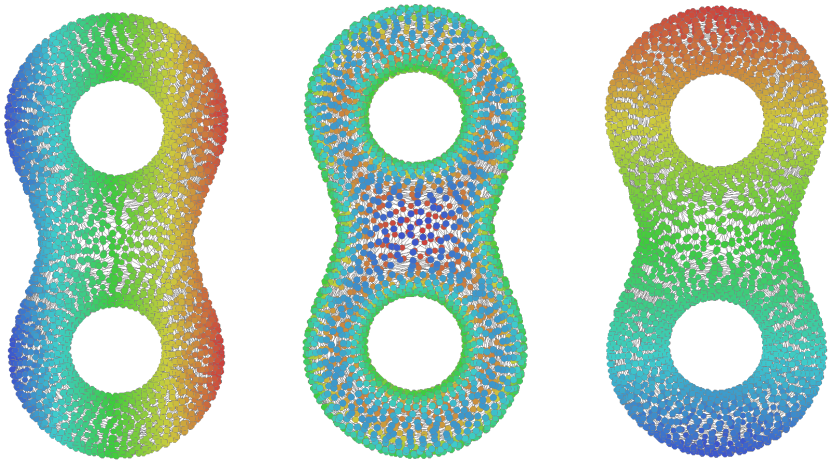
The DRGSS takes as input a triangulated mesh and a *general* scalar function f. This means that no conditions are posed on the input function except its *generality*. One of the dimensions on which to test the DRGSS was thus in the different classes of f. The aim was of choosing the most representative ones, remembering that different functions take to different graphs, even if each of them must have a number of loops equal to the shape genus g.

In this section the input functions used to test the algorithm are described.

5.1.2.1 Height function

The height function has been extensively used in literature (see for instance (TIS+95; dBvK97; SKK91; BFS00; WXS06; PSBM07) among the others); it is the immediate choice, in particular in case of natural objects (e.g. terrain models (BFS00), (TIS+95)). The level lines of f^* (see Eq. 2.6 in Chap. 2) defined by an height function correspond to the intersection between the mesh and a plane normal to the height vector.

This kind of scalar function leads to a graph that is invariant to translations and uniform scaling, but it is not invariant to rotations and deformations. The value of this f for Reeb graphs construction must thus be evaluated



(a) $f = x$ (b) $f = y$ (c) $f = z$

Figure 5.1 – The three height functions corresponding to the Cartesian axes in R^3, mapped on the double torus.

with respect to the specific application (e.g. it would not be good for shape matching (BMMP03)). We chose the three height directions corresponding to the Cartesian axes in R^3. In Figure 5.1 three examples of height function are given for the double torus.

5.1.2.2 Intrinsic function

A scalar function mapped on a shape is called *intrinsic* if its values do not change in case of translation, scaling, rotation and deformation of the shape. This kind of function produces Reeb graphs that are homeomorphic to each other when applied to a same shape in different poses or at different magnifications.

Among the available intrinsic functions (e.g.(LV99), (MP02a), (BMMP03),(TVD08)), we chose to test the DRGSS algorithm with a variant of the function proposed by (TVD08), as described in (BP12). This choice was motivated by the fact that this function starts with an heuristic (the *starting point* is chosen at random among mesh vertices, see below). This particular aspects gives us the opportunity for testing many scalar functions at once: in our experiments we chose at each run, for each mesh, a different vertex as the starting point, until all mesh vertices have been selected. In this way we

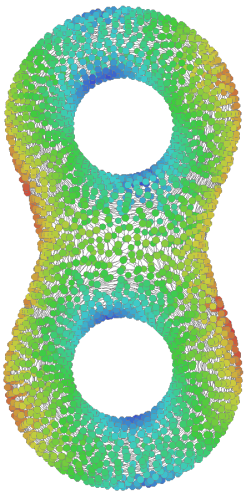


Figure 5.2 – An intrinsic function mapped on the double torus

tested the DRGSS algorithm with all the existing variations, for each mesh, of this intrinsic function, resulting in a broad test coverage.

To compute the *intrinsic* function we used the concept of *geodesic distance* on a mesh (NKI02), meant as the length of the shortest path connecting each two vertices. Another fundamental definition used is that of *diameter vertices*, that is a pair of vertices that are at the maximum geodesic distance on the mesh.

Similarly to (TVD08), in our implementation, the f function is computed through the following steps:

- find the two *diameter vertices* and compute the two distance functions (δ_1, δ_2) from these points, with the Dijkstra algorithm (Dij59): each function will assign to each vertex in the mesh its *geodesic distance* from the corresponding point;
- find the *local extrema*, i.e. the local maxima and minima of the two distance functions (δ_1, δ_2);
- identify the *feature points* (FP) by merging local extrema with some tolerance;
- the resulting function is computed as the geodesic distance between each vertex and its closest FP.

Algorithm 5.1.1: FINDDIAMETERVERTICES($mesh$)

$maxDistance \leftarrow 0$
$v_p \leftarrow null$
$v \leftarrow random$
$v_n \leftarrow$ ***findFarthestVertex***(v)
while $v_n \neq v_p$
do
$v_p \leftarrow v$
$v \leftarrow v_n$
$v_n \leftarrow$ ***findFarthestVertex***(v)
$maxDistance \leftarrow$ ***distance***(v_n, v)
return $(v_n, v, maxDistance)$

Diameter Vertices and Maximum Distance The algorithm chooses a *starting vertex* at random in the mesh and sets it as the *currentVertex*. Then it finds the farthest vertex from the *currentVertex*, with the Dijkstra algorithm (*findFarthestVertex*), and sets it as the *currentVertex*. The procedure is repeated until a fixed pair of vertices is met, namely when *currentVertex* and its farthest vertex coincide in two consecutive loops. These are the *diameter vertices* v_1 and v_2 (see Figure 5.3a).

This algorithm is a variation of the one defined in (LV99), in that we choose to continue the iteration until it reaches a stable pair of vertices, in the above sense (see Algorithm 5.1.1), while only two iterations are performed in the original version, after the random selection.

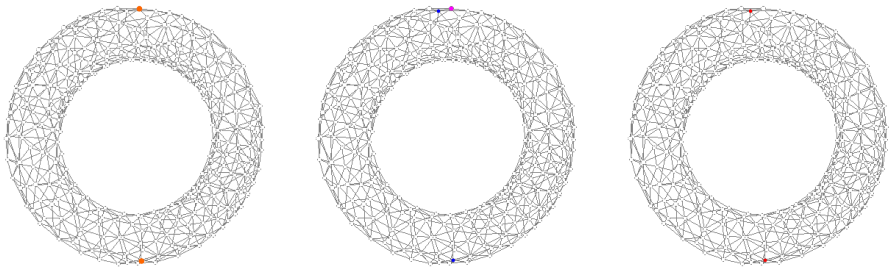
Each mesh has its specific configuration of diameter vertices that, in general, changes when changing the *starting point*. In particular if the mesh shows a symmetry but does not have a principal diameter (like in the case of the torus), a new starting point will often identify a new couple of diameter vertices. On the other hand, if the mesh is *elongated* (i.e. it has a principal diameter, like in the case of the bunny or in the case of the hand), iterating until a stable pair of vertices is found will result in quite the same couple of diameter vertices, also choosing each time a different starting point. Table 5.4 shows the number of different configurations of diameter vertices for some of the meshes of the test set.

The two diameter vertices define *two* distance functions δ_1 and δ_2 being the geodesic distances from vertices v_1, and v_2 respectively.

computing f Following (TVD08), we identify the *feature points* (FP) of the mesh by *softly* merging the *local extrema* of the two distance functions δ_1 and δ_2. Local extrema (i.e local minima and maxima) of the two functions are merged into a common feature point if they are not farther away than a certain predefined tolerance, otherwise they are simply discarded (see Figure 5.3b and 5.3c).

Algorithm 5.1.2: FINDFP(v_1, v_2)

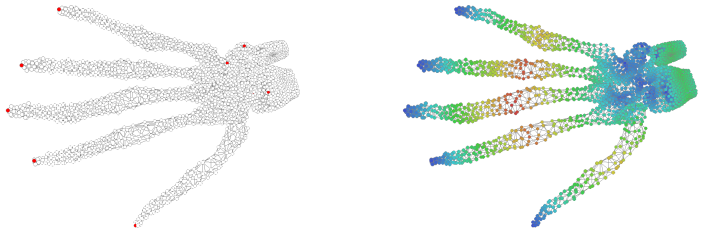
$V_1 \leftarrow$ ***findLocalExtrema***(v_1)
$V_2 \leftarrow$ ***findLocalExtrema***(v_2)
$FP \leftarrow$ ***mergeLocalExtrema***(V_1, V_2, ε)
return (FP)



(a) (b) (c)

Figure 5.3 – Diameter vertices (in orange) (a), local maxima and minima of the two distance functions δ_1 and δ_2 (in blue and purple) (b), the feature points (in red) (c).

As seen also in (MP02a), (KLT05) and (TVD08), typically feature points are located on the most prominent components of the mesh (see Fig. 5.4a). The function of choice is defined as the geodesic distance between each vertex and its closest feature point (see Fig. 5.4b), normalized w.r.t. the maximal vertex distance on the mesh to the nearest feature point. We also



(a) (b)

Figure 5.4 – Feature points (in red) (a) and the resulting distance function δ (b) for the hand with genus 5.

ensure that the function is *general* (i.e. no two vertices have the same value) while computing the geodesic distances.

