Università
di Genova

Dipartimento di
Informatica, Bioingegneria,
Robotica e Ingegneria dei Sistemi

Drawing on Surfaces

Claudio Mancinelli

Theses Series DIBRIS-TH-2022-42

Ph.D. Thesis

Università di Genova
Dipartimento di Informatica, Bioingegneria,
Robotica ed Ingegneria dei Sistemi
Ph.D. Thesis in
Computer Science and System Engineering
Computer Science Curriculum

Drawing on Surfaces

by

Claudio Mancinelli

April 2022

Ph.D. Thesis in Computer Science and System Engineering (S.S.D. INF/01)
Dipartimento di Informatica, Bioingegneria, Robotica ed Ingegneria dei Sistemi
Università di Genova

Candidate
Claudio Mancinelli
`claudio.mancinelli@dibris.unige.it`

Title
Drawing on Surfaces

Advisors
Enrico Puppo
DIBRIS, Università di Genova
`enrico.puppo@unige.it`

External Reviewers
Pierre Alliez,
National Institute for Research in Digital Science and Technology, Sophia-Antipolis,
`pierre.alliez@inria.fr`

Marco Tarini,
Department of Computer Science - Università degli Studi di Milano,
`marco.tarini@unimi.it`

Location
DIBRIS, Univ. di Genova
Via Opera Pia, 13
I-16145 Genova, Italy

Submitted On
April 2022

Abstract

Vector graphics in 2D is consolidated since decades, as is supported in many design applications, such as Adobe Illustrator [Ado21], and languages, like Scalable Vector Graphics (SVG) [W3C10]. In this thesis, we address the problem of designing algorithms that support the generation of vector graphics on a discrete surface. We require such algorithm to rely on the intrinsic geometry of the surface, and to support real time interaction on highly-tessellated meshes (few million triangles). Both of these requirements aim at mimicking the behavior of standard drawing systems in the Euclidean context in the following sense. Working in the intrinsic setting means that we consider the surface as our canvas, and any quantity needed to fulfill a given task will be computed directly on it, without resorting to any type of local/global parametrization or projection. In this way, we are sure that, once the theoretical limitations behind some given operation are properly handled, our result will always be consistent with the input regardless of the surface we are working with. As we will see, in some cases, this may imply that one geometric primitive cannot be indefinitely large, but must be contained in a proper subset of the surface. Requiring the algorithms to support real time interaction on large meshes makes possible to use them via a click-and-drag procedure, just as in the 2D case. Both of these two requirements have several challenges. On the one hand, working with a metric different from the Euclidean one implies that most of the properties on which one relies on the plane are not preserved when considering a surface, so the conditions under which geometric primitives admit a well defined counterpart in the manifold setting need to be carefully investigated in order to ensure the robustness of our algorithms. On the other hand, the building block of most of such algorithms are geodesic paths and distances, which are known to be expensive operations in computer graphics, especially if one is interested in accurate results, which is our case.

The purpose of this thesis, is to show how this problem can be addressed fulfilling all the above requirements. The final result will be a Graphical User Interface (GUI) endowed with all the main tools present in a 2D drawing system that allow the user to generate geometric primitives on a mesh in robust manner and in real-time.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor, Enrico Puppo, even if I don't think that I will be able to find words that faithfully describe how grateful I am for his support. Besides the fact that I owe to him most of the things I learned, his priceless guidance has been always present during these years, he always encouraged me to pursue my ideas, being always available for useful discussions and helping me in all the difficult moments I encountered.

I am also very grateful to Kai Hormann, which supervised me during my visiting period in Lugano, at USI (Universitá della Svizzera Italiana). He made those three months a wonderful experience, in which I learned an incredible amount of things and had the possibility of meeting wonderful people.

A special thank goes to Fabio Pellacini, being always present in helping me with my implementations and providing useful advices on every level. The calls we had were precious moments of this journey.

Part of the merit for achieving this milestone goes to Marco Livesu for his huge support, especially at the beginning of my PhD. His patience while teaching me most of the things I know about C++ programming was immeasurable.

Special thanks go to Carlo Mantegazza for being always available for useful discussions during the writing of this thesis and the paper on the cut locus, his help was crucial in every occasion.

I also would like to thank Pierre Alliez and Marco Tarini for having reviewed this thesis. Their feedback and suggestions have been very helpful to improve the presentation.

I am very grateful to my PhD committee, especially to Giuseppe Patané. He always provided useful comments and feedback at every presentation I delivered, and was always available to advise me whenever I consulted him.

For the wonderful moments shared during this journey, I am very grateful to my friends and colleagues, both in at DIBRIS and USI. Thank you all. Among them, special thanks go to Eleonora, Filippo, Davide, Veronica, Vanessa, Francesco, Federico, and Alexandros.

Last but not least, I would like to thank my family for their love and support.

Contents

Introduction

1

Introduction

Software packages for the generation of vector graphics in 2D are available in many forms nowadays. From the most basic ones, which can be found and used online, to the more advanced ones, which give the user a wide range of tools to generate and edit visual images. Such packages are extensively used in many applications, such as design, engineering and manufacturing. The concept of *geometric primitive* is at the core of this technology. Essentially, geometric primitives are basic geometric shapes that can be used and combined to generate more complex ones. Such primitives are defined in a plane endowed with the Euclidean metric, together with a Cartesian coordinate system, and the algorithms to generate them are based on this structure. To fix ideas, a straight line segment can be traced once its two endpoints are defined, since we have a closed form to describe the other points on it and, of course, in order to exploit such closed form we need the coordinates of the endpoints in a common reference system.

The purpose of this thesis is to describe several algorithms that generate the counterparts of such geometric primitives in a different domain, endowed with a different metric. In particular, we will no longer consider a plane, but a *surface*. In other words, if the plane is the piece of paper on which we are drawing, then we are concerned in understanding how to turn such piece of paper into a curved domain, in order to be able to draw on a sphere, for example. It is important to point out that our strategy will not consist in drawing on the piece of paper and wrap the paper to form a surface, but rather forget about the piece of paper (hence all the geometry we were relying on in that setting), and investigate on how the constructions we were able to do in the planar case can be extended to the surface setting. The main theoretical problem we will face in such task is that, when working in a context with a metric different from the Euclidean one, even the simplest concepts such as a straight line or a circle do not preserve all their properties. Therefore, one needs to find solutions that guarantee some kind of consistency with their Euclidean counterparts, and at the same time that are defined in an *intrinsic* way. Roughly speaking, this means that we define them using just what we have "on the surface", i.e. ignoring every other possible technique that takes place in a space different from our domain. Although being more challenging than working, for example, in $\mathbb{R}^3$ and then find a way to map the result onto the surface, this approach makes our algorithms well defined and robust regardless of the surface we are working on. Some operations may undergo to some constraint imposed by the fact that we are working on a curved domain, but once such constraints are met, we will be able to generate geometric primitives free of artifacts, being sure that the final result will be consistent with our expectations.

From a computational point of view, our algorithms must be designed keeping in mind that they need to be efficient. In fact, we aim at reproducing the behavior of the 2D drawing systems, where all the operations can be done via an intuitive click-and-drag procedure, and the result is shown in real time. For this reason, each of the proposed algorithms for the generation of primitives in this thesis has been implemented in a way that guarantees the real time interaction on meshes consisting of few million triangles. This constituted a challenge mainly for the following reason. In the Euclidean setting, most of the definitions of geometric primitives involves distances and/or straight lines, both of which can be computed in closed form. Conversely, when considering two points on a surface, we need to compute their *geodesic distance*, which means that we need to find the shortest path on the surface connecting them. Unfortunately, we do not have a closed form for the geodesic distance between two points, and similarly for the analogue of the straight lines on a surface, which are called *geodesics*. The algorithms that compute geodesic paths and distances are known to be expensive, especially if a certain amount of accuracy is requested, which is our case. A poor estimation of such quantities may lead to, e.g., wiggly geodesic circles or broken splines, which is not acceptable.

1.1 Outline

Part I We introduce the theoretical concepts upon which our algorithms rely and describe how such concepts are brought into a discrete setting. In doing so, we also review the main state-of-the-art methods and briefly compare their results with the ones obtained with our implementations.

Chapter 2 We present background notions of Riemannian geometry and report the main results that will be exploited in the sequel.

Chapter 3 We describe the data structures used to encode a discrete surface and the basic tools used in most of our implementations.

Chapter 4 We review the main methods for the computation of geodesic paths and distances, and we describe how such quantities are computed in our setting.

Part II We present our solutions for the problem of defining geometric primitives on a discrete surface. We start by describing an algorithm that estimates the cut locus of a point on mesh, which will be crucial in determining whether our construction are well defined or not. Algorithms for the generations of various geometric primitives are then described.

Chapter 5 We describe an algorithm that estimates the cut locus on a discrete surface. With respect to previous methods in the literature, our approach compute an approximation of the cut locus in seconds on large meshes with arbitrary genus.

Chapter 6 We present different techniques to generate the simplest geometric primitives on a surface. These techniques are compatible with real-time interaction and they are integrated in the context of a prototype interactive system.

Chapter 7 We present two algorithms for the tracing of Bézier curves on a mesh are presented. We show that such algorithms produce C^1 and C^2 curves for any arbitrary positioning of the control points on the surface. Such algorithms are integrated to extend our prototype system.

1.2 Notations and Conventions

The terms "smooth" and "differentiable" we will be used meaning C^∞ continuity. When this does not cause amibiguities, we will write simply 0 to denote the origin of $\mathbb{R}^n$. For example, if $\alpha : [a, b] \to \mathbb{R}^3$ is a curve in the the space, we will write $\alpha(t) \neq 0$ meaning $\alpha(t) \neq (0, 0, 0)$, $t \in [a, b]$. Moreover, the list below describes the main notations that will be used in the following. Note that, the symbols used in the continuous setting will be used in the discrete setting as well, with obvious meaning. For example, T_pM will describe the tangent space at a point p belonging to the triangle mesh M.

Continuous setting

$\mathcal{M}, S$	Complete Riemannian 2-manifold
X, Y	Vector fields on $\mathcal{M}$
$T_p\mathcal{M}$	Tangent space at a point $p \in \mathcal{M}$
$\{\frac{\partial}{\partial x^i}\} = \{\partial_i\}$	Basis of $T_p\mathcal{M}$ corresponding to the local coordinates (x^0, x^1)
$\langle \cdot\, , \cdot \rangle$	Euclidean inner product
$\langle \cdot\, , \cdot \rangle_g$	Inner product on $T_p\mathcal{M}$
$d_\mathcal{M}(\cdot, \cdot)$	Geodesic distance on $\mathcal{M}$
$d_p(\cdot)$	Geodesic distance functions sourced at $p \in \mathcal{M}$
$\mathcal{D}(U)$	Set of C^∞ real-valued functions defined on U
$\mathcal{X}(U)$	Set of C^∞ vector fields defined on U
γ	Geodesic on $\mathcal{M}$
γ_{pq}	Minimizing geodesic connecting p to q, $p, q \in \mathcal{M}$
K	Gaussian curvature

$\nabla_X Y$	Covariant derivative of the vector field Y along the vector field X
df_p	Differential of $f \in \mathcal{D}(\mathcal{M})$ at $p \in \mathcal{M}$
∇f	Riemannian gradient of $f \in \mathcal{D}(\mathcal{M})$
$\nabla^2 f$	Second covariant derivative of $f \in \mathcal{D}(\mathcal{M})$
$\mathrm{Hess} f$	Hessian of $f \in \mathcal{D}(\mathcal{M})$
Δf	Laplacian of $f \in \mathcal{D}(\mathcal{M})$

Discrete setting

M	Triangle mesh with vertices, edges and triangles denoted by V, E and T, respectively
e_{ij}	Edge connecting the vertices $v_i, v_j \in M$
t_{ijk}	Triangle having vertices $v_i, v_j, v_k \in M$
θ_i^{jk}	Angle at v_i of t_{ijk}
$\mathcal{S}(v)$	Star (region) of the vertex $v \in M$
$\mathcal{N}(v)$	1-ring (vertices) of the vertex $v \in M$
$Lk(v)$	Link of the vertex $v \in M$

PART I

Preliminaries

2

Continuous Setting

In this chapter the theoretical context in which most of the algorithms proposed in this thesis are defined is presented. Since the objects targeted by such algorithms are discrete surfaces (*meshes*), a formal definition of what we mean with “surface” will be introduced. Secondly, we will introduce the concepts that will allow us to describe the properties of such objects, such as their curvature at a given point or the length of a curve on them. The properties we are more interested in are the *intrinsic* ones, rather than the *extrinsic* ones. Before entering the details and giving formal definitions, we will elaborate a bit more on what has just been said by considering two examples that motivate the theory developed in the following.

Let us consider the unit sphere S^2 in $\mathbb{R}^3$ and let p be a point on S^2. To fix ideas, one can think about S^2 as the surface of the earth and at p as our position on it. Then, it is quite simple to convince ourselves that, locally, the surface we are living on seems flat, with two dimensions only. In fact, the maps we use to navigate the earth are 2-dimensional. When travelling far, we may need another map, which overlaps with the one we were following so that we can switch from one to the other, but all of them faithfully represent the streets surrounding us so that it is possible to reach every position on earth relying exclusively on these maps. Let now $\gamma : [a, b] \to S^2$ be a differentiable curve on S^2, and suppose that $\gamma(a) = p$, where “differentiable” means that, if $\gamma(t) = (x(t), y(t), z(t))$, then the functions $x(t), y(t), z(t)$ are differentiable. We can think about γ as a path we need to follow to reach some position $\gamma(b)$. Then $\dot{\gamma}(t)$ is the vector tangent to γ (and hence to S^2) at $\gamma(t)$. Therefore, we can think about $\dot{\gamma}(a)$ as the direction in which we need to move our first step in order to follow the trajectory defined by γ on S^2. However, $\dot{\gamma}(t) = (\dot{x}(t), \dot{y}(t), \dot{z}(t))$ is a 3-dimensional vector, which seems quite inconsistent with the fact that locally our surface is well described by a portion of the plane. Taking a step further, let us consider $\ddot{\gamma}(t)$, which geometrically is a vector orthogonal to $\dot{\gamma}(t)$ whose magnitude is equal to the curvature of γ at t. Now we can clearly see how the excess of information brought by considering 3D vectors to describe the geometric properties of γ becomes disturbing when trying to follow its trajectory on S^2. In fact, the component of $\ddot{\gamma}(t)$ we are interested in is the one that gives us information about the entity of the drifts we have to do in order to be compliant with the direction of $\dot{\gamma}(t)$. However, $\ddot{\gamma}(t)$ encodes also the curvature due to the fact that γ is curving to remain on S^2, which is totally irrelevant to us. Referring to Figure 2.1, we can say that we are interested in

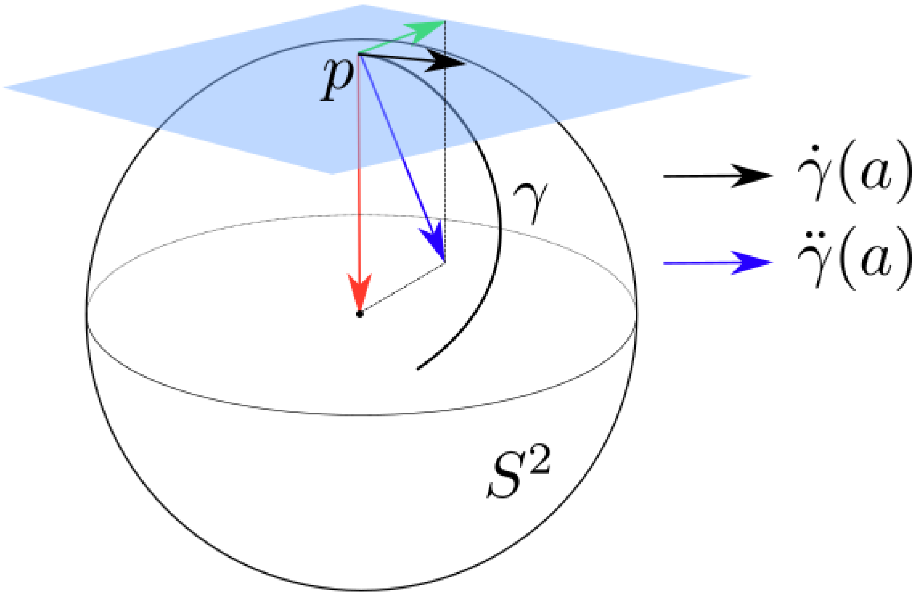


Figure 2.1: The intrinsic component (green vector) of $\ddot{\gamma}(a)$(blue vector) gives information about how much γ is "bending" on the blue plane, while the extrinsic component (red vector) is the curvature "necessary to keep γ on S^2". In fact, its magnitude coincides with the Gaussian curvature of S^2.

the variations that γ has with respect to the blue plane, i.e. the plane which $\dot{\gamma}(a)$ belongs to. The component of $\ddot{\gamma}(a)$ depicted with a green vector indicates how much γ is drifting from the point of view of someone that is standing on such plane at p, while the other one (red vector), is the component due to the fact that γ needs to bend in order to stay on the sphere. This latter component is an *extrinsic* quantity, while the former is an *intrinsic* one. In a nutshell, intrinsic properties "forget" about the embedding space and give information about what happens on the earth, while extrinsic ones depend on the ambient space. For this reason, the context in which all the theory in this chapter will be described is the one of *intrinsic geometry*, i.e. we will be interested in what happens *on* the surface ignoring everything around it. Nevertheless, sometimes we may consider the extrinsic point of view, since some concepts may have a more intuitive interpretation when viewed in such setting. We will point out when this is the case and use different notations in order to avoid ambiguities.

Let now $\alpha : [a, b] \longrightarrow \mathbb{R}^3$ be a regular parametrized differentiable curve in $\mathbb{R}^3$, where "regular" means that $\dot{\alpha}(t) \neq 0$ for every $t \in [a, b]$. Given $t \in [a, b]$, the *arc-length* of α from a is by definition

$$s(t) := \int_a^t \|\dot{\alpha}(u)\| du,$$

where

$$\|\dot{\alpha}(u)\|^2 = \langle \dot{\alpha}(u) , \dot{\alpha}(u) \rangle,$$

and $\langle \cdot , \cdot \rangle$ is the Euclidean inner product. Supose now that $\alpha(a) = p_0$ and $\alpha(b) = p_1$, where p_0 and p_1 are two points in $\mathbb{R}^3$. Then, by Fundamental theorem of Calculus we have that

$$\alpha(b) - \alpha(a) = \int_a^b \dot{\alpha}(t)dt.$$

Therefore, for every unit vector $v \in \mathbb{R}^3$, by the Cauchy-Schwartz inequality we have that

$$\langle v\,, p_1 - p_0 \rangle = \langle v\,, \alpha(b) - \alpha(a) \rangle \leq \int_a^b \|\dot{\alpha}(s)\| ds, \tag{2.1}$$

Now, by taking $v = \frac{(p_1-p_0)}{\|p_1-p_0\|}$ and comparing the first and the last term of (2.1), we conclude that every curve connecting p_0 to p_1 has length at least $\|p_1 - p_0\|$, which is the *Euclidean distance* $d_E(p_0, p_1)$ from p_0 to p_1. Of course, the curve whose length realizes such distance is the straight line $r(t) = p_0 + t(p_1 - p_0)$, $t \in [0, 1]$, which is unique up to parametrization. Summarizing, we have that the distance between two points in $\mathbb{R}^3$ is equal to the length of the shortest curve connecting them. The curves in $\mathbb{R}^3$ that realize the distance between any two points on them are straight lines, i.e. curve with zero curvature, and they are uniquely defined once two points on them are fixed. Last but not least, to give this notion of distance we only used the inner product of $\mathbb{R}^3$.

In Section 2.1 the intuitive idea of surface that we previously developed will be formalized. We will consider a set that can be covered with such charts, which maps portion of it diffeomorphically to an open set of $\mathbb{R}^2$, i.e. a disk. Upon this definition, we will build a framework that will satisfy all the requests formulated in the examples presented before. This will lead to the definition of *Riemannian manifold*.

In Section 2.2 we will characterize the curves that realize the distance between two points on a surface. In fact, since such curves are always the *straightest* among all others curves (just as straight lines in $\mathbb{R}^3$), we will formally define them. We will then introduce two important mappings that allow us to consider a local system of coordinates based on this latter definition.

In Section 2.3, the concept of curvature will be introduced. We will see that this quantity can be defined as a measure of how much the surface fails to be locally flat.

We will conclude the chapter with Section 2.4, in which the definition of *cut locus* of a point and the concept of *convexity* will be introduced. Both of these concepts will play an important role in the sequel.

For the sake of simplicity, such definitions and all the related results will be given in the 2-dimensional setting, although they are extendable to n-dimensional manifolds. Any further detail about the content of this chapter can be found in any introductory book to Differential Geometry, e.g. [dC92, Sak97, GHL04].

2.1 Riemannian Manifolds

2.1.1 Differentiable Manifolds

We will start by formalizing the concepts of local maps or charts defined previously.

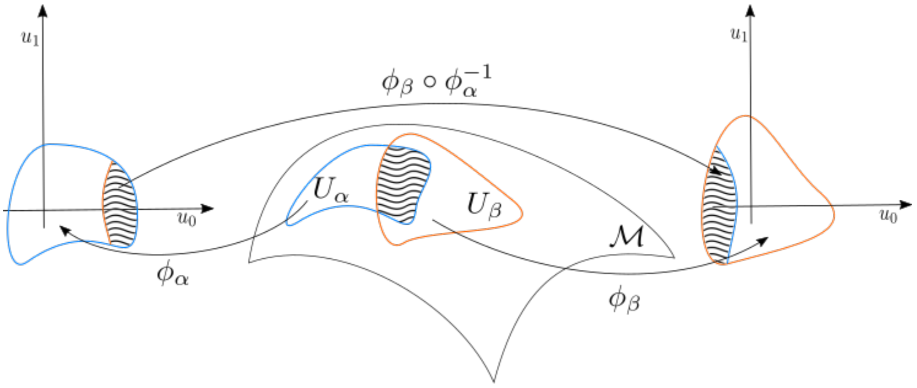
DEFINITION 2.1 (Local chart): Let $\mathcal{M}$ be a Hausdorff topological space. A pair (U, ϕ) of an open set U of $\mathcal{M}$ and a homeomorphism $\phi : U \longrightarrow \mathbb{R}^m$ from U to an open set of R^m is called a *(local) chart* and U is called a *coordinate neighborhood.*

We are now ready to give the definition of a differentiable 2-manifold. Essentially, such definition will consists in requiring that such object must locally resemble to a portion $\mathbb{R}^2$, and we will use local charts to formalize what we mean with "resemble".

DEFINITION 2.2 (Differentiable 2-manifold): Let A be an indexing set. Then, if there exists a family $\{(U_\alpha, \phi_\alpha)\}_{\alpha \in A}$ in $\mathcal{M}$ such that:

i) $\bigcup\limits_{\alpha \in A} \phi_\alpha(U_\alpha) = \mathcal{M}$,
ii) whenever $V := U_\alpha \cap U_\beta \neq \emptyset$, coordinates transformations $\phi_\beta \circ \phi_\alpha^{-1} : \phi_\alpha(V) \longrightarrow \phi_\beta(V)$ are C^∞ maps between open subsets of $\mathbb{R}^2$ (see figure below),

then $\mathcal{M}$ is a *differentiable* (or *smooth*) *manifold* of dimension 2 with an *atlas* $\{(U_\alpha, \phi_\alpha)\}_{\alpha \in A}$.



Let (u^0, u^1) denotes the coordinates in $\mathbb{R}^2$. For a chart (U_α, ϕ_α), we set $x^i_\alpha := u^i \circ \phi_\alpha$, $i = 0, 1$, which are called *local coordinates.* In the following, when we will consider the local coordinates representation of some quantity defined in a neighborhood U of a point p in a manifold $\mathcal{M}$, we will mean that a chart (U, ϕ, x^i) has been fixed. Such an assumption will be pointed out for a while, and at some point we will freely consider such coordinates implicitly referring to (U, ϕ, x^i).

At a first glance, condition ii) of the theorem above seems to impose more than what we were looking for. In fact, referring to the example of the sphere made before, at this stage we only wanted to formalize the idea that 2D maps can be used to move on S^2. For this purpose, C^0 charts would be enough, since we are only interested in moving from U_α to U_β and having consistency between the maps ϕ_α and ϕ_β during the transition. In this case, we would say that $\mathcal{M}$ is a *topological manifold*. However, the direction we wanted to follow was given by the tangent $\dot{\gamma}(t)$ of a given curve γ, obtained by differentiating the three components of γ with respect to t. In our current setting, we do not have the possibility of considering (and in particular, differentiating) objects defined in the ambient space, since we do not have one anymore. We therefore assume the smoothness of the local charts in order to exploit it to define the differentiation of functions on the manifold and, in particular, tangent vectors. For the latter, we are going to consider two different definitions: the first one has the purpose of giving an intuitive geometrical idea of what a vector tangent to a manifold is, the second one will be used when vector fields will be introduced.

When we considered the curve γ on the sphere, we observed that, since $\dot{\gamma}(a)$ is a vector tangent to γ at $\gamma(a) = p$, then such vector is also tangent to S^2 at p. In fact, it is well known that there is a plane *tangent* to S^2 at p in which the tangent vectors at p of all the curves through p are defined. It is therefore reasonable to define the tangent space to a differentiable manifold following this observation. However, since in this setting we cannot freely differentiate the components of γ as before, we will exploit the differentiable structure of the manifold to consider the tangent vector to a curve at a given point. Let now I be an open interval containing 0. A C^∞ map $c : I \longrightarrow \mathcal{M}$ such that $c(0) = p$ is called a (C^∞) *curve* through $p \in \mathcal{M}$.

DEFINITION 2.3 (Tangent vector): Let $\mathcal{M}$ be a differentiable 2-manifold and let (U, ϕ) be a chart around $p \in \mathcal{M}$. A *tangent vector* to $\mathcal{M}$ at p is an equivalence class of curves $c : I \longrightarrow \mathcal{M}$, where I and c are as above, for the equivalence relation $\sim$ defined by

$$c \sim \sigma \iff \left.\frac{d}{dt}\right|_{t=0} (\phi \circ c)(t) = \left.\frac{d}{dt}\right|_{t=0} (\phi \circ \sigma)(t).$$

The *tangent space* to $\mathcal{M}$ at p, denoted with $T_p\mathcal{M}$, is the set of all tangent vectors to $\mathcal{M}$ at p. Essentially, a tangent vector $\xi \in T_p\mathcal{M}$ is represented by curve c and, in order to be able to assess which curves have the same tangent at p, we use a chart to make this comparison between vectors in $\mathbb{R}^2$. Similarly, one can introduce the concept of a smooth function on $\mathcal{M}$. Let thus $f : \mathcal{M} \longrightarrow \mathbb{R}$ be a real-valued function on a smooth manifold $\mathcal{M}$. Then f is said to be C^∞ at $p \in \mathcal{M}$ if $f \circ \phi^{-1} : \phi(U) \longrightarrow \mathbb{R}$ is C^∞ at $\phi(p)$, where (U, ϕ) are defined as above. Note that, by condition ii), both these latter definitions do not depend on the choice of charts around p. In the following, we will denote with $\mathcal{F}(V)$ the set of all real-valued functions defined on an open set $V \subseteq \mathcal{M}$ and of class C^∞ everywhere. We also denote with $\mathcal{F}(p)$ the family of C^∞ functions defined in a neighborhood of p. We will also denote with $\mathcal{D}(\mathcal{M})$ the set of all real-valued functions defined on $\mathcal{M}$ that are differentiable, and, with obvious meaning of the notations, we will use $\mathcal{D}(V)$ and $\mathcal{D}(p)$.

Let us now c be as above, and let us consider the directional derivative $\xi f := \frac{d}{dt}\big|_{t=0} f(c(t))$ of $f \in \mathcal{F}(p)$, where $\xi = \dot{c}(0)$. Such derivative satisfies

$$\xi(af + bg) = a\xi f + b\xi g, \qquad \xi(fg) = f(p)\xi g + g(p)\xi f, \tag{2.2}$$

where $a, b \in \mathbb{R}$ and $f, g \in \mathcal{F}(p)$. The reader may recognize in the above equations the properties that the usual differentiation in the Euclidean setting satisfies. In fact, the mapping $\xi : \mathcal{F}(p) \to \mathbb{R}$ satisfying (2.2) is called a *derivation* of $\mathcal{F}(p)$. By defining $(a\xi + b\eta)f := a\xi f + b\eta f$ for derivations ξ, η and $a, b \in \mathbb{R}$, we can identify the tangent space $T_p\mathcal{M}$ as the space of all derivations of $\mathcal{F}(p)$, which is a vector space. If (U, ϕ, x^i) is a chart, then for $p \in U$ we define $\frac{\partial}{\partial x^i}(p) \in T_p\mathcal{M}$ $(i = 0, 1)$ by

$$\frac{\partial}{\partial x^i}(p)f := \left(\frac{\partial}{\partial u^i}\right)(f \circ \phi^{-1})(\phi(p)),$$

where $\frac{\partial}{\partial u^i}$ denotes the partial differentiation with respect to the i-th coordinate. Then $\{\frac{\partial}{\partial x^0}(p), \frac{\partial}{\partial x^1}(p)\}$ gives a basis of $T_p\mathcal{M}$ for each $p \in \mathcal{M}$. From now on, we also write $\frac{\partial}{\partial x^i}$, or simply ∂_i, when this does not cause ambiguity. Now let $T\mathcal{M} := \cup_{p\in\mathcal{M}} T_p\mathcal{M}$ be the set of all tangent vectors to $\mathcal{M}$ and $\tau_{\mathcal{M}} : T\mathcal{M} \to \mathcal{M}$ the map that associates p to every $\xi \in T_p\mathcal{M}$. Then $T\mathcal{M}$ carries a 4-dimensional C^∞ manifold structure and $\tau_{\mathcal{M}}$ is a C^∞ map. We call $T\mathcal{M}$ the *tangent bundle* of $\mathcal{M}$.

With the above notations, a vector $\xi \in T_p\mathcal{M}$ can be therefore written as

$$\xi = \xi^i \frac{\partial}{\partial x^i},$$

and $\xi(f)$, or simply ξf as

$$\xi f = \xi^i \frac{\partial f}{\partial x^i}.$$

Here and in the sequel we use the Einstein summation convention that dictates an implicit sum over indices appearing twice in lower and upper position in expressions. Thus, $\xi^i \frac{\partial}{\partial x^i}$ is an abbreviation for

$$\sum_{i=0}^{1} \xi^i \frac{\partial}{\partial x^i},$$

since the index i appears in upper position in ξ^i and in lower position in $\frac{\partial}{\partial x^i}$.

The *gradient field* of some $f \in \mathcal{D}(\mathcal{M})$ will play an important role in most of this thesis. In order to define it, we first need to introduce the concept of *vector field* on a manifold.

DEFINITION 2.4 (Vector field): Let $\mathcal{M}$ be a smooth 2-manifold, and suppose that every point $p \in \mathcal{M}$ a tangent vector $X_p \in T_p\mathcal{M}$ is assigned. If a map

$$X : \mathcal{M} \to T\mathcal{M}$$
$$p \mapsto X_p$$

is C^∞, then X is said to be a (C^∞) *vector field* on $\mathcal{M}$.

We will denote with $\mathcal{X}(\mathcal{M})$ the set of all vector fields on $\mathcal{M}$. The definition of a tangent vector as a derivation is quite convenient to see a vector field X as a mapping from $\mathcal{D}(\mathcal{M})$ to $\mathcal{F}(\mathcal{M})$. In details, with respect to a chart (U, ϕ, x^i) we have the vector fields $\frac{\partial}{\partial x^i} : q \mapsto \frac{\partial}{\partial x^i}(q)$ for every $q \in U$ $(i = 0, 1)$. Then for every $X \in \mathcal{X}(U)$ and $q \in U$, X_q may be written as

$$X_q = X^i(q)\frac{\partial}{\partial x^i}(q),$$

where $X^i \in \mathcal{F}(U)$, $i = 0, 1$. We can thus think of X as a mapping $X : \mathcal{D}(\mathcal{M}) \longrightarrow \mathcal{F}(\mathcal{M})$ defined as

$$(Xf)(q) = X^i(q)\frac{\partial f}{\partial x^i}(q). \tag{2.3}$$

Equation (2.3) tells us that X is a mapping that associates to every $f \in D$ its directional derivative Xf along $X_q \in T_q\mathcal{M}$, for every $q \in \mathcal{M}$. Note that, if $Xf \in \mathcal{D}(\mathcal{M})$, then it is immediate that X is differentiable, i.e. $X : \mathcal{D}(\mathcal{M}) \longrightarrow \mathcal{D}(\mathcal{M})$.

Give a function $F : \mathcal{M} \longrightarrow \mathcal{N}$ between two smooth 2-differentiable manifolds, we can now define the *differential* (or *pushforward*) of F at a given point $p \in \mathcal{M}$ as a mapping between the tangent spaces $T_p\mathcal{M}$ and $T_{F(p)}\mathcal{N}$. As we defined tangent vectors in two different ways, we will give this definition according to both representations.

DEFINITION 2.5 (Differential): Let $\mathcal{M}$ and $\mathcal{N}$ be two smooth manifolds, and let $F : \mathcal{M} \longrightarrow \mathcal{N}$ be a differentiable mapping between them. For every $p \in M$ and for each $\xi \in T_p\mathcal{M}$, let $\gamma : (-\delta, \delta) \longrightarrow \mathcal{M}$ be a smooth curve such that $\gamma(0) = p$ and $\dot{\gamma}(0) = \xi$. Let σ be the image of γ through F, i.e. $\sigma = F \circ \gamma$. Then the linear mapping $dF_p : T_p\mathcal{M} \longrightarrow T_{F(p)}\mathcal{N}$ given by $df_p(\xi) = \frac{d}{dt}\big|_{t=0}\sigma(t)$ is called the *differential* of F at p.

The above definition gives a clear geometrical interpretation of the differential: the vector tangent to a given curve through p is mapped to the vector tangent to the image of that curve through F. However, the interpretation of tangent vectors as derivations allows us to express the differential in local coordinates, which will be used in the sequel.

With the notations used above, let (U, ϕ, x^i) and (V, ψ, y^j) be two charts in p and $F(p) = (F^0(x^0, x^1), F^1(x^0, x^1))$, respectively. Then dF_p may be defined as

$$dF_p(\xi)(g) := \xi(g \circ F), \qquad g \in \mathcal{F}(F(p)),$$

i.e.

$$\begin{aligned} dF_p &: T_p\mathcal{M} \longrightarrow T_{F(p)}\mathcal{N} \\ \xi = \xi^i\frac{\partial}{\partial x^i}(p) &\mapsto \xi^i\frac{\partial F^j}{\partial x^i}(p)\frac{\partial}{\partial y^j}F(p). \end{aligned}$$

We thus see that the differential of F at p maps a vector $\xi \in T_p\mathcal{M}$ to a vector $w \in T_{F(p)}\mathcal{N}$ such that the components w^j of w are the directional derivatives of the components of F along ξ. Note that, if F is invertible, the above equation suggests that dF somehow induces a way of computing directional derivatives of the functions $g \in \mathcal{D}(\mathcal{N})$. Actually, we already exploited this fact when we defined the basis $\{\frac{\partial}{\partial x^0}(p), \frac{\partial}{\partial x^1}(p)\}$ of $T_p\mathcal{M}$. In fact, such vectors can be written as

$$\frac{\partial}{\partial x^i}(p) = d_{\phi(p)}\phi^{-1}\left(\frac{\partial}{\partial u^i}\right),$$

i.e. we used the image of the basis of $T_{\phi(p)}\mathbb{R}^2$ through the differential of ϕ^{-1} to define the basis of $T_p\mathcal{M}$. See Figure 2.2.

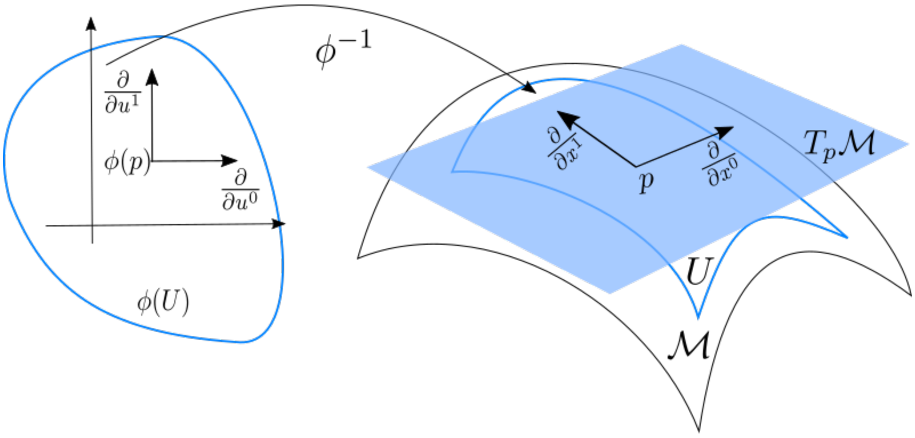

ϕ^{-1}
$\frac{\partial}{\partial u^1}$
$\phi(p)$
$\frac{\partial}{\partial u^0}$
$\phi(U)$
$\frac{\partial}{\partial x^1}$
p
$\frac{\partial}{\partial x^0}$
$T_p\mathcal{M}$
U
$\mathcal{M}$

Figure 2.2: The differential of ϕ^{-1} at $\phi(p)$ maps $\frac{\partial}{\partial u^i}$ to $\frac{\partial}{\partial x^i}$, $i = 0, 1$.

We conclude this section by giving the definition of integral curve of a vector field.

DEFINITION 2.6 (Integral curve): Given a vector field X on $\mathcal{M}$ and $p \in \mathcal{M}$, a curve $c : (-\delta, \delta) \longrightarrow \mathcal{M}$ such that $c(0) = p$ is called an *integral curve* of X through p is $X_{c(t)} = \dot{c}(t)$ holds everywhere.

2.1.2 Riemannian Metric

If we think again at the example made at the beginning of this chapter in which we considered a curve γ on S^2, we realize that we have now all the ingredients to define S^2, $\gamma(t)$ and $\dot{\gamma}(t)$ without relying on the framework provided by the embedding space $\mathbb{R}^3$. However, we did not make any progress in terms of measuring the length of a curve. This in fact would allow us to define the distance $d_{\mathcal{M}}(p, q)$ between two points p, q on a manifold $\mathcal{M}$ as the length of the shortest curve connecting them, similarly to what we have seen in the Euclidean case. The first step in this direction will be the definition of a scalar product between the vectors tangent to $\mathcal{M}$. Such product will induce a norm that can be used to measure the length of a curve on $\mathcal{M}$. The main difference with respect to the Euclidean case though, is that now vectors tangent to $\mathcal{M}$ at two different points $p, q \in \mathcal{M}$ belong to different tangent spaces, namely $T_p\mathcal{M}$ and $T_q\mathcal{M}$. We thus need to define an inner product on each tangent space, i.e, a *Riemannian metric* on $\mathcal{M}$.

DEFINITION 2.7 (Riemannian metric): Let $\mathcal{M}$ be a smooth manifold. If for every point $p \in \mathcal{M}$ an inner product g_p is assigned to $T_p\mathcal{M}$ such that the functions

$$\begin{aligned} g(X, Y) : \mathcal{M} &\to \mathbb{R} \\ p &\mapsto g_p(X_p, Y_p) \end{aligned}$$

are of class C^∞ for all $X, Y \in \mathcal{X}(\mathcal{M})$, then we call g a *Riemannian metric* on $\mathcal{M}$ and the pair (M, g) a *smooth Riemannian manifold.*

One can show that every differentiable manifold may be equipped with a Riemannian metric.

To fix ideas, the inner product g_p is a positive definite, symmetric, bilinear map $\langle \cdot\,, \cdot \rangle_p : T_p\mathcal{M} \times T_p\mathcal{M} \to \mathbb{R}$ for every $p \in \mathcal{M}$. We thus may consider the norm $\| \cdot \|_p$ of a vector $\xi \in T_p\mathcal{M}$ defined as $\|\xi\|_p^2 := \langle \xi\,, \xi \rangle_p$. When this will not cause ambiguities, in the following we will simply write $\|\xi\|$ and $\langle \xi\,, \xi \rangle$, omitting p.