5.1.2.3 Random function

To test the correctness of the DRGSS algorithm, we chose to introduce in the test set also meshes with the random scalar function. This is the *worst* function in terms of number of critical points since it has a great number of local minima, maxima, saddle points, also degenerate ones and it will generate *noisy* simplified Reeb graphs. For example, in Fig. 5.5 the function has 2367 critical points in a mesh of 3070 vertices.

For these reasons, the random function is a good test case for the topological correctness of the algorithm, because also under these conditions, the DRGSS must produce a graph that has a number of loops equal to the shape genus.

5.1.3 Shape Genus

We chose the meshes in the experiments with the aim of covering different levels of topological complexity: shapes in the test set range from genus 0 up to genus 22. We used public datasets, taking meshes from the AIM@SHAPE database (Fal04) and from the SHREC12 database (LVB$^+$12) (in particular here we selected the triangulated closed surfaces).

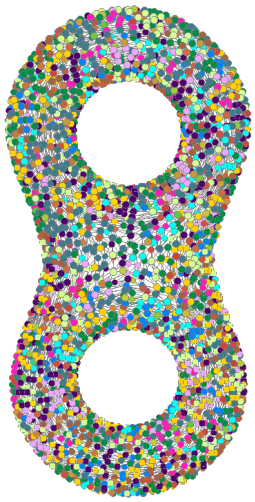


Figure 5.5 – A random function mapped on the double torus

5.1.4 Mesh density

Mesh density, intended intuitively as the number of vertices per unit of surface, is a relevant parameter for a segmentation and Reeb graph extraction algorithm, as explained in Sec. 4.1.4.1. To test the effectiveness of the representation technique here proposed (the *augmented* ULS), we put in our test set different versions of the same mesh, that have been progressively either coarsened via *vertex decimation*, either thickened with *surface subdivision* (i.e. two common mesh processing techniques (CNR12)). Meshes in the test set thus range from a few hundred to several thousand vertices.
Furthermore we also added some noise (intended as roughness of surfaces (CNR12)) to some of the meshes, the ones identified with *Noisy* in the Tables 5.2 and 5.3.

5.2 Results

With all the variants of the mapping function described in 5.1.2 and, for each of the meshes shown in Tables 5.1, 5.2, 5.3, the proposed algorithm computes the SRG corresponding to the correct genus, also when the scalar function is the *random* function (see Tab. 5.3).

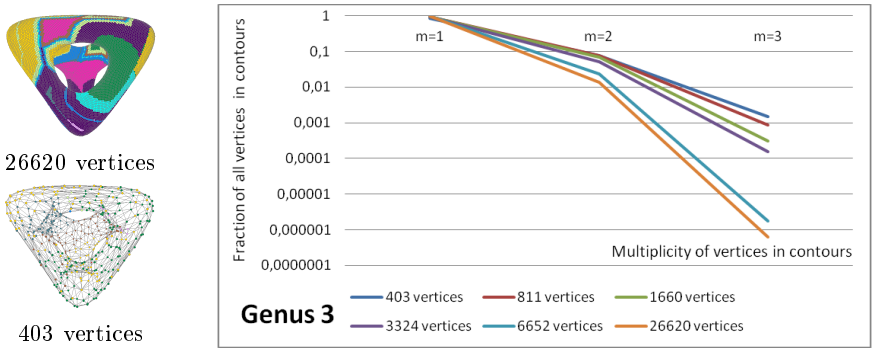


Figure 5.6 – Progressive coarsening of the same mesh (Genus 3) from 26620 vertices to 403 vertices: a smaller number of vertices causes an increase of multiplicity of edges and vertices in contours.

Some of the meshes in the test set are also illustrated in Figure 5.8, together with their SRG and the segmentations obtained.

5.2.1 The role of multiplicity

The experiments (see Fig.5.6) show that the statistics of the multiplicity of edges and vertices in contours, as described in Sec. 4.1.2.1 are, in general, proportional to the coarseness of the triangulation: the more a mesh is coarse, the more contours will show edges and vertices multiplicity.

Figure 5.6 shows the statistics for a shape with genus 3, used for the tests, that has been progressively coarsened via *vertex decimation*. The diagram on the right in Figure 5.6 shows the multiplicity of vertices in contours, measured in relative terms - as the fraction of all vertex occurrences in contours, for the same test shape, triangulated at different levels of coarseness. It is evident that the occurrence of vertices with *multiplicity* higher than 1 steadily increases as the mesh becomes coarser. In particular, the fraction of vertices with $m = 3$ in a mesh with 403 vertices is more than three orders of magnitude larger than the corresponding fraction in a mesh with 26620 vertices. This means that, statistically, the multiplicity of vertices and edges is inversely proportional to mesh density and directly proportional to genus. Thus, the most interesting test cases, in terms of higher multiplicity, will be found in meshes having high genus and low vertices density. For this reason we took care of inserting into the test set a relevant number of such

meshes.

Table 5.1 – Some of the meshes used to check the validity of the DRGSS algorithm, with their genus and number of vertices.

Mesh Name	Vertices	Genus
Horse	2450	0
Bunny	3052	0
Dinopet	4500	0
Torus	359	1
Double Torus	319	2
Genus3	782	3
HandG5	4037	5
HandG8	3639	8
Heptoroid	10851	22

5.2.2 Robustness to different mesh densities

We tested the DRGSS algorithm with meshes having different densities. We also decimated and thickened the same meshes as in the case of the double torus and the Genus3 of Table 5.2.

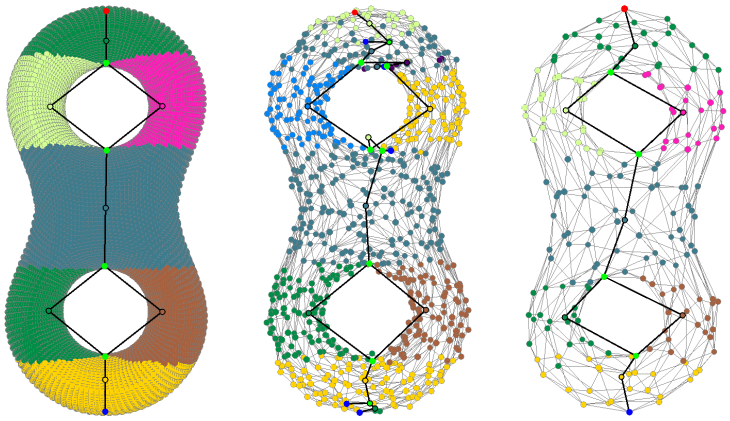
To prove the *noise resistance* of the SRG, we added, with (CNR12), random disturbance on the vertices of some meshes (the ones indicated in the latter table with the *Noisy* attribute), making their surfaces rougher.

In all the test cases the DRGSS produced a correct Reeb graph. As can be seen also in Fig. 5.7, DRGSS proved to be robust to variation in mesh density.

5.2.3 The random function

We tested the DRGSS algorithm also with a set of meshes having $f = random$. In Table 5.3 are some of the meshes of the random test set, together with their number of vertices and genus.

Also in the case of the random function, for all of the meshes of the test set, the algorithm computed a graph with a number of loops equal to the genus of the shape.

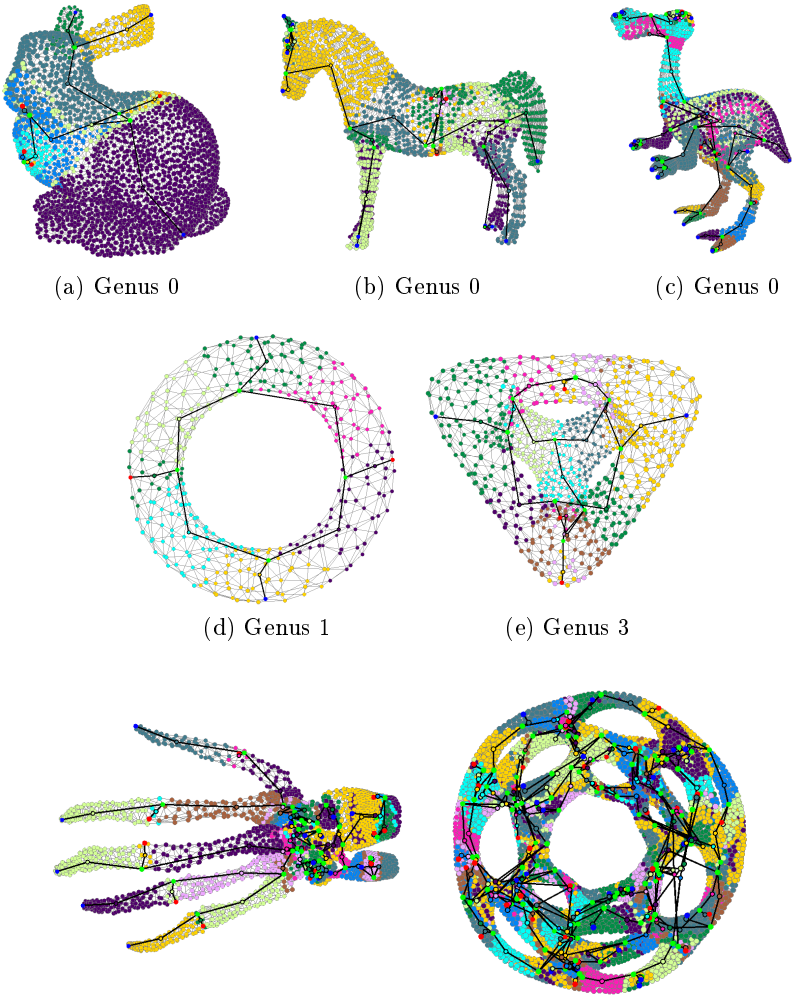


(a) 12286 vertices (b) 766 vertices (c) 190 vertices

Figure 5.7 – The original, high-resolution mesh with 12.286 vertices (a), two increasingly decimated versions with random disturbance added, with 766 vertices (b) and 190 vertices (c).

5.2.4 Intrinsic function variants

Considering the variants of the intrinsic function (see Sec. 5.1.2.2), each mesh has its specific configuration of diameter vertices that, in general, changes when changing the *starting point*. Results show that if the mesh has a symmetry but does not have a principal direction (like in the case of the torus), choosing at each iteration a different starting point will result in many different couples of diameter vertices. On the other hand, if the mesh is *elongated* (i.e. it has a principal direction, like in the case of the bunny or in the case of the hand), choosing at each iteration a different starting point will result in a few distinct couples of diameter vertices. Table 5.4 shows the number of different configurations of diameter vertices for some of the meshes of the test set.

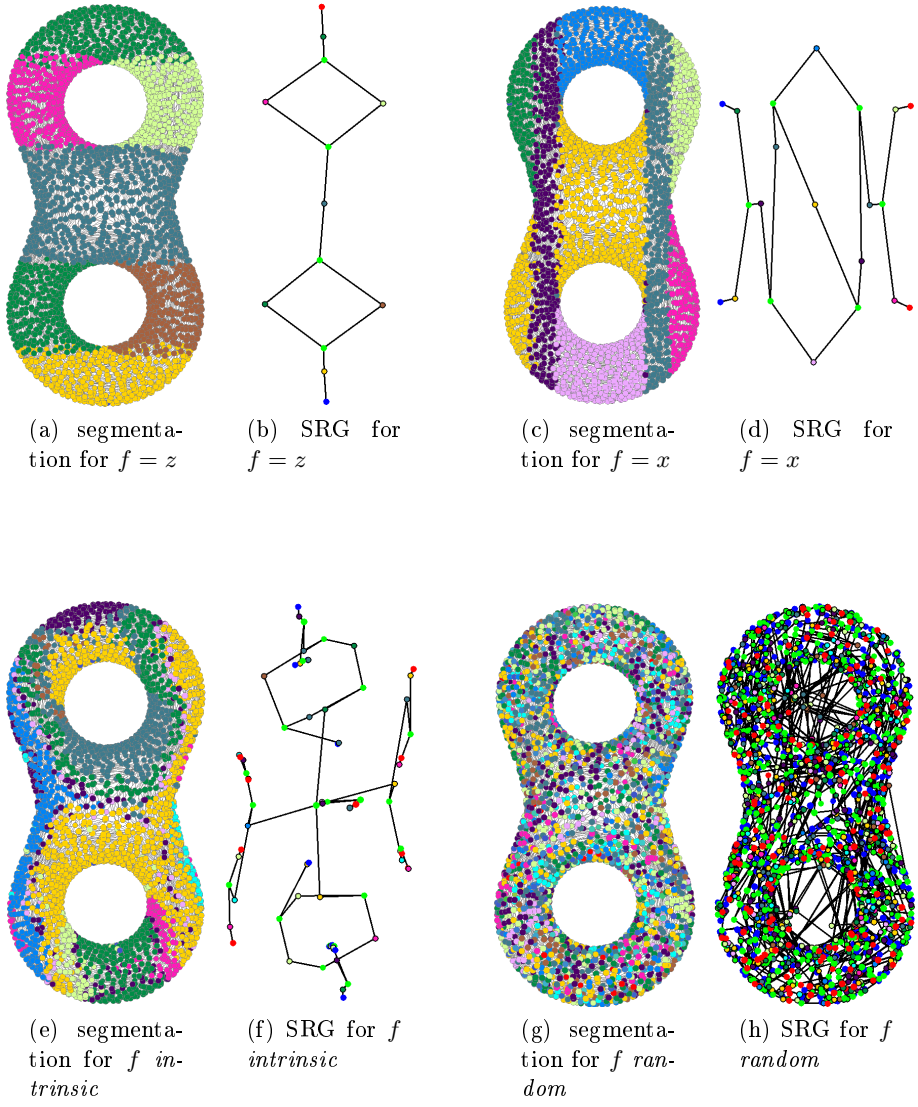


(a) Genus 0 (b) Genus 0 (c) Genus 0

(d) Genus 1 (e) Genus 3

(f) Genus 5 (g) Genus 22

Figure 5.8 – A few meshes in the test set: Reeb graphs are painted in black, segmentations are highlighted with different vertices colors. Multiple vertex memberships are not represented in these images.



(a) segmentation for $f = z$

(b) SRG for $f = z$

(c) segmentation for $f = x$

(d) SRG for $f = x$

(e) segmentation for f *intrinsic*

(f) SRG for f *intrinsic*

(g) segmentation for f *random*

(h) SRG for f *random*

Figure 5.9 – Different segmentations and Reeb graphs for the same mesh with different scalar functions f.

Table 5.2 – These are some of the variants, in terms of densities, of the meshes on which the DRGSS algorithm has been tested: the double torus mesh with vertices ranging from 12.286 to 190 and the Genus3 mesh with vertices ranging from 26.620 to 412. In some meshes (indicated in the table with *Noisy* in the name) also random disturbance has been applied (always with (CNR12)).

Mesh Name	Number of vertices	Genus
Double Torus	12286 3070 766 382 190	2
Double Torus Noisy	766 382 190	2
Genus3	26620 6652 3324 1660 828 412	3

Table 5.3 – These are the mesh tested with the random function. In all the cases the number of loops of the obtained SRG was equal to the mesh genus.

Mesh Name	Number of vertices	Genus
Bunny	3052	0
Double Torus	3070 766 382 190	2
Double Torus Noisy	766 382 190	2
Genus3	828 782 412	3
HandG5	4037	5
HandG8	3639	8
Heptoroid	10851	22

Table 5.4 – Here following, for each mesh of the test set, it is showed the resulting number of all the possible function variants that have been discovered and used for testing the DRGSS algorithm, on each mesh. Variants have been discovered taking, at each run, a different starting point in order to find mesh *diameter vertices* and then mesh *feature points*.

Mesh Name	Vertices	Number of variants of the intrinsic function
Dinopet	4500	2
Bunny	3052	2
Horse	2450	6
Torus	359	27
Double Torus	12286	41
	3070	28
	766	13
	382	8
	319	5
	190	2
Double Torus Noisy	766	11
	382	5
	190	2
Genus3	26620	1376
	6652	496
	3324	95
	1660	39
	828	28
	782	32
	412	11
HandG5	4037	2
HandG8	3639	2
Heptoroid	10851	96

Chapter 6

SRG for human striatum

Contents

6.1 Stating the problem 117
6.2 Striatum shape processing 120
6.3 SRG-based Registration of Striatal Meshes 122
6.4 Results 123
6.5 Conclusions 128

The DRGSS algorithm has been applied in medical imaging, using the SRG as graph-like shape descriptor of the human *striatum* (i.e. a part of the brain, see Fig. 6.1).

We present in this chapter the image and mesh processing pipeline that, starting from 3D T1-weighted MR images, extracts the SRG of the striatal shapes and uses it for mesh registration, decomposition and shape comparison. Further details about this application can be found in (PBP+12).

In literature, at the best of our knowledge, there are no previous works using the Reeb graph of the striatum for such purposes.

6.1 Stating the problem

Human striatum is an highly innervated group of nuclei in the brain (LSB01) (see Fig 6.1). It is of great interest from a medical point of view because it

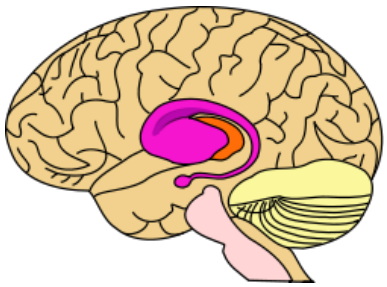


Figure 6.1 – The human striatum (Wik09).

is implicated in both motor processes and in a number of non-motor processes such as cognitive functions, learning, attention, memory, motivation, reward, and addiction (BBS+03; GPO08; KHN+07). Striatum is composed by three regions: *caudate*, *putamen*, and *nucleus accumbens*. This subdivision is important because of the different roles (Seg08) of each region. In particular, striatal structures are of high interest in diseases in which cognitive functions are impaired (like schizophrenia).

The three regions of the striatum have been studied with different techniques: both volumetric studies (BSB+03; LSB01) and local and global morphometric techniques (VGS00; KHN+07).

In the past, deformation and surface based methods have been applied: usually they require the establishment of the point to point correspondence among surfaces (RWSN09). Computing the point to point correspondence is computationally demanding (except if ad-hoc strategies are used (PZT+11)) and it often requires remeshing. On the other hand, these methods provide spatially localized shape information that is relatively straightforward to interpret.

In recent years, a set of global shape approaches have been used in medical imaging for the compact representation and analysis of neuroanatomical shapes: medial representation (SSP+03), spherical harmonic description (HLD+06), Laplace-Beltrami eigenvalues (SLK+08; RWSN09).