Now let (U, ϕ, x^i) be a chart on $\mathcal{M}$. Then we can think about the metric as a positive definite symmetric 2×2 matrix $g(p)$ for every $p \in U$. By denoting the ij^{th} entry of such matrix with $g_{ij}(p)$, we set

$$g_{ij}(p) = \langle \partial_i\,, \partial_j \rangle_p, \qquad 0 \leq i, j \leq 1.$$

For any given $p \in U$, let v, w be two vectors in $T_p\mathcal{M}$, then

$$\langle v\,, w \rangle = v^T g(p) w = g_{ij}(p) v^i w^j. \tag{2.4}$$

We will refer to $g(p)$ as *the local representation of the Riemannian metric* (or the g_{ij} of the metric) *in the chart* (U, ϕ, x^i). We denote with $g^{ij}(p)$ the ij^{th} entry of $g(p)^{-1}$. It is important to point out that, although thinking about $g(p)$ as matrix is convenient to visualize things, $g(p)$ is a tensor of type $(0, 2)$, which, roughly speaking, means that it is a linear mapping that associates to every pair of tangent vectors belonging to the same tangent space a scalar. For this reason, in the following we may refer to it also as *metric tensor.*

We may now consider the length of curves on a Riemannian manifold $\mathcal{M}$. Let $c : [a, b] \to \mathcal{M}$ be a smooth curve on $\mathcal{M}$, we define its *length* $L(c)$ by

$$L(c) = \int_a^b \|\dot{c}(t)\|_{c(t)} dt = \int_a^b \langle \dot{c}(t)\,, \dot{c}(t) \rangle_{c(t)}^{1/2} dt, \tag{2.5}$$

and its arc-length by

$$s(t) = \int_a^t \|\dot{c}(u)\|_{c(u)} du.$$

If c is *regular*, i.e. $\dot{c}(t) \neq 0$ for every $t \in [a, b]$, the arc-length $s = s(t)$ is strictly monotone increasing, since $s'(t) = \|\dot{c}(t)\| > 0$. Denoting by $t = t(s)$ its inverse, we get the parametrization of a curve by arc-length, $\bar{c}(s) = c(t(s))$, for $0 \leq s \leq L(c)$. Note that in this case we have

$$\|\dot{\bar{c}}(s)\| = \|t'(s)\dot{c}(t(s))\| = \left\| \frac{\dot{c}(t(s)}{s'(t(s))} \right\| = 1.$$

In general, a curve is said to be *normal* (or *parametrized by arc-length*) if $\|\dot{c}(t)\| \equiv 1$, and of *constant speed* (or *parametrized proportionally to arc-length*) if $\|\dot{c}(t)\|$ is constant. In the following, we will always assume a given curve to be regular.

Let now C_{pq} be the set of all curves joining p to q on $\mathcal{M}$. We define

$$d_{\mathcal{M}}(p,q) := \inf\{L(\gamma) : \gamma \in C_{pq}\}. \tag{2.6}$$

Whenever this does not cause ambiguity, we may simply write $d(p,q)$, omitting the index $\mathcal{M}$. One can show that the distance function defined above satisfies the usual axioms, i.e., for all $p, q, r \in \mathcal{M}$ we have

i) $d(p,q) \geq 0$ and $d(p,q) > 0$ if $p \neq q$,

ii) $d(p,q) = d(q,p)$,

iii) $d(p,q) \leq d(p,r) + d(r,q)$.

2.1.3 Affine Connection

When we considered the shortest curve between two points in $\mathbb{R}^3$, we observed that such curve had to be a straight line, i.e. a curve with null curvature everywhere. We would like to make a similar assessment concerning the curves in $C_{p,q}$, for $p, q \in \mathcal{M}$. However, the fact that vectors tangent to a manifold at two different points belong to different spaces is a problem also in this case. To understand why, let us consider a vector field X on $\mathbb{R}^3$ and a vector ξ at $p \in \mathbb{R}^3$. Then, to study how X varies in a neighborhood of p along direction ξ one must compute

$$\lim_{t\to 0} \frac{X(p + t\xi) - X(p)}{t}. \tag{2.7}$$

Since $\mathbb{R}^3$ is a manifold, we have that $X(p + t\xi)$ belongs to $T_{p+t\xi}\mathbb{R}^3$ and $X(p) \in T_p\mathbb{R}^3$. However, for each $p \in \mathbb{R}^3$, $T_p\mathbb{R}^3$ can be canonically identified with $T_0\mathbb{R}^3 \cong \mathbb{R}^3$, so we can freely add and subtract vectors tangent at different points because, with a slight abuse, we can say that the tangent space at every point in $\mathbb{R}^3$ is $\mathbb{R}^3$ itself. Note that this is the same thing we do when we think of $\mathbb{R}^3$ as affine space. In fact, when we considered the quantity $\ddot{\gamma}(t)$ it was clear what we meant and that such quantity was well-defined. However, when we consider a generic Riemannian manifold $\mathcal{M}$, in order to compute the curvature, or the *acceleration* of a given curve γ, we need a way to map, or *connect*, the tangent space at a point to the tangent space at any neighboring point. This leads us to the following

DEFINITION 2.8 (Affine connection): An *affine connection* ∇ on a smooth manifold $\mathcal{M}$ is a mapping

$$\begin{aligned} \nabla : \mathcal{X}(\mathcal{M}) \times \mathcal{X}(\mathcal{M}) &\to \mathcal{X}(\mathcal{M}) \\ (X, Y) &\mapsto \nabla_X Y \end{aligned}$$

which satisfies the following properties:

i) $\nabla_{fX+gY} Z = f\nabla_X Z + g\nabla_Y Z$

ii) $\nabla_X(Y+Z) = \nabla_X Y + \nabla_X Z$

iii) $\nabla_X(fY) = f\nabla_X Y + X(f)Y$

where $X, Y, Z \in \mathcal{X}(\mathcal{M})$ and $f, g \in D$.

In order to better understand this latter concept, let us consider two vector fields $X = X^i\partial_i$ and $Y = Y^j\partial_j$, $X, Y \in \mathcal{X}(\mathcal{M})$, and let us compute $\nabla_X Y$.

$$\begin{aligned}\nabla_X Y &= \nabla_{X^i\partial_i} Y^j\partial_j \\ &= X^i\nabla_{\partial_i}\left(Y^j\partial_j\right) && \text{i)} \\ &= X^iY^j\nabla_{\partial_i}\partial_j + X^i\partial_i(Y^j)\partial_j && \text{ii) and iii)}\end{aligned}$$

Note that $\nabla_{\partial_i}\partial_j$ measures how much ∂_j varies when we move in the direction ∂_i. We can thus express this in terms of the basis $\{\partial_0, \partial_1\}$ and set $\nabla_{\partial_i}\partial_j = \Gamma^k_{ij}\partial_k$. We can conclude that

$$\nabla_X Y = (X(Y^k) + \Gamma^k_{ij}X^iY^j)\partial_k. \tag{2.8}$$

Therefore, the *covariant derivative* $\nabla_X Y$ at $p \in \mathcal{M}$ is a vector belonging to $T_p\mathcal{M}$ whose coefficients in the basis of $T_p\mathcal{M}$ consist in two terms: the first one is the usual derivative of a vector field along another one, the second one keeps into account that we are on a curved domain, and hence measures how basis vectors change when we move in a given direction. We will see that, in the Euclidean case, the (differentiable) functions Γ^k_{ij}, called *Christoffel symbol*, are identically zero, confirming that the affine connection is a generalization of the usual differentiation of vector fields in the Euclidean setting.

It is possible that, so far, the way in which an affine connection *connects* tangent spaces is not so transparent. To make this more clear, we are going to show that an affine connection leads to a bona fide definition of *parallelism* between vectors. First, we introduce the concept of a vector field along a curve, and then we use the affine connection to characterize those vector fields that have null covariant derivative with respect to the tangent vectors of the curves along which they are defined.

DEFINITION 2.9 (Vector field along a curve): A vector field Y along a smooth curve $c : [a, b] \longrightarrow \mathcal{M}$ is a C^∞ mapping that associates to every $t \in [a, b]$ a tangent vector $Y(t) \in T_{c(t)}\mathcal{M}$. Y is differentiable in the sense that, for any $f \in \mathcal{D}(\mathcal{M})$, the function $t \mapsto Y(t)f$ is differentiable on $[a, b]$.

We can then compute the covariant derivative of $Y(t)$ with respect to $\dot{c}(t)$. Such quantity will be denoted with $\nabla_{\dot{c}}Y$, and it is convenient to express it in local coordinates. By choosing a chart (U, ϕ, x^i), we have that

$$\nabla_{\dot{c}}Y = \left(\frac{dY^k}{dt} + \Gamma^k_{ij}\frac{dx^i}{dt}Y^j\right)\partial_k, \tag{2.9}$$

where we set $Y(t) = Y^i\partial_i(c(t))$ and $x^i(t) = x^i(c(t))$.

DEFINITION 2.10 (Parallel vector along a curve): Let $Y(t)$ a vector field along a smooth curve $c : [a,b] \longrightarrow \mathcal{M}$. We say that $Y(t)$ is *parallel* along c if $\nabla_{\dot{c}} Y$ is identically null.

By writing the condition $\nabla_{\dot{c}} Y \equiv 0$ with respect to a chart, one can prove the following

PROPOSITION 2.11 : With c as above, let Y_a be a vector in $T_{c(a)}\mathcal{M}$. Then there exists a unique parallel vector field Y along c, called *the parallel transport of Y_a along c*, such that $Y(a) = Y_a$.

The parallel transport can thus be seen as a map $P_c^{t_0,t_1} : T_{c(t_0)}\mathcal{M} \longrightarrow T_{c(t_1)}\mathcal{M}$, where $t_0, t_1 \in [a,b]$, such that $P_c^{t_0,t_1}(\xi)$ is $Y(t_1)$, where $\xi \in T_{c(t_0)}$ and Y is the parallel transport of ξ along c. This latter interpretation allows us to express the covariant derivative in terms of the parallel transport. Namely, for $X, Y \in \mathcal{X}(\mathcal{M})$ and a smooth curve $c(t)$ on $\mathcal{M}$ with $c(0) = p$ and $\dot{c}(0) = X_p$, we have that

$$(\nabla_X Y)_p = \lim_{t \to 0} \frac{P_c^{t,0}(Y_{c(t)}) - Y_p}{t}. \tag{2.10}$$

By comparing the above equation with Eq. (2.7), we see how the definition of an affine connection on $\mathcal{M}$ allowed us to compare vectors belonging to different tangent spaces and hence to compute their derivatives with respect to other vectors.

Definition 2.8 suggests that, given a manifold $\mathcal{M}$, the choice of the affine connection which endow $\mathcal{M}$ with is not unique. However, an important result states that, for each Riemannian metric on a manifold, there is a unique affine connection, called the *Levi-Civita* connection, which satisfies certain two properties. The first one is the *compatibility with the metric*:

DEFINITION 2.12 (Compatibility with the metric): Let $(\mathcal{M}, g)$ be a Riemannian manifold. An affine connection ∇ on $\mathcal{M}$ is said to be *compatible with the metric g* if, for any $X, Y, Z \in \mathcal{X}(\mathcal{M})$, we have

$$X\langle Y, Z\rangle_g = \langle \nabla_X Y, Z\rangle_g + \langle Y, \nabla_X Z\rangle_g.$$

Roughly speaking, we can say that a connection is compatible with the metric if we can differentiate the product of two vector fields by applying the usual product rule. The second property involves just the connection, and it requires the latter to be symmetric in some sense. A connection satisfying such property is also called *torsion-free*. We do not enter the detail for brevity, and refer to [Sak97, dC92] for a formal definition. The Fundamental Theorem of Riemannian geometry states that for each Riemannian metric, there is a unique connection ∇ which is both torsion-free and compatible with the metric. Such connection is called the *Levi-Civita* connection or the *Riemannian* connection. From now on, ∇ will always indicate such connection.

One can show that the compatibility with metric of a connection implies that $\Gamma_{ij}^k = \Gamma_{ji}^k$. In particular, one can also show that the Christoffel symbols can be express in terms of the local representation of the metric in the following way

$$\Gamma_{ij}^{k} = \frac{1}{2} g^{km} (\partial_i g_{jm} + \partial_j g_{mi} - \partial_m g_{ij}), \tag{2.11}$$

where we remember that g^{km} is the km^{th} entry of the inverse of g, and that $\partial_i = \frac{\partial}{\partial x^i}$.

2.1.4 Differential Operators on Manifolds

We will now introduce the definition of the differential operators that we will use in the sequel. For every one of them, we will present their expression in terms of the Riemannian metric, since it will be used when implementing them in the discrete setting.

DEFINITION 2.13 (Gradient): Let $f \in \mathcal{D}(\mathcal{M})$. The *gradient vector* ∇f of f is the unique vector field in $\mathcal{X}(\mathcal{M})$ satisfying

$$\langle \nabla f, X \rangle = X(f), \qquad X \in \mathcal{X}(\mathcal{M}).$$

By fixing $p \in \mathcal{M}$ and putting $(\nabla f)_p = y^i \frac{\partial f}{\partial x^i}$, by Eq. (2.3) and (2.4), we can re-write the definition of ∇f above in the following way

$$g_{ij} y^i X^j = X^j \frac{\partial f}{\partial x^j},$$

which means that

$$(\nabla f)_p = g^{ij} \frac{\partial f}{\partial x^j} \frac{\partial}{\partial x^i}, \tag{2.12}$$

where we omit the dependence from p to keep a light notation.

DEFINITION 2.14 (Second Covariant Derivative): Let $f \in \mathcal{D}(\mathcal{M})$. The *second covariant derivative* of a real-valued function f is the bilinear operator defined by

$$\nabla^2 f(X, Y) = \langle \nabla_X \nabla_f, Y \rangle = XYf - (\nabla_X Y) f, \qquad X \in \mathcal{X}(\mathcal{M}).$$

Therefore, the second covariant derivative associates to each pair of vectors $X_p, Y_p \in T_p\mathcal{M}$ the covariant derivative of the gradient of f along X_p, and compute its component along Y_p. However, to retrieve its expression in local coordinates it is convenient to use the other expression. Let $X = X^i \frac{\partial}{\partial x^j}$ and $Y = Y^j \frac{\partial}{\partial x^j}$, then, by (2.3), (2.8) and the chain rule, we have

$$\begin{aligned} XYf - (\nabla_X Y)f &= X(Y^j \frac{\partial f}{\partial x^j}) - \left((X(Y^k) + \Gamma_{ij}^k X^i Y^j) \frac{\partial}{\partial x^k} \right) f \\ &= X^i \frac{\partial Y^j}{\partial x^i} \frac{\partial f}{\partial x^j} + X^i Y^j \frac{\partial^2 f}{\partial x^i \partial x^j} - \left(X^i \frac{\partial Y^k}{\partial x^i} + \Gamma_{ij}^k X^i Y^j \right) \frac{\partial f}{\partial x^k}. \end{aligned}$$

By simplifying we thus obtain

$$\nabla^2 f(X, Y) = X^i Y^j \frac{\partial^2 f}{\partial x^i \partial x^j} - \Gamma_{ij}^k X^i Y^j \frac{\partial f}{\partial x^k}.$$

Therefore, when $X = \frac{\partial}{\partial x^i}$ and $Y = \frac{\partial}{\partial x^j}$ we have

$$\nabla^2 f_{ij} = \frac{\partial^2 f}{\partial x^i \partial x^j} - \Gamma^k_{ij} \frac{\partial f}{\partial x^k}. \tag{2.13}$$

DEFINITION 2.15 (Convexity of a function): A function $f \in \mathcal{D}(\mathcal{M})$ is called (strictly) convex if its second covariant derivative is positive semidefinite (definite).

As it will be clear in the following, being able to assess the strictly convexity of a function on given domain will be important for us, which by the definition above means being able to check the positive definetess of the second covariant derivative. In order to do that, it is more convenient to introduce the *Hessian operator*, which is as a linear operator from the tangent space $T_p\mathcal{M}$ of a point $p \in \mathcal{M}$ into itself. More formally, we will consider the operator Hess : $T_p\mathcal{M} \to T_p\mathcal{M}$ defined as

$$\mathrm{Hess} f(X) = \nabla_X \nabla f(p),$$

i.e., the operator that associate to every vector $X \in T_p\mathcal{M}$ the covariant derivative of the gradient of f with respect to X. It follows from the definition that such operator is related to the second covariant derivative by

$$\nabla^2 f(X, Y) = \langle \mathrm{Hess} f(X), Y \rangle.$$

An eigenvalue and an eigenvector of $\nabla^2 f$ at a point $p \in \mathcal{M}$ are a number λ and a vector $X \in T_p\mathcal{M}$ such that for all vectors $Y \in T_p\mathcal{M}$ we have

$$\nabla^2 f(X, Y) = \lambda \langle X, Y \rangle.$$

The above formulas implies that the eigenvalues of $\nabla^2 f$ coincides with the ones of Hess$f$, so to assess its positive-definetess we may check whether the eigenvalues of Hess$f$ are positive. To do that, we need to express Hess$f(X)$ in local coordinates. By (2.8), (2.12) and the definition of Hess$f$ we have that

$$\mathrm{Hess} f(X) = \left(X\big(g^{kh} \frac{\partial f}{\partial x^h}\big) + \Gamma^k_{ij} X^i g^{jh} \frac{\partial f}{\partial x^h} \right) \partial_k,$$

where $X = X^i \frac{\partial}{\partial x^i}$. By observing that $\frac{\partial g_{ij} g^{jk}}{\partial x^h} = \frac{\partial \delta^k_i}{\partial x^h} = 0$ and applying the product rule, we have that

$$\frac{\partial g^{kh}}{\partial x^i} = -g^{km} \partial_i g_{mj} g^{jh},$$

which once substituted in the expression of Hess$f(x)$ above gives

$$\mathrm{Hess} f(X) = \left(-g^{km} \partial_i g_{mj} g^{jh} \frac{\partial f}{\partial x^h} + g^{kh} \frac{\partial^2 f}{\partial x^h \partial x^i} + \Gamma^k_{ij} g^{jh} \frac{\partial f}{\partial x^h} \right) X^i \partial_k. \tag{2.14}$$

In Section 3.2 we will see how the above formula will permit us to efficiently compute Hess$f(X)$ in the discrete setting.

DEFINITION 2.16 (Laplacian): Let $f \in \mathcal{D}(\mathcal{M})$. We define the *Laplacian* of f as the *trace* of the second covariant derivative, namely

$$\Delta f = tr \nabla^2 f = g^{ij} \nabla^2 f_{ij},$$

where we do not justify the last equality since we would need the definition of tensor. However, this implies that, in local coordinates, the Laplacian may be written as

$$\Delta f = g^{ij} \frac{\partial^2 f}{\partial x^i \partial x^j} - g^{ij} \Gamma^k_{ij} \frac{\partial f}{\partial x^k}. \tag{2.15}$$

2.1.5 Extrinsic setting and choice of the Riemannian metric

So far, we have been able to introduce all the basic quantities on $\mathcal{M}$ mentioned at the beginning of this chapter: tangent vectors, a metric, and a way of differentiating vectors with respect to other vectors, which, as we will see in the next section, will allow us to define the *curvature* of a curve on $\mathcal{M}$ in an intrinsic manner. Moreover, all these quantities have been introduced by relying exclusively on the charts that the cover $\mathcal{M}$, so all these quantities are *intrinsic*. Even if our interest will remain focused exclusively on this kind of properties, in order to have a geometrical interpretation, it is often practical to consider $\mathcal{M}$ as *embedded* surface in $\mathbb{R}^3$. Note that, when we will move to the discrete setting, we will always consider this kind of objects, so it is appropriate to formally introduce them.

DEFINITION 2.17 (Embedded surface): Let S be a subset of $\mathbb{R}^3$. We say that S is an *embedded surface* if for every $p \in S$ there exists a chart (U, ϕ) for $\mathbb{R}^3$ such that $p \in U$ and $\phi(U \cap S) = V \times \{0\}$, where $V \subset \mathbb{R}^2$.

Note that, in the above definition, we are considering $\mathbb{R}^3$ as a differentiable manifold, so ϕ is the identity map. It is convenient to introduce the following notations. Let $\{U_\alpha, \phi_\alpha\}_{\alpha \in A}$ be a covering atlas for $\mathbb{R}^3$. By the above definition, we can say that, if S is an embedded surface, then $\phi_\alpha(S \cap U_\alpha)$ can be seen as the intersection of a plane with $\phi_\alpha(U_\alpha)$ for every $\alpha \in A$, so it is possible to endow S with the differentiable structure $\{S \cap U_\alpha, \phi_\alpha\big|_{S \cap U_\alpha}\}_{\alpha \in A}$. For practical reasons, if $V_\alpha := \phi_\alpha(S \cap U_\alpha)$, it is convenient to consider the mapping $x_\alpha : V_\alpha \to S \cap U_\alpha$. In the following we will call x_α a *parametrization* in a neighborhood of $p \in V_\alpha$, and we will denote the coordinates in its domain to be (u, v). In the following, S will always denote an embedded surface of R^3. One can show that the tangent space T_pS at a point $p \in S$ is a 2-dimensional vector subspace of $T_p\mathbb{R}^3$(i.e. a plane), having basis $\{x_u, x_v\}$, where $x : V \to S$ is a parametrization of S in a neighborhood of p [Lee19, Proposition 5.8]. It is reasonable to ask ourselves which metric we should endow S with. The most natural choice is to use the *induced metric*. In fact, since two tangent vectors $v_0, v_1 \in T_pS$ are in particular vectors of $T_p\mathbb{R}^3$, a natural choice is to use the product of $T_p\mathbb{R}^3 \cong \mathbb{R}^3$ as Riemannian metric, i.e. the usual Euclidean inner product $\langle \cdot, \cdot \rangle$.

The fact that every point $p \in S$ admits a tangent plane T_pS suggests that one can consider the normal vector to such plane at p. By defining

$$N(q) := \frac{x_u \wedge x_v}{\|x_u \wedge x_v\|}(q), \qquad q \in x(V),$$

we thus obtain a differentiable field of unit normal vectors on $x(V)$. We will say that, if S admits a differentiable field of unit normal vector on the whole surface, then S is *orientable*, and the choice of such field N is called an orientation of S. From now on, we will assume that S is orientable. We can thus define a map $N : S \longrightarrow \mathbb{R}^3$ that takes its values in the unit sphere

$$S^2 = \{x \in \mathbb{R}^3 : \|x\| = 1\}.$$

We will call N the *Gauss Map* of S.

DEFINITION 2.18 (Orthogonal parametrization): Let $x : V \longrightarrow S$ a parametrization of a neighborhood of $p \in S$. Then we define the following quantities

$$E := \langle x_u , x_u \rangle, \quad F := \langle x_u , x_v \rangle, \quad G := \langle x_v , x_v \rangle.$$

Moreover, we will say that x is an *orthogonal* parametrization if $F = 0$.

2.2 Geodesics and Exponential Map

2.2.1 Geodesics

As done before, we start this section by considering the examples made at the beginning of this chapter and commenting on the progress made so far. Let us then consider the covariant derivative $\nabla_{\dot{c}} Y$ of a vector field Y along c, introduced in the previous section. Then, the choice $Y(t) = \dot{c}(t)$ suggests that $\nabla_{\dot{c}} \dot{c}$ must somehow be related to the curvature or the *acceleration* of the curve $c(t)$. To have a geometric interpretation of this vector, it is convenient to switch to the extrinsic setting.

Let us start by formalizing the discussion made about the curve $\gamma(t) : [a, b] \longrightarrow S^2$ introduced at the beginning of this chapter. In order to put ourselves in a more general context, we now assume $\gamma : [a, b] \longrightarrow S$ to be a curve on a generic surface $S \subset \mathbb{R}^3$. Suppose that γ is parametrized by arc-length, so that $\dot{\gamma}(t)$ has unit length for every $t \in [a, b]$. Let $N : [a, b] \longrightarrow S^2$ be the restriction of the Gauss map to $\gamma(t)$. For every $t \in [a, b]$, let us denote with $n(t) := \dot{\gamma}(t) \wedge N(t)$ the vector normal to γ at t. Then $\ddot{\gamma}(t)$ can be decomposed into two scalar components(see Figure 2.3)

$$\kappa_g(t) :=< n(t), \ddot{\gamma}(t) >$$
$$\kappa_N(t) :=< N(t), \ddot{\gamma}(t) >,$$

so that

$$\ddot{\gamma}(t) = \kappa_g(t)n(t) + \kappa_N(t)N(t).$$

Note that, since $n(t)$ is orthogonal to $N(t)$, $n(t)$ belongs to $T_{\gamma()t}S$. Therefore, $k_g(t)$, called *geodesic curvature*, measures how much γ is bending *on* S, which is exactly the quantity we were interested in (i.e. the green vector in Figure 2.1). Then one can define the covariant derivative of $\dot{\gamma}(t)$ at t, which, when working in the extrinsic setting, is usually denoted with $\frac{D\dot{\gamma}}{dt}$, as the projection of $\ddot{\gamma}(t)$ into the plane tangent to S at $\gamma(t)$, namely

$$\frac{D\dot{\gamma}}{dt} = \kappa_g(t)n(t).$$

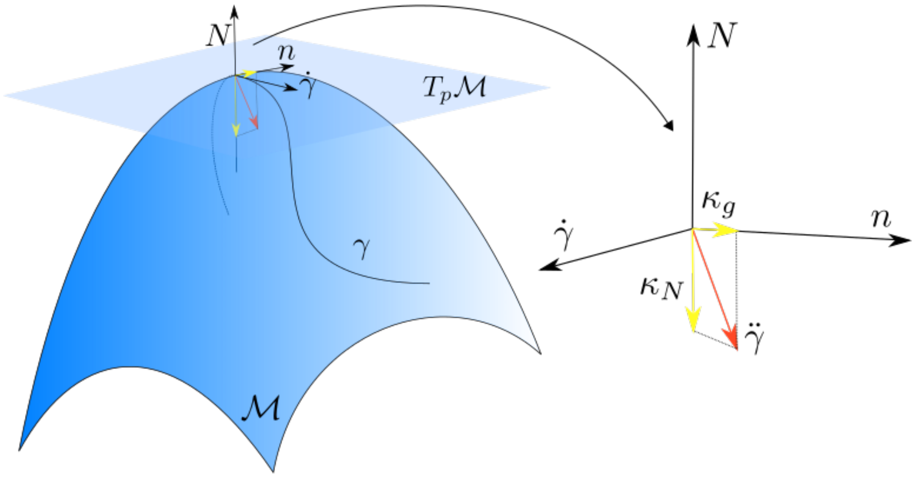


Figure 2.3: The curvature $\ddot{\gamma}$ can be decomposed in two components: κ_g, which measures how much γ is bending *on* $\mathcal{M}$, and κ_N, which measures how much γ is bending *with* $\mathcal{M}$.

The geometric interpretation of $\nabla_{\dot{c}}\dot{c}$ made above suggests that one could introduce a notion of "straightness" of curves on a manifold. In fact, since in our setting $\nabla_{\dot{c}}\dot{c}$ is the only definition of curvature we have for a curve, it seems reasonable to have a definition for the curves that has null curvature everywhere, just as straight lines in the Euclidean setting. These curves are called *geodesics*.

DEFINITION 2.19 : Let γ be a C^2 curve on a Riemannian manifold $\mathcal{M}$. If

$$\nabla_{\dot{\gamma}}\dot{\gamma}(t) \equiv 0, \tag{2.16}$$

then γ is called a *geodesic*.

If $\gamma : [0,1] \to \mathcal{M}$, then Eq. (2.16) implies that $\dot{\gamma}(t) = P_{\gamma}^{0,t}(\dot{\gamma}(0))$, which means that the vector tangent to γ at t is just the parallel transport of $\dot{\gamma}(0)$, i.e. γ proceeds straight. A curve satisfying such property is also called *auto-parallel*. With respect to chart (U, ϕ, x^i), we have that Eq. (2.16) turns into

$$\frac{d^2x^i}{dt} + \Gamma_{ij}^k \frac{dx^i}{dt}\frac{dx^j}{dt} = 0, \qquad i = 0, 1, \tag{2.17}$$

where we set $x^i(t) = x^i(\gamma(t))$. Confronting the above equation with Eq.(2.9), one can see that we are imposing the coefficients of $\nabla_{\dot{\gamma}}\dot{\gamma}$ in the basis $\{\partial_0, \partial_1\}$ to be zero. Note that, in the Euclidean setting, the g_{ij} of the metric is just the identity 2 × 2 matrix everywhere; which in turns implies, by (2.11), that the Christoffel symbols are zero everywhere. Therefore, (2.17) simply tells us that geodesics in this setting are curve with null second derivatives, i.e. straight lines.

One can show that, if γ is a geodesic, then the length of the tangent vector $\dot{\gamma}$ is constant, i.e. γ is parametrized proportionally to arc-length. Moreover, by using the theory of second order ODE, one can prove the following

THEOREM 2.20 (Local existence and uniqueness of a geodesic): Let $\mathcal{M}$ be a Riemannian manifold, $p \in \mathcal{M}$, $v \in T_p\mathcal{M}$. Then there exists $\varepsilon > 0$ and precisely on geodesic

$$\gamma : [0, \varepsilon] \longrightarrow \mathcal{M}$$

such that $\gamma(0) = p$, $\dot{\gamma}(0) = v$. In addition, γ depends smoothly on p and v.

We thus see that, locally, geodesics share another property with straight lines, which is the uniqueness once a starting point and a direction is fixed.

2.2.2 Exponential and Logarithmic Map

Note that if $\gamma(t)$ is a solution of Eq. (2.17), so is $\gamma(\lambda t)$ for any constant $\lambda \in \mathbb{R}$. Denoting the geodesic of Theorem 2.20 satisfying $\gamma(0) = p$, $\dot{\gamma}(0) = v$ by γ_v, we have that

$$\gamma_v(t) = \gamma_{\lambda v}\left(\frac{t}{\lambda}\right), \qquad \lambda > 0, t \in [0, \varepsilon].$$

In particular, $\gamma_{\lambda v}$ is defined on $[0, \frac{\varepsilon}{\lambda}]$. Since by Theorem 2.20 γ_v depends smoothly on v, there exists $\varepsilon_0 > 0$ such that for $\|v\| = 1$ γ_v is defined at least on $[0, \varepsilon_0]$. Therefore, for any $w \in T_p\mathcal{M}$ with $\|w\| \leq \varepsilon_0$, γ_w is defined at least on $[0, 1]$.

DEFINITION 2.21 (Exponential map): Let $\mathcal{M}$ be a Riemannian manifold, $p \in \mathcal{M}$, and $V_p := \{v \in T_p\mathcal{M} : \gamma_v \text{ is defined in } [0, 1]\}$. The mapping

$$\begin{aligned} \exp_p : V_p &\longrightarrow \mathcal{M} \\ v &\longmapsto \gamma_v(1) \end{aligned}$$

is called the *exponential map* of $\mathcal{M}$ at p.

From a geometrical point of view, $\exp_p(v)$ associates to every $v \in T_p\mathcal{M}$ the point obtained by travelling along $\gamma_v(t)$ for an amount equal to $\|v\|_p$. To fix ideas, the straight line segment $r(t) = p_0 + t(p_1 - p_0)$ introduced at the beginning of this chapter do the same thing in the Euclidean setting. In fact, $r(t)$ is a map that associates to each t a point q obtained by moving from p_0 in the direction $w := (p_1 - p_0)$ for an amount $t \cdot \|p_1 - p_0\|$, where the norm we are considering is the Euclidean one. One can show [dC92, Proposition 2.9] that the exponential map $\exp_p$ maps a neighborhood V of $0 \in T_p\mathcal{M}$ diffeomorphically onto a neighborhood of $p \in \mathcal{M}$. We can now give the following definition:

DEFINITION 2.22 (Normal ball): Let $\mathcal{M}$ be a Riemannian manifold, $p \in \mathcal{M}$. Let V be the neighborhood of $0 \in T_p\mathcal{M}$ within which $\exp_p$ is a diffeomorphism. Then $\exp_p V = U$ is called a *normal neighborhood* of p. If $\mathcal{B}_\varepsilon(0)$ is a ball centered at $0 \in T_p\mathcal{M}$ with radius $\varepsilon > 0$, such that $\overline{\mathcal{B}_\varepsilon}(0) \subset V$, then $\exp_p \mathcal{B}_\varepsilon(0) = B_\varepsilon(p)$ is a *normal ball* centered at p with radius ε.

Let now e_0, e_1 be a basis of $T_p\mathcal{M}$, which is orthornormal with respect to the inner product $\langle \cdot, \cdot \rangle_p$ on $T_p\mathcal{M}$. Expressing each vector $v \in T_p\mathcal{M}$ in terms of this basis we obtain a map

$$\begin{aligned} \Phi : T_p\mathcal{M} &\to \mathbb{R}^2 \\ v = v^0 e_0 + v^1 e_1 &\mapsto (v^0, v^1) \end{aligned}$$

For the next definition, it is convenient to identify $T_p\mathcal{M}$ with $\mathbb{R}^2$ via Φ, and the reason will be clear soon enough. By what said above, there exists a neighborhood U of p which is mapped by $\exp_p^{-1}$ diffeomorphically onto a neighborhood of $0 \in T_p\mathcal{M}$.

DEFINITION 2.23 (Normal coordinates): The local coordinates defined by the chart $(U, \exp_p^{-1})$ are called (Riemannian) *normal coordinates* with center p.

Note that, by the identification $T_p\mathcal{M} \cong R^2$ via Φ, $(U, \exp_p^{-1})$ is a proper chart in the sense of Definition 2.1. The mapping $\exp_p^{-1}$ is also called *logarithmic map* at p, and it will be denoted with $\log_p$. A geometric interpretation of the logarithmic map can be as follows. With the notations of above, $v := \log_p(q)$ is a vector in $T_p\mathcal{M}$ such that the uniquely defined geodesic γ_v such that $\gamma_v(0) = p$ and $\dot{\gamma}_v(0) = v$ satisfies $\gamma_v(1) = q$, i.e. $\exp_p(v) = q$, for every $q \in U$, as in Figure 2.4.

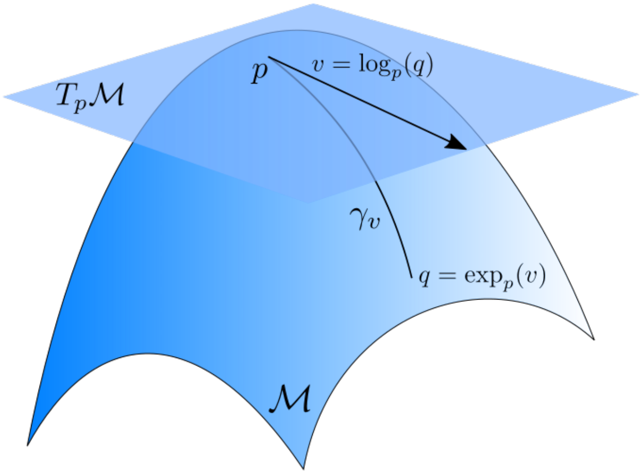


Figure 2.4: The exponential map at p associates to a vector $v \in T_p\mathcal{M}$ the point $q = \gamma_v(1)$, while the logarithmic map associates to a point q in a neighborhood of p a vector $v \in T_p\mathcal{M}$ which is the tangent vector at p of the geodesic joining p with q.

Similarly to what said above about the exponential map, one can see the logarithmic map as the counterpart of the difference between two points p_0, p_1 in the affine space $\mathbb{R}^3$. In fact, $w := (p_1 - p_0)$ is a vector such that $p_0 + w = p_1$.

We state now an important result known as *Gauss lemma.*

LEMMA 2.24 (Gauss): Let $p \in \mathcal{M}$, $v \in T_p\mathcal{M}$, $\gamma(t) := \exp_p(tv)$ the geodesic with $\gamma(0) = p$, $\dot{\gamma}(0) = v$, $t \in [0, 1]$. If v is contained in the domain of definition of $\exp_p$, then for any $w \in T_p\mathcal{M}$

$$\langle (dexp_p)_v v \, , (dexp_p)_v w \rangle = \langle v \, , w \rangle,$$

where $(dexp_p)_v$ denotes the differential of $\exp_p$ at v, and it is applied to the vectors $v, w \in T_v T_p\mathcal{M} \cong T_p\mathcal{M}$ considered as vectors tangent to $T_p\mathcal{M}$ at the point v.

The above lemma implies that $\exp_p$ is a radial isometry in the sense that the length of the radial component of any vector tangent to $T_p\mathcal{M}$ is preserved. Moreover, we have that the image under the exponential map of every ball centered at $0 \in T_pM$, which thus intersect orthogonally the radius, is orthogonal to every radial geodesic.

We want now to relate the definition of geodesics to the curves in C_{pq} that realize the distance between $p, q \in \mathcal{M}$. In other words, since we have seen that geodesics are curve with zero curvature everywhere, we ask ourselves whether these curves are the shortest one among all the curves connecting two arbitrary points on $\mathcal{M}$. For this purpose, it is convenient to introduce the following terminology.

PROPOSITION 2.25 : Let $p \in \mathcal{M}$, U a normal neighborhood of p. Then for every $r > 0$ such that the ball B centered at p with radius r is normal, i.e. $B \subset U$, and for any $q \in \partial B$, there exists precisely one minimizing geodesic from p to q. If $v = log_p(q)$, then such geodesic is $\gamma(t) := exp_p(tv)$, $t \in [0, 1]$. Here, "minimizing" means that such curve is the shortest among all the curves in C_{pq}, i.e. $d(p, q) = L(\gamma)$.

The above proposition tells us that every point $p \in \mathcal{M}$ admits a neighborhood U in which radial geodesics from p behave like straight lines in the following sense. For every $q \in U$, there exists a unique geodesic (i.e. a curve with zero curvature) $\gamma : [0, 1] \longrightarrow \mathcal{M}$ such that $\gamma(0) = p$, $\gamma(1) = q$ and such that $L(\gamma) = d(p, q)$. In the following, we will denote such geodesic with γ_{pq}. From what said above, it is clear that if a curve c realizes the distance between p and q, then c is for sure a geodesic, but Proposition 2.25 suggests that not all geodesics emanated from $p \in \mathcal{M}$ are minimizing. In fact, consider a point p on the unit sphere S^2, and let v be a vector in its tangent space. Then the geodesic $\gamma(t) := \exp_p(tv)$ is a great circle arc on the sphere. However, once $\gamma(t)$ cross the antipodal point q of p, then it is clear that γ is no longer minimizing, since the points "beyond" q can be reached with a shorter geodesic $\sigma := \exp_p(t(v + \pi))$, i.e. by going in the opposite direction with respect to v. In Section 2.4.1 we will address the problem of determining the set of points beyond which a geodesic emanated from a given point is no longer minimizing. For now, we are interested in stating the conditions under which a minimizing geodesic can always be found for every choice of $p, q \in \mathcal{M}$.

DEFINITION 2.26 (Geodesically completeness): A Riemannian manifold $\mathcal{M}$ is *geodesically complete* if, for all $p \in \mathcal{M}$, the exponential map $\exp_p$ is defined for all $v \in T_p\mathcal{M}$, i.e. if any radial geodesic $\gamma(t)$ from p is defined for all values of $t \in \mathbb{R}$.

An example of a manifold which is not geodesically is $\mathbb{R}^3 \setminus \{0\}$, since some geodesics will hit 0 and thus stop to be defined for some t. We can now state the Theorem of Hopf-Rinow.

THEOREM 2.27 (Hopf-Rinow): Let $\mathcal{M}$ be a Riemannian manifold. The following statements are equivalent:

i) $\mathcal{M}$ is a complete metric space where the distance function d from p to q in $\mathcal{M}$ is defined as the minimum lengths of all curves in C_{pq}.

ii) The closed and bounded sets of $\mathcal{M}$ are compact.

iii) There exists $p \in \mathcal{M}$ for which $\exp_p$ is defined for every $v \in T_p\mathcal{M}$.

iv) $\mathcal{M}$ is geodesically complete.

Furthermore, any of the above statements imply

v) Any two points $p, q \in \mathcal{M}$ can be joined by a minimizing geodesic, i.e. by a geodesic of length $d(p, q)$.

For our purposes, the implication i) $\implies$ v) is the most important. Not that i) is a very natural hypothesis which holds in particular whenever $\mathcal{M}$ is compact. In the following, if not stated otherwise, we always denote with $\mathcal{M}$ a complete Riemannian manifold.

At this point, we gave all the definitions needed to address the problems raised at beginning of this chapter, which constitute a building block upon which the theory of the next sections will be presented.

2.3 Curvature and Jacobi Fields

When considering a surface embedded in $\mathbb{R}^3$, we have an intuitive notion of whether a surface is flat or curved. For example, if the implicit form of such surface is linear, we know that it is flat, while when we think about a sphere or a torus, we know that they are curved. This intuitive idea of curvature is expressed by the Gaussian curvature, which will be one of the main concept of this chapter. Note that, when moving to higher dimensions, there are several notions of curvatures, that capture aspects of non-linearity of the manifold with varying details. However, in our setting we only need to define the Gaussian curvature, which, intuitively measures how much a Riemannian manifold deviates from being Euclidean, i.e. flat. Let us consider a point p on a Riemannian manifold $\mathcal{M}$. We know from the Gauss Lemma that the $\exp_p$ is a radial isometry, i.e. it preserves lengths along radial geodesics. It is reasonable to ask ourselves what happens along non-radial directions or, in other words, if there is way of quantifying how much radial geodesics "pull away" from each other as they move away from p. To investigate this, we will start by introducing the *Riemannian curvature*, which will lead to the definition of Gaussian curvature.

We will then introduce special vector fields (called *Jacobi fields*) along radial geodesics that, roughly speaking, measure how much two nearby geodesic

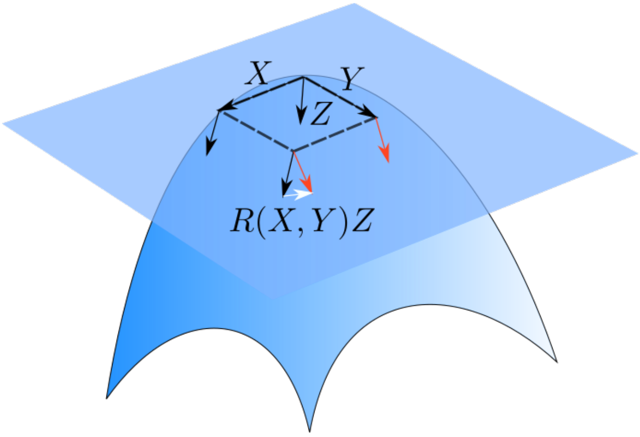


Figure 2.5: The Riemannian curvature describe the difference in parallel transporting a vector Z around an infinitesimal parallelogram spanned by two vectors X and Y.

spread apart as they move away from their common starting point. This definition will allow to relate the exponential map to the curvature.

We will conclude by stating the Gauss-Bonnet theorem, an important result that will be helpful to have a geometric interpretation of some definitions in Section 3.1 and to understand the theoretical limitations of some algorithms in Chapter 6.

2.3.1 Curvature

DEFINITION 2.28 (Riemannian curvature): The *Riemannian curvature* R of a Riemannian manifold $\mathcal{M}$ is a correspondence that associates to every $X, Y \in \mathcal{X}(\mathcal{M})$ a mapping $R(X, Y) : \mathcal{X}(\mathcal{M}) \longrightarrow \mathcal{X}(\mathcal{M})$ given by

$$R(X,Y)Z = \nabla_X \nabla_Y Z - \nabla_Y \nabla_X Z - \nabla_{[X,Y]} Z, \qquad Z \in \mathcal{X}, \tag{2.18}$$

where $[X, Y]$ is the *Lie bracket* [dC92]. Since in this thesis we will always consider $R(X, Y)Z$ in cases where $[X, Y] = 0$, we can forget about the last term of the above equation and consider the Riemannian curvature defined as

$$R(X,Y) = \nabla_X \nabla_Y - \nabla_Y \nabla_X .$$

Keeping in mind the definition of covariant derivative through Eq.(2.10), we see that, when computing $\nabla_X \nabla_Y Z$, we first move Z by infinitesimal parallel transport in the direction Y and then in the direction X, while when forming $\nabla_Y \nabla_X Z$ the order is reversed. The $R(X, Y)Z$ expresses the difference between parallel transporting Z along an infinitesimal parallelogram with sides X and Y, i.e. the dependence of parallel transport on the chosen path. Therefore, we clearly see that, if $\mathcal{M} = \mathbb{R}^2$, then $R(X, Y)Z \equiv 0$ for every $X, Y, Z \in \mathcal{X}(\mathcal{M})$, since parallel vector fields in $\mathbb{R}^2$ are constant fields.

DEFINITION 2.29 (Gaussian curvature): Given a point p on Riemannian 2-manifold $\mathcal{M}$, the *Gaussian curvature* K at p is given by

$$K = \frac{\langle R(\partial_0, \partial_1)\partial_1 \, , \partial_0 \rangle_p}{G}, \tag{2.19}$$

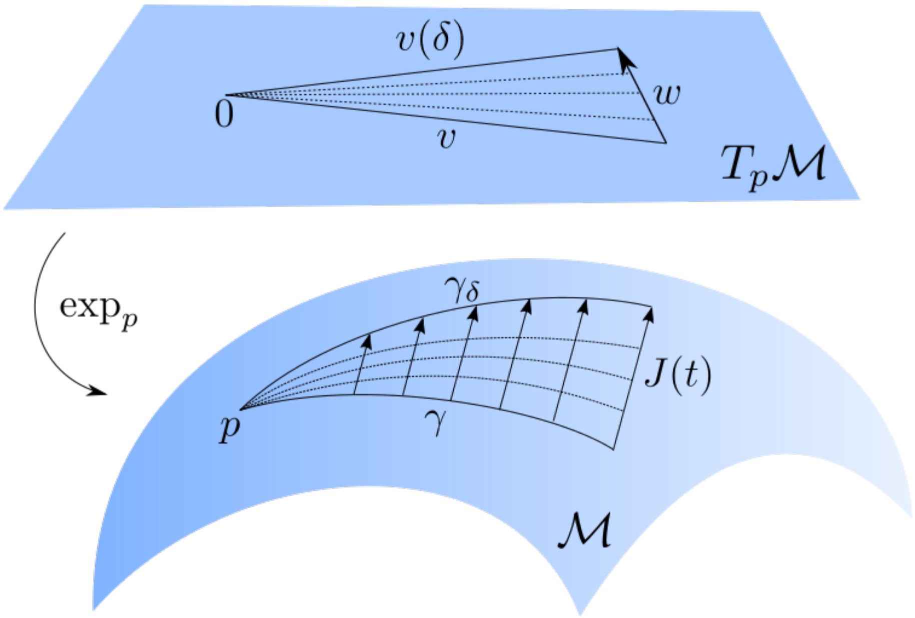


Figure 2.6: The mapping $\alpha(t,s)$ maps the rays $tv(s)$ in $T_p\mathcal{M}$ (dotted straight lines) into geodesics γ_s (dotted black curves) through the exponential map. For illustrative purposes, we consider only the case $s \in [0,\delta]$.

where $g(p)$ is the $g_{ij}$of the metric at p and G denotes its determinant.

A geometric interpretation of the above definition could be as follows. We first observe that $\mid g(p) \mid = \|\partial_0\|^2\|\partial_1\|^2 - \langle\partial_0\,,\partial_1\rangle$ is the squared area of the parallelogram spanned by ∂_0 and ∂_1. From what said before, the numerator in Eq.(2.19) is the component along ∂_0 of the difference we get when parallel transporting ∂_1 along such parallelogram.

2.3.2 Jacobi Fields

We want now to examine the relationship between curvature and exponential map. Let $p \in \mathcal{M}$, and $v, w \in T_p\mathcal{M}$, where $w \in T_vT_p\mathcal{M} \cong T_p\mathcal{M}$. Let us consider the curve $v(s)$ in $T_p\mathcal{M}$ defined as $v(s) := v + sw$, $s \in [-\delta,\delta]$. Then $v(s)$ is such that $v(0) = v$, $\dot{v}(0) = w$. Geometrically, one can think of $v(s)$ as a fan of vectors in $T_p\mathcal{M}$ spanned by $v - \delta w$ and $v + \delta w$. We then define a C^∞ map $\alpha : [0,1] \times [-\delta,\delta] \longrightarrow \mathcal{M}$ as

$$\alpha(t,s) := \exp_p(tv(s)) = \exp_p t(v + sw).$$

For each fixed s, the curve $\gamma_s : t \mapsto \alpha(t,s)$ is a radial geodesic from p such that $\dot{\gamma}_s(0) = v(s)$. Roughly speaking, the parameter s in $\alpha(t,s)$ determine the direction along which we "shoot" a geodesic and parameter t determine the range of the shot (see Figure 2.6). Let us put $J(t) := \frac{\partial\alpha}{\partial s}(t,0)$, which is a vector field along $\gamma_v := exp_p(tv)$, $0 \leq t \leq 1$. Note that $J(t) = (d\exp_p)_{tv}tw$ and hence, in particular, $J(1) = (d\exp_p)_v w$. We will see that $\mid J(t) \mid$ gives an estimation on how much geodesic spread apart with respect to the rays $tv(s)$ in $T_p\mathcal{M}$. By using the notation $\dot{J} = \nabla_{\dot{\gamma}} J$ and $\ddot{J} = \nabla_{\dot{\gamma}}\nabla_{\dot{\gamma}} J$, one can show that $J(t)$ satisfies

$$\ddot{J} + R(\dot{\gamma}(t), J(t))\dot{\gamma}(t) = 0, \tag{2.20}$$

which is called the *Jacobi equation.*

DEFINITION 2.30 (Jacobi field): Let $\gamma : [0, \ell] \to \mathcal{M}$ be a geodesic in $\mathcal{M}$. A vector field along γ is said to be a *Jacobi field* if it satisfies the Jacobi equation (2.20).

We are now ready to see how curvature and exponential map are related.

PROPOSITION 2.31 : If $\gamma : [0, \ell] \to \mathcal{M}$ is a geodesic parametrized by arc-length, (i.e., $\|\dot{\gamma}(t)\| \equiv 1$), and K is Gaussian curvature at p, then

$$\mid J(t) \mid = t - \frac{1}{6}Kt^3 + o(t^3), \tag{2.21}$$

To give a geometric interpretation to the above formula, let us consider again

$$\alpha(t, s) = exp_p tv(s), \qquad t \in [0, \varepsilon], s \in [-\delta, \delta],$$

where ε is small enough to guarantee that $\exp_p tv(s)$ is defined, and $v(s)$ is a curve in $T_p\mathcal{M}$ such that $\|v(s)\| = 1$, $v'(0) = w$. Let us consider the rays $t \mapsto tv(s)$, $t \in [0, \delta]$, that start from $0 \in T_p\mathcal{M}$ and deviate from the ray $t \mapsto tv(0)$ with velocity

$$\left\| \left(\frac{\partial}{\partial s} tv(s) \right)(0) \right\| = \|tw\| = t.$$

On the other hand, Eq. (2.21) tells us that geodesics $t \mapsto \exp_p(tv(s))$ deviate from $\gamma(t) = exp_p tv(0)$ with a velocity that differs from t by $-\frac{1}{6}Kt^3$. This tells us that, locally, the geodesics spread apart less than the rays in $T_p\mathcal{M}$ in the case $K > 0$, while they deviate from each other faster if $K < 0$. To fix ideas, an example of the former case are geodesics starting from the north pole of the sphere. Eventually, they will end up converging on the same point (the south pole). On the other hand, geodesics starting from the same point on the *Poincarré disk*, which are Euclidean circle arcs, pull away from each other faster than straight lines and never meet. We conclude this section with the definition of *conjugate point*, which will be needed when we will talk about the cut locus in Section 2.4.1.

DEFINITION 2.32 (Conjugate point): Let $\gamma : [0, \ell] \to \mathcal{M}$ be a geodesic. The point $\gamma(t_0)$ is said to be *conjugate* to $\gamma(0)$ along γ, $t_0 \in (0, \ell]$, if there exists a Jacobi field J along γ, not identically zero, with $J(0) = J(t_0) = 0$.

It can be shown that, on manifolds with constant positive curvature K, if we consider two nearby geodesics emanated from the same point p (e.g. $\alpha(t, s_0)$ and $\alpha(t, s_0)$, for some $s_0, s_1 \in [-\delta, \delta]$), then these two geodesics initially diverge, but then converge again, until they meet at a point, which is called indeed the first conjugate point of p. Note that this does not happen on manifold with non-positive curvature: if $K = 0$, then straight lines diverge linearly, while if $K < 0$ it can be shown that they diverge exponentially [Sak97]. As a concrete example, one may think again at the case of the sphere: if we shoot two geodesics from the north pole, then eventually these two geodesics will meet each other at the south pole. In fact, every point p on the sphere has as conjugate point its antipodal point. Conjugate points are crucial in the definition of *cut locus*, as we will see in Section 2.4.

2.3.3 Gauss-Bonnet Theorem

In this section, we will state an important result in the differential geometry of surfaces, i.e. the *Gauss-Bonnet* theorem. We will report such theorem in its local version, and refer to [dC76]. We will use such result to understand the theoretical limitations in some algorithms presented in Chapter 6, and to introduce a discrete notion of curvature at the vertices of a discrete surface in Section 3.1. Since one of our purpose is having an intuitive definition of curvature, in this section we will take an extrinsic approach, the intrinsic formulation of the Gauss-Bonnet theorem can be found for example in [Cha06]. This choice will allow us to give an equivalent formulation of (2.19) which may be more intuitive. In the following, we will therefore use the notations introduced at the end of Section 2.1.2, and consider a surface S embedded in $\mathbb{R}^3$.

Let us consider a continuous mapping $\gamma : [0, \ell] \longrightarrow S$ from the closed interval $[0, \ell]$ to S. We will say that γ is a *simple, closed, piece-wise regular, parametrized curve* if the following conditions hold:

i) $\gamma(0) = \gamma(\ell)$

ii) For $t_0, t_1 \in [0, \ell)$, $t_0 \neq t_1$ implies $\gamma(t_1) \neq \gamma(t_2)$

iii) There exists a subdivision

$$0 = t_0 \leq t_1 \leq \ldots \leq t_k \leq t_{k+1} = \ell,$$

of $[0, \ell]$ such that γ is differentiable and regular (i.e. $ga\dot{m}ma(t) \neq 0$) in each $[t_i, t_{i+1}]$, $i = 0, \ldots, k$.

Essentially, we are requesting γ to be a closed curve i), that does not self-intersect ii), which fails to have a well defined tangent line only at a finite number of points iii). The points $\gamma(t_i)$ are called *vertices* of γ and the traces $\gamma([t_i, t_i + 1])$ are called *regular arcs* of gamma, $i = 0, \ldots, k$. It is usual to call the trace of $\gamma[0, \ell]$ a *closed, piece-wise, regular curve.* For our purpose, it is convenient to think about γ as the boundary ∂R of a region $R \subset S$, which we assume to be homeomorphic to a disk. Moreover, we will say that γ is *positively oriented* is for each $\gamma(t)$, belonging to a regular arc, the positive orthogonal basis $\{\dot{\gamma}(t), n(t)\}$ is such that $n(t)$ "points toward" R or, more precisely, for any curve $\beta : I \longrightarrow R$ such that $\beta(0) = \gamma(t)$ and $\dot{\beta}(0) \neq \dot{\gamma}(t)$, we have that $\langle \dot{\beta}(0)\, , n(t) \rangle \geq 0$.

Let now $x : U \longrightarrow S$ be a parametrization of S and let $R \subset x(U)$ be a bounded region of S. If f is a differentiable function on S, then it is easily seen that the integral

$$\iint_{x^{-1}(R)} f(u,v)\sqrt{EG - F^2}dudv$$

does not depend on the parametrization x. This integral has, therefore, a geometrical meaning and is called the *integral of f over the region R*, and it is usually denoted as

$$\iint_R f d\sigma.$$

THEOREM 2.33 (Local Gauss-Bonnet): Let $R \subset x(U)$ be a region of S homeomorphic to a disk, where x is an orthogonal parametrization of S. Let $\gamma : I \longrightarrow S$ be such that $\partial R = \gamma(I)$. Assume that γ is positively oriented, parametrized by arc length s and let $\gamma(s_0), \ldots, \gamma(s_k)$ and $\theta_0, \ldots, \theta_k$ be the vertices and the external angles of γ, respectively. Then

$$\sum_{i=0}^{k} \int_{s_i}^{s_{i+1}} \kappa_g(s)ds + \iint_R K d\sigma + \sum_{i=0}^{k} \theta_i = 2\pi, \tag{2.22}$$

where $\kappa_g(s)$ denoted the geodesic curvature (see Section 2.2).

An interesting case is when the boundary of R is a geodesic polygon, i.e. $\kappa_g(s) \equiv 0$ along every regular arcs of γ. Then (2.22) becomes

$$\iint_R K d\sigma + \sum_{i=0}^{k} \theta_i = 2\pi,$$

by substituting $\theta_i = \pi - \hat{\theta}_i$, where $\hat{\theta}_i$ are the internal angles we have

$$\sum_{i=0}^{k} \hat{\theta}_i = \iint_R K d\sigma + (k-1)\pi. \tag{2.23}$$

Therefore, the sum of the internals angles of a geodesic polygon is fully determined by the total curvature of the region circumscribed by the polygon itself. We note that if $K = 0$ then we obtain the well known formula for the sum of the interior angles of a planar polygon.

We would like now to present a definition of the curvature within this setting equivalent to (2.19), which may be more intuitive. The notations will be the same as of above with one difference: we will assume that the boundary of R has no vertices, i.e. we consider a curve parametrized by arc-length $\gamma : [0,\ell] \longrightarrow x(U)$ such that trace of $\gamma([0,\ell]) = \partial R$. We further suppose that x is a parametrization of a neighborhood of some point $p \in S$. Let us consider a unit vector V_0 tangent to S at $\gamma(0)$, and let us denote with $V(s)$, $s \in [0,\ell]$ its parallel transport along γ. By denoting with $\phi(s)$ the angle formed by $V(s)$ with x_u, and putting $\Delta\phi := \phi(\ell) - \phi(0)$, we have the Gaussian curvature K at p satisfies

$$K = \lim_{R \to p} \frac{\Delta\phi}{A(R)},$$

where $A(R)$ denotes the area of the region R, and the limit is taken through a sequence of regions B_n that converges to p.

2.4 Cut Locus and Convexity

In Section 2.2 we stated the conditions under which the minimizing geodesic between two points $p, q \in \mathcal{M}$ always exists. However, we did not talk about their uniqueness outside a normal ball. Differently from what happens in R^n, where every straight line realizes the distance between any of two points on it, on a Riemannian manifold there could be more than one minimizing geodesics joining two points. Moreover, we have seen that if a curve realizes the distance between two points, then such curve must be a geodesic, but the converse is no longer true. For these reasons, in Riemannian geometry, the concept of convexity is more complicated, since, for example, even a normal ball (i.e. a ball diffeomorphic to a Euclidean one) may not be convex.

In this section we will investigate more in details how geodesics behave globally. We will start by introducing the concept of *cut locus* of a point $p \in \mathcal{M}$, which is the closure of the set of points that can be joined to p with more than one minimizing geodesic. Since Poincaré introduced the concept of cut locus [Poi05], this subject was studied by different researchers under different perspectives and in different times. Initially, the cut locus has been studied mainly from a geometrical point of view, while in the last decades important results have been stated about the relationship of the cut locus at a point and the behavior of the geodesic distance function sourced at such point. Since for the purpose of this thesis we are more interested in this latter type of results, we will not give a comprehensive analysis of this subject, and refer the reader to [Sak97, GHL04, CE75] for further details. We will conclude by giving the definition of *strongly convex ball*, which will play an important role in the following.

2.4.1 Cut Locus

Let $\mathcal{M}$ be a complete Riemannian manifold and let $p \in \mathcal{M}$. Let us consider the normal geodesic $\gamma_v : [0, +\infty] \longrightarrow \mathcal{M}$ emanating from p such that $\dot{\gamma}_v(0) = v$, $v \in T_p\mathcal{M}$. We define the following quantity

$$t_v := \sup\{t > 0 : \gamma_v(t)\big|_{t\in[0,t]} \text{ is minimizing, namely } d(p, \gamma_v(t)) = t\}.$$

Of course, we have $0 < t_v \leq +\infty$ and, if $t(v) < +\infty$, then it corresponds to the last value such that $\gamma_v(t)\big|_{t\in[0,t_v]}$ realizes the distance between each of its point. In this case, we call $\gamma_v(t_v)$ a *cut point* of p along γ. We will start by giving a fundamental property of t_v.

PROPOSITION 2.34 : Suppose that $\gamma_v(t_v)$ is a cut point of p along γ_v. Then at least one of the following conditions a) and b) holds:

a) $\gamma_v(t_v)$ is the first conjugate point of p along γ_v,

b) There exists a vector $u \in T_p\mathcal{M}$, $u \neq v$ such that $\gamma_u(t_v) = \gamma_v(t_v)$.

In other words, the above Proposition tells us that the first point beyond which a geodesic γ from p stops of being minimizing is either the first conjugate point of p along γ, or it is a point that can joined to p with another minimizing geodesic σ different from γ.

DEFINITION 2.35 (Cut locus): The set of all cut points of p along every geodesic emanated from p, is called the *cut locus* $C(p)$ of p. More formally

$$C(p) := \exp_p \tilde{C}(p),$$

where

$$\tilde{C}(p) := \{t_v v : v \in T_p\mathcal{M}, t_v < +\infty\}$$

is called the *tangent cut locus* of p.

In the following, we will denote with $\mathcal{U}_p$ the set $\{v \in T_p\mathcal{M} : t_v > 1\}$, i.e. $\mathcal{U}_p$ is the set of vectors v in $T_p\mathcal{M}$ such that the geodesic $t \mapsto \gamma_v(t) = \exp_p(tv)$ is minimizing in the interval $[0, 1]$.

PROPOSITION 2.36 : With the notation introduced above, we have that $C(p) \cup \mathcal{U}_p = \mathcal{M}$ and $C(p) \cap \mathcal{U}_p = \emptyset$. Moreover, the exponential map is a diffeomorphism between $\mathcal{U}_p$ and the open set $\mathcal{M} \setminus C(p)$, consequently $C(p)$ is closed.

One can show that if p is a cut point of q along γ_{pq} then q is a cut point of p along γ_{qp}. In particular, $q \in C(p)$ if an only if $p \in C(q)$. As an example, one can consider $p \in S^2$. Then the cut locus of p consists of its antipodal point q and, in this case, such point is both conjugate along all the geodesics emanating from p and such that all geodesics joining p and q have the same length. However, this simple structure of the cut locus is quite unusual. Actually, it has been proved that a compact Riemannian 2-manifold for which the cut locus of every point consists in just one point must be isometric to S^2. In fact, even for the 2-dimensional, the determination of the cut locus of a point on a Riemannian manifold is far from being trivial. We are going now to present the results that will be exploited in Chapter 5. In the following, we will denote with d_p the *distance function* (or *distance field*) that associates to every $q \in \mathcal{M}$ its geodesic distance from p, i.e.

$$\begin{aligned} d_p : \mathcal{M} &\to \mathbb{R} \\ q &\mapsto d(p, q). \end{aligned}$$

With this definition, we can report a well known result concerning the smoothness of d_p.

PROPOSITION 2.37 : The function d_p defined above is of class C^∞ on $\mathcal{M} \setminus \{C(p) \cup p\}$, and its gradient $\nabla d_p(q)$ at $q \in \mathcal{M} \setminus \{C(p) \cup p\}$ is given by

$$\nabla d_p(q) = \dot{\gamma}_{pq}(d_p(q)),$$

where we remember that γ_{pq} denotes the unique minimizing geodesic from p to q parametrized by arc-length. In particular, we have that $\|\nabla d_p(q)\| = 1$. Moreover, suppose that there exist at least two normal minimizing geodesics joining p to q. Then d_p is not differentiable at q.

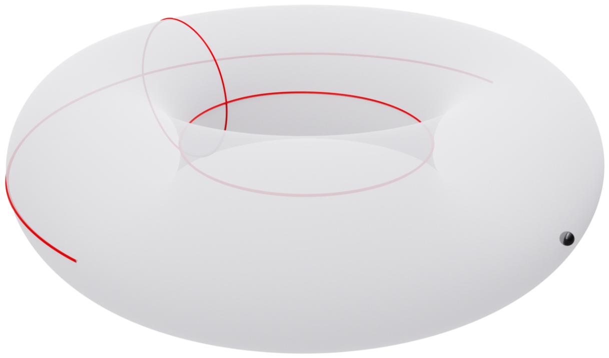


Figure 2.7: Example of the cut locus (red) of a point (black bullet) on a torus.

Note that, for what said in Section 2.3.2, conjugate points along a given geodesic from p are critical points of $\exp_p$, since at those points the differential of the exponential map is zero. Therefore, cut points may be characterized both with geometric arguments (conjugate points, number of minimal geodesics) and with analytical arguments (critical points of $\exp_p$, points at which d_p is not differentiable). We will see how these definitions has been used to implement algorithms that estimate the cut locus on discrete surfaces in Chapter 5.

DEFINITION 2.38 (Injectivity radius): Let p be a point in $\mathcal{M}$. We define the *injectivity radius* $i_p(\mathcal{M})$ at p as

$$\sup\{r > 0 : exp_p\big|_{B_r(0)} \text{ is a diffeomorphism}\},$$

where $B_r(0)0 \in T_p\mathcal{M}$ is a ball centered at $0 \in T_p\mathcal{M}$ with radius r.

PROPOSITION 2.39 : With the notations used above, we have that

$$i_p(\mathcal{M}) = d(p, C(p)),$$

and $p \mapsto i_p(\mathcal{M})$ is a continuous function from $\mathcal{M}$ to $\mathbb{R}^+ \cup \{+\infty\}$.

Second order differential properties at the cut locus have been investigated only more recently in a weak sense, like in the sense of distributions, or of viscosity, or of barriers [GOV22, MMU14, Nee07]. Roughly speaking, all such methods study the behavior at the cut locus by approximating the distance function arbitrarily well with a smooth function. Mantegazza and colleagues show the equivalence of analyses in the sense of distributions and of viscosity, and that the analysis in the sense of barriers implies the other two [MMU14]. Cordero-Erausquin and colleagues proved that the distance function fails to be semiconvex at the cut locus, by showing that the Hessian is unbounded from below [CEMS01]. Neel used the heat kernel as a natural mollification of the distance function and, among other results, proved that the norm of the Hessian blows to infinity at the cut locus [Nee07]. We will report here the result we will be using in Chapter 5, due to Générau [Gén20], referring to cited literature for any further detail on the subject.

DEFINITION 2.40 : Let $\psi : \mathcal{M} \to \mathbb{R}$ be a continuous function, and $p \in \mathcal{M}$. We say that the Laplacian of ψ at p is $-\infty$ *in the sense of barriers* if for any $A > 0$, there exists a smooth function $\phi : \mathcal{M} \to \mathbb{R}$ defined in a neighborhood of p such that

$$\phi \geq \psi, \quad \phi(p) = \psi(p), \text{ and } \Delta\phi(p) \leq -A.$$

THEOREM 2.41 : Let $\mathcal{M}$ be a smooth manifold without boundary of dimension $n \geq 2$, and $p \in \mathcal{M}$. The Laplacian of d_p is $-\infty$ at every point of $C(p)$ in the sense of barriers.

Concerning topological properties, we first report an important result due to Myers ([Mye35, Theorem 4, Vol. II])

THEOREM 2.42 : The cut locus $C(p)$ of point p on a closed analytic surface S is a linear graph. The endpoints of $C(p)$ are conjugate to p, and are cusps turned towards p of the locus of first conjugate points to p. An arc of $C(p)$ containing no points conjugate to p and no interior points which can be connected to p with more than 2 minimizing geodesics is a regular analytic arc.

The term *analytic surface* means that S can be locally represented through (real) analytic functions. Since we will apply this results in the case of embedded surface in the sense of Definition 2.17 with no boundary, this result holds in our setting. Note that Myer also proves that the number of endpoints of $C(p)$ (as a graph) is finite ([Mye35, Vol I]). These results have been generalized to higher dimensions by Buchner [Buc77].

We will now report a result due to Mantegazza and Mennucci which summarizes most of the geometrical and topological properties of the cut locus that we will need in the sequel. Such results will be adapted to our context in the following sense. We will omit the contributions regarding concepts that we did not define so far, and we will restrict the result to the case in which $\mathcal{M}$ is a Riemannian 2-manifold endowed with a C^∞ metric. For further details we refer to [MM02].

PROPOSITION 2.43 : Let $p \in \mathcal{M}$, where $\mathcal{M}$ is a smooth and connected Riemannian manifold. Then the squared distance function from p is C^∞ in $\mathcal{M} \setminus C(p)$, which is an open neighborhood of p. Moreover, $C(p)$ has Hausdorff dimension at most 1, and the subset of $C(p)$ where d_p fails to be differentiable is locally a finite union of smooth curves.

We now relate the topology of $\mathcal{M}$ with the one of $C(p)$. For the sake of brevity, we omit the introduction of basic concepts of Algebraic Geometry such as homology groups and retractions. The reader not familiar with these definitions is referred to [Ful97, CLO98].

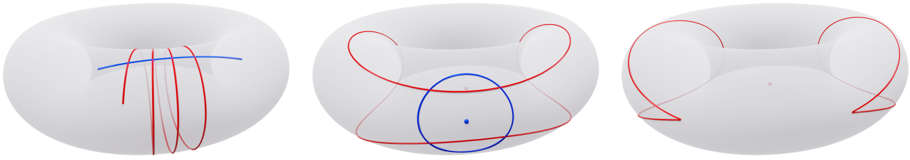


Figure 2.8: The intersection of a strongly convex set (blue) with a weakly convex one (red) could be not topologically consistent with what happens in the Euclidean plane: geodesics may intersect in more than one point (*left*) and circles may intersect in more than two points (*middle*); a circle with radius greater than the injectivity radius of its center may not be even homeomorphic to a Euclidean circle, and may be not smooth at the cut locus of the center (*right*).

PROPOSITION 2.44 : Let $\mathcal{M}$ be a compact connected 2-dimensional Riemannian manifold, and let p be a point on $\mathcal{M}$. Then

i) $\mathcal{M} \setminus \{p\}$ deformation retracts to $C(p)$ and $\mathcal{M} \setminus C(p)$ deformation retracts to p.

ii) $H_i(\mathcal{M}, \mathbb{Z}) \cong H_i(C(p), \mathbb{Z}) \qquad i = 0, 1,$

where $H_i(A, \mathbb{Z})$ denotes the i-th homology group.

Condition i) tells us that we can *continuously* deform $\mathcal{M} \setminus p$ into $C(p)$ and $\mathcal{M} \setminus C(p)$ into p, while, roughly speaking, condition ii) tells us that $\mathcal{M}$ and $C(p)$ have the same number of connected components and the same number of holes.

2.4.2 Convexity

As said before, in Riemannian geometry there several notions of convexity that one may consider, all of which coincide in R^n.

DEFINITION 2.45 (Convexity): Let K $(\neq \emptyset)$ be a subset of a Riemannian manifold $\mathcal{M}$. Then K is

- *weakly convex*, if for any two points $p, q \in K$ there exists a unique normal geodesic γ_{pq} such that γ_{pq} is entirely contained in K, and it is the unique minimizing geodesic in K connecting p to q.
- *strongly convex* if for any $p, q \in K$ there exists a unique normal minimizing geodesic $\gamma_{pq} \in \mathcal{M}$, and γ_{pq} is entirely contained in K.

To understand the difference between these two definitions one can consider the examples in Figure 2.8. Note that a geodesic is strongly convex if its length does not exceed the radius of injectivity of one of its endpoints, otherwise it is weakly convex. When one intersects a strongly convex geodesic with another geodesic which is not, then the number of intersections could be arbitrarily many (Figure 2.8, *left*). Similar arguments apply to the case of geodesic circles: if one of the two circles is not strongly convex, then the intersection could

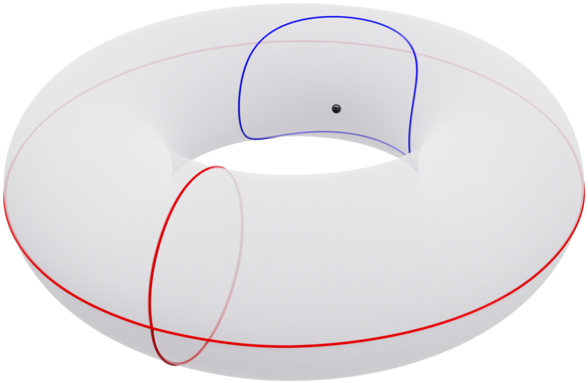


Figure 2.9: The cut locus (red) of a point p (black dot) on a torus. The convex ball (blue) centered at p cannot extend further, since otherwise it would contain pairs of points connected with a shortest path crossing the outer equator, hence not entirely contained in such ball.

be not topologically consistent with what happens in the Euclidean setting, where the intersection of two circles is either empty, or consists in one or two points. Note that the red circle in Figure 2.8(*middle*) is weakly convex, since for every point there exists a unique minimizing geodesic *within* such circle connecting them, which is not the case for the circle on the right, which is not even homeomorphic to its Euclidean counterpart.

DEFINITION 2.46 (Convexity radius): For every point $p \in \mathcal{M}$, let

$$r_p := \sup\{r > 0 : B_r(p) \ \text{ is strongly convex}\},$$

the r_p is called the *convexity radius* of p.

It is well known that if $\mathcal{M}$ is a Riemannian manifold with bounded curvature, then $r_p > 0$ for every $p \in \mathcal{M}$. Nevertheless, the estimate of the convexity radius of a point, or the convexity radius of the manifold (defined as the minimum among the convexity radii of its point) is still a matter of research interest nowadays (see for example [Dib17, Xu18]). Most of the classical results proved lower and upper bound in terms of the injectivity radius and upper bound of the curvature. However, in the following we will determine the convexity radius of a point p by considering the Hessian of the distance function d_p. In fact, Whitehead proved that if Hessd_p is positive definite inside a ball $B_r(p)$, i.e. d_p is strictly convex, then $B_r(p)$ is strongly convex [Whi33]. The proof of this fact can be carried out also by arguments similar to the ones used in [Sak97, Theorem 5.3]. Figure 2.9 shows the cut locus (red curve) of a point (blue bullet) on a torus, and a strongly convex ball (blue circle) centered at that point.

3

Discrete Setting

In the previous chapter we introduced the theory upon which the algorithms described in the following will be built. We will now see how such concepts can be brought into a discrete setting, where we cannot afford of considering the neighborhood of a point $p \in \mathcal{M}$ in the usual sense, i.e. as infinite set of points on $\mathcal{M}$ surrounding p. The discretization of $\mathcal{M}$ will be a *mesh* M, made of vertices, edges, and triangles, which will be denoted by V, E and T respectively. We assume the vertices $\{v_1, \dots, v_n\}$ of M to be points of $\mathcal{M}$, i.e. they belong to some sampling of points made on the continuous surface. We will denote with $t_{ijk} \in F$ the triangle having vertices $\{v_i, v_j, v_k\}$, and with e_{ij} the edge connecting v_i with v_j. For every vertex $v_i \in M$ there exists at least one triangle in T (plane) triangle having v_i as one of its vertices. We further require such triangulation to be a *simplicial complex*, which means that the intersection of every pair of triangles is either the empty one, or a common vertex, or a common edge. We do not allow configurations in which a triangle share its edge with more than one triangle or in which its edge intersect the interior of another triangle. Moreover, we would like to ensure that M preserve the property of a topological manifold, i.e. that it is possible to consider a covering of C^0 charts that maps homeomorphically the neighborhood of a vertex to a disk in $\mathbb{R}^2$. To state this property formally, we need to introduce some definitions and notations. Given a vertex $v \in M$, the set of triangles incident to v will be called the *star* of v, and will be denoted with $\mathcal{S}(v)$, while the set of vertices which are the other endpoints of all the edges incident in v is called the 1-*ring* of v, and will be denoted with $\mathcal{N}(v)$, where the choice of the letter $\mathcal{N}$ is due to the fact that we will often refer to this set as the (discrete) *neighborhood* of v. For $k \geq 2$, we define the k-*ring* of v as the set of vertices that adjacent to the vertices of the $(k-1)$-ring. Finally, since $\mathcal{N}(v)$ consists of a set of vertices that are connected pairwise with and edge, we will call the *link* $Lk(v)$ of v the union of $\mathcal{N}(v)$ and such edges. We will then require the link of every vertex $v \in M$ to be a single closes loop. If we think about the set $Lk(v) \cup \mathcal{S}(v)$ as a discretization of the mathematical neighborhood of v, then under the assumption made above this latter requirement is equivalent to ask that $Lk(v) \cup \mathcal{S}(v)$ is homeomorphic to a plane disk, which somehow "discretize" the definition of topological manifold seen in Section 2.1. More details about the existence of such triangulation can be found in [Cai35, Whi40, Whi57]. In the following, we will say that M is a *triangle mesh* having $\mathcal{M}$ as *carrier*.

We will now show how the concepts introduced in Chapter 2 can be adapted in order to be defined in this (discrete) domain. We will start by defining the tangent spaces for the vertices, the edges and the triangles of M. Then, we will give the definitions of two possible metrics that can be defined on M pointing out the nuances between these choices. The definition of covariant derivative will be somehow more subtle: in fact, we will only give the definition of (discrete) parallel transport, and this by (2.10) implicitly means that our discrete covariant derivative will be the one obtained by such definition. Note that, in the literature, more sophisticated methods exist that propose a discretization of this concept, see e.g. [LTGD16].

3.1 Tangent Spaces, Metric and Parallel Transport

We are now interested in defining the tangent space T_pM of a point p on M. Note that, with the term "point" we are not restricting ourselves exclusively to the vertices of M, but we are considering also the points in the interior of edges and triangles. In the following, we will call p a *mesh point*.

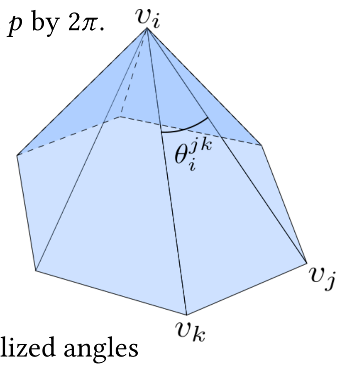
As pointed out in Section 2.1, the smoothness of the charts in Definition 2.2 allowed us to introduce the concept of vectors tangent to a differentiable manifold, which lead to the definition of tangent spaces. Since in our current setting we lack such differentiable structure, we need a way of "discretizing" such concepts. This can be done in two ways: either we see M as a topological manifold, and we define the tangent space at a given point $p \in M$ by finding a suitable discrete counterpart of the definition in the continuous setting, or we see M as an approximation of an embedded surface, for which we find a local parametrization around every vertex $v_i \in M$. The former approach will be the one described in this section. While a mixture of the two will be used in Section 3.2.2 to define discrete differential operators on M and in Section 3.2.3 we justify this latter choice. Note that these two different conceptions of M will also be considered in Chapter 4, where we introduce the discretization of the geodesics.

We thus aim at reproducing the concept of tangent space introduced in Section 2.1, so for every point p we want to define a 2-dimensional vector space in which the vectors tangent to M at p belong.

Let us start by assuming that p lies inside a triangle t_{ijk}. It is clear that, for every curve $\gamma : [a, b] \to M$ through p, there exists $a_p, b_p \in \mathbb{R}$ such that $\gamma\big|_{[a_p,b_p]}$ is a plane curve entirely contained in t_{ijk}. Therefore, in this case, the 2-dimensional space in which all the vectors tangent to M at p lie can be identified with the plane containing t_{ijk}. Similarly, if p lies on an edge e, once the two triangles t_0, t_1 sharing e are flattened onto a common plane, the considerations made before apply also in this case, too. Note that, in both cases, one can define a system of coordinates in T_pM such that the exponential map $\exp_p$ from T_pM to $U \subset M$ is an isometry, where U in the first case is t_{ijk} and in the second is $t_0 \cup t_1$.

If p is vertex instead, then such mapping cannot be isometric, in general. In this latter case, we use a well known approach in the literature (see e.g.

[ZMT07, KCPS13, SSC19b]) which consist in mapping the triangles in the star of p by rescaling the total angle they form about p by 2π.



In details, if $V = \{v_1, \dots, v_n\}$, suppose that $p = v_i$ for some $i = 1, \dots, n$. Let us denote with θ_i^{jk} the angle at v_i of the triangle $t_{ijk} \in \mathcal{S}(v_i)$ (see inset).

Then by letting

$$\Theta_i := \sum_{t_{ijk} \in \mathcal{S}(vi)} \theta_i^{jk} \tag{3.1}$$

be total interior angle at v_i, we define the normalized angles

$$\tilde{\theta}_i^{jk} := \frac{2\pi\theta_i^{jk}}{\Theta_i}$$

which sum to 2π. Therefore, we can now consider a plane T_pM in which all the neighbors of v_i can be represented. In fact, by putting $\mathcal{S}(v_i) = \{t_{ij_0j_1}, \dots, t_{ij_{m-1},j_m}\}$, we can map an outgoing edge e_{ij_0} into the x-axis of T_pM, and define a polar coordinate system (ρ, θ) consistently with the rescaling of the angles done above. By putting

$$\theta_{ija} := \sum_{p=0}^{a-1} \tilde{\theta}_i^{j_pj_{p+1}},$$

every vertex $\{v_{j_0}, \dots, v_{j_m}\}$ in $\mathcal{N}(v_i)$ can be uniquely represented with the pair $(\rho_{ij_a}, \theta_{ij_a})$, where ρ_a is the length of e_{ij_a} and θ_{ij_a} is the cumulative angle defined above, $a = 0, \dots, m$. It is therefore clear that, in this case, $\exp_p$ will be just a radial isometry, since we are just preserving the lengths of the outgoing edges (i.e. the radial geodesics) from p.