As we have seen in previous chapters, Reeb graphs have an interesting property: they are more robust than other skeletal descriptors since they are less sensitive to small and localized changes on the mesh (BMMP03; SLK+08).

In this application we will use the SRG as a compact descriptor for the

striatal shapes for the purposes of automatic inter-subject mesh registration, automatic mesh decomposition, and for the inter-group striatal shapes comparison.

6.1.1 Automatic inter-subject mesh registration

Surface-based registration techniques are computationally expensive in that they imply establishing the point to point correspondence among the vertices of the meshes.

In this work we propose a SRG-based registration technique, that uses the SRG nodes, instead of the mesh vertices, to register a group of meshes.

6.1.2 Automatic mesh decomposition

Striatal sub-segmentation is important because each striatal region is differently affected in diseases like schizophrenia (BSB$^+$03). The sub-segmentation of the striatum is a challenging task, in that the signal intensity alone is not sufficient to distinguish among striatum sectors (FSB$^+$02), and manual or semi-automatic methods are still regarded as the gold standard. We used the DRGSS algorithm to automatically decompose the striatal mesh into its three primary anatomical regions (caudate, putamen and nucleus accumbens). The proposed sub-segmentation does not require an a-priory knowledge of the striatal surface representation (e.g. number of mesh vertices).

6.1.3 Inter-group striatal shapes comparison

In this application we used the SRG as shape descriptor to detect shape variations of the striatum between different groups of subjects. The objective was to detect and analyse neuroanatomical morphological differences (both within and between groups).

6.2 Striatum shape processing

6.2.1 The dataset

In this application we used 3-D T1-weighted MR brain images of 40 subjects: 22 normal controls, and 18 neuroleptic-naïve patients.

MRI brain scans were acquired using a 1.5 T Siemens Magnetom (Erlangen, Germany) as detailed in (LTS$^+$01). The voxel size of the MR images was $1.5 \times 1.5 \times 1.0$ mm^3 and the size of the images varied from $256 \times 256 \times 150$ to $256 \times 256 \times 170$ voxels. The triangulated meshes have been obtained from the MRI volumes by non-rigid deformation of the same triangulated surface model, using an interactive 3D software tool (LRMT99; KHN$^+$07). Finally the obtained striatal shapes were decomposed into two mesh surfaces representing the left and right striatum separately (see (PBP$^+$12) for details).

6.2.2 Computing SRG

The DRGSS algorithm was applied to each mesh to obtain a simplified Reeb graph (SRG) and a mesh partitioning. To best fit the DRGSS algorithm to the striatum problem we adopted some special provisions, as described in the following paragraphs.

6.2.2.1 The scalar function

In this specific application we used a variant of the intrinsic function described in (BP12). In its original implementation indeed, this function is computed applying an heuristic: a *starting point* is selected at random among mesh vertices to find the *diameter vertices* and this choice influences the computation of the scalar function f and thus the SRG. Applying the DRGSS to the striatum problem required to make the SRG independent from this random choice and uniquely defined when applied to the same mesh. The devised solution was to choose the *starting point* corresponding to the *maximal* distance between the *diameter vertices*.

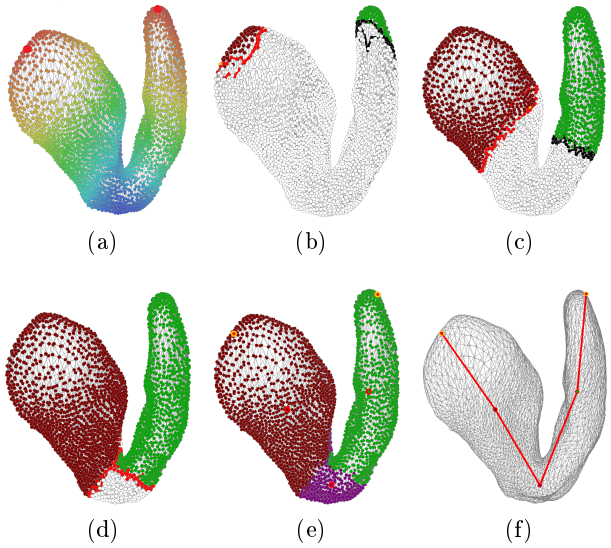
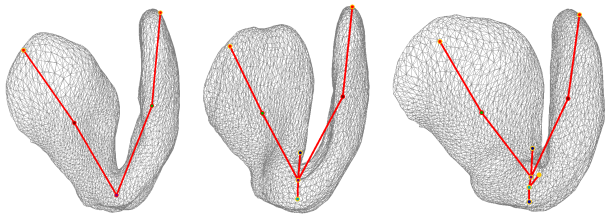


Figure 6.2 – Vertices colors represent increasing values (from dark red to dark blue) of the f function; feature points are magnified and represented in red. (a). Contours start at each function minima (b) and they are evolved in the direction of ascending values of the f function (c). Two contours merge in a new one (d). In this application each node in the SRG is either a minimum point or the centroid of a segment (e), arcs of the SRG describe adjacency relations (f).



(a) (b) (c)

Figure 6.3 – SRG obtained by applying the proposed algorithm to three examples of striatal shapes.

6.2.2.2 The segmentation

Once the mesh has been swept, the SRG is built assigning a node to each segment (nodes are positioned in segments centroid). Graph arcs encode adjacency relations between segments. The graph is completed by the nodes representing the function minima, together with an arc between each minimum and the centroid of the segment generated by the contour started at that minimum.

The selected f may have several local maxima which generate extra branches in the Reeb graph (see Fig. 6.3). But all the striatal meshes have genus 0, thus their Reeb graphs are guaranteed to contain no loops. All extra branches in the Reeb graph have a common root and thus the corresponding segments could be safely merged with this root segment. This also involved recomputing the balanced centroid of the root segment (as a consequence of the merging of the extra segments). As a result of this procedure, all the obtained SRG were made of 5 nodes and 4 edges.

6.3 SRG-based Registration of Striatal Meshes

The striatal meshes were registered using an iterative algorithm based on a 7-parameter linear registration by matching their SRG descriptors using Full Procrustes Superimposition (Ken89) (FPS). The algorithm is composed of three consecutive steps:

- Step 1: One of the N graphs x_i, is randomly selected as the *reference graph* x^r. All other graphs x_i are registered to this one by mini-

mization of the sum of square differences $\sum_{j=1}^{V} \|x_j^r - (s_i \mathbf{R}_i x_{i,j} + \mathbf{t}_i)\|^2$ w.r.t. $s_i, \mathbf{t}_i, \mathbf{R}_i$ using FPS. Here $V = 5$ is the number of nodes in each graph; x_j^r and $x_{i,j}$ are the 3D coordinates of the j-th node of the reference graph and of the i-th individual graph, respectively; s_i, $\mathbf{t}_i$, $\mathbf{R}_i$ are the isotropic scaling factor, translation vector, and rotation matrix, for the i-th graph respectively. After this step each graph x_i is transformed (with the $T(\tilde{s}, \tilde{\mathbf{t}}, \tilde{\mathbf{R}})$) in a graph $\tilde{x}_i$ aligned to the reference graph x^r.

- Step 2:The registration accuracy might be significantly impaired if the reference graph x^r is not a good representative of the mean SRG in the dataset. For this reason a second iterative registration routine is implemented:

  (i) The routine computes the mean shape $\tilde{x}^m$ (as the arithmetic mean of the previously registered graphs $\tilde{x}_i$).

  (ii) Graphs $(\tilde{x}_i)$ are aligned to $\tilde{x}^m$ by minimizing $\sum_{j=1}^{V} \|\tilde{x}_j^m - (\tilde{s}_i \tilde{\mathbf{R}}_i \tilde{x}_{i,j} + \tilde{\mathbf{t}}_i)\|^2$ w.r.t. $\tilde{s}_i, \tilde{\mathbf{t}}_i, \tilde{\mathbf{R}}_i$ using FPS.

  (iii) The mean shape $\tilde{x}^m$ is updated and the transformation $T(\tilde{s}, \tilde{\mathbf{t}}, \tilde{\mathbf{R}})$ is composed to the one calculated at the previous stages.

  Points (ii) and (iii) are repeated until there are no *significant* improvements to the SRG superimposition or a maximum number of iterations is achieved.

- Step 3: The composed transformation $T(\tilde{s}, \tilde{\mathbf{t}}, \tilde{\mathbf{R}})$ that best matches each SRG to the mean graph shape is applied to the corresponding original meshes, from which each SRG was extracted.

6.4 Results

6.4.1 SRG-based Registration of Striatal Meshes: a quantitative assessment

We tested the SRG-based registration routine described in Section 6.3 for the linear alignment of striatal surfaces (left and right striatum separately).

To make a quantitative comparison with the traditional mesh-based approach, the same striatal meshes were also linearly aligned to each other with the same method illustrated in Section 6.3 but applied to the whole mesh. In this more traditional approach the linear transformation is extracted using all the vertices of the mesh. This transformation is then applied to the corresponding SRG for comparison.
We compared the SRG based registration also in respect to a second surface-based registration routine, similar to the one in (KHN+07): a mean mesh was calculated and all the individual meshes were affinely registered to that using Arun's method (AHB87); the mean mesh was finally registered back to the previously (affinely) registered meshes using again (AHB87). Examples of the SRG and striatal meshes, prior and after registration, are shown in Fig. 6.4.
Since a ground truth for the striatal surface registration was not available, we quantitatively compared the results of the SRG-based registration of striatal meshes to the aforementioned (using FPS and Arun's method) more traditional surface-based methods. Results were quantitatively validated using the Hausdorff Distance (HD) (HKR93) (see Table 6.1). This provides a measure of the maximum symmetrical distance between two surfaces and thus can be used to validate the surface registration. Particularly, two measures based on the HD were computed:

- HD_1: mean of the HD computed between each registered surface and the mean surface mesh. HD_1 can be interpreted as a measure of the dispersion of the registered surfaces as compared to the mean shape, and therefore as a measure of the remaining mis-alignment.