In all the three possible scenarios discussed above, it is clear that the most natural choice is to define the inner product in T_pM as the usual Euclidean inner product $\langle \cdot\,, \cdot \rangle$. Let us now consider the edge e_{ij} for some $v_i, v_j \in V$. For what said above, θ_{ij} encodes the direction of such edge in the tangent space of v_i. In other words, the vector $(\rho_{ij}, \theta_{ij}) \in T_{v_i}M$ is the vector tangent to the (trivial) geodesic joining v_i to v_j at v_i. Suppose now that we want parallel transport some vector $\xi \in T_pM$ from $T_{v_i}M$ to $T_{v_j}M$. Note that, since v_j is a neighbor of v_i, e_{ij} can also be expressed in the coordinate system of v_j, i.e. by $(\rho_{ij}, \pi + \theta_{ji})$, where π accounts for the fact that θ_{ji} is the direction of v_i in the tangent space of v_j, as shown in Figure 3.1. It is clear then that, to parallel transport ξ from $T_{v_i}M$ to $T_{v_j}M$, we need to rotate it by the angle $(\pi + \theta_{ji}) - \theta_{ij}$. By doing this, we are ensuring that ξ will keep a constant angle with respect to the tangent vector of e_{ij} when moving from one tangent space to the other, i.e. ξ does not undergo to any variation along e_{ij} during the transition. Similarly, one can parallel transport a vector from a vertex v to a triangle t (and vice versa) by thinking of the centroid of the triangle as a vertex of the mesh, and expressing its direction in the tangent space of v and proceeding as before, while the parallel transport a vector from a triangle to an adjacent one boils down to a translation once we flatten the two triangle in common plane.

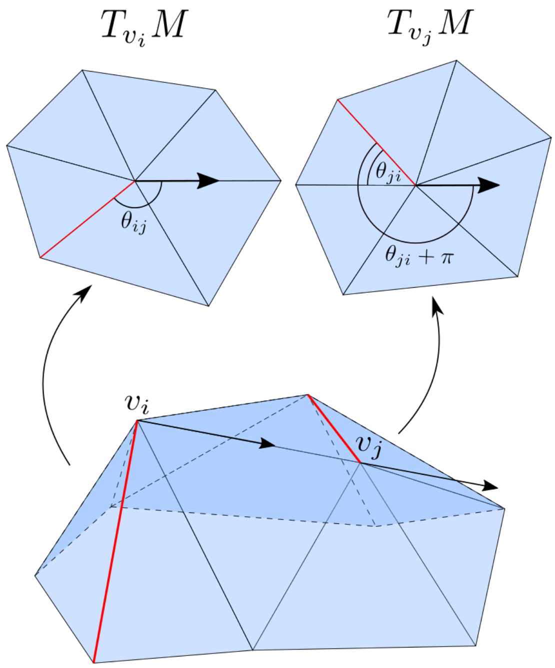


Figure 3.1: The edge e_{ij} can be represented both in the tangent space of v_i and in the tangent space of v_j. Viewed as a geodesic connecting such vertices, its tangent vector at v_i forms an angle θ_{ij} with respect to the reference direction $\theta = 0$ (red). We can then express such direction in $T_{v_j}M$ as $\theta_{ji} + \pi$.

Therefore, parallel transport in the discrete setting consists in rotations that keep a vector "constant" when moving them from a tangent space to another. Such definition suggests a useful interpretation of the polar coordinates in the tangent space of a vertex v_i, which can be thought as normal coordinates (Definition 2.23) at v_i. In fact, we are considering a neighborhood around v_i, and we are mapping the points in such neighborhood through the logarithmic map, since the pair (ρ_{ij}, θ_{ij}) represent the tangent vector at v_i of the geodesic e_{ij} connecting v_i to v_j. Furthermore, it is clear that every mesh point p belonging to a triangle in $\mathcal{S}(v_i)$ can be expressed in such coordinates system by considering the straight line segment that join such point to v_i.

3.2 Differential Operators

Quite often, a scalar field is sampled at the vertices of a mesh, which as said before discretizes a given domain. Computational analysis of such field may require evaluating its first and second order derivatives and, possibly, tracing the integral curves of its gradient. To fix ideas, consider the case in which such scalar field is the geodesic distance function from a given point on a mesh (sampled at its vertices). Then the integral lines of such field are the geodesic emanated from such point.

In the discrete setting, the knowledge of a scalar function (or a *signal*) on a mesh M is limited to a finite number of points, which usually are the vertices of M. However, in order to compute first and second order derivatives of such function, one needs to somehow extend it to the whole M. In the present thesis, this will always be done by linear interpolation, which means that we assume the function to be linear inside the triangles of M, where we evaluate it by interpolating its values at the vertices. Assuming the piecewise linearity of the function could overly simplify its nature, especially if M is coarse.

Working on a discrete domain not only influences the definition of a function on it, but also the way in which one differentiate such function. To fix ideas, one can think of the first order finite differences method to estimate the gradient of a function f in the univariate case: since the points at which the values of f are known do not have a continuous neighborhood, one approximates the quantity

$$\lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

with $f(x_{i+1}) - f(x_i)/(x_{i+1} - x_i)$, where $\{x_0, \ldots, x_k\}$ is a sampling of the domain at which f is known. When moving to a discretization of a 2-dimensional curved domain, things get more complicated, since we also need to find a way of discretizing a covariant differentiation.

In this section, we will start by considering the planar case, i.e. we postpone the case of discrete curved domain after reviewing the main methods that estimate the gradient of scalar function defined over a triangle planar mesh. The idea is to understand the pros and cons of each of these methods in the former case, and exploit such knowledge to design a robust way of estimation differential quantities in the latter one. This seems reasonable since the pathologies that a given method has in the planar case will be preserved when considering a curved domain, so in this way we are somehow splitting the analysis in two parts, where we isolate the problems due the discretization of the domain from the ones induced by the discretization of the covariant differentiation. The final purpose, will be a unique framework in which the first and second order derivatives of scalar field on a (non-planar) triangle mesh M can be computed in an efficient and robust way.

3.2.1 Gradient Field Estimation on Triangle Meshes

This section includes contents from a co-authored paper [MLP19] that has been re-formatted for this thesis.

A said before, in the geometry processing and FEM literature, a scalar field is often extended from vertices to the interior of higher dimensional triangles by linear interpolation. Under this approach, a constant gradient is associated to each triangle with a straightforward computation, thus providing the simplest form of evaluation of the gradient field. Although sufficient for many applications, the piecewise constant gradient has several limitations and drawbacks: being not continuous, the gradient has divergent covariant derivative at the interface between triangles; singularities are forced to lie just at vertices of the mesh; and integral curves are discretized into polylines that travel parallel to each other inside each element, so that their distribution does not depend just on the field, but also on the orientation of edges and on the vertex valences of the underlying mesh (see Figure 3.2).

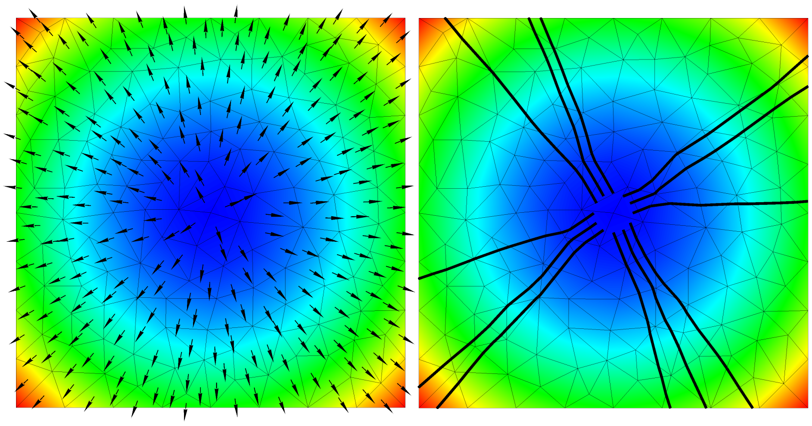


Figure 3.2: Per-face gradient estimation on a paraboloid with minimum at the center of the domain (left). Although the estimate is very accurate, integral lines traced from a neighborhood of the minimum do not diverge uniformly, but rather travel in four bundles of nearly parallel lines (right). This behavior is induced by the discrete piecewise constant nature of the gradient field. Starting the tracing from the star of a vertex with low valence exacerbates the problem.

As discussed in [DGDT16], discrete vector fields can be given per face, per edge, or per vertex. Despite the entity to which the gradient estimate is attached, the gradient field can be extended to the whole mesh either in a piecewise constant manner – i.e., defining a local region surrounding each vertex/edge/face and assuming the gradient to be constant within it, or via interpolation – e.g., linearly inside each simplex. When vector fields are extended to the whole mesh by linear interpolation, they can be traced exactly inside each region [KRG03, NJ99].

In [MLP19], common methods for estimating a per-vertex gradient field linearly interpolated within each element have been reviewed, and a comparison of their accuracy with respect to the standard method to compute a per-face constant gradient field has been presented. We will start by describing the methods considered, and then briefly summarize the results of such work. Note that, even if such comparisons has been made considering planar 2D meshes and 3D meshes, further experiments confirmed that the considerations made in that context may be extended also in the case of surface meshes. This latter statement will be clarified at the end of this section through more specific arguments. In the following, we will use the following notations.

Let $\Omega \subset \mathbb{R}^2$ be a compact domain and M be a triangle mesh having Ω as carrier. As done before, with denote with V and T the vertices and the triangles of M, respectively. Let $f : \Omega \longrightarrow \mathbb{R}$ be a smooth scalar function on Ω. We assume to know the value of f only at the vertices of M. The discrete version of f is therefore a collection $F = \{f_1, \dots, f_n\}$, where n is the number of vertices in V and each f_i corresponds to the function value sampled at vertex v_i, for all $i = 1, \dots, n$. The problem addressed in the following is that of estimating the gradient ∇f on Ω, on the basis of the discretizations F and M.

Per-Triangle linear Estimation (PTE)

Our bottom line is a method that estimates a constant gradient at each triangle. We start by extending the values in F to the triangles of M by linear interpolation, which means that in each triangle t of M we consider the (unique) linear function

$$\tilde{f}_t(p) = \sum_{v_i \in t} v_i \lambda_i f_i,$$

where p is a generic point of t, the v_i's are the vertices of t and the λ_i's are the barycentric coordinates of p with respect to the v_i's. In this model, function $\tilde{f}$ estimates f as a piecewise-linear function, which is continuous over Ω and differentiable only in the interior of its triangles. The gradient of $\tilde{f}$ is thus constant inside every triangle t and it is associated either to the whole t, or conventionally to its centroid c_t, depending on the applications. For a triangle t with vertices v_i, v_j, v_k it is easy to show that we have

$$\nabla f_t = (f_j - f_i)\frac{(v_i - v_k)^\perp}{2A_t} + (f_k - f_i)\frac{(v_j - v_i)^\perp}{2A_t}. \tag{3.2}$$

where $e^\perp$ denotes edge e rotated by 90°, and A_t is the area of t. In fact, by a corollary of the divergence theorem, we have that the integral $\nabla\tilde{f}$ over t is equal to the integral of $\tilde{f}$ along the edges of t. Since $\tilde{f}$ is linear by definition along edges, we can use the trapezoidal rule to compute such integral, which gives Eq. (3.2).

In the piecewise linear model of $\tilde{f}$ the gradient is not defined at vertices of M. The methods we review in the following assume that f is a higher order function, smooth at edges and vertices of M. The different methods exploit different facts that hold in the continuous case, and try to bring them to the discrete setting. All such methods work either by averaging (integration) or by approximation (fitting), because no exact model can be assumed for f in the generic case.

Note that, once the gradient has been estimated at all vertices, the gradient field can be extended by linear interpolation inside cells of any order. It is therefore continuous in Ω and overall more accurate than the piecewise constant field reviewed in the previous section [MLP19]. In the following, with a slight abuse, we will identify f with $\tilde{f}$.

Average Gradient on Star (AGS)

A common procedure in discrete differential geometry consists of estimating a differential property at a point p as the average value of the same property in a neighborhood of p [MDSB03].

More formally, in our case, we can write

$$\nabla f(p) \simeq \frac{1}{A_{B(p)}} \int_{B(p)} \nabla f da, \tag{3.3}$$

where $B(p)$ is a neighborhood of p and $A_{B(p)}$ is its area.

Now, given a vertex v of M, we can use the method in the previous section to estimate (an average value of) ∇f in the triangles of the star of v, and compute the integral as a sum of constant terms, obtaining

$$\nabla f_v \simeq \frac{1}{\sum\limits_{t \in \mathcal{N}(v)} w_t} \sum_{t \in \mathcal{N}(v)} w_t \nabla f_t, \tag{3.4}$$

where ∇f_t is the value computed with Equation 3.2 and w_t is the weight assigned to triangle t incident at v. Correa et al. [CHM09] studied the influence of different weights on the accuracy of results on tetrahedral meshes, They experimented using the *inverse distance of centroid of τ from v*, the *(solid) angle of τ at v*, and the *volume of τ*, where τ denotes a tetrahedron incident at v. Note that, in our case, weighting with the area corresponds to applying the divergence theorem to compute Equation 3.3 on the star of v (or, equivalently, on a centroidal decomposition of triangles, as defined in [MDSB03]), by assuming the linear model inside each incident triangle. According to the experiments in [CHM09, TW97] however, the inverse distance of centroid and the angle weights perform similarly, and overall better than the volume weight. Furthermore, in the experiments made in [MLP19], we found the angle weight to be more robust against anisotropic meshes (i.e., meshes containing elongated elements), hence we consider such weight throughout. In summary, w_t in Equation 3.4 will be the measure of the angle of t at v (where dependence of w_t on v has been omitted to keep a lighter notation).

Least Squares fit of Directional Derivatives (LSDD)

This approach consists in estimating first a few directional derivatives of f at v_i, and imposing their relation with the gradient. Let be $\{v_0, ..., v_{k_i}\}$ the vertices belonging to the 1-ring of v_i. Taylor's expansion of f at the first order allows us to write:

$$f(v_j) - f(v_i) \approx \nabla f \cdot (v_i - v_j),$$

for every $j = 0, ..., k_i$. The idea is to build a linear system exploiting the above approximation, i.e. writing

$$f(v_j) - f(v_i) = \nabla f \cdot (v_i - v_j), \qquad j = 0, ..., k_i. \tag{3.5}$$

Note that the second term of (3.5) is the directional derivative of f along vector $(v_i - v_j)$, hence the name. Since k_i is usually greater than the dimension of the space, 2, the linear system is usually overdetermined and it only admits a least squares solution. As observed in [CHM09], this can be addressed as a weighted least squares problem, where a weight is assigned to each equation, which is inversely proportional to the square of the corresponding edge length. Let A_i be the $k_i \times 2$ matrix obtained by collecting all the $(v_j - v_i)$, let W_i the diagonal matrix of weights, and let D_i be the column matrix consisting of all the $f(v_j) - f(v_i)$. Then, the weighted least squares solution is obtained by resolving the 2×2 linear system

$$A_i^T W_i A_i \nabla f(v_i) = A_i^T W_i D_i, \tag{3.6}$$

where $W_i(j,j) = 1/d_{ij}^2$ and d_{ij} is the length of edge v_iv_j. Note that such a system must be solved at every vertex. The unweighted solution (i.e., $W_i = I$) turned out to be less robust in all the experiments made in [MLP19].

Linear regression (LR)

The last approach we review consists of approximating function f in the neighborhood of v_i with a polynomial π_i of given degree, by setting a system of linear equations that asks π_i to assume the given values of F at all vertices of a given k-ring of v_i. After the fitting polynomial has been obtained, the gradient of f at v_i is estimated analytically as the gradient of π_i.

In [MLP19], we considered quadratic polynomials and 1-rings, which are extended to 2-rings only if the number of neighbors of v_i is insufficient to fix all degrees of freedom, so we will describe this method consistently with such choice. We have

$$\pi_i(x,y) = a_ix^2 + b_iy^2 + c_ixy + d_ix + e_iy + f_i, \tag{3.7}$$

where coefficients $P_i^T = [a_i, b_i, c_i, d_i, e_i, f_i]$ are unknown. For each vertex v_j in the neighborhood of v_i (including v_i itself), we impose $\pi_i(v_j) = f_j$, thus obtaining a linear system with as many equations as the vertices in the neighborhood of v_i. Again, we address it as a weighted least squares problem, by assigning a weight to each equation, and we obtain the coefficients of the best fitting polynomial by solving the system

$$A_i^T W_i A_i P_i = A_i^T W_i F_i, \tag{3.8}$$

where:

- A_i is a $k_i \times 6$ matrix containing one row per vertex in the neighborhood of v_i (including v_i itself); the row corresponding to vertex $v_j = (x_j, y_j)$ contains values $(x_j^2, y_j^2, x_jy_j, x_j, y_j, 1)$;
- F_i is a column vector containing the values f_j corresponding to the vertices v_j in the neighborhood of v_i;
- W_i is a diagonal matrix of weights, each inversely proportional to distance from v_i with a Gaussian decay:

  $$W_i(j,j) = \frac{1}{L\sqrt{2\pi}} e^{-\frac{d_{ij}^2}{L^2}}$$

  with d_{ij} being the length of edge v_iv_j ($d_{ii} = 0$) and L being the average edge length in the mesh.

Once the coefficients of π_i are known, the gradient at $v_i = (x_i, y_i)$ is given trivially by

$$\nabla f(v_i) = (2a_ix_i + c_iy_i + d_i, 2b_iy_i + c_ix_i + e_i).$$

Note that we have to assemble and solve such a system at every vertex, hence this method is the most expensive in the set we review. As for the previous method, we found the weighted solution of the least squares problem to perform better than the unweighted solution in all our experiments.

Outcome of the comparison

The four techniques presented in previous section were tested using analytic functions, comparing numerical estimates with the ground truth. In order to analyze different situations, a parametric family of non-polynomial periodic functions was used together with meshes with different characteristics. We considered the domain $\Omega = [0,1] \times [0,1]$ and the following parametric family of functions:

$$f_{a,b}(x,y) = a\ \sin(bx)\ \cos(by).$$

Parameters a and b control the amplitude and the frequency of the function, respectively. Figure 3.3 shows four plots of $f_{a,b}$ for different values of a and b.

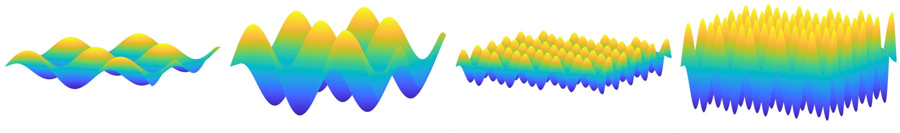


Figure 3.3: Four examples of the test function $f_{a,b}$ used in [MLP19]. Parameter a controls the amplitude, whereas parameter b controls the frequency. From left to right: $f_{2,10}$, $f_{5,10}$, $f_{2,30}$, $f_{5,30}$.

We used three different types of discretization for the domain Ω: a *structured* mesh made of equilateral triangles, an *unstructured mesh* obtained by triangulating a Poisson sampling of the domain, and an *anisotropic mesh* obtained by triangulating a rectangle and squeezing it to fit the square. Closeups of these three tessellations is shown in Figure 3.4.

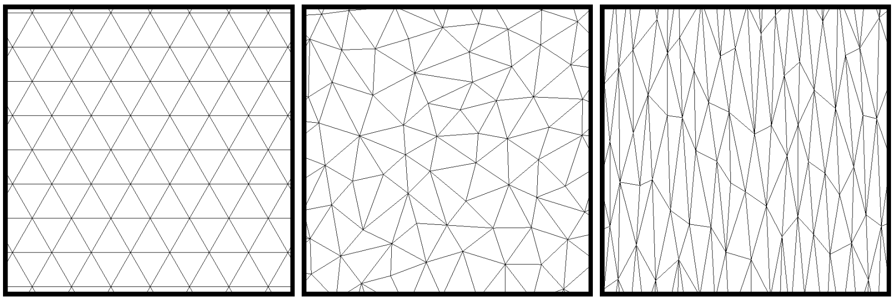


Figure 3.4: Close up of the test meshes. From left to right: structured, unstructured and anisotropic.

We also used different error metrics, in order to isolate the error in estimating the direction and the magnitude of the gradient. The analysis of our results, not surprisingly, showed that PTE perform worse than vertex-based methods. The only exception being performances at the boundaries, and the estimation of the sole magnitude of the gradient on coarse meshes, where separate computation per triangle offers an advantage over methods that use a bigger stencil. On the other hand, a smaller stencil turned out to be counterproductive when considering the estimation of the direction of the gradient, especially near critical points of the signal. In fact, sampling the signal just along three directions (the vertices of the triangle) gives not enough information for an accurate estimation, especially if nearby the signal undergoes to abrupt changes. In other words, the vertices of the triangle give a poor approximation of the neighborhood in Eq. (3.3) and this affects the estimation of the gradient, as shown in Figure 3.5.

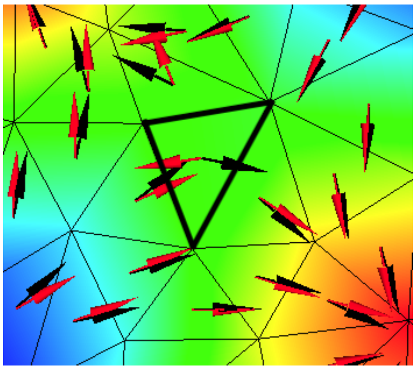


Figure 3.5: Example in which PTE (red arrows) fails in estimating the direction of the gradient (black arrows). The geometry of the highlighted triangle is such that the estimation of the gradient through Eq. (3.2) does not take into account of the maximum the ground truth is pointing to.

AGS and LSSD are more resilient to this kind of errors, but they may offer a poor approximation of the gradient near singularities. In fact, even if the stencil used is larger, and span more directions around the vertex we are considering, it still depends on the local geometry of the mesh, and it may happen that some (relevant) information is lost. Moreover, these effects are more evident at boundary vertices. In fact, in these cases the estimation relies on an asymmetric discretization of the neighborhood $B(p)$, since we have only half of it to our disposal. Roughly speaking, the absence of edges or triangles that "pull" the gradient to a different direction make the estimation to be based just on what happens on one side (the one in which we know the signal).

For LR the situation is rather different. Note that estimating a gradient at the boundary with LR is equivalent to using LR to extrapolate the signal beyond the boundary. It is well known that extrapolation methods are very unstable, unless a prior on the data can be assumed. The assumption underlying LR is that the signal is a quadric, which is not true in general, and in particular for the test function used to run the experiments. In fact, LR does not fail systematically at all critical points, but when it does, it may return very poor results. Figure 3.6 summarizes what has just been said showing the heat map of the error made by the four methods when estimating the gradient of $f_{0.1,11}$. In detail, the error shown is $\|\nabla f(v_i) - \nabla \tilde{F}(v_i)\|$, where $\nabla f(v_i)$ is the gradient estimated with a given method, and $\nabla \tilde{F}(v_i)$ is analytic gradient of $f_{0.1,11}$ computed at v_i.

We conclude this section by reporting a table that summarizes the results obtained in [MLP19], and we refer to such paper for every further detail on the subject (such as the analysis on 3D meshes which was not commented in this thesis). For a given mesh, we define L to be its average edge length, hence

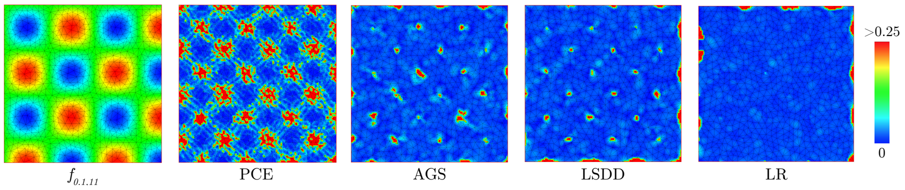


Figure 3.6: Heatmaps of the total error measured on the 2D unstructured mesh with test function $f_{0.1,11}$.

$1/L$ to be the *sampling frequency*. In the experiments, we did not investigate progressively finer meshes; we rather decided to keep the tessellation fixed and to act on the frequency of the test function (i.e., parameter b), in order to study the interaction between domain discretization and signal frequency. Note that the two approaches are equivalent. The quantity L/P in the table below, stands for the ratio signal frequency over sampling frequency.

Method	**Overall performance**			**Resilience to:**			Comp. cost	**Pathologies**
	interior		**boundary**	**poor L/P**	**anisotropy**	**noise**		
	direction	*magnitude*						
PCE	- -	+	-	+	-	- -	++	critical points
AGS	++	-	-	-	+	+	++	critical points
LSDD	++	-	-	-	+	+	-	critical points
LR	++	-	- -	-	++	+	- -	critical points on boundary

Table 3.1: Summary of the performance of the various methods on an ordinal scale (- -, -, +, ++) on all aspects analyzed. Overall, LR is the preferable algorithm to use in terms of robustness, while AGS offers the best trade off between accuracy and efficiency.

3.2.2 Differential Operators: A Unified Framework

Even if the analysis made in [MLP19] dealt with planar and 3D meshes, i.e. complexes that do not require the use of a non-linear geometry, further experiments confirmed that the considerations made applied also to the case of surface meshes. In fact, the (discrete) curvature of the mesh does not change the fact that methods such as PTE, AGS or LSSD heavily depend on the quality of the neighborhood used to compute the gradient. In details: PTE considers just one triangle, while, in the other two cases, once the contributions of the neighboring triangles or vertices has been parallel transported to the tangent space of a given vertex, there are no differences between surface and planar setting in terms of pathologies. Of course, results can be worsened by the fact that the contributions of nearby vertices are parallel transported. At this regard, it is important to point out another major difference between the setting considered in [MLP19] and the current one. Both in the case of planar meshes and 3D meshes, the discretization of the domain implies the restriction of the knowledge of the signal to the vertices of the mesh. Nevertheless, such domain is *exactly* represented by the faces (triangles or tetrahedra) of the mesh, which is not the case when considering a curved domain. In other words, when we triangulate a planar domain Ω we are considering a subdivision of

Ω into triangles such that the union of such triangle is exactly Ω, so a coarse tessellation of Ω, for example, is just affecting the way in which the signal is approximated (since we assume its linearity inside the triangles), but the underlying domain is still faithfully represented by the mesh. This is no longer true when considering the triangulation of a smooth manifold: the triangles are an approximation of the geometry, so in this case a poor discretization of the domain does not affect just the signal, but also the accuracy in representing the exact domain in which the signal is defined.

As it will be clear in the following, an accurate estimation of the gradient near critical points is of paramount importance for our purposes, so the pathologies that AGS and LSSD present near such points are not acceptable. On the other hand, LR is more robust away from boundaries, since the geometry around the vertex is used to fit a quadratic polynomial which in turn will determine the gradient at that vertex. Nevertheless, we cannot afford to solve a linear system for every vertex of the mesh whenever we need to compute the gradient of a function, since our target are high-tessellated meshes and every operation need to be real-time on them.

We therefore considered the approach proposed by Xu in [Xu13]. The main idea is similar to what described for the LR algorithm, although in this case the quadratic fitting is used not only to estimate the derivatives of the signal, but also to approximate the local parametrization of the continuous surface discretized by the mesh, and use such knowledge to compute the local representation of the metric.

Our implementation follows this exact construction, except for the choice of points used to fit the quadric. At the end of this section, we will present some examples that show how our choice improve the accuracy of the algorithm. Based on such technique, we also implemented a discrete Riemannian Hessian operator which, at the best of our knowledge, has never been proposed before.

Local Representation of the Metric through Quadric Fitting

Let us consider an embedded surface $S \subset \mathbb{R}^3$ as in Definition 2.17. Then, for every $p \in S$, we know that we can consider a local parametrization $x : U \longrightarrow S$ in a neighborhood V of p. In order to simplify the notations, sometimes we may write the coordinates (u, v) of $\mathbb{R}^2$ as (u^0, u^1), consistently with what done in Chapter 2.

We will start by introducing some notations and re-writing some quantities in a more convenient way. In particular, if

$$x_{u^i} = \frac{\partial x}{\partial u^i}, \quad x_{u^i u^j} = \frac{\partial^2 x}{\partial u^i \partial u^j}, \qquad i, j = 0, 1,$$

then $g_{ij} = \langle x_{u^i}, x_{u^j} \rangle$, where g_{ij} denotes the local representation of the metric. Therefore, by denoting with G the determinant of g_{ij} and remembering that its inverse g^{ij} has the form

$$g^{ij} = \frac{1}{G} \begin{pmatrix} g_{11} & -g_{01} \\ -g_{10} & g_{00} \end{pmatrix},$$

the expression of the gradient in (2.12) may be written as

$$\begin{aligned}\nabla f &= (x_u, x_v) g^{ij} (f_u, f_v)^T \\ &= g_u^{\nabla} f_u + g_v^{\nabla} f_v,\end{aligned} \tag{3.9}$$

where $g_u^{\nabla} = (g_{22} x_u - g_{12} x_v)/G$ and $g_v^{\nabla} = (g_{11} x_v - g_{12} x_u)/G$. We will also write g_{ijk} to denote the product $\langle x_{u^i}, x_{u^j u^k} \rangle$. With these notations the expression (2.11) of the Christoffel symbol may be re-written as

$$\begin{aligned}\Gamma_{ij}^k &= \frac{1}{2} g^{km} (g_{mji} + g_{jmi} + g_{imj} + g_{mij} - g_{jim} - g_{ijm}) \\ &= g^{km} g_{mji},\end{aligned} \tag{3.10}$$

where we exploited the fact that $g_{ijk} = g_{ikj}$, since the derivatives of x commute. Then, by (2.13), we can write

$$\begin{aligned}\nabla^2 f_{00} &= f_{uu} - \frac{1}{G} (H_u^{00} f_u + H_v^{00} f_v) \\ \nabla^2 f_{01} &= f_{uv} - \frac{1}{G} (H_u^{01} f_u + H_v^{01} f_v) \\ \nabla^2 f_{11} &= f_{vv} - \frac{1}{G} (H_u^{11} f_u + H_v^{11}) f_v),\end{aligned} \tag{3.11}$$

where

$$\begin{aligned}H_u^{00} &= g_{11} g_{000} - g_{01} g_{100} & \qquad H_v^{00} &= g_{00} g_{100} - g_{01} g_{000} \\ H_u^{01} &= g_{11} g_{001} - g_{01} g_{101} & \qquad H_v^{01} &= g_{00} g_{101} - g_{01} g_{001} \\ H_u^{11} &= g_{11} g_{011} - g_{01} g_{111} & \qquad H_v^{11} &= g_{00} g_{111} - g_{01} g_{011}\end{aligned}$$

Putting together the above formulas with (2.15) we have

$$\Delta f = g_u^{\Delta} f_u + g_v^{\Delta} f_v + g_{uu}^{\Delta} f_{uu} + g_{uv}^{\Delta} f_{uv} + g_{vv}^{\Delta} f_v v, \tag{3.12}$$

where

$$g_u^{\Delta} = -(g_{00}(g_{11} g_{011} - g_{01} g_{111}) + 2 g_{01}(g_{01} g_{101} - g_{11} g_{001}) + g_{11}(g_{11} g_{000} - g_{01} g_{100}))/G^2,$$
$$g_v^{\Delta} = -(g_{00}(g_{00} g_{111} - g_{01} g_{011}) + 2 g_{01}(g_{01} g_{001} - g_{00} g_{101}) + g_{11}(g_{00} g_{100} - g_{01} g_{000}))/G^2,$$
$$g_{uu}^{\Delta} = g_{11}/G, \qquad g_{uv}^{\Delta} = -2 g_{01}/G, \qquad g_{vv}^{\Delta} = g_{00}/G.$$

Even if (3.11) turned out to be very useful to obtain (3.12), which will be used to define our discrete Laplacian operator, for our purposes it is important to find a similar expression for Hess$f$, since it will be the object we will use to assess the positive-definetess of the second covariant derivative. For this reason, we observe that, by substituting (3.10) in (2.14) we can write

$$\mathrm{Hess} f(\xi) = g^{\ell m} \Big(\frac{\partial^2 f}{\partial u^m \partial u^i} - g_{jim} g^{jh} \frac{\partial f}{\partial u^h} \Big) \xi^i x_{u^\ell},$$

where $\xi = \xi^i x_{u^i} \in T_p S$. By defining the matrix $\mathcal{H}_i^\ell$ as the matrix having the ik-th entry equal to

$$g^{\ell m} \Big(\frac{\partial^2 f}{\partial u^m \partial u^i} - g_{jim} g^{jh} \frac{\partial f}{\partial u^h} \Big),$$

we have that Hess$f(\xi)$ is a linear operator from $T_p S$ to itself satisfying

$$\mathrm{Hess} f(\xi) = \mathcal{H}_i^\ell \xi^i x_{u^\ell}. \tag{3.13}$$

The reader familiar with basic notions of tensor algebra may notice that we have just switched from the notion of second covariant derivative, which is a tensor of type $(0,2)$, to the one of Hessian operator, which is a tensor of type $(1,1)$ (i.e. a matrix). Some authors called the Hessian operator the former [Sak97, CE75], while others use this term for the latter [dC92, AMP08]. Since both of them are important in our context, we use the notation and terminology consistent with [AMP08], and refer the reader to Chapter 5 of such book for further details about this subject.

The idea proposed in [Xu13] can be seen as a mixture of the techniques describe in the LR and the LSSD algorithm. In fact, the shape of a neighborhood V of a vertex $v_k \in M$ is approximated with a quadratic polynomial, in order to compute the partial derivatives of the parametrization x (similarly to what seen in the LR algorithm), and the same techniques is used to approximate the partial derivatives of a scalar function f defined on M exploiting its Taylor expansion in V (as in the LSSD methods). Since we have seen in the planar case that both of these approaches are more accurate when the neighborhood considered is as similar as possible to a ball, instead of considering the 1-ring of v_k as done in [Xu13], we perform a uniform sampling of m points around on a geodesic circle of radius r around v_k. In this way, we considerably reduce the dependence on the connectivity of the mesh. At the end of this section we will present some examples that show how this choice considerably improves the accuracy of the method and we will give more details about the choice of m and r.

We start by considering a uniform sampling $\{\theta_1, \dots, \theta_m\}$ of the interval $[0, 2\pi]$, and we trace k straightest paths of length r with initial direction $(r\cos(\theta_a), r\sin(\theta_a)) \in T_vM$, $a = 1, \dots, m$. Let $\mathcal{B} := \{p_1, \dots, p_m\}$ be endpoints of such geodesics. In order to simplify the notation, we put $p_0 = v_k$. Since in this case we want more flexibility, instead of fixing a quadratic polynomial as done before, it is more practical to consider the basis functions

$$\{Q_\ell(u,v)\}_{\ell=0}^5 := \{1, u, v, \frac{1}{2}u^2, uv, \frac{1}{2}v^2\},$$

where the factor $\frac{1}{2}$ is present just to simplify the computations. Our first step is to find an approximation of x using these basis functions. To do that, we put $q_0 = (0,0)$ and $q_a = (r\cos(\theta_a), r\sin(\theta_a))$, $a = 1, \dots, m$, and we look for coefficients $c_\ell \in R^3$ such that

$$\sum c_\ell Q_\ell(q_a) = p_a, \qquad a = 0, \dots, m,$$

in the least-squares sense. To do that, we consider the matrix $A \in \mathbf{M}_{m+1\times 6}(\mathbb{R})$ of the form

$$\begin{pmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ 1 & r\cos(\theta_1) & r\sin(\theta_1) & \frac{1}{2}r^2\cos^2(\theta_1) & r^2\cos(\theta_1)\sin(\theta_1) & \frac{1}{2}r^2\sin^2(\theta_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & r\cos(\theta_m) & r\sin(\theta_m) & \frac{1}{2}r^2\cos^2(\theta_m) & r^2\cos(\theta_m)\sin(\theta_m) & \frac{1}{2}r^2\sin^2(\theta_m), \end{pmatrix}$$

i.e $A(\ell, a) = Q_\ell(q_a)$, $\ell = 0, \dots, 5$, $a = 0, \dots, m$. We know that if

$$C = (A^TA)^{-1}A^T \in \mathbf{M}_{6\times m+1}(\mathbb{R}),$$

then $(c_0, \dots, c_5) = C(p_0, \dots, p_m)^T$. Note that, in this case, we are not using the weight matrix W, since the points are at the same distance from v_k. This choice turned out to be more robust in our experiments, although it seems interesting to investigate how other options may affect or ameliorate the accuracy of the method. It is important to point out that the technique used to sample the points ensure that, if $m \geq 5$, then A has full rank, and hence $A^T A$ is always invertible.

We can thus define our local parametrization around v_k as

$$x(u, v) = \sum_{\ell=0}^{5} c_\ell Q_\ell(u, v).$$

Moreover, given discrete function f sampled at the vertices of M, we can consider the values $f(p_a)$, obtained by linear interpolation, $a = 0, \dots, m$. Similarly to what done in the LSSD algorithm, by considering the second order Taylor expansion of f at v_k, we can define $f(u, v) = \sum_{\ell=0}^{5} d_\ell Q_\ell(u, v)$, where $d_\ell = C(f_0, \dots, f_m)^T$. A straightforward computation shows that, if we denote with C_1, C_2, C_{11}, C_{12} and C_{22} the second, third, fourth, fifth and sixth rows of C, respectively, then at $(0, 0)$ the following identities hold

$$\begin{aligned} x_{u^i} &= C_i(p_0, \dots, p_m)^T, & i &= 0, 1 \\ f_{u^i} &= C_i(f_0, \dots, f_m)^T, & i &= 0, 1 \\ x_{u^i u^j} &= C_{ij}(p_0, \dots, p_m)^T, & 0 \leq i &\leq j \leq 1 \\ f_{u^i u^j} &= C_{ij}(f_0, \dots, f_m)^T, & 0 \leq i &\leq j \leq 1 \end{aligned} \tag{3.14}$$

By substituting (3.14) into (3.9), we get an approximation of the gradient operator as follows

$$\nabla f(v_k) \approx \sum_{a=0}^{m} w_a f(p_a), \qquad w_a = g_u^\nabla c_1^{(a)} + g_v^\nabla c_2^{(a)}, \tag{3.15}$$

where $c_i^{(a)}$ is the a-th component of C_i. We can thus build a sparse matrix $\mathcal{G} \in \mathbf{M}_{3n \times n}(\mathbb{R})$ such that, if $F = (f_1, \dots, f_n)^T$ are the discrete values of f sampled at the n vertices of M, then

$$\begin{pmatrix} \nabla f(v_1)^x \\ \vdots \\ \nabla f(v_n)^x \\ \nabla f(v_1)^y \\ \vdots \\ \nabla f(v_n)^y \\ \nabla f(v_1)^z \\ \vdots \\ \nabla f(v_n)^z \end{pmatrix} \approx \mathcal{G} F,$$

where $\nabla f(v_k) = (\nabla f(v_k)^x, \nabla f(v_k)^y, \nabla f(v_k)^z)$, $k = 1, \dots, n$. However, in order to do that, we need to distribute the contribution of every point p_a in (3.15), since the p_a's are mesh point and not vertices. In details, by denoting with $\{v_{a_{\ell_0}}, v_{a_{\ell_1}}, v_{a_{\ell_2}}\}$ the vertices of the triangle t_a containing p_a, $a = 0, \dots, k$, we denote its barycentric coordinates in t_a as $\lambda_{a_{\ell_0}}, \lambda_{a_{\ell_1}}, \lambda_{a_{\ell_2}}$, so that

$$p_a = \lambda_{a_{\ell_0}} v_{a_{\ell_0}} + \lambda_{a_{\ell_1}} v_{a_{\ell_1}} + \lambda_{a_{\ell_2}} v_{a_{\ell_2}}, \qquad a = 0 \dots, k.$$

By assuming the piecewise linearity of f within the triangles of M, (3.15) may be re-written as

$$\nabla f(v_k) \approx \sum_{h=1}^{n} \sum_{a=0}^{m} \bar{w}_{a_h} f(v_{a_h}), \qquad \bar{w}_{a_h} = \begin{cases} (0,0,0) & v_{a_h} \notin t_a \\ \lambda_{a_h} w_a & v_{a_h} \in t_a \end{cases}$$

So far, we did not express the dependence from v_k of the $\bar{w}_{a_i}$'s in order to keep a notation as light as possible. However, since the points $\{p_1, \dots, p_m\}$ obviously depend on v_k, we shall now make such dependence explicit by writing $\bar{w}^k_{a_i}$. Moreover, since $\bar{w}^k_{a_i}$ is a vector in $\mathbb{R}^3$, we will write $\bar{w}^k_{a_i} = (\bar{w}^{k^x}_{a_i}, \bar{w}^{k^y}_{a_i}, \bar{w}^{k^z}_{a_i})$ We can therefore define $\mathcal{G}$ as the sparse matrix such that

$$\begin{aligned} \mathcal{G}(i,j) &= \sum_{a=0}^{m} \bar{w}^{i^x}_{a_j} \\ \mathcal{G}(ni,j) &= \sum_{a=0}^{m} \bar{w}^{i^y}_{a_j} \\ \mathcal{G}(2ni,j) &= \sum_{a=0}^{m} \bar{w}^{i^z}_{a_j}. \end{aligned}$$

The sparsity of $\mathcal{G}$ is due to the fact that the only vertices contributing to the estimation of $\nabla f(v_k)$ are the ones of the triangles containing the points $\{p_1, \dots, p_m\}$.