- HD_2: mean of the HD computed for each mesh with respect to each other mesh. HD_2 can be interpreted as a measure of the mean maximal difference between each pair of surfaces in the database (after their alignment), and therefore also as a measure of the shape variability within the given dataset.

Using the SRG we obtained a registration accuracy that was qualitatively (see Fig. 6.4) and quantitatively (as measured by the HD, see Table 6.1) comparable to, and in some cases outperforming, the registration accuracy obtained by the surface-based approaches. The surface-based registration algorithms estimate optimal registration parameters in presence of point

Table 6.1 – Accuracy of the SRG-based registration compared to the surface-based registration. The accuracy of the striatal mesh registration was evaluated with two measures derived from the Hausdorff distance.

	SRG-based		surface-based			
	FPS		FPS		Arun	
	Left	Right	Left	Right	Left	Right
HD_1 [mm]	3.4751	2.9124	3.4187	3.0510	3.4264	3.1653
HD_2 [mm]	4.8307	4.1326	4.8512	4.3999	4.9386	4.6012

correspondence, but this point correspondence is computationally demanding, especially for dense surfaces, and it often requires remeshing. Remeshing could in turn influence shape comparisons done on surfaces (RWSN09). Our experimental results suggest that in case of 7-parameters linear motion the SRG is a good descriptor for doing mesh registration. This SRG-based registration only requires triangulated closed meshes and it is not limited to meshes having the same number of vertices. The implications of these results might thus be extended to shapes other than the human striatum, although further testing is required.

6.4.2 SRG-based Surface Decomposition: a qualitative assessment

The automatic SRG extraction and mesh decomposition provides three distinct mesh sectors from each striatal surface. By visual inspection, results of the mesh decomposition were consistent within (left and right striatum) and between subjects, for the whole database. An example of mesh decomposition produced by our pipeline is depicted in Fig. 6.2.(e).
A probabilistic map of the resulting striatal decomposition was then obtained by calculating the probability of each surface vertex to be assigned to each mesh sector in the whole database. This mesh decomposition into three sectors corresponded to the anatomical sub-segmentation of human striatum into *nucleus accumbens*, *caudate* and *putamen* (depicted in Fig. 6.5 in violet, green and red colors, respectively).

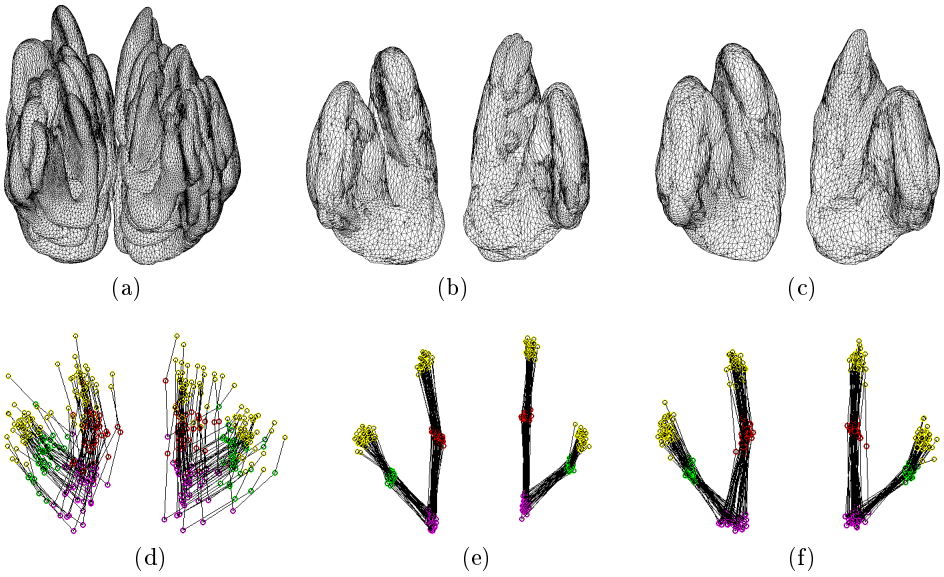
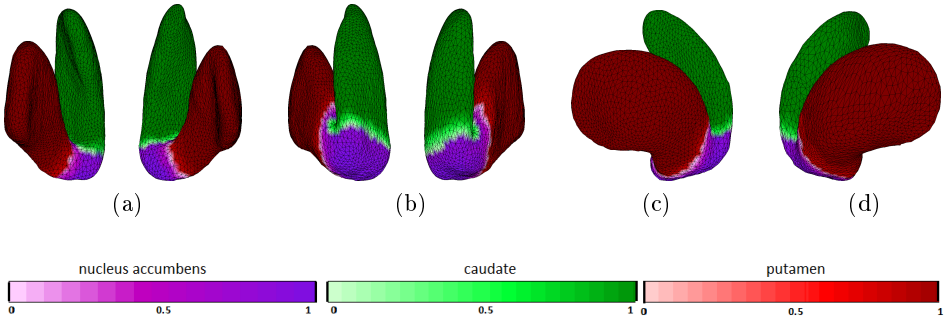


Figure 6.4 – A comparison between the SRG-based and surface-based registration performed on the whole database of 40 subjects. In the first column, striatal meshes in the native space (a) are shown with their corresponding SRG (d). In the second column, the SRG in the native space (d) are aligned by SRG-based registration (e) and the obtained transformations are applied back to the corresponding meshes (a) to obtain (b). In the third column, meshes in the native space (a) are aligned by a surface-based registration using FPS (c) and the obtained transformations are applied to the corresponding SRG graphs (d) to obtain (f).



(e)

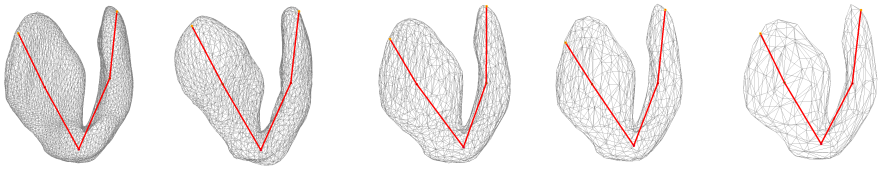
Figure 6.5 – In this figure it is shown a probabilistic map, for the left and right striatal separately, of the mesh decomposition, on the whole database. Meshes are depicted in frontal (a), posterior (b), and lateral (c and d) views. Maps were obtained by calculating the probability of each surface vertex to be assigned to each mesh sector. It is easy to see how the mesh decomposition of the striatal shapes into three sectors actually corresponds to the anatomical sub-segmentation of human striatum into nucleus accumbens (violet), caudate (green) and putamen (red).

6.4.3 Inter-group comparison

Fig. 6.7 shows the mean SRG in schizophrenia and in control groups. The analysis of the mean SRG obtained showed that the SRG descriptor is sensible to mesh variations. The use of simplified Reeb graphs as a tool for studying the intra-group shape variability (or for discriminating among groups based on their shapes) has not been explored quantitatively, even if these results inspires further investigations on the use of the SRG descriptor for the detection and analysis of neuroanatomical morphological differences within and between groups.

6.4.4 Stability of the SRG to Mesh Resolution

Applying the SRG algorithm to meshes progressively decimated we can see that the resulting segmentation is stable, and the corresponding SRG are consistent (see Fig. 6.6).



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

Figure 6.6 – The SRG is robust to mesh resolution. These are SRG of the same mesh progressively decimated: 3293 vertices (a), 1647 vertices (b), 824 vertices (c), 413 vertices (d), 207 vertices (e).

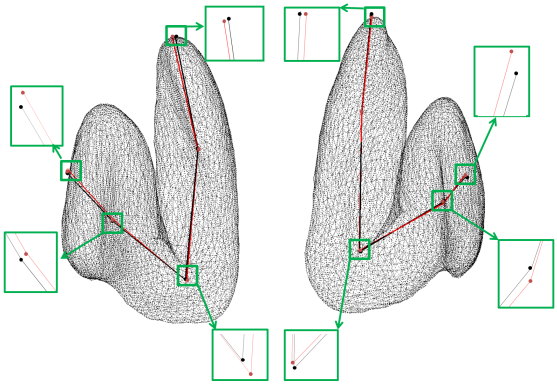
6.5 Conclusions

The results presented in Sec. 6.4.1 show that the registration obtained via SRG is qualitatively comparable and quantitatively (as measured by the Hausdorff distance) outperforming the surface-based registration (see Tab. 6.1), while being computationally simpler. SRG-based registration does not require neither the computation of the point correspondence among mesh vertices, nor is limited to meshes having the same number of vertices.

The experiments described in Sec. 6.4.2 show the efficacy of the proposed method for mesh partitioning: the sub-segmentation of the striatal surfaces into caudate, putamen, and nucleus accumbens is qualitatively meaningful, even if this method is still not quantitatively validated.