For the sake of brevity, we will not enter into the details about state-of-the-art methods for the computation of the Laplacian on discrete surfaces, referring to the work of Wardetzky et al. [WMKG07] in which the variety of discrete Laplace operators are described. Essentially, a discrete Laplace operator is described by its linear action on vertex based functions:

$$(Lf)_i = \sum_j \omega_{ij}(f_j - f_i),$$

where usually j varies in the set of indices of the vertices in the 1-ring of v_i. Among the various discretizations of the Laplace operator, the most popular is the so called *cotangent* Laplacian, introduced in [Mac49]. Since the weights ω_{ij} in the cotangent Laplacian are estimated measuring the cotangent of the angles opposite to the edge e_{ij}, the robustness of this method depends on the tessellation of M (see for example [SSC19b, Section 5.4]). This has been further confirmed in our experience while working on the cut locus (Chapter 5), where an accurate estimation of the Laplacian was crucial for our purposes. Our implementation turned out to be more resilient to the tessellation of the mesh (see Section 5.4).

We are now ready to define our discrete Laplace operator. Similarly to what done with gradient, by substituting (3.14) in (3.12) we obtain

$$\Delta f(v_k) \approx \sum_{a=0}^{m} \omega_a f(p_a), \tag{3.16}$$

where $\omega_a = g_u^{\Delta} c_1^{(a)} + g_v^{\Delta} c_2^{(a)} + g_{uu}^{\Delta} c_{11}^{(a)} + g_{uv}^{\Delta} c_{12}^{(a)} + g_{vv} c_{22}^{(a)}$ and $c_{ij}^{(a)}$ is the a-th component of C_{ij}.

As before, we can build a sparse matrix $L \in \mathbf{M}_{n\times n}(\mathbb{R})$ such that

$$\begin{pmatrix} \Delta f(v_1) \\ \vdots \\ \Delta f(v_n) \end{pmatrix} \approx LF.$$

Of course, also in this case we need to suitably distribute the contribution at the vertices of M. Therefore, with the notations consistent with above, we will have that L will be such that

$$L(i,j) = \sum_{a=0}^{m} \bar{\omega}_{a_j}^{i}, \qquad \bar{\omega}_{a_j}^{i} = \begin{cases} 0 & v_{a_j} \notin t_a \\ \lambda_{a_j} \omega_a^i & v_{a_j} \in t_a \end{cases}.$$

Concerning the Hessian operator, the advantage of Eq. (3.13) is that when considering a vertex $v_k \in M$, by (3.14) we can write

$$\mathcal{H}_i^{\ell}(v_k) \approx g^{\ell m}\big(C_{i+1m} - g_{jim} g^{jh} C_h\big)(f_0, \dots, f_m)^T,$$

where the dependence from v_k is just to keep in mind that $\mathcal{H}_i^{\ell}(v_k)$ maps vectors from $T_{v_k}M$ into vectors of $T_{v_k}M$. This allows us to store a matrix $\mathcal{H} \in \mathbf{M}_{4n\times n}\mathbb{R}$ such that

$$\mathcal{H}F = H,$$

where H is a $4n \times 1$ vector having at the k-th, $2k$-th, $3k$-th and $4k$-th entry the value $\mathcal{H}_0^0(v_k)$, $\mathcal{H}_0^1(v_k)$, $\mathcal{H}_1^0(v_k)$ and $\mathcal{H}_1^1(v_k)$, respectively, $k = 1, \dots, n$. For the sake of brevity, we omit the details about how constructing $\mathcal{H}$, since the procedure is similar to what seen for the gradient and the Laplacian.

3.2.3 Discussion

Even if it is a minor change with respect to what proposed by Xu in [Xu13], the uniform sampling $\{p_1, \dots, p_m\}$ is of paramount importance for the robustness of the algorithms described above. In fact, relying on the connectivity of the mesh to sample the signal lead to similar problems discussed in Section 3.2.1: the accuracy of the estimation at v_k heavily depends on the geometry of its neighborhood, and, according to the directions such neighborhood spans, we may lack important pieces of information. Moreover, this problem becomes more evident near singularities. On the other hand, by using the 1-ring of a vertex, one does not need to assume piecewise linearity of the signal inside the triangle, since its partial derivatives are estimated using a second-order version of LSSD (Section 3.2.1), while in our case we basically only values obtained through linear interpolation of the input function within the triangles of the mesh. Nevertheless, the experiments made so far suggest that it is better to have a less accurate information, but uniformly sampled around a point, rather than working with the exact values of the function but sampled according to the geometry of the mesh. Figure 3.7 show two examples. In both examples, the input signal is the distance function d_x sourced at some point $x \in M$, where

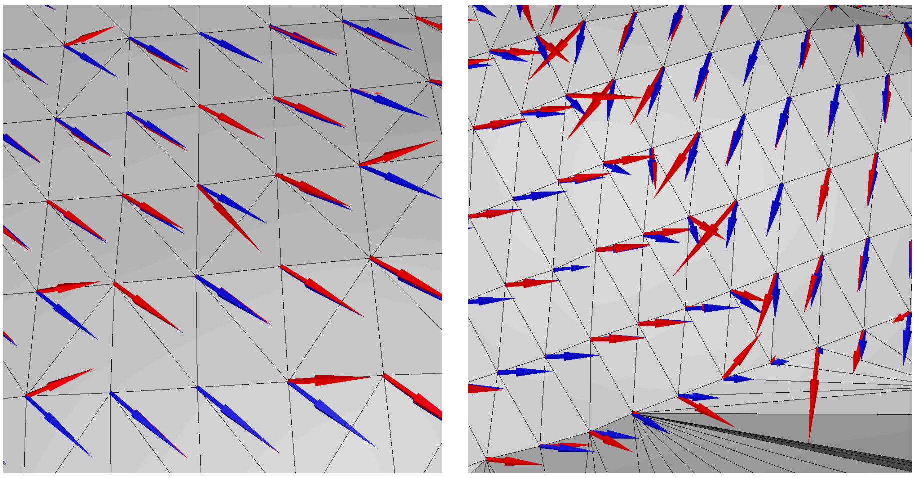


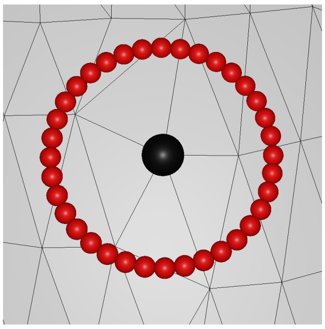
Figure 3.7: The estimation of the gradient of a distance field d_x using the approach proposed in [Xu13] (red) and the one described in the previous section(blue). Even in regions far from the cut locus, estimating the gradient by relying on the connectivity of the mesh may give poor results (*left*). In proximity of the cut locus, this behavior is more evident (*right*).

M, in this particular case, is the Stanford bunny[1]. The gradient estimated with our approach is depicted with blue arrows, while red arrows are used for the gradient computed with the algorithm proposed in [Xu13]. Estimating the gradient using the 1-ring (or the 2-ring in case the matrix C is singular), may lead to a poor estimation of the gradient at several vertices. Interestingly enough, the error is more evident when the vertex at which we are estimating the gradient has a dual edge that points towards the direction of steepest ascent of d_x, consistently with what observed in the planar case (see Figure 3.5). This kind of problem is more evident while approaching the cut locus. Even if the point of the cut locus are not critical points of the d_x, since d_x at those point is not even differentiable, the estimation of the gradient at the vertices around $C(x)$ seems always to be poor when using stencils based on the geometry of the mesh. In Figure 3.7(*right*), we can see two fronts of ∇d_x meeting at $C(x)$. Even if our method is *not* the groundtruth, the behavior seems consistent with what should happen in the continuous setting: since d_x is smooth away from $C(x)$, which has Hausdorff dimension 1, one expects to see the two fronts of the gradient to "break" at just one polyline across some vertices at which the discontinuity is evident (which is what happens for the blue arrows). When looking at the red arrows instead, we see that the discontinuities around $C(x)$ are less localized. We think that this happens because using a stencil whose geometry is dictated by the one of M around a vertex inhibits the possibility of catching the variations to which d_x undergoes around $C(x)$, similar to what observed with *PTE*, *AGS* and *LSSD* near critical points.

Concerning our approach, we estimated the gradient by sampling $m = 36$ point and by choosing r as the average edge length of the 1-ring of v_k (see

[1] Courtesy of the Stanford University Computer Graphics Laboratory

inset).



This choice gave good results in all our experiments, although a thorough analysis to assess the optimal values for m and r has not been carried out yet. For example, it is reasonable to think that, on highly anisotropic meshes, choosing r as the average edge length may lead to stencils that are too big, in the sense that, in some directions, we may consider points lying on triangles having vertices that do not belong to the 1-ring of the vertex we are considering. Nevertheless, also the Laplace operator defined with this technique turned out to be more robust than the cotangent Laplacian, as briefly discussed in Section 5.5. On this regard, it seems interesting to compare our discrete Laplace operator with the *diamond Laplacian* proposed by Bunge et al. in [BBA21], which, in a nutshell, discretize the Laplace operator using a more balanced stencil than the cotangent Laplacian.

In the future, we plan to perform a thorough analysis in the spirit of [MLP19], considering surfaces meshes and extending the study to second order differential operators.

4

Geodesic Paths and Distances

Most of the algorithms presented in this thesis have been designed and implemented with one *conditio sine qua non*: efficiency on highly-tessellated meshes. This means that such algorithms are required to be compatible with real-time interaction on complex meshes (few millions of triangles), i.e. to be able to produce the output in less than $0.1s$. In every case, the most expensive operations, which need particular care in order to achieve such a goal, are the computations of geodesic paths and distances. In this chapter, we will describe how such queries can be answered in the discrete setting. Differently from what has been seen in the smooth setting, where geodesics are both auto-parallel (i.e. straight) and locally shortest, this is not always true on meshes. For this reason, the algorithms required to trace *straightest geodesics* will be different from the ones that compute *shortest geodesics*. We will also describe various techniques to compute a geodesic distance field on M, i.e. scalar function defined at the vertices of M that associates to every vertex its (geodesic) distance from the source, which may consist either in a point or a set of points on M.

Broadly speaking, there exists two major classes of methods that address the problems mentioned above, which differ on how the mesh M is considered. In one case, M is viewed as an exact description of the geometry, hence the domain is polyhedral; while in the other case, M represents an approximation of a smooth surface. Neither class of approaches is to be preferred a priori, since each may be best-suited to a particular task and in a particular setting according to a variety of trade offs in terms of accuracy, storage cost, run time performances, and scalability. We will therefore describe the main algorithms present in the literature to address such problems keeping into account the different setting in which they solve them. We will then conclude by presenting the approaches used in our implementations, which have been obtained by mixing and/or modifying well known state-of-the-art methods, in order to obtain an optimal trade-off between accuracy and efficiency.

For the sake of brevity, the description of the state of the art will focus on the ideas behind the main methods, entering into details only when needed for the description of our algorithms. For any further details about this subject, we refer to the surveys [CLPQ20, BMSW11].

4.1 PDE-based Methods

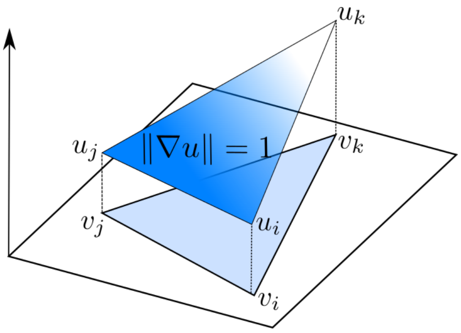
Many methods for computing geodesic distances are based on formulating the problem in terms of partial differential equations (PDEs) on a smooth manifold $\mathcal{M}$. By discretizing and solving these PDEs, a solution on a mesh M representing the smooth surface can be found. These methods are usually suitable to compute the geodesic distance field from a point or a set of points. To explicitly trace a shortest geodesic between two points, one needs to exploit the knowledge of the geodesic distance field sourced at one of them and, for example, trace a suited integral curve of the gradient field of such function. In fact, by Proposition 2.37 we know that the integral curves of the vector field on $\mathcal{M}$ ∇d_p are geodesic emanated from $p \in \mathcal{M}$. On this regard, we have also seen that $\|\nabla d_p(q)\| = 1$ for every $q \in \mathcal{M} \setminus \{C(p) \cup \{p\}\}$. Roughly speaking, all PDEs methods leverage on this fact, since they are all trying to minimize the residual of the eikonal equation, defined as

$$\begin{aligned} \|\nabla u(x)\| &= 1 \qquad x \in \Omega \\ u(x) &= 0 \qquad x \in \partial\Omega. \end{aligned} \tag{4.1}$$

By identifying Ω with $\mathcal{M}$, the solution u of the above equation will be a scalar function u on $\mathcal{M}$ such at any point of $\partial\mathcal{M}$, which can be thought as the source set, u should be zero, while at every other point the gradient of u should equate one, just as the geodesic distance field sourced at $\partial\mathcal{M}$.

PDE-based methods can be categorized according to the type of equation considered when formulating the problem in the smooth setting. At a high level, there are two basic classes of methods: *wavefront-based* and *diffusion-based*. We will describe only one method for each category and refer the reader to [CLPQ20] for further details on this type of methods.

Fast Marching



The idea behind the *fast marching method* for triangle meshes [KS98] is to solve the eikonal equation (4.1) by propagating a front starting from the source. In fact, since solving the eikonal equation using standard linear FEM is far from being trivial due to its non-linearity, the approach used consists in iteratively updating the solution while the front is propagated. Such a strategy is very similar to Dijkstra's algorithm, which can be described as follows. We start by putting the distance at the source set to zero, and to infinity anywhere else. We then use a region-growing strategy to update the remaining distances in an "upwind" order, i.e. we consider the node with smallest distance first. The fast marching method proceeds similarly. The key difference is that distances are not updated according to paths along arcs, but by solving an equation that approximates through finite differences the eikonal equation. If the values u_i, u_j of two vertices v_i, v_j are known, one picks a third value u_k for the third vertex v_k such that the gradient of u inside the triangle computed with PTE has unit norm.

Heat Method

The *heat method* has been introduced in [CWW13] and exploits a relationship between the geodesic distance function d and the short-time heat kernel $k_t(x, y)$, where the latter measures the heat transferred from a source x to a destination y after time t. Such a relationship is dictated by the *Varadhan's formula* [Var67]:

$$d(x, y) = \lim_{t \to 0} \sqrt{-4t \log k_t(x, y)}.$$

Such formula can be interpreted as follows: if a point-wise source of heat centered at point x diffuses for a very short time t, then the resulting heat distribution looks nearly identical to the geodesic distance function, up to a simple transformation of the value at each point. Since computing a numerical approximation u_t of the heat kernel such that the Varadhan's transformation $u_t \mapsto \sqrt{-4t \log u_t}$ yields to a good approximation of $d(x, y)$ is quite challenging, the idea behind the heat method is to leverage the knowledge of the magnitude of the gradient of the distance function, and estimating its direction by solving the heat equation

$$\frac{d}{dt} u_t = \Delta u_t \tag{4.2}$$

$$u_0 = \delta_x, \tag{4.3}$$

where δ_x is a Dirac delta centered at x. Therefore, the pipeline of the heat method can be described as follows. First, it determines an approximation u_t of the heat kernel by solving the heat equation above. Define the vector field $X = -\nabla u_t / \|\nabla u_t\|$, where the minus sign accounts for the fact that d increases when moving away from x while the heat kernel does the opposite. Identify d with minimizer of $\int_M \|\nabla d - X\|^2$, or, equivalently, by solving the corresponding Euler-Lagrange equations $\Delta d = \nabla \cdot X$.

Concerning performances, the advantage of the heat method with respect to the fast marching method is that the matrices used to solve linear systems needed to approximate the heat kernel and to solve the Euler-Lagrange equation can be prefactored. This means that the computation of a geodesic distance field consists in two steps: a pre-processing phase in which a sparse Cholesky factorization [CDHR08] of the matrices involved in the computation is built, and the effective computation of the distance field by solving the above mentioned linear systems. Note that the pre-processing phase depends only on the geometry of the mesh M, so the query of another distance field sourced at some other point of M can be answered very efficiently once the matrix has been pre-factorized. On the other hand, the fast marching method needs to propagate a new front every time the source is relocated, so it is outperformed in terms of efficiency. In terms of accuracy, the results reported in [CWW13] show that fast marching tends to achieve smaller maximum error, whereas the heat method does better on average (see [CWW13, Table 1]).

4.2 Polyhedral methods

The methods described in this section are based on the assumption that M is an exact representation of the geometry, which means that M is viewed as a topological manifold endowed with a *piece-wise linear structure* [Cai35,Whi40]. In this class we have the so called *exact* methods that, roughly speaking, define the distance between two points just as in the smooth case, i.e. as the length of the shortest curve joining them, which, by the piecewise nature of the mesh, is a polyline made of straight line segments. The idea on which most of the methods that fall in this group rely on, is due to Mitchell et al. [MMP87], in which a characterization of what is a shortest path (or a minimizing geodesic) in this setting is given. The MMP (Mitchell-Mount-Papadimitriou) algorithm proposed in this paper is commonly viewed as a landmark in the research of polyhedral geodesic algorithms, and many subsequent approaches have proposed methods to increase its efficiency and robustness without violating the constraint that the final result should be the *exact* polyhedral distance. However, these methods can be time-consuming for large-scale applications, so several algorithms have been proposed that approximate the polyhedral distance, which can be applied in scenarios where the accuracy is not an essential feature. In this class fall the *graph-based* and *local* methods, which will both play an important role in this thesis.

Differently from PDE-based methods, polyhedral methods are well suited for geodesic paths tracing. In some cases, the technique can be space-consuming and not very useful from a practical point view when dealing with highly-tessellated meshes. However, we will see that such query can be answered efficiently by combining a graph-based approach with a local method.

4.2.1 Shortest and Straightest Paths on Triangle Meshes

As said above, polyhedral methods compute geodesic distances by considering curves of minimum length between points. Therefore, it is clear that a characterization of such curves is needed. The piece-wise flat structure of a triangle mesh M helps in giving an idea of what happens in some trivial cases. For example, if two points p and q lie in the same triangle t, then the shortest path connecting is a straight line segment within t. Similar arguments can be made if p and q belong to adjacent triangles: we can unfold the two triangle in a common plane, connect p and q with a straight line segment, and map the result on M. In general, if a shortest path does not cross any vertex, once the triangle crossed by such a path are flattened into a common plane, then it

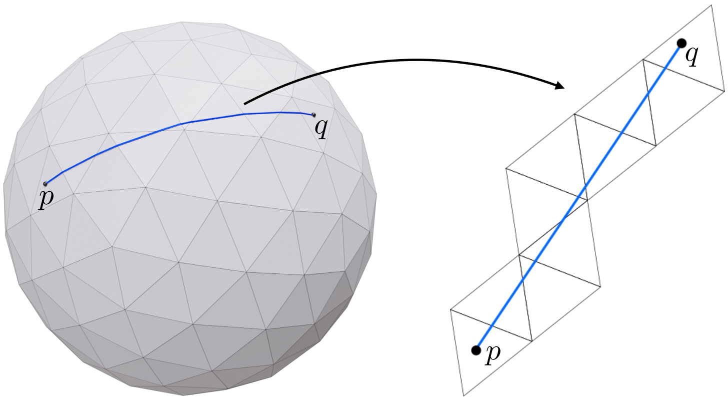
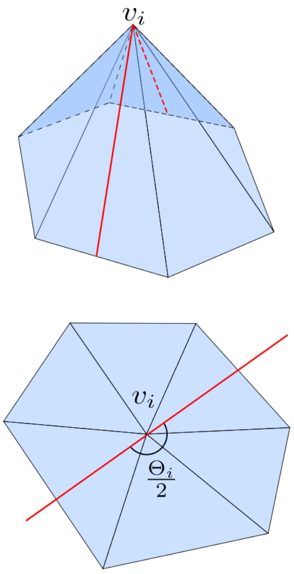


Figure 4.1: Due to the piecewise linear nature of the mesh, the shortest path between p and q is a polyline made of straight line segments, which reduces to a straight line once the strip of triangles crossed by this path is flattened.

reduces to a straight line segment, as shown in Figure 4.1. Note that, in these cases, we obtain a path connecting the two points which are both straightest *and* shortest. However, when p and q get far away from each other, things get more complicated.

Straightest Paths



Let $p \in M$ be a point in M and let u be a direction in T_pM. For the moment, suppose that p is not a vertex. We are interested in tracing the straightest possible path from p with direction u. It seems reasonable to start by tracing the straight line segment with direction u until we hit an edge of the triangle containing p, and then unfold the neighboring triangles to extend this ray indefinitely. Here we are implicitly leveraging the fact that the tangent space of a point q in a triangle is identified with the plane that contains this triangle, so it is natural to ask ourselves what happens when the said ray hits a vertex v_i of M. In that case, the idea proposed by Polthier and Schmies in [PS98] is, roughly speaking, to impose the path to be “straight” in the tangent space $T_{v_i}M$. More formally, if $(1, \theta_u)$ are the polar coordinates of the incoming direction in $T_{v_i}M$, then we impose the outgoing direction to have coordinates $(1, \pi+\theta_u)$. This is equivalent to require that the incoming and outgoing directions splits the total angle Θ_i in half, where Θ_i is defined as in (3.1), see inset.

Discrete Curvature

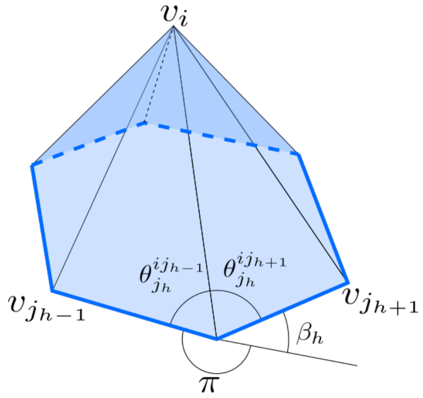
Before describing what characterizes a shortest path on M, it is convenient to introduce a way of measuring the "flatness" of a vertex $v_i \in M$. We thus define the *angle defect* Ω_i as

$$\Omega_i = 2\pi - \Theta_i,$$

The angle is often viewed as a discrete analogue of the Gaussian curvature. In fact, if we put $R := \overline{\mathcal{S}(v_i)}$, then $\partial R = Lk(v_i)$ (thick blue line in the inset). Since the Gauss-Bonnet theorem holds also in this setting, we can write

$$\iint_R K d\sigma + \sum_{h=0}^{m} \beta_h = 2\pi, \tag{4.4}$$

where $\{v_{j_0}, \dots, v_{j_m}\} = \mathcal{N}(v_i)$ and β_h is the external angle of $Lk(v_i)$ at v_{j_h}, $h = 0, \dots, m$ (see inset).



Note that this simplified form is due to the fact that the integral of geodesic curvature along the piecewise linear boundary of R is zero. With notations consistent with the ones of Section 3.1, and assuming that indexes are meant modulo m, we have that

$$\beta_h = \pi - (\theta_{j_h}^{ij_{h-1}} + \theta_{j_h}^{ij_{h+1}}).$$

Therefore, when summing over h, we can re-arrange the terms and write

$$\sum_{h=0}^{m} \beta_h = \sum_{h=0}^{m} \pi - (\theta_{j_{h-1}}^{ij_h} + \theta_{j_h}^{ij_{h-1}}) = \sum_{h} \theta_i^{j_h j_{h-1}} = \Theta_i,$$

where again the indices are meant modulo m. Therefore, (4.4) becomes

$$\iint_R K d\sigma = 2\pi - \Theta_i.$$

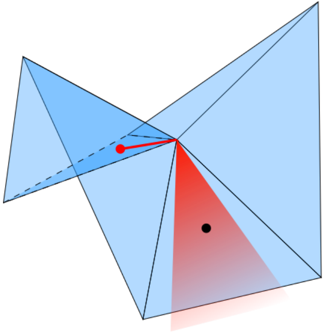
We will therefore refer to vertex with positive, zero and negative curvature (or angle defect) as *spherical* (or *cone-like*), *Euclidean* and *hyperbolic* (or *saddle-like*), respectively.

Shortest Path

Let us now consider two points $p, q \in M$ directly opposite a spherical vertex v_i (i.e. a vertex at which the curvature is positive). What we said about straightest path, suggests that the shortest route between p and q is to consider the straightest path from p to v_i and from v_i to q. However, this is wrong, since if we think at v_i as a bump, or a hill, one can find a shorter path by walking "around" it rather than walking over it. In fact, Sharir and Schorr proved in [SS84] that, on a convex mesh, i.e. a mesh M such that every vertex $v_i \in M$ is such that $\Omega_i > 0$, a shortest path never passes through a vertex.

On the other hand, if v_i is a saddle vertex ($\Omega_i < 0$), then one can find many shortest paths passing through it. In fact, as we thought about cone-like vertices as hills, here we may think about hyperbolic ones as passages between two mountains: in order to reach the other side, it is more cost-effective to go through the passage than crossing the ridge of one of the mountains. In fact, suppose that you want to find the shortest path connecting the red point with the black point in the inset. It is clear that it is much quicker to pass straight through the vertex than to walk up and down across the nearby triangles.

However, the big difference when considering the discrete setting, is that, in this case, the passage "collapses" into just one vertex, so *all* the shortest paths that connect two points at the opposite side of a hyperbolic vertex will pass through it. In fact, if we keep the red point fixed, there will be infinitely many outgoing directions that yield a shortest path; these directions form a wedge-like region of angle $| \Omega_i |$ (red).



The above considerations have been formalized by Mitchell et al. in [MMP87], in which they give a full characterization of shortest paths in the polyhedral setting, which is summarized in the following lemma [MMP87, Lemma 3.5].

LEMMA 4.1 : The general form of a geodesic path is a path which goes through an alternating sequence of vertices and (possibly) empty edge sequences such that the unfolded image of the path along any edge sequence is a straight line segment and the angle of the path passing through a vertex is greater than or equal π. The general form of an optimal path is the same as that of a geodesic path, except that no edge can appear in more than one edge sequence and each edge sequence must be simple.

In the above lemma, "optimal path" stands for shortest paths, and an edge sequence is called *simple* when no edges appear more than once in it. The condition on the angle of the path passing through a vertex exclude spherical vertices by the definition. In fact, suppose that a path γ passes through a vertex v_i while going from the triangle t_0 to the triangle t_1. Let ℓ_j be the segment of γ on t_j leading to v, $j = 0, 1$. Then γ splits the star of v_i in two portions: the first is the strip of adjacent triangles $\{t_0, \dots, t_1\}$ that goes from t_0 to t_1 moving around v_i in a clockwise order, the other goes from t_0 to t_1 in a counterclockwise order. Let $\{v_{a_0}, \dots, v_{a_m}\}$ be the vertices in $\mathcal{N}(v_i)$ that belong to the first portion and $\{v_{b_0}, \dots, v_{b_m}\}$ the ones in the second portion. Let ϕ_a^0 and ϕ_b^0 be the angles formed by ℓ_0 with e_{ia_0} and e_{ib_0}, respectively. Similarly, let ϕ_a^1 and ϕ_b^1 be the angles formed by ℓ_1 with $e_{ia_{m-1}}$ and $e_{ib_{m-1}}$. We define

$$\Theta_i^+ := \phi_a^0 + \sum_{j=1}^{m-1} \theta_i^{a_j a_{j+1}} + \phi_a^1, \qquad \Theta_i^- := \phi_b^0 + \sum_{j=1}^{m} \theta_i^{b_j b_{j+1}} + \phi_b^1 \tag{4.5}$$

as the total angles of these two portions. Note that the summations go from one to $m - 1$ because we are measuring the clockwise and the counterclockwise

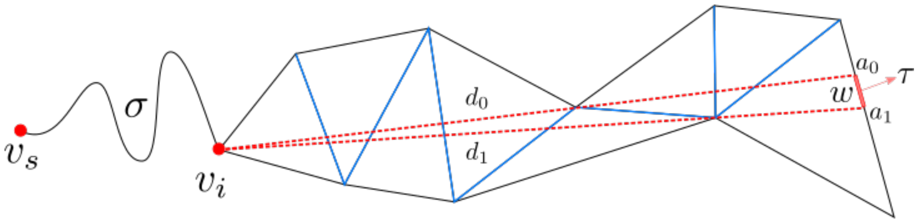


Figure 4.2: The window is data structure $w = (a_0, a_1, d_0, d_1, \sigma, \tau)$, where a_0, a_1 denote the endpoints of w; d_0, d_1 denote the corresponding distances from a_0, a_1 to the pseudosource v_i; σ denotes the geodesic distance from v_i to the source v_s; and τ denotes the direction of the propagation. Adapted from [SSK+05].

angles from ℓ_0 to ℓ_1. Then the angle of γ at v_i is defined as the minimum between Θ_i^+ and Θ_i^-. Note that, if $\Omega_i > 0$, then such an angle is always smaller than π.

4.2.2 Exact Methods

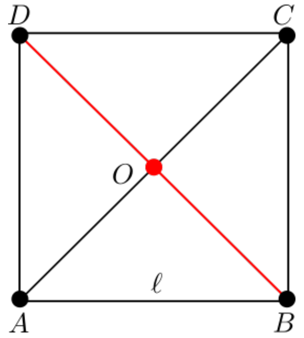
Mitchell et al. [MMP87] proposed the first practical algorithm for geodesic computation on polyhedral surfaces, which is commonly referred to as MMP (Mitchell-Mount-Papadimitriou) algorithm. The main idea is to use the characterization of a shortest path made in Lemma 4.1 to propagate a front from the source with a technique that the authors call *continuous Dijkstra*. In fact, this procedure somehow extends the well known Dijkstra's algorithm [Dij59] from graphs to polyhedral surfaces, since it views the edges of the mesh as nodes of a graph. Instead of distance values, they use a dedicated data structure, called *window*, which encodes all locally shortest paths in an unfolded triangle strip. Essentially, windows are intervals on edges that a shortest path may cross. Whenever the front hits a hyperbolic vertex v_i, then a new front is propagated from v_i, which has been called in the following works a *pseudosource*. In this case, the propagation of the windows starts again from v_i as if it were the original source, hence the name. This not essential but for technical reasons it makes the algorithm faster. It may happen though, that the propagations started from two pseudosources "intersect", i.e. there may be edges where windows overlap. In this case, the optimization of windows on an edge is accomplished by trimming such windows into disjoint ones according to the smaller distance in the overlapping part. Since the cost of the algorithm is positively correlated to the number of windows arriving at each edge, in order to minimize this number, the wave front propagation paradigm in Dijkstra's algorithm and fast marching is used. This means that windows across M are propagated from near to far by maintaining a priority queue. This paradigm ensures that redundant windows will be trimmed at earliest possible stage. Then, the geodesic distances can be computed by finding the optimal windows on the edges of M. If x is hyperbolic vertex, then its distance from the source has been labeled during the propagation, since it is a pseudosource, otherwise the distance is retrieved in $O(\log n)$, where n denotes the number of vertices of M. Since the authors proved that at most $O(n^2)$ windows are created by this algorithm, it can be easily derived that MMP computes the geodesic distance from a vertex v to all the other vertices of M in $O(n^2 \log n)$.

The MMP algorithm is commonly viewed as landmark in the research of polyhedral geodesic algorithms. Its distinct distribution is the *window propagation framework* which contains three major components: window propagation, window pruning (e.g. trimming) and window management (e.g. priority queue). Several approaches in the literature have adopted this framework in order to propose an algorithm that computes exact geodesic polyhedral distances in the most efficient possible, and they differ from their unique techniques used in the three components. For the sake of brevity, we report the main ones [CH90,SSK+05,XW09,XWL+15,QHY+16] and refer the reader to [CLPQ20] for further details. In a nutshell, the best performing algorithm to date is the VTP algorithm (Vertex-oriented Triangle Propagation), proposed by Qin et al. [QHY+16].

4.2.3 Graph-Based methods

Even the most efficient method for computing exact polyhedral distances is far from being compatible with real-time performances on average-sized meshes. Therefore, one needs to find a trade off between accuracy and efficiency, which means giving up on computing the exact distance between two points and relying on more efficient approaches that provide an estimation of it. *Graph-based* methods rely on the assumption that the shortest geodesic distance between any pair of point p, q can be approximated by considering a set $(p, v_0, \dots, v_k, q)$, where $v_0 \dots, v_k$ belong to a finite set V_G of points on M, such that the distance between any pair of points of V_G is precomputed and stored in the edges E_G of a graph $G = (V_G, E_g)$.

To fix ideas, suppose M is made of two triangles obtained by splitting the square A, B, C, D by connecting A and C with one of the diagonal of the square (see inset). Now suppose that we want to estimate the distance between B and D.



By allowing ourselves to move just along the edges of M (black lines), the best we can do is to approximate such distance with 2ℓ, where ℓ is the length of the side of the square. However, by considering the graph $G = (V_G, E_G)$, such that

$$V_G = \{A, B, C, D, O\}$$
$$E_G = (AB, BC, CD, DA, AC, OB, OD),$$

where O is the centroid of square, we can “go” from B to D by “walking” along BO and OD (red), i.e. we obtain the correct distance $\sqrt{2}\ell$. The idea behind graph-based method is to refine the initial mesh M by adding *nodes* and connecting them with *arcs* in order to improve the estimation of the distances that we would have if we allow ourselves to move just along the edges of M.

This idea has been probably proposed for the first time in the PhD thesis of Lanthier [Lan00]. Such methods differ for the choice of points of V_G and the edges of E_G and the strategy to build the graph. Once G is provided, geodesic distances and shortest paths can be easily resolved though shortest path queries on G, most frequently with standard Dijkstra search [Dij59]. In [Lan97, LMRS01], Lanthier et al. proposed three different strategies to build a graph G made of edges that can cross the triangles of M. The first step consists in defining the vertices V_G of G by adding *Steiner points* along the edges of M, while in the second one the graph of edges E_G is built, which interconnects the vertices of V_G with arcs that walk either along edges or across faces. The three above mentioned techniques differ in how the Steiner points are distributed along the edges (same number of points distributed uniformly on all the edges or different number of points such that the distance between them on all the edges is the same), and in the way in which they are connected, i.e. by which arcs one add to E_G. Lanthier et al. [LMRS01] proved that if the number of Steiner points is large enough, they can approximate the geodesic shortest path within additive bound that is a function of the length of the longest edge in M.

Several other techniques have been proposed that deal with the positioning of Steiner points and with the accuracy of the resulting method [MM97, ALM00, AMS00, AMS05]. However, the results in these works have mostly a theoretical interest, since the proposed algorithms are too slow for practical purposes [BMSW11].

Ying et al. [YWH13] proposed a graph based method base on the observation made in Lemma 4.1: since a shortest path on a polyhedral mesh may cross a vertex only if it is a hyperbolic one, they proposed to build a *Saddle Vertex Graph* (SVG) having the vertices of M as nodes, and one arc in E_G for any *direct* path between a pair of vertices, where a path is said to be direct if it does not cross any vertex. Therefore, in this case the arcs are polygonal paths across edges. However, since encoding all the direct paths between any pair of vertices may result in a too large data structure for practical purposes, they proposed to connect each vertex v_i with the K vertices belonging to a geodesic disc centered at v_i. They compared SVG with the heat method and report that SVG is faster for $K < 100$, comparable for $100 < K < 500$ and slower, but more accurate for larger values of K.

Wang et al. [WFW$^+$17] improved the efficiency of SVG proposing the *Discrete Geodesic Graph*, in which they relax the hypothesis that no shortest path should cross a spherical vertex. They start by observing that cones resulting from the window propagation in the MMP algorithm become progressively narrower. If a cone is long enough, then any direct path from the source v_s to a vertex v_t in the cone can be approximated by the sum of two shortest paths v_sv and vv_t, where v is a vertex preceding v_t and lying on the boundary of the cone. Therefore, while generating direct paths as in the SVG method, they set an early termination of window propagation when the length of the cone reaches a certain threshold, which depends on the parameter defining a window W

that bounds the cone and on a tolerance threshold ε. The experiments carried out by the authors show that DGG is the best graph-based algorithm in terms of efficiency and accuracy to date.

4.2.4 Local Methods

The methods reviewed above are well suited for computing geodesic distances. Both exact methods and graph-based methods can be used to compute a shortest path but, as it will be clear soon enough, there are methods that are tailored for this type of query. Therefore, in this section we will focus on methods that compute the shortest path between two vertices v_s and v_t. The idea behind *local methods* is to compute an *initial guess* P, which is a path connecting v_s and v_t which is not needed to be the shortest one, and then refine P in order to turn it into a geodesic path. Note that, in the last statement we used the term "geodesic" and not "shortest", because these methods only guarantee the result to be *locally* shortest, i.e. it is not possible to assess if another globally shortest path exists. To clarify the previous statement, let us start by describing how to locally assess that a given path is geodesic or not. To this end, it is important to remember the discussion made in Section 4.2.1, where we observed that, in the discrete setting, the concepts of shortest and straightest do not coincide, in general. Polthier and Schmies [PS98] observed that a path passing through vertex v_i divides its total angle into two components (Θ_i^+ and Θ_i^- in (4.5)), and define geodesic paths in terms of these quantities. In particular, they state that a path is

- locally straightest if $\Theta_i^+ = \Theta_i^-$,
- locally shortest if $\Theta_i^+ \geq \pi$ and $\Theta_i^- \geq \pi$.

Note that if $\Omega_i = 0$, i.e. if v_i is Euclidean, both conditions hold and the parallelism with the smooth setting is preserved. However, we notice that there are infinitely many shortest paths passing through an hyperbolic vertex, i.e. any solution $\Theta_i^+ + \Theta_i^- = \Theta_i$ with $\Theta_i^+ \geq \pi$ and $\Theta_i^- \geq \pi$ defines a locally shortest path through v_i.

Many local methods in the literature exploit this classification to turn the initial guess P into a path which is locally shortest. Essentially, P is represented as a concatenation of mesh points $p_0, \dots, p_n$, such that p_i and p_{i+1} belong to the same triangle t, $i = 0, \dots, n$. The idea is to iteratively flatten the mesh around each one of these points and update their position in order to locally straighten or shorten the path. This procedure is iterated until convergence, that it, when all the points locally satisfy the geodesic criterion ([Wan04, MVdC04]). Xin et al. [XW07] strive for locally shortest geodesics, but rather than using the local angle criteria expressed in [PS98], they adopt an equivalent concept based on the Fermat principle, which states that light always follows the shortest optical path. As in [Wan04,XW07], this method is guaranteed to converge to a path that is locally shortest everywhere in the sense of [PS98]. Recently, Sharp and Crane [SC20a] proposed an algorithm that shortens the initial guess by flipping the edges of the mesh and updating the path accordingly to such flip. They proved that their local flipping procedure is guaranteed to reduce the length in the general case. The comparison made in Chapter 7 shows that our method is, on average, one order of magnitude faster than the one proposed in [SC20a].

Other methods in the literature address this problem by updating the all path at each iteration. In a nutshell, instead of focusing on one point, they move all the points in the path by minimizing a suited energy. Since we will not use this kind of approach in the following, we omit their description for the sake of brevity and refer to [CLPQ20] for further details.

4.2.5 Geodesic Tracing

With *geodesic tracing* we mean the tracing of a curve γ which is locally straigthest everywhere in the previously described sense. Therefore, we are interested in finding a piece-wise linear curve such that $\Theta_i^+ = \Theta_i^-$ whenever a vertex v_i is crossed. The theoretical bases for solving this problem on a polyhedral surface has been laid by Polthier and Schmies in [PS98], as described above. Polthier and Schmies [PS98] proposed two alternative methods to integrate a vector field on a mesh, one based on Euler integration, and the other based on the fourth order Runge-Kutta method. In the implementation of the algorithms described in the sequel, the former will always been used. We report the definition given by the author and describe how it has been implemented in our setting.