Finally, Sec. 6.4.3 shows that, despite its compactness, SRG can be used as a compact descriptor for group specific mean-shapes of the human striatum.

Due to its geometrical and topological properties, the SRG is thus effective as a basis for inter-subject registration, shape representation, and mesh decomposition of striatal surfaces.



(a)

Figure 6.7 – The figure shows the mean SRG in schizophrenia (depicted in red) and in normal controls (depicted in black) overlaid to the mean striatal surface in frontal view. This is an illustration of the potentialities of the SRG as a descriptor for group-specific mean shapes, although we are aware that the role of this descriptor as a tool to discriminate among groups has not been validated, and no medical conclusions should be derived from it.

Chapter 7

Conclusions

Reeb graphs are compact and faithful shape descriptors: thanks to their topological properties, when embedded in R^3, they encode both the topological and the geometric features of a shape and for this reason they have found several applications in different fields of Computer Graphics.

Since Reeb graphs introduction in Computer Graphics (SKK91), a lot of contributions have been proposed to find a computationally simpler alternative to the level lines of the scalar function f defined on the shape.

This work introduces a 1-skeleton representation of the level lines induced on a closed, orientable, triangulated 2-manifold, by a *general* scalar function f defined on the vertices of the triangulation.

The *augmented ULS* is made of existing mesh edges and vertices, with possible *multiplicity* $m \geq 1$. Contours are *ULS* connected components, and they are in the same way *augmented* with *multiplicity*. Contours proved to have a *topological correspondence* (see Chap. 2) with the connected components of the level lines induced by f on the surface. These theoretical properties lead to the implementation of the *Discrete Reeb Graph and Surface Segmentation* algorithm (DRGSS), that computes in one-pass both the *correct* Reeb graph and the surface segmentation for *any* input general scalar function f.

The resulting graph is called *simplified* Reeb graph (SRG) because all the nodes corresponding to *regular* vertices between two *critical* vertices are merged in the centroid of the corresponding segment.

The resulting segmentation is a partition of mesh vertices. This work introduces special provisions to preserve segments connectedness that could be compromised in case of *low density* meshes. Indeed, when mesh density (intended informally as the number of vertices per unit of surface) is low, it can happen that one or more faces are shared between more than two segments. The DRGSS algorithm identify those faces and let some of their vertices to be *multiple*, i.e. to belong to more than 1 segment.

The DRGSS algorithm is a *sweep* algorithm in which contours are initialized at f minima and evolve in the direction of ascending value of f. At each step the DRGSS algorithm uses only information local to the considerate *candidate* vertex. Furthermore, contour events can be detected by looking at candidate vertex *multiplicity* only. Solely in case of a split/merge event the entire contour is walked in order to identify its connected components. But also in this case only informations in the 1-neighborhood of the contour are used. When all contours have reached a maximum, both the SRG and the segmentation are given (see Chap. 4). The computational complexity of the overall algorithm is thus $\mathcal{O}(n \log n)$ (where n is the number of vertices in X), the same as the best sweeping algorithm (CMEH$^+$03) in literature.

Extensive experimental validation (see Chap. 5) has been carried out considering different test dimensions: shape genus, mesh density, scalar function. The test set included shapes ranging from genus 0 up to genus 22. Different mesh densities have been considered, either for different meshes either considering different density values for the same mesh. The input scalar functions have been selected among different classes: height functions, intrinsic functions, random functions.

For all the considered test dimensions, the DRGSS algorithm always computes a Reeb graph that has the same number of loops as the shape genus, also when the *random* scalar function is used.

This is the only algorithm for both Reeb graph extraction and mesh segmentation that, at the best of information available, has been reportedly validated with the random function in literature.

Finally, in Chap. 6, an application is presented in which the SRG is used, with very good results, as a descriptor for the human striatum (i.e. a part of the brain). SRG nodes are used in place of mesh vertices to register a group

of striatal meshes, obtaining a registration accuracy comparable with, and in some cases outperforming, the surface-based methods. In this context, the SRG is also applied to automatic striatal mesh decomposition and an application of the SRG graph for inter-group comparison of striatal shapes is shown.

These results inspire future works in several directions, both theoretical (e.g. in terms of how the contours can be advanced, exploiting shape properties), both toward other possible applications.

Bibliography

[AEHW06] Pankaj K. Agarwal, Herbert Edelsbrunner, John Harer, and Yusu Wang. Extreme elevation on a 2-manifold. *Discrete and Computational Geometry*, 36:553–572, 2006.

[AFS06] Marco Attene, Bianca Falcidieno, and Michela Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer*, 22:181–193, 2006.

[AHB87] K. Arun, Thomas Huang, and Steven Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern. Ana.l Mach. Intell.*, 9(5):698–700, 1987.

[AKM+06] Marco Attene, Sagi Katz, Michela Mortara, Giuseppe Patané, Michela Spagnuolo, and Ayellet Tal. Mesh segmentation - a comparative study. In *Shape Modeling and Applications, 2006. SMI 2006. IEEE International Conference on*, page 7, june 2006.

[Ban70] Thomas Francis Banchoff. Critical points and curvature for embedded polyhedral surfaces. *The American Mathematical Monthly*, 77(5):475–485, 1970.

[BBS+03] Adam M. Brickman, Monte S. Buchsbaum, Lina Shihabuddin, Erin A. Hazlett, Joan C. Borod, and Richard C. Mohs. Striatal size, glucose metabolic rate, and verbal learning in normal aging. *Cognitive Brain Res.*, 17(1):106 – 116, 2003.

[BDBP09] Stefano Berretti, Alberto Del Bimbo, and Pietro Pala. 3D mesh decomposition using Reeb graphs. *Image and Vision*

Computing, 27(10):1540 – 1554, 2009. Special Section: Computer Vision Methods for Ambient Intelligence.

[BFS00] Silvia Biasotti, Bianca Falcidieno, and Michela Spagnuolo. Extended Reeb graphs for surface understanding and description. In Gunilla Borgefors, Ingela Nyström, and Gabriella di Baja, editors, *Discrete Geometry for Computer Imagery*, volume 1953 of *Lecture Notes in Computer Science*, pages 185–197. Springer Berlin / Heidelberg, 2000.

[BGSF08] Silvia Biasotti, Daniela Giorgi, Michela Spagnuolo, and Bianca Falcidieno. Reeb graphs for shape analysis and applications. *Theoretical computer science*, 392:5–22, 2008.

[BHEP04] Peer-Timo Bremer, Bernd Hamann, Herbert Edelsbrunner, and Valerio Pascucci. A topological hierarchy for functions on triangulated surfaces. *Visualization and Computer Graphics, IEEE Transactions on*, 10(4):385 –396, july-aug. 2004.

[Bia04] Silvia Biasotti. *Computational Topology Methods for Shape Modelling Applications*. PhD thesis, Universitá degli Studi di Genova, May 2004.

[BMMP03] Silvia Biasotti, Simone Marini, Michela Mortara, and Giuseppe Patané. An overview on propertiers and efficacy of topological skeletons in shape modeling. In *Shape Modeling International 2003 (SMI'03)*. IEEE, 2003.

[BMS00] Silvia Biasotti, Michela Mortara, and Michela Spagnuolo. Surface compression and reconstruction using Reeb graphs and shape analysis. In *Proceedings of 16th Spring Conference on Computer Graphics*, pages 175–184. ACM press, 2000.

[BP12] Laura Brandolini and Marco Piastra. Computing the Reeb graph for triangle meshes with active contours. In *In Proc. of ICPRAM 2012, Volume 2*, pages 80–89. SciTePress, 2012.

[BSB+03] Monte S. Buchsbaum, Lina Shihabuddin, Adam M. Brickman, Ruben Miozzo, Radovan Prikryl, Robert Shaw, and Kenneth Davis. Caudate and putamen volumes in good and poor outcome patients with schizophrenia. *Schizophr. Res.*, 64(1):53 – 62, 2003.

[BVLD09] Halim Benhabiles, Jean-Philippe Vandeborre, Guillaume Lavoué, and Mohamed Daoudi. A framework for the objective evaluation of segmentation algorithms using a ground-truth of human segmented 3D-models. In *IEEE International Conference on Shape Modeling and Applications (Shape Modeling International 2009)*, Beijing, China, June 26-28 2009. short paper.

[BW12] Margherita Barile and Eric W. Weisstein. Betti numbers, 2012. URL: `http://mathworld.wolfram.com/BettiNumber.html`.

[Car09] Gunnar Carlsson. *Topology and Data.* Bulletin of the American Mathematical Society, 2009.

[CDST97] Bernard Chazelle, David P. Dobkin, Nadia Shouraboura, and Ayellet Tal. Strategies for polyhedral surface decomposition: An experimental study. *Computational Geometry*, 7(5-6):327 – 342, 1997.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT PRESS, 2009.

[CMEH+03] Kree Cole-McLaughlin, Herbert Edelsbrunner, John Harer, Vijay Natarajan, and Valerio Pascucci. Loops in Reeb graphs of 2-manifolds. In *Proceedings of the nineteenth annual symposium on Computational geometry*, SCG 2003, pages 344–350, New York, NY, USA, 2003. ACM.

[CNR12] Visual Computing Lab ISTI CNR. Meshlab, 2012. URL: `http://meshlab.sourceforge.net/`.

[Coh73] Marshal M. Cohen. A course in simple-homotopy theory, 1973.