DEFINITION 4.2 ([PS98, Definition 21]): Let M be a polyhedral surface with a polyhedral tangent vector field v on M, let $y_0 \in M$ be an initial point, and let $h > 0$ a (possibly varying) step size. For each point $p \in M$, let $\delta(t, p, v(p))$ denote the unique straightest geodesic through p with initial direction $v(p)$ and evaluated at parameter value t. A single iteration step of the *geodesic Euler method* is given by

$$y_{i+1} := \delta(h, y_i, v(y_i)).$$

The produces a sequence of points $\{y_0, y_1, \dots\}$ on M which are connected by straightest geodesic segments of length h. For each $i\{0, 1, \dots\}$, we define

$$\gamma(ih + t) := \delta(t, y_i, v(y_i)), \qquad t \in [0, h]$$

and obtain a piece-wise straightest, continuous curve $\gamma : [0, \ell) \longrightarrow M$ of some length ℓ such that each segment $\gamma\big|_{[ih,(i+1)h]}$ is a straightest geodesic.

Note that, in the definition above, the existence of a tangential vector field on M is assumed. However, in our setting we always start with just one point p and a direction w. In this case, the authors state that such a vector field should be computed during a pre-processing step. On this regard, let us consider the case in which p lies within a triangle t_0, and hence w is a tangent vector defined in the plane containing t_0. The unique straightest geodesic satisfying these initial conditions is therefore the straight line segment emanating from

p with direction w. We therefore trace such segment until we hit the boundary of t_0 at some point p_0. Let us assume that p_0 is not a vertex, and let t_1 be the triangle adjacent to t_0 along the edge on which p_0 lies. We can then flatten t_0 and t_1 onto a common tangent plane, and extending the straight line segment defined in t_0 to t_1. If p_0 is a vertex instead, suppose $p_0 = v_i$ for some $v_i \in V$. We parallel transport w from t_0 to $T_{v_i}M$ as described in Section 3.1. With an abuse of notation, we will call such a vector w for simplicity, The polar coordinates (ρ_w, θ_w) in $T_{v_i}M$ uniquely determine the vertices v_{j_a} and $v_{j_{a+1}}$ such that $\tilde{\theta}_{j_a} \leq \theta_w \leq \tilde{\theta}_{j_{a+1}}$, where $\tilde{\theta}_{j_a}$ and $\tilde{\theta}_{j_{a+1}}$ are defined as in Section 3.1. We therefore parallel transport w from $T_{v_i}M$ to t_1. Again, the straightest geodesic in t_1 emanated from v_i with direction w is uniquely defined, we can therefore trace a straight line segment within t_1 and determine the next triangle t_2. In other words, we construct the tangential vector field in Definition 4.2 by parallel transporting the initial direction from cell to cell. Note that, if γ is the piece-wise linear curve consisting of all the straight line segments traced inside every triangle, then by construction we have that γ is a locally straigthest curve in the sense of [PS98]. In fact, within a triangle γ is just a straight line segment, and when moving from one triangle to another, the parallel transport of the direction ensures that γ remains "straight", in the sense of [PS98].

Although the above method is the most popular and used, it has some limitations. Suppose for example that we are interested in approximating the exponential map at some vertex v_i, and, to do that, we trace straightest geodesics in every possible direction from v_i. Suppose that one of them, say γ hits a spherical vertex v_j. Then γ splits the beam of nearby curves into two groups which meet discontinuously on the opposite side of v_j, since the tangent spaces we are considering to define them are different and, most importantly, it is not possible to smoothly pass from one to the other. Note that, if v_j is hyperbolic, then we also fail in covering the wedge of all possible outgoing directions of shortest paths beyond v_j. Kumar et al. [KSH$^+$03] proposed a method that uses the discrete normals of the mesh rather that tangent spaces to define stragihtest geodesics. This allows to trace beams of the curves that leave less "blind spots" when passing a vertex, as demonstrated in [BMBZ02]. In our implementation, we do not use such techniques since we do not have the same requirements of Biermann et al. [BMBZ02], and the method of Polthier and Schmies [PS98] perfectly fits our purposes.

Note that, at the best of our knowledge, none of the existing methods ensure to have consistency between shortest and straightest paths in the following sense. Suppose that to have a discrete exponential map constructed as above, i.e. by tracing a finite number of straightest paths from one vertex v_i. Besides the aforementioned problems regarding the injectivity of such mapping, the shortest paths connecting v_i with the points obtained in this way would not coincide with the geodesic rays emanated from v_i. It seems therefore interesting to investigate if there is way of defining a metric and a covariant derivative such that the curves having null geodesic curvature with respect to the said covariant derivative are also shortest paths with respect to such a metric. It seems reasonable to think that one needs a proper definition of a (discrete) covariant derivative, instead of relying on the one obtained through the parallel transport. On this regard, the work of Liu et al. [LTGD16] seems a good starting point to investigate this problem. In fact, straightest geodesics defined using the covariant derivative proposed in [LTGD16] would not be straight line segments within the triangles, since, roughly speaking, the tangent space of a triangle is not assumed to be just a plane, so one need a way to parallel transport vectors within it. Of course, in order to have consistency, one needs to define shortest path differently from what done above. We plan to investigate this possibility in future works.

4.3 Our approach

The methods described in Chapter 6 and 7 heavily rely on efficient algorithms to compute geodesic distance field and shortest paths. Besides efficiency, these algorithms need also be accurate, since otherwise the continuity and/or smoothness of the result would be affected. As an example, consider the geodesic ball in Figure 4.3. To trace it, we considered the isoline of the distance field d_p sourced at the red point p. Since d_p is defined at the vertices of the mesh, we linearly interpolate d_p along the edges in order to make the ball cross the edges and the triangles of the mesh. Since the mesh is made of 1 million of triangles, the linear interpolation of the field does not visibly affect the result, in the sense that triangles are small enough and the piece-wise linear nature of the curve is not perceived (see close up on the bottom). However, since the estimation of the distance field is not enough accurate, we obtain a wiggly geodesic circle, which is not acceptable.

Nevertheless, such circle has been traced in real-time, that is, once the center has been selected, by dragging the mouse the user can see in real-time the isoline corresponding to the position of the mouse. Therefore, the computation of d_p must support the tracing of geodesic balls via click-and-drag. Similar arguments apply to shortest paths tracing: the accuracy of the result is crucial in order to obtain a pleasant output and, in most cases, we need to trace many of them in real-time. In this section, we describe the approaches used to satisfy these requests.

4.3.1 Locally Shortest Paths Tracing

The method described below has been proposed in [MNPP22] and a comparison with the *FlipOut* algorithm of Sharp and Crane [SC20a] will be presented in Chapter 7.

Our algorithm to compute locally shortest geodesic paths is derived by combining insights from the work of Lee and Preparata [LP84] and Xin and Wang [XW07]. The algorithm consists of three phases: (i) extraction of an initial strip; (ii) shortest path in a strip; and (iii) strip straightening.

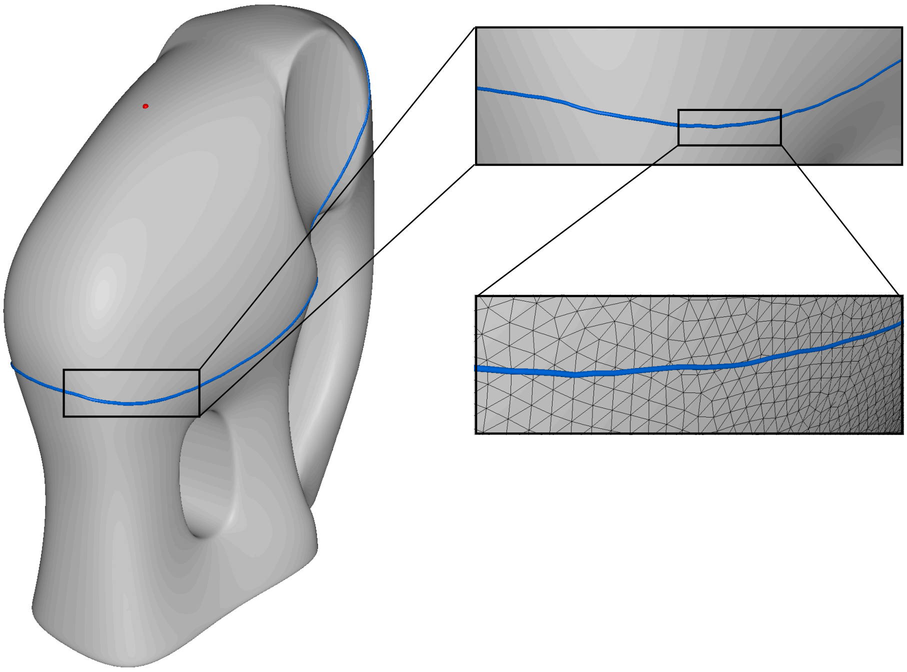


Figure 4.3: If the estimation of the distance field is not enough accurate, when computing its isoline we may obtain a wiggly geodesic circle, even on high-tessellated meshes.

Phase (i), which has been overlooked in previous approaches, is critical as it can become the bottleneck on large meshes (see, e.g., the discussion in [SC20a, Section 5.2.1]). Given two mesh points P and Q, we compute a strip of triangles that connects them, performing a search on a dual graph. This graph is defined on the mesh having as nodes the centroids of the triangles of M. Each node is connected with the centroids of the three adjacent triangles and each arc encodes the distance between two nodes, computed by flattening the triangles in a common plane. Since every node of this graph has valence 3, its navigation turns out to be very efficient.

We experienced a relevant speedup over the classical Dijkstra search by using a shortest path algorithm based on the small-label-first (SLF) and large-label-last (LLL) heuristics [Ber98], which do not require a priority queue, but a double ended queue. Instead, the SLF heuristics adds a new node to either the front, or the back of the queue, according to the estimated distance of that node, compared to the distance of the first node in the queue. The LLL heuristics moves a node from the front to the back of the queue if its distance is larger than the average distance of nodes in the queue. The SLF and LLL heuristics govern the insertion and extraction of weighted nodes in the queue. Besides, we weight each node as in a classical A* search, with the sum of its current distance from the source plus its Euclidean 3D distance to the target. This heuristic prioritizes the exploration of triangles closer to the destination in terms of Euclidean distance, improving performance on most models.

In phase (ii), the strip is unfolded in the 2D plane and the shortest path

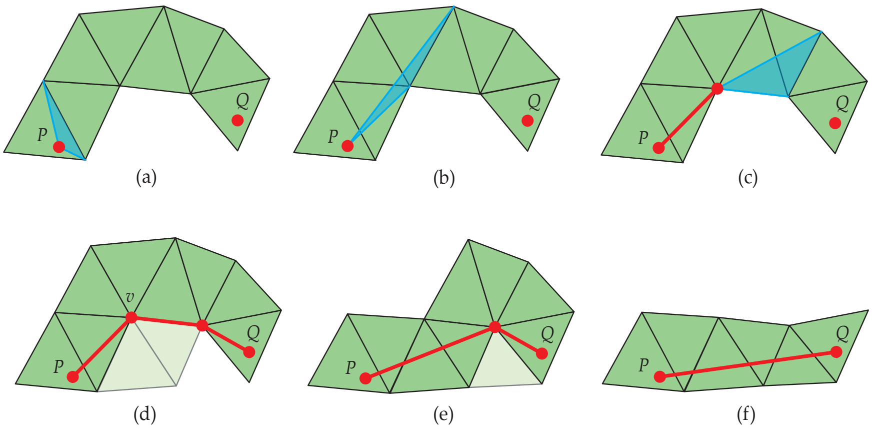


Figure 4.4: Shortest path computation. Given a source P and a target Q an initial strip of triangles is found with a search on the dual graph of the mesh. (a) A shortest path within the strip is found by propagating a funnel, which is initialized with its apex at P and its front at the first edge crossing the strip. (b) The edges of the strip are processed one by one, to tighten the front of the funnel. (c) When the funnel collapses, a new vertex, called a pseudo-source, is added to the path and the apex of the funnel is moved to the pseudo-source. (d) When Q is reached, some reflex vertices may still lie on the path. (e) Reflex vertices are analyzed for possible removal, starting at the vertex v causing the sharpest turn. (f) The final path is found when no more reflex vertices can be removed.

within it is computed in linear time with the funnel algorithm [LP84]. See Fig. 4.4(a-b-c) for an example.

In phase (iii), in order to obtain the locally shortest path on the mesh, we remove reflex vertices from the strip where possible. To this aim, Xin and Wang find the reflex vertices that can be removed by computing angles about a vertex inside and outside the strip, respectively [XW07]. However, in our experiments, the computation of angles slows down the algorithm, because the star of each reflex vertex is retrieved from a data structure that is not in cache memory. Instead, we select the reflex vertex v that creates the largest turn in the polyline and, similarly to [XW07], we update the strip by substituting the current semi-star of v inside the strip with its other semi-star. We perform the unfolding and the funnel algorithm on the new strip: if v still remains on the path, then it is frozen; we repeat this procedure until all reflex vertices either are removed or become frozen. See Fig. 4.4(d-e-f) for an example. This procedure is repeated iteratively until no turns in the path are found, at which point the 2D path is mapped back into a surface geodesic path and returned as result. Another special stopping case is when the larger turn is found at the same vertex twice in a row: this happens when the path passes through a hyperbolic vertex and nothing can be done to further shorten the path. This iterative algorithm is usually efficient as the input strip is made of up to a few hundreds of faces even on dense meshes.

The comparison with [SC20a] made in Chapter 7, shows that we consistently

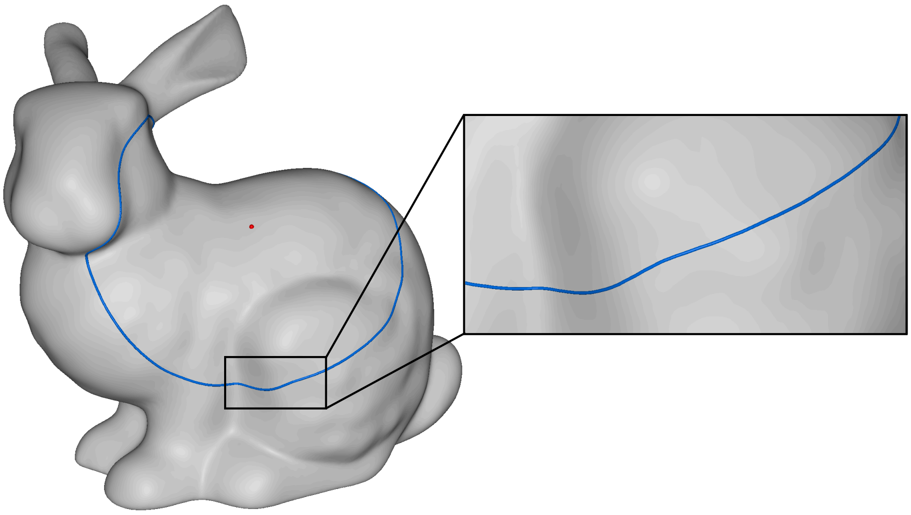


Figure 4.5: Geodesic circle traced on a 200k triangles Bunny. Even if the mesh is coarser than the one in Figure 4.3, the isoline of the distance is smooth and faithfully represent the behavior of the geodesic distance field.

beat it for about one order of magnitude in speed. The breakup of times presented suggests that the speed-up stems mostly from Phase (i).

4.3.2 Geodesic Distance Fields Computation

The method described below has been proposed in [MP22].

Our approach to estimate geodesic distance fields is a graph-based method similar to the techniques described in [NPP22] and [WFW+17], on a different graph though. In fact, our graph has all the vertices of the mesh as nodes, and arcs from each vertex to all vertices in its k-ring, where k is a parameter. Each arc is weighted with the length of the shortest path between two vertices, computed as described in the previous section.

As it will be clarified next, we will need to compute the geodesic distance field for a generic mesh point $p \in M$. To do that, we first compute the exact distances from p to the vertices of its containing triangle t and to the vertices opposite to the edges of t, and we initialize the distances of the corresponding nodes of the graph accordingly. We then propagate the distance field using the heuristic algorithm described above for the dual graph.

The parameter k used in building the graph provides a trade-off between the accuracy and the cost of the solver. A low value of k may compromise the smoothness of the distance field, hence resulting in wiggly isolines. For example, the geodesic circle in Figure 4.3 has been traced using the graph proposed in [NPP22], where the arcs of the graph connect a vertex v to its 1-ring and to every vertex in its 2-ring which can be connected to v with a dual edges. On the other hand a too high value for k may slow down computations considerably, thus hindering interaction. We experimented that $k = 3, 4$ are choices compatible with real-time interaction on high-tessellated meshes (few millions of triangles) and provide an estimation of geodesic distances sufficiently accurate for our purposes. For example, the geodesic circle in Figure 4.5 has been traced on a 200k triangles mesh. Even if the resolution is way lower than the example on the elephant showed in Figure 4.3, the result is smooth and the tracing is performed in real-time via a click-and-drag procedure. In future works, we plan to carry out a thorough comparison with state-of-the-art methods. On this regard, Nazzaro et al. have compared their implementation (which can be thought as the case $k = 1.5$ of our method) with the exact geodesic distances and experiments show that their method makes in average an error between 0.1 and 0.2 percent (see [NPP22]).

PART II

Geometric primitives on discrete surfaces

5

Practical Computation of the Cut Locus on Discrete Surfaces

This section includes contents from a co-authored paper [MLP21] that has been re-formatted for this thesis.

5.1 Introduction

Computational problems in the geodesic metric of Riemannian manifolds are becoming more and more relevant in geometry processing [AOCBC15, KCPS15,MRCK21,Sch13,SSC19b], optimal transport [BvdPPH11,LD11,SRGB14, SDGP⁺15, Sol18, MCSK⁺17], and machine learning [BYF⁺19, MKK21, MBBV15, RGA⁺20, SRC⁺20]. Many results, however, make assumptions about well definiteness of differential quantities of the distance function, uniqueness of shortest paths, injectivity of the exponential map, etc. However, such properties are violated at the cut loci of points.

Therefore, methods that make assumptions about staying away from the cut locus, without properly knowing it, might be hindered not just on a global basis, but on a local basis too. For example, the cut locus sets a tight bound to the injectivity of the exponential map in local surface parametrization [HA19] and it impacts the smoothness of the solution to the Monge problem in optimal transport [Vil08, Vil11].

Surprisingly enough, there are very few algorithms for computing the cut locus. Besides, existing algorithms are slow, or dependent on several parameters, or limited to specific classes of surfaces, or suffer of all such limitations (see Sec. 5.2).

In this chapter, we focus on real analytic surfaces represented as polygonal meshes, and propose a practical and efficient algorithm for finding the cut locus in this setting. The presented algorithm is independent of the method used to estimate the distance function, which is taken as an input and defined at the vertices of the mesh. Starting from the point farthest from the source (which surely belongs to the cut locus), we exploit Theorem 2.41 to grow a spanning tree that floods the entire mesh, and locally aligns with the cut locus. The method relies on a unique parameter that the user can tune interactively to obtain an approximation of the cut locus by pruning the spanning tree. For surfaces of genus higher than zero, we restore the correct topology by closing the necessary loops. As a result, we obtain a cut locus that is geometrically

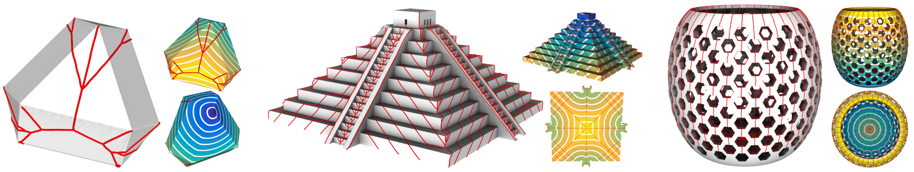


Figure 5.1: We can compute the cut locus of the geodesic distance function on any shape. From the simplest ones (left) to the most complex, both geometrically (middle) and topologically (right). Our output can be as smooth as the real cut locus (left, middle), or encoded in the edges of the underlying mesh (right). Both results are practically relevant for applications.

approximated, being made of edges of the mesh; and topologically accurate, having the same homology of the underlying manifold. For geometrically more accurate results, we also provide an algorithm to smooth the curves that compose the cut locus.

This method has three key practical advantages with respect to prior art: (i) the user can intuitively tweak a single parameter in real time with immediate feedback; (ii) it is orders of magnitude faster, thus permitting to operate on discrete manifolds with substantially higher complexity, both geometric and topological; (iii) it is independent of the algorithm used to compute the distance function, thus allowing the user to trade-off between accuracy and speed.

The method has been evaluated on several shapes as well as different algorithms for computing the distance function. We demonstrate its applications to the computation of the radius of injectivity of the exponential map, and to visibility-aware mesh cutting for texture mapping. The code is released in the public domain at https://github.com/Claudiomancinelli90/CutLocus.

5.2 Related Works

The first two tools for computing the cut locus that were proposed in the literature are *Loki* [ST02] and *Thaw* [IS04]. They both have limited capabilities, and were mainly developed with the purpose of supporting theoretical investigations. Loki is based on a polynomial approximation of the exponential map defined from a periodic parametrization of the surface, and supports only surfaces with genus one. Thaw supports only convex surfaces.

Misztal and colleagues [MBAM11] compute a retraction of the surface to the cut locus by means of an advecting front, represented as a piecewise linear curve. The cut locus is detected from self-intersections of the propagated front. Geodesics are computed by relying on a local parametrization through (2.17), and using finite differences to compute derivatives. Results are presented just on parametrized tori, and reported running times are about half an hour. While in principle this method scales to arbitrary surfaces, such an extension would require developing more general ways to compute geodesics, and to propagate the advecting front. Specifically, many geodesics are radially cast during front propagation, hence it may become very expensive to maintain the front accurate, especially far from the source, where it stretches.

Dey and Li [DL09] compute a subset of the cut locus, defined as the set of points where two minimizing geodesics meet after spreading apart by a certain amount. Geodesics are discretized with shortest paths on a graph, and are traced from all pairs of sampled points that lie closer than a given threshold. If two geodesics starting at nearby points diverge, then such points are added to the cut locus. The method depends on several parameters, but it is proven to converge to the cut locus by increasing the density of sampling and reducing the thresholds for distances between adjacent points and spread between geodesics. The output is just a discrete collection of points, which may be connected to form a complex. The authors report about ten minutes to compute the cut locus on a mesh of about 280K triangles.

Générau and colleagues [Gén20, GOV22] provide a comprehensive account of the subject, and propose a method to compute an approximation of the cut locus on a rigorous mathematical basis. They define a λ-*cut locus* as a regularized subset of the cut locus. They prove that such set can be approximated arbitrarily well with the solution of a variational problem, depending in turn on another parameter m. The solution converges to the exact cut locus when m goes to infinity and λ goes to zero. The approximation is obtained by resolving the variational problem with finite element methods. The method can work on general surfaces, but results are presented just on simple shapes (multi-tori of genus 1, 2, 3); reported times are of about an hour to process a mesh with 100K triangles.

None of these previous techniques is capable of combining the performances of the method described in this chapter with its ability to compute the cut locus on discrete manifolds with arbitrary geometric or topological complexity. In Sec. 5.4, we validate our geometric accuracy by comparing it against the ground truth on the torus [GMST05], as well as against numerical methods that converge to the exact solution [DL09, Gén20].

5.3 Method

Our method alternates discrete differential geometry and topology computations. We take as input an approximation of a smooth surface $\mathcal{M}$ approximated by a polygonal mesh $M = (V, F)$, where V and F are the vertices and faces, respectively. Without lack of generality, we assume all faces of F to be triangles. In the following, $x \in M$ is the *source* for which we evaluate the cut locus, and d_x is the distance function from it. We assume that a method is given to compute the distance function from any given point on M to all the vertices of V, as this computation is orthogonal to our contribution. In Section 5.6 we experiment with different methods. The method relies on four properties of the cut locus that has been introduced in Section 2.4.1, which we recall below.

(P1) The Laplacian Δd_x of the distance function is $-\infty$ in the sense of barriers at the cut locus (Theorem 2.41);

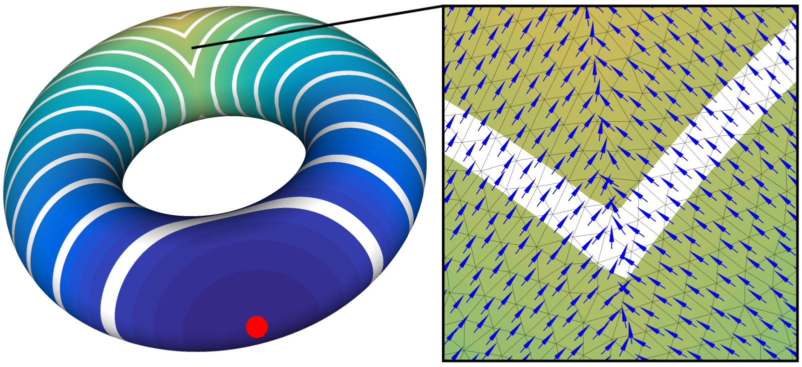


Figure 5.2: The distance function emanating from the red source is smooth everywhere except at the cut locus, where its gradient is discontinuous (closeup).

(P2) The cut locus is a finite linear graph, having the local structure of a tree (Theorem 2.42);

(P3) The cut locus has the same homology as $\mathcal{M}$: it is a tree for genus zero surfaces, and it contains $2g$ cycles otherwise, with g being the genus of $\mathcal{M}$ (Proposition 2.44);

(P4) The cut locus is piece-wise smooth. Specifically, it is C^∞ at all cut points which are not conjugate (Proposition 2.43).

Our method consists of two steps, plus one optional smoothing step. The steps jointly address the properties (P1-P4) listed above. We first compute the cut locus in the form of a tree, exploiting its relation with the Laplacian of the distance function (Property P1) and its local structure (Property P2). This step is already sufficient to provide a valid solution for objects of genus zero. In the second step, we ensure the correct homology for objects of higher genus (Property P3). In the third (optional) step, we smooth the cut locus, following the gradient of the distance function (Property P4). In the following sections we provide the technical details.

5.3.1 Cut locus from spanning tree

We know that the distance function d_x is not differentiable at points that can be joined with more than one geodesic to x (which are also called *regular* or *ordinary* points). In fact, the gradient ∇d_x points towards the cut locus and breaks at it, as illustrated in Figure 5.2. Moreover, it has been proved that regular points are dense in $C(x)$ [Bis77] and that the number of conjugate points is finite [Mye35], so the set of all regular points is for sure a good approximation of the cut locus of x. Our method aims to determine such a set.

The distance function d_x can be approximated arbitrarily well with a smooth function $\tilde{d}_x$ (barrier), whose Laplacian diverges to $-\infty$ at the cut locus as the approximation improves (Property P1). Therefore, if we estimate the discrete Laplacian of the distance function sampled at the vertices of M, we expect it

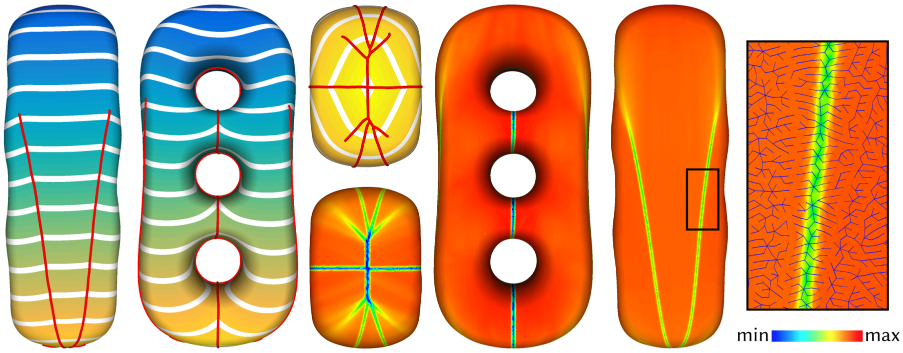


Figure 5.3: Left: cut locus of a three torus with respect to a distance function sourced at its topmost vertex. Right: the valleys of the Laplacian of the distance function clearly demarcate the paths of the cut locus. We catch these paths with a spanning tree that grows by locally prioritizing lower Laplacian values (closeup).

to become highly negative close to the cut locus. We base our construction on the above observation, designing a spanning tree that floods the entire mesh while aligning to the valleys of the discrete Laplacian, and then pruning its branches in order to retain only the portions of it that are at the cut locus (Figure 5.3).

ORG Spanning Tree

Knowing that the cut locus has the local structure of a tree (Property P2), to retain its branches we rely on the construction of an Ordered Region Growing (ORG) spanning tree T. For this, we were inspired by techniques that extract line structures from higher dimensional data, such as blood vessels from medical images [YCS00] and curve-skeletons [LGS12].

We set the root of T at the mesh vertex y^* that maximizes function d_x. Being the global maximum, this point is guaranteed to be in the cut locus. We then initialize a priority queue Q with y^*, and we grow T by iteratively extracting the top element from Q, creating new arcs with all its neighbors that have not been included in T yet, and inserting these points into Q. The process stops when the whole mesh is flooded, hence the queue Q becomes empty. We set the priority of each point y at $-\Delta d_x(y)$. This ensures that the tree expands following the deepest valleys of the Laplacian, thus aligning to the branches of the cut locus (Property P1). See the closeup to the right side of Figure 5.3.

Thinning

By construction, tree T spans the whole surface. Since we are operating in a discrete setting, if parallel branches of T are connected by transversal edges of mesh M, we consider them to span the portion of surface between them. On the other hand, we know that the cut locus has Hausdorff dimension one (Proposition 2.43). Therefore, we should not allow nodes of T to contain in their 1-ring in M any node of T, other than their parent and siblings in T. We

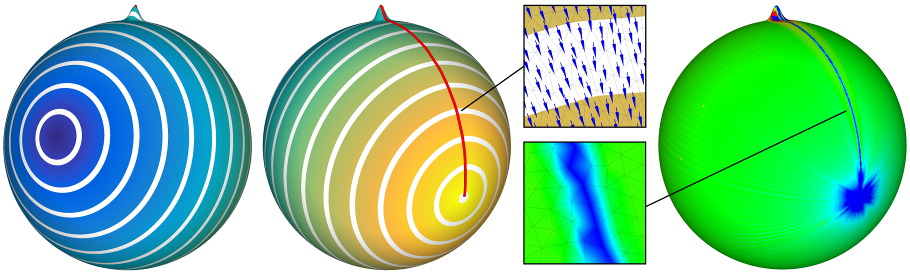


Figure 5.4: On a sphere with a bump, the cut locus of a point x (blue) is a geodesic line from the top of the bump to the antipodal point $\bar{x}$ (yellow). Despite weak in terms of discontinuity of gradients, our Laplacian-based detection system clearly defines a valley along the whole cut locus (closeups). Branches that accumulate around the antipodal point $\bar{x}$ may be filtered out.

prune T by removing the weaker nodes (in terms of their Laplacian) that do not fulfill this property. In order to preserve the integrity of the tree, we apply a thinning filter, akin [RKS00], again prioritized on the value of the Laplacian. We insert all the leaves y of T in a priority queue Q, this time by using $\Delta d_x(y)$ as priority. When an element y is extracted from Q, we check whether any of its neighbors, different from its parent and siblings in T, also belong to T. If there exits one such neighbor y' such that $\Delta d_x(y') < \Delta d_x(y)$, then we remove y from T, and we add its parent to Q if it has become a leaf node. Although this filter does not guarantee to fulfill the constraint above everywhere, it maintains the integrity of the tree, avoiding to discard whole branches which might contain strongly negative values of the Laplacian, just because some of their intermediate nodes are weak. In practice, we found it to produce better results than other thinning strategies that we have tried.

Filtering

Having flooded the entire mesh, even after thinning, the tree T contains many spurious branches, which do not belong to the cut locus. Spurious branches are not easy to remove automatically, because the paths of the cut locus may be very unstable. Consider for instance the example in Figure 5.4. It is well known that the cut locus of a point x on a sphere consists just of its antipodal point $\bar{x}$. However, if the sphere is perturbed with a bump, the cut locus of x is extended to a geodesic curve that connects the top of the bump to $\bar{x}$. Note that the bump can be arbitrarily small and arbitrarily close to x, hence the cut locus on the bumpy sphere can be as extended as a maximal semi-circumference.

Since we are working on a discrete approximation M of the manifold $\mathcal{M}$, we may easily miss existing bumps or, conversely, create artificial bumps. Branches caused by small bumps will be weak in terms of Laplacian, since geodesic lines from x reach them at a very narrow angle, hence easily confused with noise.

We thus decided to let the user clean the tree T from spurious branches, pruning it from the leaves, by means of a simple filter that can be tuned

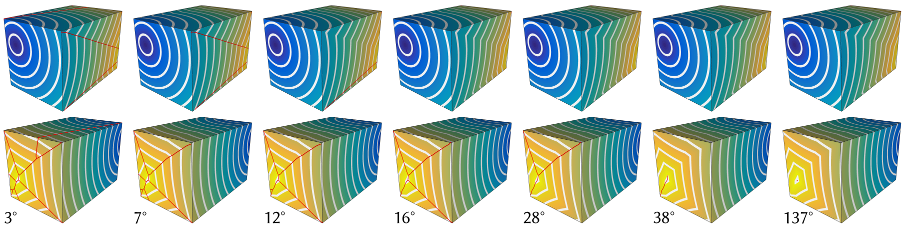


Figure 5.5: Increasing the threshold on the gradient angle, we obtain progressively simpler version of the cut locus, where "weak" branches that are nearly differentiable are pruned. Pushing the threshold to the extreme, we retain only the point antipodal to the source (right).

interactively in real time. The filter is controlled with a geometric parameter, which is independent of the specific dataset and is very intuitive: the angle between gradients that approach the cut locus from opposite sides. For each leaf node y of G_C, we consider the gradients $\nabla d_x(y_i)$ for all neighbors y_i in the 1-ring of y, we parallel transport such gradients to the tangent plane at y, and we compute the maximum angle between any two of them. If this angle is below a given threshold θ, then we prune y from T, and we proceed recursively along the branch it belongs to.

Note that, by pushing threshold θ towards relatively large angles, we may also remove the weaker branches of the cut locus, where function d_x is *nearly* differentiable, thus obtaining progressively simpler cut loci (Figure 5.5).

This may be useful in a variety of applications, since it is equivalent to consider a smoothed version of the manifold, or of its distance function. An alternative to the above filter consists of pruning the tree based on a threshold on the value of the Laplacian. While the two filters provide similar results, we privilege the one based on gradients because it is most intuitive and independent of the underlying shape. However, in some cases the filter based on the Laplacian allows the user to obtain better results. We therefore support switching between the two filters, or combining them.

Moreover, the filtering process can use two alternative policies: one based on pruning, in which the leaves that exceed the threshold are recursively removed; and the other based on growing, in which the tree is expanded from the root, including nodes that fulfill the given threshold. Pruning is more conservative, while growing is more aggressive. Consider again the example in Figure 5.4: both the gradient and the Laplacian are “strong" only at the antipodal point and at the tip of the bump. If pruning is applied, then the whole ridge shown in the image would be retained; conversely, if growing is applied, just the antipodal point would be retained, and the bump would be deemed noise. The default policy, which has been applied in most of our experiments, is the conservative one. However, with models containing many tiny details, such as the ones in Figure 7.16, the aggressive strategy provides cleaner results.

In our experiments, we experienced the presence of spurious tiny branches incident at the main ridges even after filtering. We provide an additional filter, which can be used to remove short branches to obtain clean paths. The

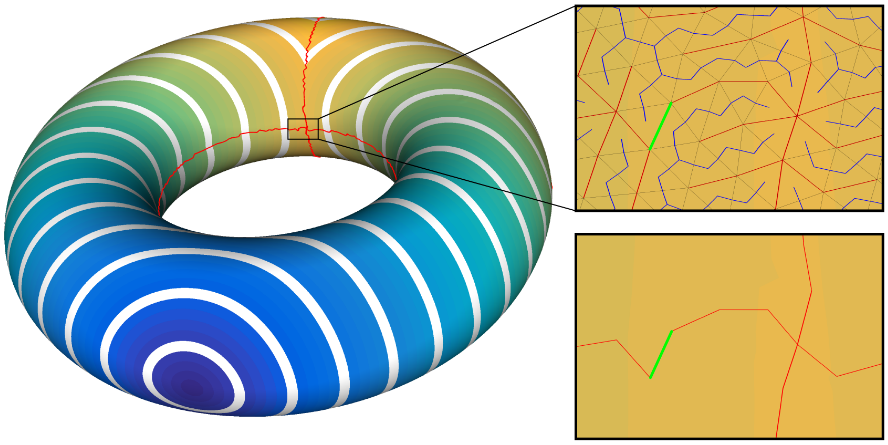


Figure 5.6: The cut locus has the same homology of the underlying manifold. We use a field-aware variant of the tree cotree algorithm [EW05] to transform the spanning tree into a system of loops, always guaranteeing the correct topology. In the upper closeup, red and blue lines denote the tree and cotree, respectively. Light green edges are the homology generators. The bottom closeup shows a detail of the output cut locus.

filter can be tuned either based on the number of edges in a branch (for high resolution meshes, we found that removing branches shorter than three edges is a reasonable choice in all cases), or on the length of the branch relative to the size of the object (for coarser meshes, removing branches shorter than 0.01 the length of the diagonal of the bounding box is a reasonable choice).

The filtered tree T_C provides our final approximation of the cut locus for objects of genus zero.

5.3.2 Homology

If M has genus $g > 0$, we know that its cut locus must have the same homology (Property P3), thus it must contain exactly $2g$ cycles. To restore the correct homology, we employ a variation of the greedy homotopy basis algorithm proposed in [EW05]. The original algorithm finds the shortest homotopy basis centered at a mesh vertex in $O(n \log n)$, by first growing a shortest spanning tree emanating from a source node, and then growing a spanning tree in the dual mesh (a.k.a. cotree [Epp03]), covering the edges not in the primal tree. This procedure leaves exactly $2g$ edges that are covered neither by the primal nor by the dual trees. These edges are the generators of the homology basis, and – bridging disjoint branches in the primal tree – form the wanted system of loops.

In our specific case, we are interested in finding the system of loops that best aligns with the cut locus. To this end, we initialize the tree with T_C, and extend it so as to flood the entire mesh with a new tree $\bar{T}$. Next, we set the root of the cotree $\bar{C}$ at one of the triangles incident to the source x, and we expand it across edges that do not belong to the primal tree $\bar{T}$. Our goal is to position the homology generators as close as possible to the cut locus. To do so, we ensure that both the tree and the cotree grow at constant speed in all directions, thus forcing opposite fronts of the trees to cover roughly the same distance before they collide. We obtain the desired result by prioritizing the growth of the tree $\bar{T}$ with values of d_x, and the growth of the cotree $\bar{C}$ with $-d_x$ (interpolating the function at the centroid of each triangle). In most cases, the endpoints of a generator are already nodes of T_C, and it is sufficient to add such edges to the graph to close the loop (Figure 5.6). In case a generator e is not connected to T_C at its endpoints, we climb the branches of $\bar{T}$ until we reach a node of T_C, and we close the loop by adding the corresponding paths to T_C.

In the latter case, the portion of cut locus that closes the loop is a subset of tree $\bar{T}$, hence misaligned with the valleys of the Laplacian. We explain here why our choice to grow $\bar{T}$ according to values of d_x not only does not sacrifice geometric accuracy, but rather enhances it. Remembering that the Laplacian Δd_x is the divergence of the gradient ∇d_x, the previous step of the algorithm is extremely effective at detecting *strong* branches of the cut locus, characterized by strongly convergent gradients, but is less effective at detecting *weak* branches, characterized by nearly parallel gradients. Weak branches of the cut locus, which may have been missed in the previous step (e.g., due to aggressive thresholding), roughly align with the local gradient, which provides a valid guidance for tracing them (if a valid starting point in the cut locus is known).

Tree $\bar{T}$ is initialized with the subset of the cut locus selected at the previous step, and is then expanded according to growing values of d_x, hence aligning with ∇d_x. As a whole, this hybrid tree aligns to the valleys of the Laplacian where the gradients clearly converge, and to the gradients where they are nearly parallel. This allows us to have the best of both indicators, using each one of them in the places where it is more appropriate. With this technique, we were able to reconstruct high quality cut loci of shapes with non trivial topology even operating with approximated distance functions, often characterized by less accurate Laplacian fields (Figure 5.11). The result of this step is a graph G_C providing a discrete approximation of the cut locus.