[COH+12] Fang Chen, Harald Obermaier, Hans Hagen, Bernd Hamann, Julien Tierny, and Valerio Pascucci. Topology analysis of time-dependent multi-fluid data using the Reeb graph. *Computer Aided Geometric Design*, (0):–, 2012.

[CSA00] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. In *Proceedings of the eleventh*

annual ACM-SIAM symposium on Discrete algorithms, SODA '00, pages 918–926, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[dBvK97] Mark de Berg and Marc van Kreveld. Trekking in the alps without freezing or getting tired. *Algorithmica*, 18:306–323, 1997.

[DE93] Cecil Jose A. Delfinado and Herbert Edelsbrunner. An incremental algorithm for Betti numbers of simplicial complexes. In *Proceedings of the ninth annual symposium on Computational geometry*, SCG 1993, pages 232–239, New York, NY, USA, 1993. ACM.

[DE95] Cecil Jose A. Delfinado and Herbert Edelsbrunner. An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere. *Computer Aided Geometric Design*, 12(7):771 – 784, 1995.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DN09] Harish Doraiswamy and Vijay Natarajan. Efficient algorithms for computing Reeb graphs. *Computational Geometry*, 42(6-7):606 – 616, 2009.

[DN12a] Harish Doraiswamy and Vijay Natarajan. Computing Reeb graphs as a union of contour trees. *Visualization and Computer Graphics, IEEE Transactions on*, PP(99):1, 2012. `doi:10.1109/TVCG.2012.115`.

[DN12b] Harish Doraiswamy and Vijay Natarajan. Output-sensitive construction of Reeb graphs. *Visualization and Computer Graphics, IEEE Transactions on*, 18(1):146 –159, jan. 2012.

[Ede01] Herbert Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, New York, NY, USA, 2001.

[EH10] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*. Amer Mathematical Society, 2010.

[EHZ03] Herbert Edelsbrunner, John Harer, and Afra Zomorodian. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discrete and Computational Geometry*, 30(1):87–107, 2003.

[Fal04] Bianca Falcidieno. AIM@SHAPE project presentation. In *Proc. Shape Modeling Applications 2004*, page 329, june 2004.

[Fou12] The Eclipse Foundation. Eclipse, 2012. URL: `http://www.eclipse.org//`.

[FSB+02] Bruce Fischl, David Salat, Evelina Busa, Marilyn Albert, Megan Dieterich, Christian Haselgrove, Andre van der Kouwe, Ron Killiany, David Kennedy, Shuna Klaveness, Albert Montillo, Nikos Makris, Bruce Rosen, and Anders M. Dale. Whole brain segmentation: automated labeling of neuroanatomical structures in the human brain. *Neuron.*, 33:341–355, 2002.

[GPO08] Jessica A. Grahn, John A. Parkinson, and Adrian M. Owen. The cognitive functions of the caudate nucleus. *Prog. Neurobiol.*, 86(3):141 – 155, 2008.

[GWH01] Michael Garland, Andrew Willmott, and Paul S. Heckbert. Hierarchical face clustering on polygonal surfaces. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, I3D '01, pages 49–58, New York, NY, USA, 2001. ACM.

[HA03] Franck Hetroy and Dominique Attali. Topological quadrangulations of closed triangulated surfaces using the Reeb graph. *Graphical Models*, 65(1-3):131 – 148, 2003.

[Hen94] Michael Henle. *A Combinatorial Introduction to Topology*. Courier Dover Publications, 1994.

[HKR93] Daniel P. Huttenlocher, Gregory A. Klanderman, and William J. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):850 – 863, sep 1993.

[HLD+06] Jaeuk Hwang, In Kyoon Lyoo, Stephen R. Dager, Seth D. Friedman, Jung Su Oh, Jun Young Lee, Seog Ju Kim, David L.

Dunner, and Perry F. Renshaw. Basal ganglia shape alterations in bipolar disorder. *Am. J. Psychiatry*, 163(2):276–285, 2006.

[HSKK01] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 203–212, New York, NY, USA, 2001. ACM.

[HWW10] William Harvey, Yusu Wang, and Rephael Wenger. A randomized o(m log m) time algorithm for computing Reeb graphs of arbitrary simplicial complexes. In *Proceedings of the 2010 annual symposium on Computational geometry*, SoCG 2010, pages 267–276, New York, NY, USA, 2010. ACM.

[Ken89] David G. Kendall. A survey of the statistical theory of shape. *Statist. Sci.*, 4(2):87–99, 1989.

[KG00] Zachi Karni and Craig Gotsman. Spectral compression of mesh geometry. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 279–286, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[KHN+07] Juha Koikkalainen, Jussi Hirvonen, Mikko Nyman, Jyrki Lötjönen, Jarmo Hietala, and Ulla Ruotsalainen. Shape variability of the human striatum - effects of age and gender. *NeuroImage*, 34(1):85 – 93, 2007.

[Kle27] Felix Klein. *Vorlesungen uber nicht-euklidische Geometrie.* Rosemann, 1927.

[KLT05] Sagi Katz, George Leifman, and Ayellet Tal. Mesh segmentation using feature point and core extraction. *The Visual Computer*, 21:649–658, 2005.

[Knu98] Donald Ervin Knuth. *The Art of Computer Programming Vol. 2: Seminumerical Algorithms.* Addison Wesley, 3rd edition, 1998.

[Kob05] Jianhua Wu Leif Kobbelt. Structure recovery via hybrid variational surface approximation. *Computer Graphics Forum*, 24(3):277–284, 2005.

[KT96] Alan D. Kalvin and Russel H. Taylor. Superfaces: polygonal mesh simplification with bounded error. *Computer Graphics and Applications, IEEE*, 16(3):64 –77, may 1996.

[LKA06] Jyh-Ming Lien, John Keyser, and Nancy M. Amato. Simultaneous shape decomposition and skeletonization. In *Proceedings of the 2006 ACM symposium on Solid and physical modeling*, SPM '06, pages 219–228, New York, NY, USA, 2006. ACM.

[LLKR07] Yu-Shen Liu, Min Liu, Daisuke Kihara, and Karthik Ramani. Salient critical points for meshes. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, SPM '07, pages 277–282, New York, NY, USA, 2007. ACM.

[LLS+05] Yunjin Lee, Seungyong Lee, Ariel Shamir, Daniel Cohen-Or, and Hans-Peter Seidel. Mesh scissoring with minima rule and part salience. *Computer Aided Geometric Design*, 22(5):444 – 465, 2005. Geometry Processing.

[LLT03] Thomas Lewiner, Hélio Lopes, and Geovan Tavares. Optimal discrete morse functions for 2-manifolds. *Computational Geometry*, 26(3):221 – 233, 2003.

[LRMT99] J. Lötjönen, P.-J. Reissman, I.E. Magnin, and T.Katila. Model extraction from magnetic resonance volume data using the deformable pyramid. *Med. Image Anal.*, 3(4):387 – 406, 1999.

[LSB01] Martin Lauer, Dieter Senitz, and Helmut Beckmann. Increased volume of the nucleus accumbens in schizophrenia. *J. Neural Transmission*, 108:645–660, 2001.

[LTS+01] M.P Laakso, J Tiihonen, E Syvälahti, H Vilkman, A Laakso, B Alakare, V Räkköläinen, R.K.R Salokangas, E Koivisto, and J Hietala. A morphometric mri study of the hippocampus in first-episode, neuroleptic-naïve schizophrenia. *Schizophr. Res.*, 50(1 - 2):3 – 7, 2001.

[LV99] Francis Lazarus and Anne Verroust. Level set diagrams of polyhedral objects. In *Fifth Symposium on Solid Modeling*, pages 130–140. ACM, 1999.

[LVB+12] Guillaume Lavoue, Jean-Philippe Vandeborre, Halim Benhabiles, Mohamed Daoudi, K. Huebner, and Michela Mortara, Michela Spagnuolo. Shrec'12 track: 3d mesh segmentation. In *Eurographics 2012 Workshop on 3D Object Retrieval*, 2012.

[LZ04] Rong Liu and Hao Zhang. Segmentation of 3D meshes through spectral clustering. In *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on*, pages 298 – 305, oct. 2004.

[LZ07] Rong Liu and Hao Zhang. Mesh segmentation via spectral embedding and contour analysis. *Computer Graphics Forum*, 26(3):385–394, 2007.

[Mil63] John Milnor. *Morse Theory*. Princeton University Press, 1963.

[Mil64] John Milnor. On the Betti numbers of real varieties. In *Proc. Amer. Math. Soc.*, 1964.

[Mor31] Marston Morse. The critical points of a function of n variables. *Trans. Amer. Math. Soc*, 33(1):72–91, 1931.

[MP02a] Michela Mortara and Giuseppe Patané. Affine-invariant skeleton of 3D shapes. *Shape Modeling and Applications, International Conference on*, 0:245–252, 2002.

[MP02b] Michela Mortara and Giuseppe Patané. Shape-covering for skeleton extraction. *International Journal of Shape Modeling*, 08(02):139–158, 2002.

[MPS+04] Michela Mortara, Giuseppe Patané, Michela Spagnuolo, Bianca Falcidieno, and Jarek Rossignac. Blowing bubbles for multi-scale analysis and decomposition of triangle meshes. *Algorithmica*, 38:227–248, 2004.

[NGH04] Xinlai Ni, Michael Garland, and John C. Hart. Fair Morse functions for extracting the topological structure of a surface mesh. *ACM Trans. Graph.*, 23(3):613–622, August 2004.

[NKI02] Marcin Novotni, Reinhard Klein, and Insitut Für Informatik Ii. Computing geodesic distances on triangular meshes. In *In Proc. of WSCG-2002*, pages 341–347, 2002.

[PBP+12] Antonietta Pepe, Laura Brandolini, Marco Piastra, Juha Koikkalainen, Jarmo Hietala, and Jussi Tohka. Simplified Reeb graph as effective shape descriptor for the striatum. In JoshuaA. Levine, RasmusR. Paulsen, and Yongjie Zhang, editors, *Mesh Processing in Medical Image Analysis 2012*, volume 7599 of *Lecture Notes in Computer Science*, pages 134–146. Springer Berlin Heidelberg, 2012.

[Pia12] Marco Piastra. Self-organizing adaptive map: Autonomous learning of curves and surfaces from point samples. *Neural Networks*, 2012.

[PSBM07] Valerio Pascucci, Giorgio Scorzelli, Peer-Timo Bremer, and Ajith Mascarenhas. Robust on-line computation of Reeb graphs: simplicity and speed. *ACM Trans. Graph.*, 26, July 2007.

[PSF09] Giuseppe Patané, Michela Spagnuolo, and Bianca Falcidieno. A minimal contouring approach to the computation of the Reeb graph. *IEEE Transactions on Visualization and Computer Graphics*, 15:583–595, 2009.

[PZT+11] Antonietta Pepe, Lu Zhao, Jussi Tohka, Juha Koikkalainen, Jarmo Hietala, and Ulla Ruotsalainen. Automatic statistical shape analysis of local cerebral asymmetry in 3D T1-weighted magnetic resonance images. In *In Proc. of MICCAI 2011 MedMesh workshop*, pages 127–134. R.R. Paulsen and J.A. Levine, 2011.

[Ree46] George Reeb. Sur les points singuliers d une forme de Pfaff completement integrable ou d une fonction numerique. In *Comptes rendus de l'Academie des Sciences 222*, pages 847–849, 1946.

[Rob10] Charles Roberts. *Introduction to Mathematical Proofs: A Transition*. CRC Press, 2010.

[RWS11] Vanessa Robins, Peter John Wood, and Adrian P. Sheppard. Theory and algorithms for constructing discrete Morse complexes from grayscale digital images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(8):1646 –1658, aug. 2011.

[RWSN09] Martin Reuter, Franz-Erich Wolter, Martha Shenton, and Marc Niethammer. Laplace-Beltrami eigenvalues and topological features of eigenfunctions for statistical shape analysis. *Comput. Aided Design*, 41(10):739 – 755, 2009.

[SAA09] Maytham Safar, Khalid Alenzi, and Saud Albehairy. Counting cycles in an undirected graph using DFS-XOR algorithm. In *Networked Digital Technologies, 2009. NDT '09. First International Conference on*, pages 132 –139, july 2009.

[SCOGL02] Olga Sorkine, Daniel Cohen-Or, Rony Goldenthal, and Dani Lischinski. Bounded-distortion piecewise mesh parameterization. In *Visualization, 2002. VIS 2002. IEEE*, pages 355 –362, nov. 2002.

[Seg08] Carol A. Seger. How do the basal ganglia contribute to categorization? their roles in generalization, response selection, and learning via feedback. *Neurosci. Biobehav.l R.*, 32(2):265 – 278, 2008.

[Sha04] Ariel Shamir. A formulation of boundary mesh segmentation. In *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on*, pages 82 – 89, sept. 2004.

[Sha08] Ariel Shamir. A survey on mesh segmentation techniques. *Comput. Graphics Forum*, 27(6):1539–1556, 2008.

[SK91] Yoshihisa Shinagawa and Tosiyasu L. Kunii. Constructing a Reeb graph automatically from cross sections. *Computer Graphics and Applications, IEEE*, 11(6):44 –51, nov 1991.

[SKK91] Yoshihisa Shinagawa, Tosiyasu L. Kunii, and Yannick L. Kergosien. Surface coding based on Morse theory. *Computer Graphics and Applications, IEEE*, 11(5):66 –78, sep 1991.

[SKK02] Thomas Sebastian, Philip Klein, and Benjamin Kimia. Shock-based indexing into large shape databases. In *Computer Vision - ECCV 2002*, volume 2352 of *Lecture Notes in Computer Science*, pages 83–98. Springer Berlin / Heidelberg, 2002.

[SKSI95] Yoshihisa Shinagawa, Tosiyasu L. Kunii, Hideyuki Sato, and Masumi Ibusuki. Modeling contact of two complex objects, with an application to characterizing dental articulations. *Computers and Graphics*, 19(1):21 – 28, 1995.

[SLK+08] Yonggang Shi, Rongjie Lai, Sheila Krishna, Ivo Dinov, and Arthur W. Toga. Anisotropic Laplace-Beltrami eigenmaps: bridging Reeb graphs and skeletons. In *In Proc. of CVPR 2008 Workshop*, pages 1–7, Anchorage, AK, USA, 2008. IEEE Computer Society.

[SSCO08] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Consistent mesh partitioning and skeletonization using the shape diameter function. *Visual Comput.*, 24:249–259, 2008.

[SSGD03] Hari Sundar, Deborah Silver, Nikhil Gagvani, and Sven Josef Dickinson. Skeleton based shape matching and retrieval. In *Shape Modeling International, 2003*, pages 130 – 139, may 2003.

[SSP+03] Y. K. Vetsa Sampath, Martin Styner, Stephen M. Pizer, Jeffrey A. Lieberman, and Guido Gerig. Caudate shape discrimination in schizophrenia using template-free non-parametric tests. In *In Proc. of MICCAI 2003*, page Part II, 2003.

[STK02] Shymon Shlafman, Ayellet Tal, and Sagi Katz. Metamorphosis of polyhedral surfaces using decomposition. *Computer Graphics Forum*, 21(3):219–228, 2002.

[SY07] Scott Schaefer and Cem Yuksel. Example-based skeleton extraction. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 153–162, Aire-la-Ville, Switzerland, 2007. Eurographics Association.

[TGSP09] Julien Tierny, Attila Gyulassy, Eddie Simon, and Valerio Pascucci. Loop surgery for volumetric meshes: Reeb graphs re-

duced to contour trees. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1177 –1184, 2009.

[TIS+95] Shigeo Takahashi, Tetsuya Ikeda, Yoshihisa Shinagawa, Tosiyasu L. Kunii, and Minoru Ueda. Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data. *Computer Graphics Forum*, 14(3):181–192, 1995.

[TVD06] Julien Tierny, Jean-Philippe Vandeborre, and Mohamed Daoudi. 3D mesh skeleton extraction using topological and geometrical analyses. In *14th Pacific Conference on Computer Graphics and Applications*. Pacific Graphics, 2006.

[TVD08] Julien Tierny, Jean-Philippe Vandeborre, and Mohamed Daoudi. Enhancing 3d mesh topological skeletons with discrete contour constrictions. *The Visual Computer*, 24:155–172, 2008.

[TVD09] Julien Tierny, Jean-Philippe Vandeborre, and Mohamed Daoudi. Partial 3D shape retrieval by Reeb pattern unfolding. *Computer Graphics Forum*, 28(1):41–55, 2009.

[VGS00] Hans-Peter Volz, Christian Gaser, and Heinrich Sauer. Supporting evidence for the model of cognitive dysmetria in schizophrenia - a structural magnetic resonance imaging study using deformation-based morphometry. *Schizophr. Res.*, 46(1):45 – 56, 2000.

[vis10] visimp. Vtkreebgraphs, 2010. URL: `https://visimp.cs.unc.edu/2010/10/26/reeb-graphs/`.

[War83] Frank W. Warner. *Foundations of differentiable manifolds and Lie groups*. Springer, 1983.

[Wei12] Eric W. Weisstein. Barycentric coordinates, 2012. URL: `http://mathworld.wolfram.com/BarycentricCoordinates.html`.

[Wik09] Wikipedia. Human striatum, 2009. URL: `http://en.wikipedia.org/wiki/File:BrainCaudatePutamen.svg`.

[Wik12a] Wikipedia. Betti numbers, 2012. URL: `http://en.wikipedia.org/wiki/Betti_number`.

[Wik12b] Wikipedia. Genus, 2012. URL: `http://en.wikipedia.org/wiki/Genus_(mathematics)`.

[Wik12c] Wikipedia. Klein bottle, 2012. URL: `http://en.wikipedia.org/wiki/Klein_bottle`.

[Wik12d] Wikipedia. Moebius strip, 2012. URL: `http://en.wikipedia.org/wiki/Moebius_strip`.

[WXS06] Naoufel Werghi, Yijun Xiao, and Jan Paul Siebert. A functional-based segmentation of human body scans in arbitrary postures. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 36(1):153 –165, feb. 2006.

[XSW03] Yijun Xiao, Paul Siebert, and Naoufel Werghi. A discrete Reeb graph approach for the segmentation of human body scans. In *3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings. Fourth International Conference on*, pages 378 – 385, oct. 2003.

[ZH04] Yinan Zhou and Zhiyong Huang. Decomposing polygon meshes by means of critical points. In *Multimedia Modelling Conference, 2004. Proceedings. 10th International*, pages 187 – 195, jan. 2004.

[Zom05] Afra Zomorodian. *Topology for computing*. Cambridge University Press, 2005.