5.3.3 Smoothing

Graph G_C is made of edges of mesh M, thus it necessarily contains wiggly paths. This approximation is suitable to many applications, e.g., if we want to prevent any computational technique from crossing the cut locus. However, we know that the cut locus of a smooth manifold $\mathcal{M}$ consists of smooth lines (Property P4). We try to obtain a smoother (yet still piecewise-linear) structure by pushing the nodes of G_C closer to the true cut locus, freeing them from the edges of M, while remaining on its surface.

We apply a tangent space smoothing algorithm driven by gradient ∇d_x. We know that the gradient of d_x is oriented towards the cut locus from both sides

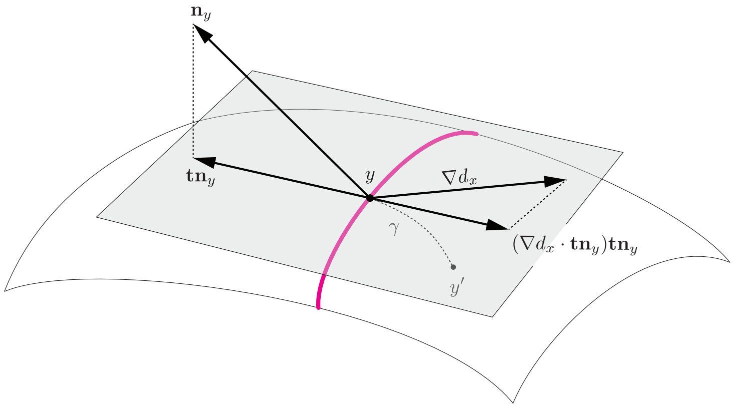


Figure 5.7: Construction for gradient-driven smoothing. We estimate the discrete tangential normal $\mathbf{tn}_y$ of the cut locus at y. The displacement vector is proportional to the component of the gradient $\nabla d_x(y)$ along $\mathbf{tn}_y$. Curve γ is a geodesic line cast from y in direction $\text{sign}(\nabla d_x(y) \cdot \mathbf{tn}_y)\mathbf{tn}_y$ for length $\lambda|\nabla d_x(y) \cdot \mathbf{tn}_y|$.

of it (Figure 5.2); since we actually estimate the gradient of the smooth barrier $\nabla\tilde{d}_x$, we expect its component orthogonal to the cut locus to be null at the cut locus. Thus, we iteratively displace each node y of G_C in the direction of the normal to the curve through y in its tangent plane, for an amount that depends on the component of $\nabla\tilde{d}_x(y)$ along this normal, until such component becomes null. Note that, a displaced node y is no longer a vertex of M, but it lies inside a triangle t_y of M, and it is encoded with barycentric coordinates wrt t_y; we estimate the gradient at y by linearly interpolating the gradients at the vertices of t_y, parallel transported at y. Consecutive nodes are connected with shortest geodesic paths to obtain the final cut locus.

Refer to Figure 5.7 for a visual explanation of the smoothing step. Let y_p, y, y_n be three consecutive nodes of G_C, y being a regular node. We first estimate the normal of the 3D curve through y_p, y, y_n with standard finite differences, as in [LBS05], then we project such a vector to the tangent plane at y to recover the tangential normal $\mathbf{tn}_y$. We displace y by casting a geodesic path from y in tangent direction $\mathbf{tn}_y$ for a length $\lambda\ell(\nabla d_x(y) \cdot \mathbf{tn}_y)$ where λ is a damping parameter, which is initialized at 0.5 and halved at each iteration, and ℓ is the average edge length. Note that the sign of the dot product may reverse the direction of $\mathbf{tn}_y$ according to the component of the gradient along it. Note also that the geodesic path will hardly cross more than one edge per iteration, while it provides a safe way to follow the intrinsic metric.

For a leaf node y_l the algorithm is analogous, just the displacement of y_l occurs in the opposite direction wrt the parallel transport of the normal at its neighbor. A branching node y_b is simply displaced tangentially towards the centroid if its neighbors on G_C, as the gradient estimated at branching points is usually not reliable.

The proposed approach readily pushes the discrete cut locus to the smooth cut locus in areas where the former has a non vanishing angle with the local gradient $\nabla d_x(y)$. Extremely weak portions of the cut locus are less affected,

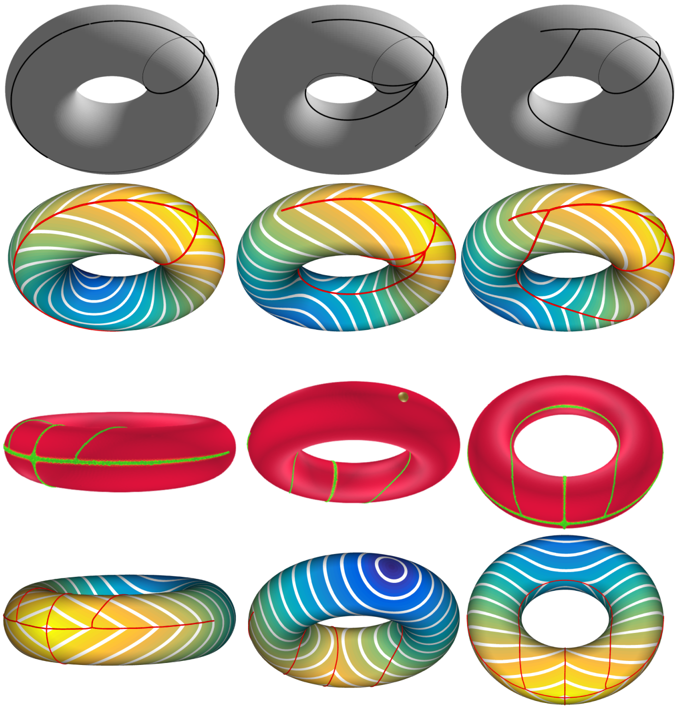


Figure 5.8: Visual comparison with the analytic cut loci showed in [GMST05] (top), and with the numerical method described in [Gén20] (bottom). We manually tried to replicate the same sources, obtaining visually indistinguishable results. For the comparison with [Gén20], both meshes contain 100K triangles. We computed our cut locus in less than one second. Our competitor is based on a convoluted numerical FEM scheme, which requires 17 Gauss quadrature points per element. The author declares that the computation terminated in less than 1 hour.

because the tangent curve and the local gradient are nearly parallel. We eventually apply one step of classical tangent space smoothing to relax these areas, too.

5.4 Results

We have implemented our algorithm in C++, using libraries Yocto/GL [PNC19], CinoLib [Liv19] and libigl [JP+18] for geometry processing. We have validated our results in a variety of experiments reported in this section. The method performs efficiently on meshes with a complexity up to a few million triangles, and demonstrates to produce plausible results on all models, correctly locating the cut locus even in places where the distance function seems rather smooth upon visual analysis (see, e.g., Figure 5.6 and Figure 5.10).

Validation

From a topological standpoint, our method is always guaranteed to produce the correct result, meaning that the cut locus has the same homology of the

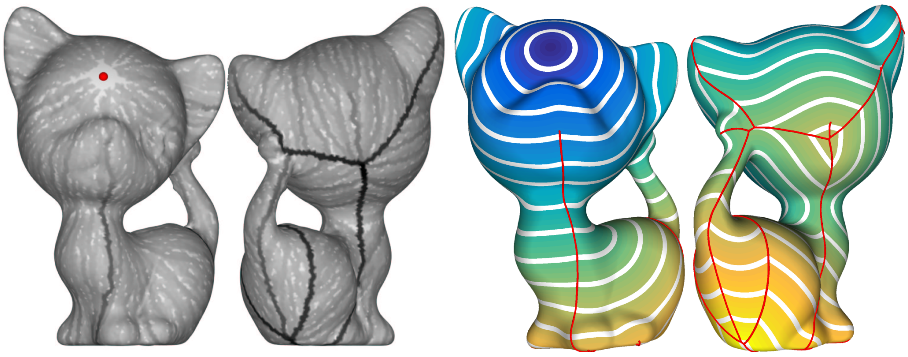


Figure 5.9: Visual comparison with the method proposed in [DL09] (left). We manually pinpointed the source on a different mesh. The mesh used in [DL09] contains about 60K triangles and the computation takes about 12 seconds, while our model contains 250K triangles and computation takes about 3 seconds. Reported times in [DL09] are about ten minutes for other meshes of complexity comparable to ours.

underlying manifold. From a geometric point of view, the algorithm has a strong theoretical foundation in the continuous setting, but in practice it relies on a discretization of the Laplace-Beltrami operator, and on heuristics to build the connectivity of the cut locus. Therefore, we cannot guarantee the exact location of the cut locus on the manifold. In Figures 5.8 and 5.9, we validate the geometric accuracy of our algorithm by comparing our outputs with the ground truth on a torus [GMST05], and with the output of prior approximated methods, which are guaranteed to converge to the exact solution [DL09,Gén20]. We obtain visually indistinguishable results on the torus, and a very similar result on the kitten. For the latter, based on our understanding and practical experience, we ascribe the tiny differences in the actual paths to the different mesh tessellations.

Scalability and performances

We experimented our method on a variety of meshes, ranging from about 100K to about 2M triangles. High-resolution meshes for all models have been obtained via isotropic remeshing in Meshlab [CCC$^+$08], to warrant a stable estimation of differential properties. All experiments are executed on a laptop equipped with a 2.9 Ghz quad-core Intel Core i7 and 16 GB RAM, running on a single core. Table 5.1 reports running times for the various phases of the method. Considering time for pre-processing (once per mesh) and computation of the distance function (once per source point, possible with three different methods), on large meshes our method runs at least two and up to three orders of magnitude faster than any other competitor. The time for the part due to our contribution (i.e., excluding the computation of the distance function) is interactive, and several orders of magnitude faster. Note that this is relevant because, while the other methods require to run the algorithm again from the beginning each time the parameters are changed, we can do parameter tuning interactively after the time consuming part (i.e., computation of the distance

			Pre-processing	Distance function			Cut locus computation		Total
model	triangles	genus	min-max	VTP	heat	graph	spanning tree	homology	min-max
torus	100K	1	1.13 - 2.78	0.78	0.03	0.01	0.06	0.10	0.17 - 0.95
diamond	160K	0	1.86 - 5.46	3.11	0.06	0.01	0.12	–	0.13 - 3.23
block	170K	0	1.90 - 5.62	3.20	0.06	0.01	0.13	–	0.14 - 3.33
2-torus	200K	2	2.35 - 6.41	2.55	0.07	0.01	0.16	0.21	0.38 - 2.91
3-torus	200K	3	2.17 - 6.84	2.29	0.07	0.01	0.14	0.19	0.33 - 2.88
raindeer	200K	0	2.45 - 5.08	1.92	0.06	0.02	0.18	–	0.20 - 2.10
sphere	240K	0	2.82 - 10.04	5.64	0.10	0.02	0.22	–	0.24 - 5.86
kitten	250K	1	2.70 - 7.73	2.67	0.09	0.02	0.18	0.24	0.44 - 3.09
nefertiti	460K	0	5.37 - 17.77	3.97	0.19	0.04	0.42	–	0.46 - 4.39
bunny	500K	0	5.52 - 19.45	8.16	0.19	0.04	0.39	–	0.43 - 8.55
fertility	500K	4	5.55 - 19.21	7.92	0.20	0.04	0.41	0.50	0.95 - 8.83
pyramid	660K	0	8.46 - 38.24	11.64	0.30	0.08	0.73	–	0.81 - 13.45
bust	700K	0	9.18 - 39.89	10.65	0.32	0.09	0.78	–	0.87 - 11.43
octopus	800K	0	9.93 - 28.10	14.42	0.31	0.09	0.87	–	0.95 - 15.29
basket	1.1M	260	13.94 - 97.98	13.87	0.62	0.12	1.18	1.36	2.66 - 16.41
fertility	2.0M	4	22.92 - 173.68	66.09	2.99	0.20	1.81	2.10	4.11 - 70.00

Table 5.1: Statistics on models used in the experiments and related running times in seconds. Preprocessing includes times for: evaluating metric tensors and related vectors for differential computations; pre-factorization in case the heat method is used; construction of the graph in case the graph method is used (the latter is one order of magnitude smaller than the rest). Spanning tree computation includes also the thinning filter. The total time depends heavily on the method used for the distance function (min with graph, max with VTP [QHY+16]) and does not include pre-processing times; graph achieves minimum times even including pre-processing; while heat becomes the most expensive for large meshes because of the cost of pre-factoring. Applying a filter upon parameter tuning works in real time, taking less than 0.01 seconds on all models, and it is not included in the table.

function) is complete.

Impact of mesh resolution

As any other technique that approximates continuous entities with a finite discretization, our method performs best on dense and regular samplings. In particular, in our experiments we noticed that even though theory ensures that the Laplacian goes to $-\infty$ at the cut locus, the discrete Laplace-Beltrami may yield values near zero where the local gradients are almost parallel, hence the cut locus is very weak. In Figure 5.10 we show a typical failure case. Notwithstanding the feeble signal in the Laplacian, our method was able to reconstruct the corresponding loop and enforce the correct topology. Similar issues that arise outside closed loops cannot be recovered. Considering the computational efficiency of our method, the easiest way to overcome these issues is to operate on dense meshes.

Impact of distance computation

We have experimented with three different methods for computing the distance function: VTP [QHY+16], which provides an efficient variant of the original MMP algorithm [MMP87]; the heat method [CWW13]; and a simple graph-based solver [NPP20]. VTP requires no pre-processing, and provides an exact solution in the polyhedral metric, but it is rather slow on large meshes.

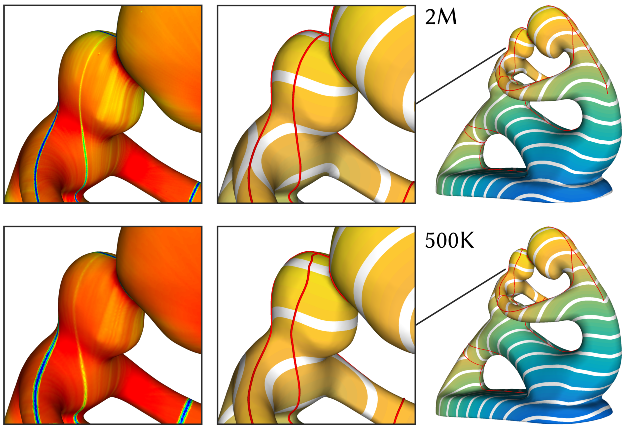


Figure 5.10: Mesh resolution may impact the result of our algorithm. In the denser mesh, the Laplacian field (left closeup) clearly identifies a weak path. In the coarse mesh the same path is more blurred. Since that portion belongs to a closed loop, our topological step correctly reconstructs it, but if the same path was open, it would have been difficult to retain it because the gradients are almost parallel, as the level sets of the distance function suggest.

The heat method requires solving a linear system of the same size of the mesh. The matrix can be pre-factorized once per mesh, and the solution after factorization amounts to a matrix vector multiplication, hence it is quite fast. It provides an approximated estimate of the distance function for an underlying smooth manifold, and converges to the exact solution as a parameter t tends to zero. We used the implementation in [JP$^+$18] with the default parameter, as recommended by the authors [CWW13]. The graph-based solver relies on a graph with one node per vertex, and one arc for each edge and dual edge of the input mesh. The graph is built during pre-processing, and a solve consists of a Dijkstra-like visit, which is quite fast, too. We have used the implementation in [PNC19]. Compared with the exact polyhedral solution, the accuracies of the heat method and of the graph solver are similar, but with different artifacts, as discussed below.

All images in the paper, except 5.11, were generated by using VTP. In 5.11, we compare results obtained with the three methods on two objects. We were able to successfully compute the cut locus with all three methods. Nevertheless, we could observe some interesting differences between the alternative approaches, especially in terms of the Laplacian they yield. VTP consistently produces a neat Laplacian that is completely free from noise. The paths of the cut locus are very sharp and easy to identify. The graph based approach produces a Laplacian with biased noise, where the propagation paths can be clearly identified. Apart from that, the paths of the cut locus remain sharp and easy to identify. On the positive side, the noise-level stripes produced by this method turn out to be useful to trace the weak branches of the cut locus, because they align with the gradient of the function. The heat method was the most challenging for us because, by solving a Poisson problem, it generates a function that is smooth

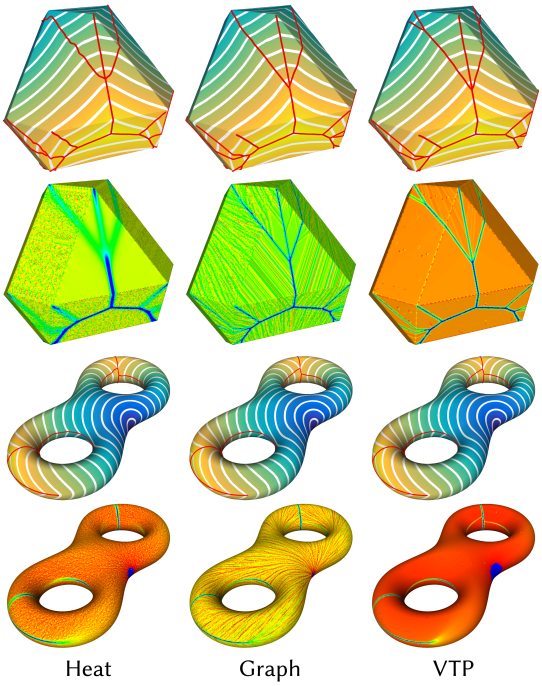


Figure 5.11: Cut loci computed with two approximated methods (the heat method [CWW13] and a graph solver [NPP20]), and an exact polyhedral method (VTP [QHY⁺16]). We show the distance function with the output (top), and its Laplacian (bottom). The exact method produces fields that clearly demarcate the branches of the cut locus. With approximated methods, noise occasionally blurs the Laplacian around the branches of the cut locus.

everywhere, cut locus included. This results in a blurred Laplacian field, where strong branches of the cut locus are still clearly identified, but weaker ones fade away (e.g. the three green paths in the upper part of the diamond), or completely disappear (e.g. the closure of the inner loops in the double torus). As a result, the paths of the cut locus computed with the heat method tend to slightly misalign from the other methods, albeit the result is still acceptable for most applications.

In terms of speed, we report timings in Table 7.1. The cost of VTP is dominant over all other phases of our algorithm. The heat method and the graph solver are quite fast (after pre-processing); the graph method scales better on large meshes, though, in terms of both pre-processing and solve times. All in all, for non time critical applications where quality dominates timing, VTP is by far the best solution. On the other hand, the graph based approach provides a good balancing between accuracy and running times, also proving to scale better than the other methods on meshes containing millions of elements.

Impact of filtering

The interactive filter described in Sec. 5.3.1 ranks each edge of the ORG spanning tree according to the angle formed by the gradients of the distance

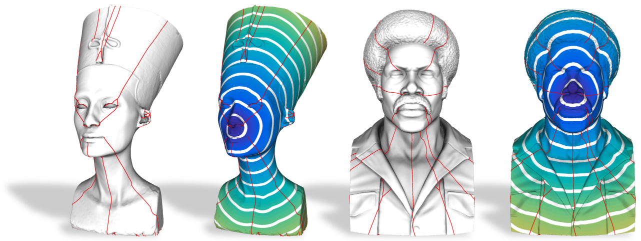


Figure 5.12: Our method can cope with complex shapes with small details and a rough surface, too. In these cases, it may be necessary to tweak the filter to a higher threshold in order to remove spurious branches and obtain a clean cut locus.

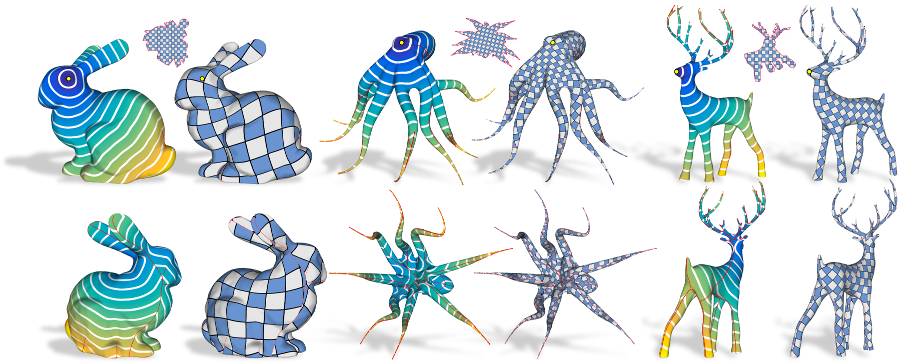


Figure 5.13: Given a mesh and a point of view (yellow dot), cutting the mesh along the cut locus positions texture seams furthest from it, making discontinuities least visible. We provide three alternative examples on complex shapes containing elongated structures. UV maps (shown in the middle insets) were computed with ARAP [LZX$^+$08].

function at its two sides. For free branches, this is a quite reliable estimate of their “strength" in the cut locus, namely of whether or not they are caused by relevant features of the model, or just by noise. However, even important branches may fade into regions with nearly parallel gradients.

The reconstruction of homology loops is quite robust and independent of filtering. We consistently experienced correct reconstructions even if parts of the loops were filtered out before the homology part is performed. On the contrary, weak free branches are quite unstable and may need to be recovered by tweaking the filter properly. The exact position of terminal points of such branches, which are all conjugate points in the exact cut locus, are hard to detect. Therefore, the length of such branches is necessarily approximated. This is especially evident for rough surfaces containing many small details, as the ones depicted in Figure 5.12.

Meshes with open boundaries

Our current implementation supports watertight meshes only. In principle, its extension to meshes with open boundaries is straightforward. Of course, the methods for computing the distance field must support meshes of this kind; and some shortest paths would contain boundary edges and not be geodesics in

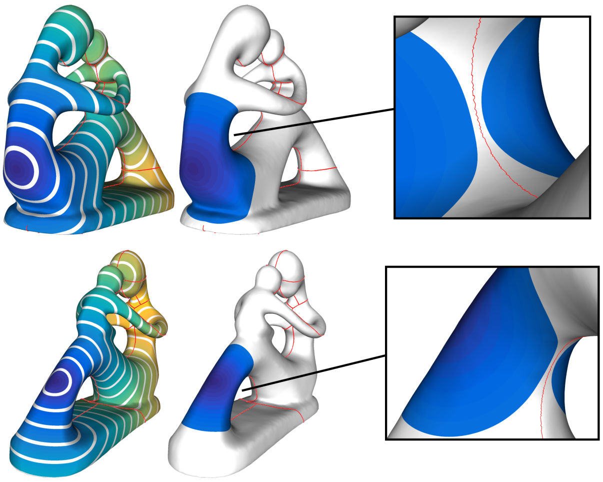


Figure 5.14: Starting from a geodesic distance function emanating from a single source, our method allows to precisely retrieve its distance from the cut locus, thus defining the radius of the maximal ball under which the exponential map is injective. This construction is relevant for many techniques in machine learning and optimal transport (Sec. 5.1).

a geometric sense. In terms of the algorithm, the point furthest from the source is no longer guaranteed to belong to the cut locus, if it lies on the boundary. In that case, the root of the ORG spanning tree could be set at the point with the lowest Laplacian, which is the best guess in this scenario. Besides these tiny details, perhaps the most critical issue concerns the reliable estimate of the Laplacian field. As already observed for differential quantities of the first order, the absence of a complete neighborhood for boundary points makes the estimate of gradients unreliable [MLP19]. We expect the estimate of second order differential quantities to be even worse, possibly affecting the efficacy of our method. Thus, special care would be needed to process boundary points.

5.4.1 Applications

Being a fundamental descriptor of a distance function living on a surface manifold, the cut locus lends itself to a variety of alternative uses. While demonstrating them all is outside of the scope of this paper, in this section we showcase two practical applications involving the cut locus, in the context of global and local surface parameterization.

POV-aware texture mapping

In texture mapping, objects that are not topological disks must be cut open prior being flattened to the plane. Cuts introduce discontinuities in the map, and often accumulate distortion, resulting in visual artifacts. Techniques for texture mapping try to hide cuts in the least visible parts of the surface, so that the weak spots of the map are not immediately perceived [SH02]. To this end,

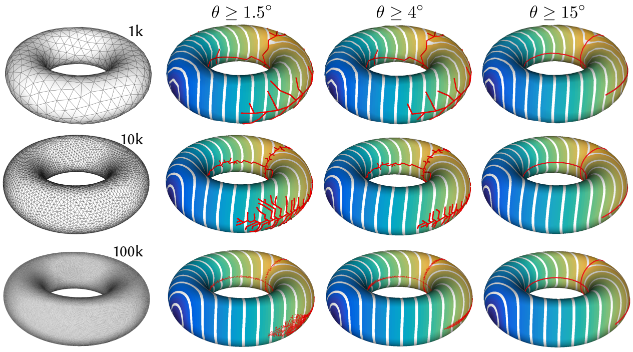


Figure 5.15: Alternative cut loci computed with our method using progressively finer meshes (rows) and wider angle thresholds θ (colums). With enough filtering, our method recovers the cut locus correctly on all models. Short branches have been filtered and smoothing have been applied to models in the last column only.

the cut locus reveals itself to be a practical tool. For objects that are largely observed from a known viewpoint – e.g., for digital sculptures in a virtual museum, or for objects for which a visual saliency map is known – one can initialize a distance function that emanates from a specific point of interest, and cut the mesh open through the cut locus of such function. There are two nice consequences: (i) since the cut locus and the manifold are homotopic, cutting through the cut locus will provably generate a topological disk, suitable to texture mapping; (ii) since the cut locus maximizes the distance from the source, cuts will be naturally hidden when the mesh is observed from the selected viewpoint. We implemented an interactive tool that allows the user to select a saliency point on a mesh, and automatically cuts it at the cut locus of the distance function that emanates from it. 5.13 shows a few results obtained with our tool.

Maximal injectivity disk

Methods that exploit a local parameterization of the surface typically rely on a heuristically computed radius, under which the exponential map is supposed to remain injective (Sec. 7.1). Considering its minimal computational overhead, our method allows to enhance these methods, providing a precise estimate of the maximum radius that verifies this assumption. The check is pretty straightforward: considering a point x and the distance function d_x emanating from it, the maximum radius $r_{\max}$ can be computed as

$$r_{\max} = \arg\min_{p} d_x(p) \quad s.t. \quad p \in C(x)\ .$$

Note that a disk having radius $r_{\max}$ centered at x will be tangent to itself at the cut locus. In practice, one may want to consider a slightly smaller radius $r' = r_{\max} - \epsilon$, which guarantees the full injectivity of the map. In 5.14 we show a few examples of nearly maximal disks centered at different points of a manifold with complex genus.

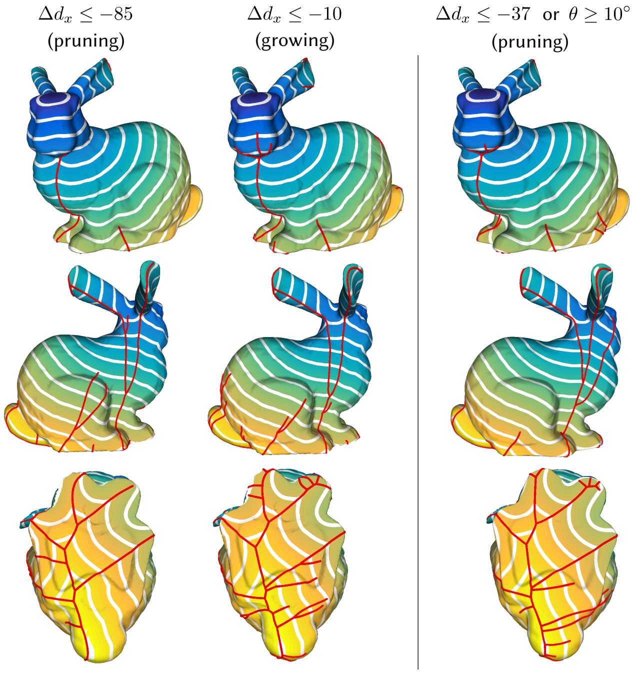


Figure 5.16: Cut loci computed on the Stanford bunny with its original tessellation and filled holes (left) and a regular isotropic remeshing of it (right), computed with Meshlab [CCC+08]. Left: because of the artifacts shown in the closeups in Fig. 5.17, the pruning and the growing policies are not able to retain the cut locus and, at the same time, filter the spurious branches of the spanning tree. Right: on a regular mesh all versions of the filter perform equally well

5.5 Limitations

Our method assumes a fairly accurate estimate of first and second order differential quantities of the distance function. The discrete methods discussed in Sec. 5.3.1 achieve this on meshes with high resolution and isotropic elements. Since our method is fast, we found it easy to remesh unsatisfactory models prior processing.

On coarse meshes, our algorithm still reconstructs the cut locus correctly, as long as the resolution is not too low and the mesh elements are slightly regular (5.15). On coarse meshes, or when the elements are poor(e.g. badly shaped, vertices with low-valence and/or irregular 1-ring, etc.), the weak branches of the cut locus are hard to distinguish from noise. In that case, while the homology is always recovered correctly, the geometry may become imprecise, some weak branches may be lost, and some spurious branches may be retained. For instance, the Stanford bunny has a complicated cut locus with many free branches that are hard to detect (5.16). On the original mesh, the irregular tessellation may interrupt the valleys of the Laplacian that demarcate the cut locus, or introduce bumps along them, or spurious local minima (5.17). Such a poor estimation may cause some points to become either too strong, or too weak, thus hindering the action of filters. Note that the “strength” of a branch

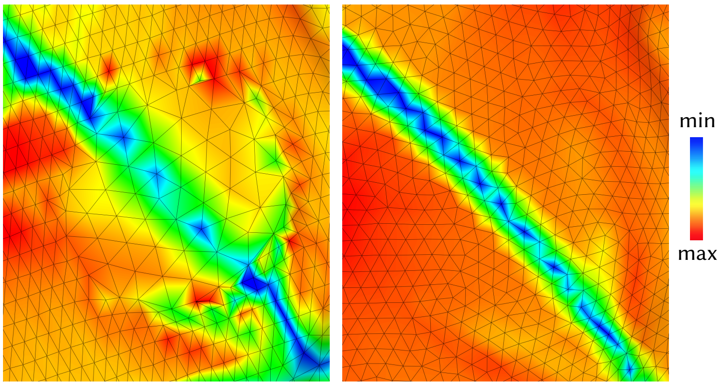


Figure 5.17: Estimating the Laplacian on an irregular (left) and regular (right) mesh: a poor tessellation may break the valleys of the Laplacian (discontinuous blue line) and introduce spurious local minima. Both types of artifacts may hinder the action of our filters, as exemplified in 5.16

depends on the location of the source, the geometry of the shape, and the discretization. Therefore, in general it is not possible to determine a priori which mesh density is suitable to detect all branches. The pruning policy needs an aggressive threshold to filter the branches ending at spurious local minima, also missing the weak branches of the cut locus (5.16, left). On the other hand, with the growing policy some branches of the cut locus are truncated too soon, because of bumps and cracks along the valleys of the Laplacian (5.16, middle). On a regular tessellation, both policies allow us computing an accurate estimate of the cut locus with reasonable thresholds (5.16, right).

At this regard, it is worth pointing out that the discrete Laplacian operator described in Section 3.2.2 turned out to be more resilient to irregular tessellation. Although even on regular tessellation the valleys slightly more evident with respect to the cotangent Laplacian (Figure 5.18, *left*), the difference is way more evident when computing the Laplacian of a distance field on the original Stanford bunny (Figure 5.18, *right*). Even if the branches are not identical in the two cases, since the tessellation are different and the source is not placed exactly in the same point, we can notice that our method still manage to clearly identify the main branches of the cut locus, while the cotangent method provides mostly weak branches that often blur away. Moreover, the presence of “false local minima” (isolated blue regions in the heat map of the bunny in the bottom-right of Figure 5.18) in this latter case may affect the expansion of the spanning tree, while, modulo the limitations discussed above, is more robust from this point of view.

We did not experiment with extreme cases characterized by very sparse meshing and long and skinny elements, such as many of the meshes found in the Thingi10k repository [ZJ16]. In that case, it may be convenient to use intrinsic triangulations [SSC19a] and related differential estimators [SC20b]. The rest of our method would work unchanged, but it needs being implemented in the framework of intrinsic triangulations.

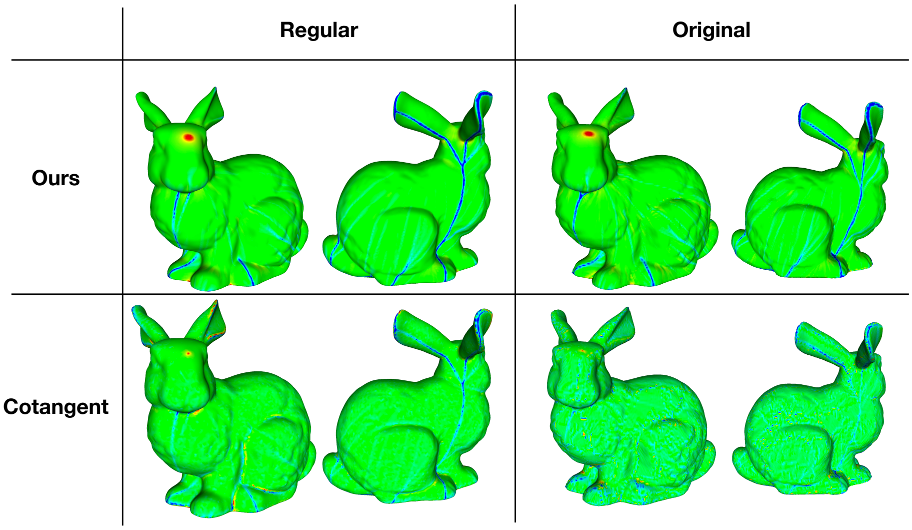


Figure 5.18: Estimation of the Laplacian with the operator described in Section 3.2.2 (*top*) and the cotangent Laplacian (*bottom*. On regular meshes (*left*), the performances of two methods is comparable, whereas on the original Stanford bunny our method demonstrate higher resilience to irregular tessellations.

5.6 Concluding remarks

We have presented a novel method to compute the cut locus that is practical and fast. The method depends on a unique intuitive parameter that can be tuned interactively to filter out artifacts arising from small details of the surface, or from discretization. The method works on surfaces of any genus, always recovering the correct topology of the cut locus; it works on shapes with sharp creases; and on rough shapes with many small details, too.

We conjecture that out method converges to the true cut locus as the mesh becomes denser, but proving this fact requires further work. In summary, all methods we adopted for computing the distance field can be shown to converge to the true geodesic distance; and the method to compute the differential quantities also converges for smooth functions. By applying such estimators to a denser and denser mesh, the estimated Laplacian should converge to the Laplacian of the distance function away from the cut locus, and to the Laplacian of a smooth barrier function near the cut locus. Thus, we expect that for any given value $A < 0$ there exists a mesh dense enough that $L(y) < A$ for all y at the cut locus. However, since no bound from below to the Laplacian away of the cut locus is known, in general the Laplacian alone is not sufficient to characterize all and only the points of the cut locus. This fact further motivates the additional criteria that we apply in our method. We foresee two interesting avenues for future work. For the computation of the cut locus, we plan to improve on our current method to achieve a reliable fully automatic detection that works well in all practical scenarios. The most challenging issue in this direction is to determine where the weak free branches end. It would also be interesting to exploit the boundary structure provided by MMP-like algorithms, as in [LCT11]. Based on such structure, the exact cut locus in the polyhedral metric can be computed. However, such a cut locus would consist of a dense tree, with one leaf at each parabolic vertex, thus being useful only for strictly polyhedral objects without any curved surface. It is an open problem how to define suitable pruning strategies to obtain a cut locus that is descriptive for curved objects approximated with a mesh, too.

Finally, we plan to explore the capabilities of our approach for the computation of the medial axis, which is a widely popular shape descriptor used for shape compression, matching and skeletonization [TDS⁺16]. Indeed, the medial axis can be defined as the cut locus of a distance field emanating from the boundaries of a geometric domain, growing inwards. Despite in this work we focused our attention on distance fields emanating from a single (point-like) source, in his work Générau showed that the Laplacian goes to $-\infty$ also for distance fields emanating from a general hypersurface embedded in the manifold domain [Gén20], creating a connection with the λ-medial axis [CL05]. In its current state, our algorithm is not able to reconstruct a proper connectivity for this more general case, but since the theoretical foundation still holds, it would be interesting to work at different tools to filter the Laplacian field and generate the medial connectivity, both for 2D and 3D manifolds.

6

Vector Graphics on Surfaces Using Straightedge and Compass Constructions

This section includes contents from a co-authored paper [MP22] that has been re-formatted for this thesis.

6.1 Introduction

The ancient Greek mathematicians developed a set of geometric techniques, which go under the name of *straightedge and compass constructions*, to draw planar geometric figures. Such constructions do not require taking any explicit measure, they are granted by Euclid’s first three postulates, and are based on two idealized tools: the straightedge, which can extend indefinitely the straight-line through any pair of points; and the compass, which can trace circles with its needle and pencil points at any two points in the plane. Besides, all intersections between straight lines and circles drawn with such tools can be found.

In the Euclidean setting, the straightedge and compass constructions can be substituted with simpler closed form solutions, though, as it is customary in 2D drawing systems. However, when addressing similar operations on a surface, one must rely on the computation of distance fields and geodesic lines. Such building blocks are indeed similar in nature to those available in the straightedge and compass framework. In the context of our effort to bring vector graphics to surface domains [MNPP22,NPP22], we thus investigate how to port such constructions to the manifold setting.

We address the problem with two complementary approaches. The first approach performs constructions in a tangent plane and then maps the result to the surface. The second approach extends the concepts of straightedge and compass to the geodesic metric and operates directly on the surface.

Euclidean constructions rely on properties that no longer hold under the geodesic metric, due to the intrinsic curvature of the surface. Because of that, both approaches fail in producing results that preserve *all* properties of their Euclidean counterpart. In fact, even the topological properties of straight lines and circles do not hold on a surface without additional conditions: geodesic lines may self-intersect or mutually intersect multiple times; and a generic

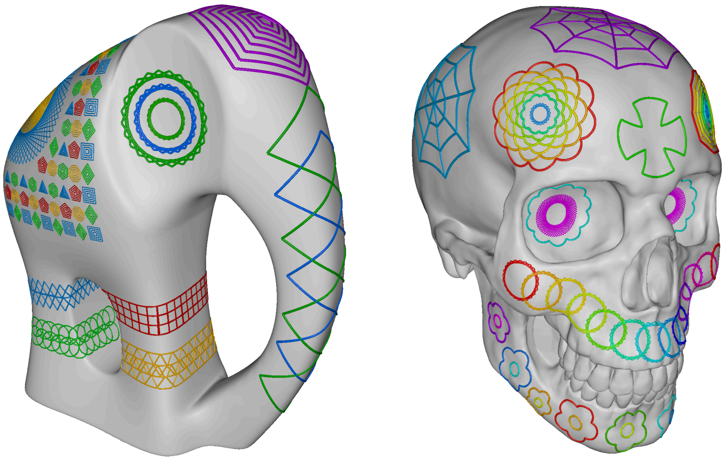


Figure 6.1: Examples of drawings obtained interactively with our prototype system on two meshes, each consisting of 1M triangles.

isoline of the distance field is not even guaranteed to be homeomorphic to a circle.

In order to address the topological limitations, we constrain our constructions to occur on sufficiently local subsets of the surface domain. Concerning the metric aspects, the first approach suffers from *geodesic distortion*, which is caused by the curvature when mapping Euclidean geometries from the tangent plane to the surface domain. On the contrary, geodesic lines and circles are well-behaved while working directly on the surface, as long as they the are "small enough". This fact increases our leeway in imposing some local properties.

We integrate all our constructions in a prototype system that supports interactive drawing with geodesic polygons and circles. We also support affine transformations of primitives and all the usual editing operations, such as copy, paste, and delete. We achieve real time interaction on meshes consisting of up to a few million triangles. This is made possible thanks to efficient algorithms to compute geodesic distances and shortest paths.

6.2 Related work

6.2.1 Intrinsic Geometry of Surfaces

The straightedge and compass constructions rely on basic theorems of the Euclidean geometry that relate lengths and angles. When trying to define similar relations on a surface, curvature must be taken into account. This subject was thoroughly investigated in the classical theory of intrinsic geometry of surfaces. See the books by Cheeger and Ebin [CE75] and by Chavel [Cha06] for comprehensive accounts. Referring just to the cases addressed in this paper, the local version of the Gauss-Bonnet theorem (Theorem 2.33) relates the internal angles of a geodesic polygon to the curvature of the region it encloses. Such result explains the challenge in addressing constructions that require geodesic lines of given lengths *and* forming given angles. See, e.g., the isosceles triangle in 6.5.4.

Alexandrov investigated thoroughly the relations between quantities measured on a surface with their counterpart on surfaces with constant curvature (a.k.a. CAT – Cartan-Alexandrov-Topogonov – spaces) [Ale48]. In a nutshell, geodesic lines, which are cast from a common source along different directions, tend to converge if the curvature of the space is positive, and to diverge if it is negative. Based upon these facts, many comparison theorems involving Alexandrov and CAT spaces have been proposed in the literature. See Alexander et al. [AKP19] for a recent account on this subject; interestingly enough, the title of the chapter addressing geodesic triangles is *The ghost of Euclid.*

6.2.2 Vector Graphics

Vector graphics in 2D is a consolidated subject, supported in many systems and tools at industrial level [W3C10,Ado21,Ink21,Aut21,Pil21,Pix22]. Until recently, vector graphics on surfaces under the geodesic metric was considered too computationally expensive to be supported. Traditional methods to decorate a surface resort to parametrization and mapping, but this approach is prone to seams and distortion, as discussed by Nazzaro et al. [NPP22] and Yuksel et al. [YLT19]. The literature concerning tools for geodesic computations is vast, though, and has been recently surveyed by Crane et al. [CLPQ20]. Some recent contributions demonstrated that such technology is mature enough to support interactive editing directly on surfaces [MNPP22,NPP22,SC20a].

6.3 Basic Straightedge and Compass Constructions in the Plane

Straightedge and compass constructions involve just points, (segments of) straight lines, and (arcs of) circles in the Euclidean plane. They consist of iteratively applying the following five basic constructions:

- line through two existing points;
- circle through one point with center another point;
- intersection point of two non-parallel lines;
- intersection points of a line and a circle;
- intersection points of two circles.

Typical constructions usually start from few objects in the plane. In the most complex constructions – e.g., for polygons with many sides – the five operations above may be iterated many times, producing a number of intermediate objects, possibly much larger than the number of objects in the final result. Figure 6.2 illustrates the manifold counterparts of the five basic constructions

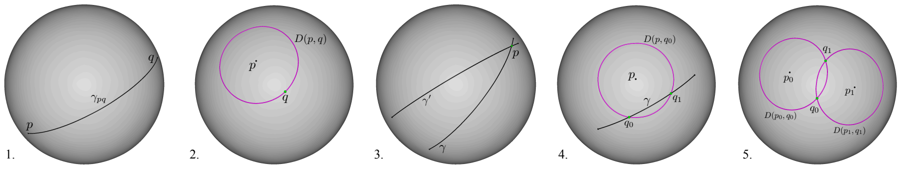


Figure 6.2: The five basic constructions on a sphere. The black curves are geodesic lines, while the curves in magenta are geodesic circles. We denote with $D(p,q)$ the geodesic circle centered at point p and passing through point q.

on a sphere: straight-line segments are substituted with shortest geodesic paths; and circles are substituted with isolines of the distance field from a point.

6.3.1 The Geodesic Arsenal

Throughout the paper, we will rely on the following primitive operations to be performed on the surface S under the geodesic metric. The implementation has been described in Chapter 3.

- *Geodesic-tracing:* given point $p \in S$ and a tangent direction $t \in T_pS$, trace a geodesic through p with tangent vector t at p; this is equivalent to a point-wise evaluation of the exponential map at p.
- *Tangent:* given a curve γ on S and one of its points p, return the direction $t \in T_pS$ tangent to γ at p; if γ is a geodesic line, this is indeed equivalent to a point-wise evaluation of the log map at p.
- *Shortest-path:* given points $p, q \in S$, return the shortest geodesic path γ_{pq} connecting them;
- *Distance-field:* given $p \in S$, compute the distance field $d_p : S \longrightarrow \mathbb{R}$ where $d_p(q) := d(p,q)$;
- *Isoline:* given the distance field d_p and a point $q \in S$ return the isoline of d_p that goes through q;
- *Intersect:* given any two lines on S, not necessarily geodesic, return their intersection points.

In the context of 6.4, we will only rely upon the first three primitives, namely the point-wise evaluation of the exp and log map and the shortest path between two points. In Section 6.5 we will also make use of the other primitives, to reproduce the straightedge and compass tools directly on the surface S.

6.4 Constructions in Tangent Space

The constructions described in this section are based on the following idea: given an initial configuration of points of S, we use the log map centered at a suited point $c \in S$ to map such points onto the tangent space T_cS. We then apply the Euclidean construction in T_cS, and finally map the result onto S through $\exp_c$. In order to preserve topological consistency, we assume all the points involved in a construction to be contained in a convex ball centered at c. Some constructions may work on a large neighborhood as well, though.

Since T_cS is a 2-dimensional vector space, we do not need any extension of the straightedge and compass tools. However, most of the properties of a given construction will be lost after applying the exponential map. Remarkably, this approach and the one described in Section 6.5 are somehow complementary: in many cases, the properties that one looses by using the former, can be preserved by using the latter, and vice-versa, Table 6.1 summarizes the results obtained with both approaches.

6.4.1 Operations with segments

Figure 6.3 shows some basic constructions in the Euclidean case, which are extended to the manifold setting in a straightforward way. These constructions are the only ones addressed in this paper, which are also insensitive to the method used to implement them. Everything works fine because they are based solely on distance and collinearity, whose properties are preserved in the manifold setting.

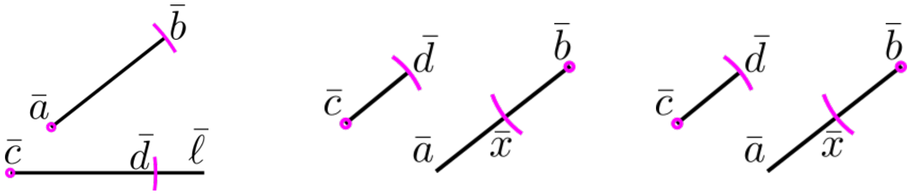


Figure 6.3: Transferring the length of a segment onto another one (*left*); adding two segments (*center*); and subtracting two segments (*right*).

Given a line segment $\bar{a}\bar{b}$ in the plane and a line $\bar{\ell}$ through another point $\bar{c}$, find a point $\bar{d}$ on $\bar{\ell}$ such that $\bar{a}\bar{b}$ and $\bar{c}\bar{d}$ have the same length. In the plane, the aperture of the compass is taken at $\bar{a}\bar{b}$, then the needle point is placed at $\bar{c}$ and a circle is traced; point $\bar{d}$ is taken at an intersection of the circle with line $\bar{\ell}$.[1] In the manifold case, we start with geodesic line segment ab and a geodesic line ℓ on S. We first lift b to a point $\bar{b}$ on the tangent plane T_aS through the log map; likewise, we lift ℓ to a radial line $\bar{\ell}$ on T_cS. Then we use a standard compass to find the length of segment $\bar{a}\bar{b}$ on T_aS; and we use the same compass to draw a circle centered at c on T_cS. We find the intersection between this circle and line $\bar{\ell}$; and finally we map the intersection point to S with the exp map.

Given two line segments $\bar{a}\bar{b}$ and $\bar{c}\bar{d}$ in the plane, extend $\bar{a}\bar{b}$ at $\bar{b}$ for a length equal to $\bar{c}\bar{d}$. In the plane, segment $\bar{a}\bar{b}$ is extended to a line with the straightedge; the aperture of the compass is taken at $\bar{c}\bar{d}$ and a circle is traced by placing the needle point at $\bar{b}$; the intersection $\bar{x}$ of this circle with the line is taken,

[1] We are assuming a non collapsible compass here; the same result can be also achieved with a collapsible compass, through a more involved procedure though.

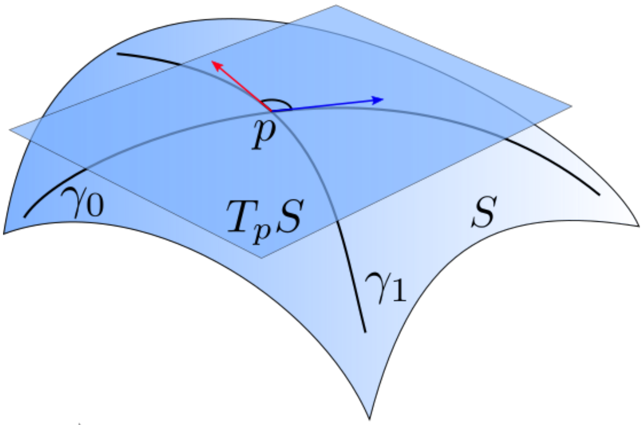


Figure 6.4: Two geodesic lines γ and γ' intersecting at point $p \in S$ form an angle defined by their tangents at p on the tangent plane T_pS (red and blue arrows)

which lies on the opposite side of $\bar{a}$ wrt $\bar{b}$; line segment $\bar{a}\bar{x}$ is the result. The construction in the manifold case is analogous to the previous and is omitted for brevity.

Given two line segments $\bar{a}\bar{b}$ and $\bar{c}\bar{d}$ in the plane, shorten $\bar{a}\bar{b}$ at $\bar{b}$ by the length of $\bar{c}\bar{d}$. In the plane, the aperture of the compass is taken at $\bar{c}\bar{d}$ and a circle is traced by placing the needle point at $\bar{b}$; the intersection $\bar{x}$ of this circle with $\bar{a}\bar{b}$ is taken; line segment $\bar{a}\bar{x}$ is the result. The construction in the manifold case is also analogous to the previous ones and is omitted for brevity.

6.4.2 Operations with angles

Let γ_0 and γ_1 be two geodesics intersecting at p; the angle between them at p is defined from their tangent directions in the tangent plane T_pS. See Fig. 6.4 for an example.

In the plane, an angle is defined by two half-lines $\bar{\ell}_a$ and $\bar{\ell}_b$ incident at a point $\bar{c}$, which can be built with the straightedge, given two points $\bar{a}$ and $\bar{b}$ lying on them, respectively. This angle can be bisected as follows. Place the needle point of the compass at $\bar{c}$, trace any circle and let $\bar{p}$ and $\bar{q}$ be its intersections with $\bar{\ell}_a$ and $\bar{\ell}_b$. Place the needle point at $\bar{p}$, and next at $\bar{q}$, with aperture $\bar{p}\bar{q}$ trace another two circles; let y be any of their two intersection points. The line $\bar{\ell}_y$ through $\bar{c}$ and $\bar{y}$ bisects the angle at $\bar{c}$. An additional property of the bisector is that all its points are equidistant from $\bar{\ell}_a$ and $\bar{\ell}_b$.

Analogously, given two geodesics γ_{ca} and γ_{cb} intersecting at c, we extend their tangent vectors at c to lines $\bar{\ell}_a$ and $\bar{\ell}_b$ in T_cS, and use the Euclidean construction to find line $\bar{\ell}_y$ as above; then we map $\bar{\ell}_y$ to a geodesic γ_{cy} emanating from c and through $y = \exp_c(\bar{y})$. Line γ_{cy} bisects the angle, in the sense that the angles formed by its tangent at c and the tangents of the two input lines γ_{ca} and γ_{cb} at c are equal, by construction. Figure 6.5 (left) illustrates this construction.

However, the points of γ_{cy} in general will *not* be equidistant from the input

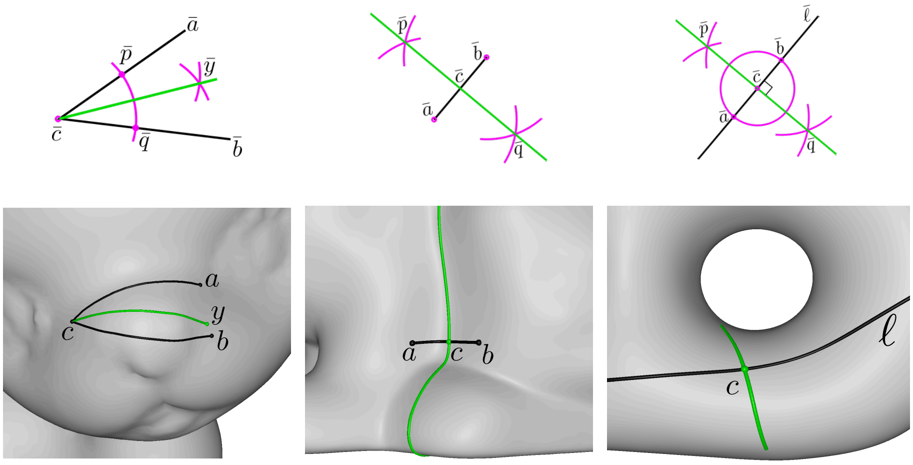


Figure 6.5: Euclidean constructions of the angle bisector(*top left*), segment bisector (*top center*) and perpendicular to a line at a point (*top right*). The corresponding constructions in the manifold case (*bottom*) are obtained by mapping the lines, which are obtained with the Euclidean constructions in the tangent plane at c, to S through the exp map.

lines. In fact, the locus of equidistant points from the two lines is not a geodesic line in general, and finding it is beyond the scope of this paper, as it requires using the distance fields from γ_{ca} and γ_{cb}, while we limit our distance fields to have their sources at single points (see Section 6.3.1).

A number of other constructions deal with operations on angles, such as copying an angle, adding or subtracting angles, or creating angles of a few specified amplitudes. These problems are somehow local to the point c at the tip of the angle, and can be addressed by finding the tangents of the geodesic lines that define the angles at play, resolving the Euclidean construction in the tangent plane, and using the resulting directions to map the geodesics to the surface S. For this reason, we do not analyze such constructions in detail.

6.4.3 Perpendicular to a line and the Square-set operator

Perpendicular bisector and midpoint

In the plane, the bisector is constructed as follows. Given points $\bar{a}, \bar{b} \in \mathbb{R}^2$, first use the straightedge to trace the straight-line segment joining them. Then place the needle point of the compass at $\bar{a}$ and the pencil point at $\bar{b}$ and trace a circle; repeat the same operation with needle at $\bar{b}$ and pencil at $\bar{a}$. Let $\bar{c}, \bar{d}$ be the intersection points of the two circles; use the straightedge to trace segment $\bar{c}\bar{d}$. The *straight line* line through $\bar{c}, \bar{d}$ intersects segment $\bar{a}\bar{b}$ *orthogonally* and *at its midpoint* $\bar{c}$; this is also the *locus of points that have equal distance* from $\bar{a}$ and $\bar{b}$. The Euclidean construction is depicted at the top of Figure 6.5(middle).

Let now a, b be two points in S and let γ be the shortest geodesic connecting them. We cannot apply the Euclidean construction in either tangent plane T_aS or T_bS and then map the result to S, as it would not have any of the above

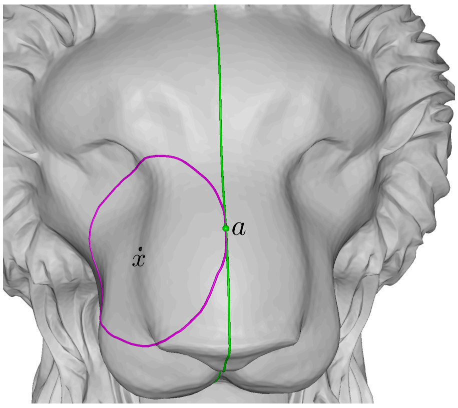


Figure 6.6: Line tangent to a circle centered at x and through a: the square set is placed at a and oriented according to the tangent of γ_{xa} at a.

properties in general, and we would also obtain different results in the two cases. We rather apply the Euclidean construction in tangent plane T_cS, where c is the midpoint of γ. In order to find c, we first find point $\bar{c}$ in the Euclidean construction on T_aS and we map it to S through the exp map; c is the midpoint of γ by construction. Now we consider the tangent t_c of γ at c, and we proceed as before to find the vector $t_c^\perp$ orthogonal to t_c. The result $\gamma^\perp$ is the geodesic line tangent to $t_c^\perp$ at c. The result is shown at the bottom of Figure 6.5 (middle).

Note that this construction satisfies just two of the three propertie of its Euclidean counterpart, since our result it is not the locus of points equidistant from a and b. In Section 6.5.2 we propose a method that constructs a curve whose points satisfy this last property, without being a geodesic though.

Perpendicular to a line at a point

In the plane, let $\bar{\ell}$ be a line and $\bar{c}$ a point on it, we want to find a line through $\bar{c}$ and orthogonal to $\bar{\ell}$. To this aim, it is sufficient to trace any circle centered at $\bar{c}$, finding its intersections $\bar{a}, \bar{b}$ with $\bar{\ell}$, and then finding the bisector of line segment $\bar{a}\bar{b}$.

Such construction can be ported to the manifold setting as above. The advantage in this case is that we already know the position of c on the geodesic γ, so we just work in T_cS. Fig. 6.5 (right) shows both constructions. The same method can be used to find the tangent at a point a to a circle centered at x and through a. This is in fact the perpendicular to geodesic segment ax and passing through a. Fig. 6.6 shows such construction.

The Square-set as derived operator

Given a curve γ on S and a point c on it, the above construction can be used to compute the vector $t^\perp \in T_cS$ orthogonal to the tangent t_c of γ at c. This procedure implements an operation that we call *Square-set*, which will be used as an atomic operation in the following.

6.4.4 Regular Polygons

In the Euclidean setting, the construction of a regular n-gon boils down to construct a straight line segment of length $\cos(\frac{2\pi}{n})$. If one starts with two points O, e on the plane, and set the length of the straight line segment Oe to be 1, then it is well known that a segment of length ℓ is constructible using straightedge and compass if and only if ℓ can be obtained from 1 using the operations $+, -, \cdot, \div$ and $\sqrt{\cdot}$. By the law of cosines, we have that the side of an n-gon inscribed in the unit circle centered at O has length $d = \sqrt{2 - 2\cos(2\pi/n)}$. Hence, if we can construct a segment of length $\cos(\frac{2\pi}{n})$, then d is constructibile. In 1796, Gauss stated a sufficient condition for $\cos(\frac{2\pi}{n})$ to be constructible, and in 1837 Pierre Wantzel proved that such condition is also necessary. The final result states that a regular n-gon can be constructed with straightedge and compass if and only if n is of the form

$$n = 2^j \cdot p_1 \cdot p_2 \cdots p_m,$$

where p_i is a prime number of the form $p_i = 2^{2^{k_i}} + 1, k_i \in \mathbb{N}$, for all $i = 1, \dots, m$. Once a chord of length d has been constructed, then we can use the compass to transfer such length $n - 1$ times on the circle. Of course, the result will be a regular n-gon because we are just duplicating the triangle of vertices $(0, 0), (1, 0), (\cos(\frac{2\pi}{n}), \sin(\frac{2\pi}{n}))$ $n - 1$ times. Note that, since $\sin(\alpha)^2 = 1 - \cos(\alpha)^2$, the constructability of $\cos(\alpha)$ implies the one of $\sin(\alpha)$.

In the manifold case, given two points $c, v_1 \in S$, if r is the length of the geodesic connecting them, then the above construction can be used to determine the n vertices $\{\bar{v}_1, \dots, \bar{v}_n\} \in T_pS$ of a regular n-gon, and then define $v_i := \exp_p(\bar{v}_i)$ for $i = 1, \dots, n$. In fact, since T_pS is a 2-dimensional vector space, we can define a system of coordinates having c as origin, an putting $\bar{v}_1 = log_c(v_1) = (0, r)$. Then the final result can be obtained by connecting v_i to v_{i+1} with shortest paths, for $i = 1, \dots, n$ (here and in the folllowing, we implicitly mean that the subscripts have to be considered modulo n). In this way, we are constructing a geodesic n-gon which satisfies just two of the properties of its Euclidean counterpart: if $t_i \in T_pS$ is the tangent of the radial geodesic γ_i connecting c with v_i at c, then the angle formed by t_i and t_{i+1} is $\frac{2\pi}{n}$, and the length of γ_i is r, $i = 1, \dots, n$. We will see that the former property is ensured by construction and the definition of angle given in Section 6.4.2 and the latter is a consequence of the fact the exponential map is a radial isometry (see Section 2.2).

The first advantage of retrieving the points $\{v_1, \dots, v_n\}$ through a construction that takes place in T_pS is that such points will be always defined. Nervetheless, the topological correctness of the result depends on the radius of injectivity of p (think about the case of a cylinder of radius R and $r \geq 2R$). Moreover, this method seems well suited to define affine transformations. In fact, rotation by angle θ and scaling by a factor λ can be applied in T_cS by choosing $\bar{v}_1 = (\cos(\theta), \sin(\theta))$ and by multiplying r by λ. Also translation is straigtforward: it suffices to parallel transport the vector $\bar{v}_1 \in T_cS$ to $T_{c'}S$ along a shortest path connecting c to c' and then repeat the construction in $T_{c'}S$, where c' is another point on S. The details on the implementation of such operations are given in Section 6.7.

We will now describe the constructions implemented in our drawing system. There are many straightedge and compass constructions for regular polygons, We limit ourselves to describe the Euclidean constructions and presenting the results obtained by mapping such constructions on S as described above. Details about the implementation of the exponential mapping and shortest paths tracing will be given Section 6.7. From now on, r will denote the length of the geodesic $\gamma(t)$ connecting two fixed points $c, v_1 \in S$. All the constructions below will take place the tangent plane T_cS, within which we define a system of coordinates having c as origin, the straight line having $log_c(v_1)$ as tangent at the origin as x-axis, and the perpendicular to such line at $\bar{c} = (0,0)$ as y-axis. In such a reference frame, we define $\bar{v}_1 := (r, 0)$ and we denote with C the circle centered at $\bar{c}$ with radius r.

EQUILATERAL TRIANGLE Let $\bar{v}_1, \bar{v}_1'$ be the points at which C intersect the x-axis, i.e $\bar{v}_1 = (r, 0)$ and $w = (-r, 0)$. Place the compass at w and trace a circle with radius r. Let $\bar{v}_2, \bar{v}_3$ be the intersections of such circle with C. Then $\{\bar{v}_1, \bar{v}_2, \bar{v}_3\}$ is an equilateral triangle.

SQUARE By proceeding as above, we construct two points $\bar{v}_1 = (r, 0)$ and $\bar{v}_3 = (-r, 0)$. The points $\bar{v}_2$ and $\bar{v}_4$ are the intersection of C with the y-axis, i.e $\bar{v}_2 = (0, r)$ and $\bar{v}_3 = (0, -r)$.

PENTAGON Let $\bar{v}_1 = (r, 0)$, $\bar{v}_1' = (-r, 0)$ and $\bar{s} = (0, r)$ constructed as above. Let $\bar{m}$ be the midpoint of the line segment $\bar{s}\bar{c}$. Place the needle of the compass at $\bar{m}$ and the pencil at $\bar{s}$ and trace a circle. Let $\bar{n}_0, \bar{n}_1$ be the intersection of such circle with the line through $\bar{v}_1'$ and $\bar{m}$. W.l.o.g, we assume $\bar{n}_0$ to be the closest one to $\bar{v}_1'$, and let us denote with r_i the distance between $\bar{v}_1'$ and n_i, $i = 0, 1$. Let C_i be the circle centered at $\bar{v}_1'$ with radius r_i, $i = 0, 1$. Then $\{\bar{v}_3, \bar{v}_4\} = C_0 \cap C$ and $\{\bar{v}_2, \bar{v}_5\} = C_1 \cap C$.

HEXAGON Let $\bar{v}_1 = (r, 0)$ and $\bar{v}_4 = (-r, 0)$ constructed as above. Place the needle at $\bar{v}_1$ and trace a circle of radius r. Let $\bar{v}_2, \bar{v}_6$ be the intersections of such circle with C. Place the needle at $\bar{v}_4$ and construct $\bar{v}_3, \bar{v}_5$ in the same way. Then $\{\bar{v}_1, \bar{v}_2, \dots, \bar{v}_4\}$ is a regular hexagon.

OCTAGON Once a square is constructed as described above, an octagon can be obtained by intersecting the angle bisector of every quadrant with C.

DECAGON Proceed as in the construction of the pentagon until the circle centered at $\bar{m}$ through $\bar{s}$ is traced. Let $\bar{n}$ the intersection with such circle with the straight line segment connecting $\bar{v}_1$ with $\bar{m}$. Let r_1 be the length of the segment $\bar{n}\bar{v}_1$ on C. Then the vertices $\bar{v}_2$ and $\bar{v}_{10}$ are the intersections of the circle C_1 centered at $\bar{v}_1$ with radius r_1 with C. The other vertices can be found

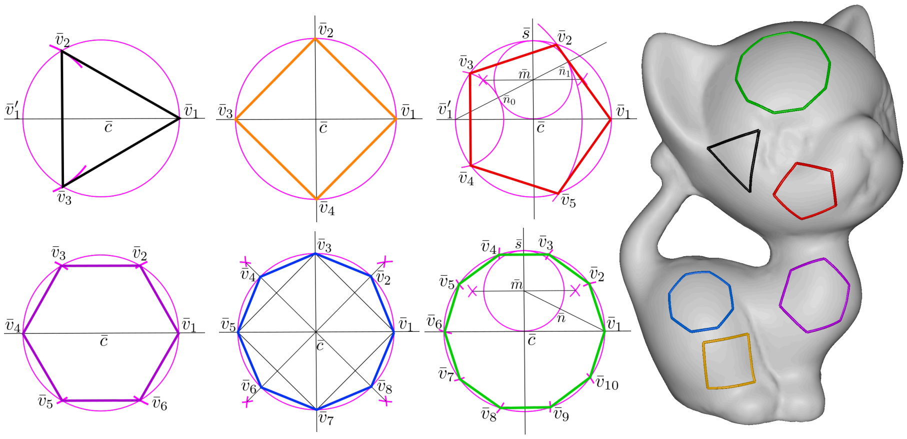


Figure 6.7: Euclidean construction of an inscribed regular n-gon for $n = 3, 4, 5, 6, 8, 10$ (*left*) and the results obtained by mapping such constructions on a mesh (*right*).

by iteratively intersecting a circle with the same radius r_1, centered at vertices found at the previous iteration.

Figure 6.14 summarizes the constructions described in this section both in the Euclidean and in the manifold setting.

6.4.5 Parallelogram, rhombus and rectangle

With the notations used in the previous section, let $\bar{v}_1 = (r, 0)$ and $\bar{v}_3 = (-r, 0)$. Place the needle at the origin and trace any circle, which we will denote with $\widehat{C}$. Pick any point $\bar{v}_2$ on $\widehat{C}$ and consider the line through $\bar{v}_2$ and $\bar{c}$. If $\bar{v}_4$ is the other point at which this line intersects $\widehat{C}$ then $Q := \{\bar{v}_1, \bar{v}_2, \bar{v}_3, \bar{v}_4\}$ is a parallelogram.

Note that if $\bar{v}_2 = (0, R)$, with R being the radius of $\widehat{C}$, then Q is a rhombus, while if $\bar{v}_2$ does not lie on the y-axis but $R = r$, then Q is a rectangle. In the particular case in which $\bar{v}_2 = (0, r)$ then Q is a square. See Figure 6.8.

Also in this case, we have no guarantees of equal length of opposite sides or equal angles at opposite corners; let alone the notion of "parallel sides", which is ill defined on a manifold. The only guarantee is that opposite semi-diagonals lie on a geodesic through c and have equal lengths, and, consequently, opposite angles at the center are equal. The rhombus has the additional property that the diagonals are orthogonal. And the rectangle has all four semi-diagonals with the same length.

In summary, all constructions above can guarantee only properties related to lengths and angles that depend just on the radial geodesics emanating from the center c, at which the tangent plane is placed. This is a consequence of the fact the exponential map is a radial isometry, and that the angles between two geodesics are defined in the tangent space of their interesection. Note that the sides of the geodesic polygons are traced only *after* their corners have been mapped to S through the exp map. The length of such lines, as well as the angles they form at the corners, are influenced from the Gaussian curvature

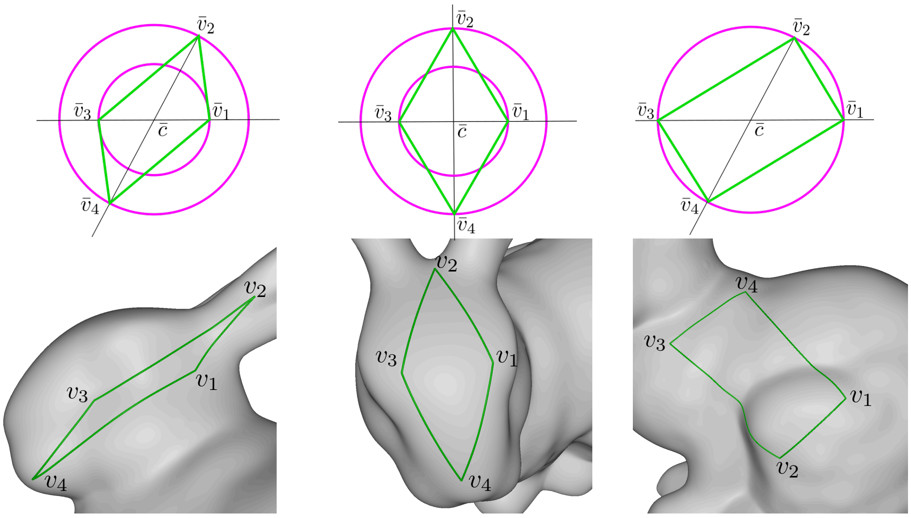


Figure 6.8: Euclidean constructions of a parallelogram, a rhombus and a rectangle (*top*) and the results obtained by mapping such constructions to a mesh (*bottom*).

of S in the region covered by the polygon: the more the Gaussian curvature around c varies, the more the shape of the geodesic polygon will differ from its Euclidean counterpart.

6.4.6 Remarks

Overall, the constructions of inscribed polygons cannot ensure any property concerning the length of their sides as well as their internal angles. However, some considerations can be made about both quantities in order to understand how much the curvature of the surface affects the shape of a regular n-gon obtained with the above constructions. For the sake of brevity, we restrict ourselves to a high-level discussion, with the purpose of just giving an idea of what kind of results may be used to better understand how the shape of our polygons may be influenced by the curvature around the center c. For more details about such result, we refer to [CC89, pages 197-198] and [Ber07, Sec. 6.4].

With the notations used above, let consider two points $\bar{v}_1, \bar{v}_2 \in T_c\mathcal{M}$ picked on the circle C centered at $\bar{c}$. Let T be triangle having vertices $\{\bar{c}, \bar{v}_1, \bar{v}_2\}$ and $\mathcal{T}$ the one having vertices $\{c, v_1, v_2\}$. Then both T and $\mathcal{T}$ have two sides of length r and the angle formed by such sides is equal to $2\pi/n$. However, T is a plane triangle, while $\mathcal{T}$ is not. So one could say that the all the differences between these two triangles are somehow localized in the length ℓ of the geodesic connecting v_1 to v_2 and in the angles α_1, α_2 at such vertices. For what said in Section 2.3.2, it is clear ℓ depends on how much $\exp_c$ fails in being an isometry, and, therefore, it can be estimated by considering a Jacobi field along γ_{cv_1}. In the case of manifolds with constant curvature, Jacobi fields have a closed form, which have been used to prove the well known *Toponogov triangle comparison theorem.*

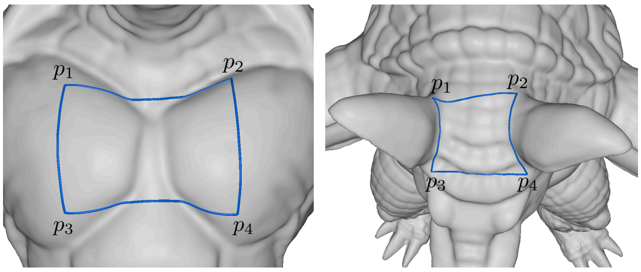


Figure 6.9: Two examples of squares drawn in bumpy regions of a mesh. In some cases, the geodesic distortion is more evident when the length of the sides is considered (left), in other instances we better notice it by looking at the angles (right).

After introducing some notations, we will state this latter result in our specific setting and refer to [Ber07] for further details about this subject. In the following, we assume $r < r_p$. If K denotes the Gaussian curvature on S, let us put

$$\delta := \inf K \qquad \Delta := \sup K,$$

and let us denote with $S(\sigma)$ the surface having constant curvature σ. Let T_δ and T_Δ be the triangles with vertices $\{c', v_1', v_2'\} \in S(\delta)$ and $\{c'', v_1'', v_2''\} \in S(\Delta)$, respectively. Suppose these triangles have two sides of length r and the angle between such sides is $2\pi/n$. Then, by the Toponogov triangle comparison theorem we have

$$d_{S_\Delta}(v_1'', v_2'') \leq \ell \leq d_{S_\delta}(v_1', v_2'),$$

where $d_{\mathcal{M}}(\cdot, \cdot)$ denotes the geodesic distance measured with the metric of the manifold $\mathcal{M}$. This means that the more K varies around c, the more ℓ could differ from $\|v_2 - v_1\|$. Concerning the angles at p_1 and p_2, by re-writing (2.23) for the case $k = 2$ we have that

$$\sum_{i=1}^{3} \alpha_i = \iint_{\mathcal{T}} K d\sigma + \pi,$$

where $\alpha_0, \alpha_1, \alpha_2$ are the angles at c, v_1, v_2, respectively. This means that the sum of the internal angles of $\mathcal{T}$ differs from π by an amount which is equal to the integral of the curvature in its interior. To fix ideas, one can think about the case in which $\mathcal{M}$ is a unit sphere. Then $K \equiv 1$ and the above formula tells us that the excess of $\sum_{i=1}^{3} \alpha_i$ over π is equal to the area of $\mathcal{T}$.

Summarizing the above considerations, we can say that the more the surface is far from being flat, i.e. the more the Gaussian curvature is great in norm, the more the shape of $\mathcal{T}$ would differ from the one of its Euclidean counterpart T. Figure 6.9 show two examples of two squares drawn in bumpy regions of a mesh, which look very different from the one shown in Figure 6.7.

6.5 Direct Constructions on the Surface

We now change approach, by defining the equivalent tools for the straightedge and compass directly on S. Referring to the geodesic arsenal defined in Section

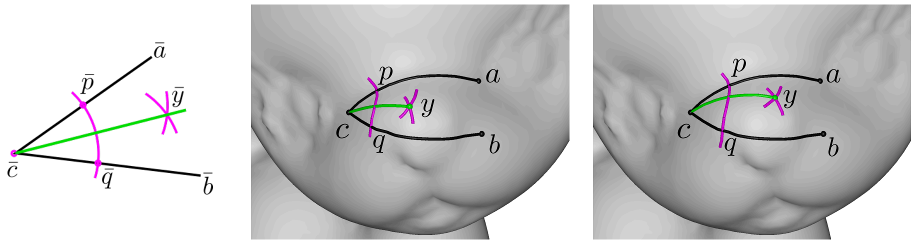


Figure 6.10: The Euclidean construction to bisect an angle (*left*) fails when ported to a surface: the resulting lines (*center* and *right*) do not bisect the angle at c and they are different depending on the choice of points p and q.

6.3, the operator *Shortest-path* allows us to trace geodesic segments between any two endpoints; and the joint use of *Tangent* and *Geodesic-tracing* allows us to extend such a segment indefinitely from both sides. We thus define the derived operation *Geodesic-line* that traces an arbitrarily long line through a pair of points, generalizing the straightedge to the manifold setting.

Likewise, the *Geodesic-compass* is a derived operation defined as the *Isoline* through a given point of the *Distance-field* from another center point. Note that, the *Distance-field* alone does not belong to the straightedge and compass framework, because it implicitly takes measures. On the other hand, since this operator is anyhow necessary to implement the *Geodesic-compass*, we will use it also directly to address constructions where the basic tools fail.

We address the five basic constructions listed in Section 6.3 by means of *Geodesic-line* (1); *Geodesic compass* (2); and *Intersect* (3, 4, 5), as depicted in Figure 6.2. Besides, we will make use of the *Square-set* operator, as defined in Section 6.4.3.

Since we need topological consistency between the result of our constructions on S and their Euclidean counterpart, we will always assume that we are considering strongly convex objects in the sense of Section 2.4.

In the following, we review some straightedge and compass constructions, showing their extension to the manifold setting with this approach, as an alternative to the constructions in tangent plane presented in the previous section. Again, we will see that these constructions can preserve only some of the properties that are guaranteed in the Euclidean case.

6.5.1 Angle bisection

The Euclidean construction depicted in Figure 6.5 (top-left) fails when ported to a surface with our geodesic tools. See Fig. 6.10. The geodesic line through c and y neither bisects the angle at c, nor its points are equidistant from the input lines. Moreover, the result depends on the radius chosen to find p and q.

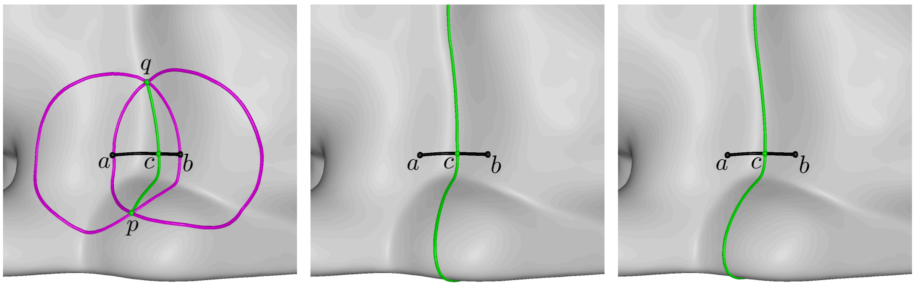


Figure 6.11: The bisector of a geodesic segment computed by reproducing the Euclidean construction (*left*); by the zero isoline of the difference of distance fields from a and b (*center*); and by tracing a geodesic from the midpoint of the segment along the orthogonal direction computed with the *Square-set* (*right*). The last construction is equivalent to the one in Section 6.4.3.

6.5.2 Line segment bisector and midpoint

The Euclidean construction described in Section 6.4.3 also fails on a surface. If we use *Geodesic-line* and *Geodesic-compass* to obtain points p, q, the two geodesic paths γ_{ab} and γ_{pq} in general will not intersect at the midpoint of γ_{ab}, nor they will be orthogonal at c. Concerning distances, we only know that p and q are equidistant from a and b, but distances can be different at all other points of γ_{pq}. See Figure 6.11 (left).

We thus resort to our additional tools. Let d_a, d_b be the two distance fields with sources at a and b, respectively. Compute the difference field $d_{ab} = d_a - d_b$; the point p computed before belong to the zero isoline of this field. If we extract the *Isoline* of d_{ab} through p, the resulting line $\gamma_{ab}^{\perp}$ will intersect orthogonally γ_{ab} at its midpoint. See Figure 6.11 (center). This construction has the further property, which we did not have with the construction in Section 6.4.3, that all points of $\gamma_{ab}^{\perp}$ are equidistant from a and b. However, $\gamma_{ab}^{\perp}$ is not a geodesic line, hence not *straight* in the manifold sense. Finally, the construction presented in Section 6.4.3 can be replicated by first finding the midpoint c of geodesic γ_{ab}, as above, and then applying the *Square-set* operator at c to find the perpendicular line. See Figure 6.11 (right).

6.5.3 Circle through three non-collinear points

In the plane, given three non-collinear points $\bar{a}, \bar{b}, \bar{c}$, this construction can be done by first computing the perpendicular bisectors of segment $\bar{a}\bar{b}$ and $\bar{b}\bar{c}$; then intersecting such two bisectors at point $\bar{o}$; and finally tracing the circle centered at $\bar{o}$ and through $\bar{a}$ (and, consequently, through $\bar{b}$ and $\bar{c}$). See Figure 6.12(left). The same procedure trivially gives the circle circumscribed to a triangle $\bar{a}\bar{b}\bar{c}$.

This construction relies on the fact that all points on a bisector are equidistant from the endpoints of the input segment, a property which is not fulfilled in the manifold case when the bisector is a straight line, as in Figure 6.12(middle).

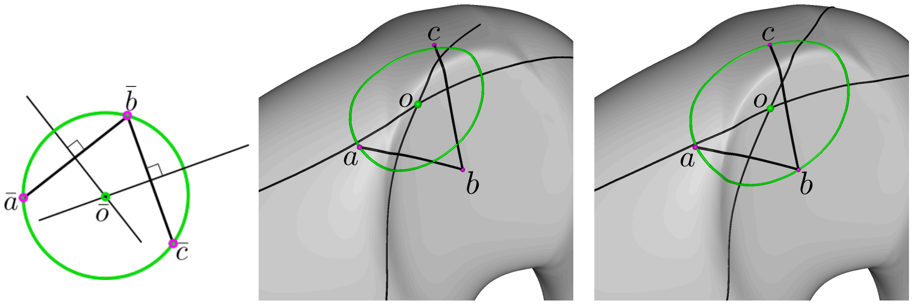


Figure 6.12: Euclidean constructions of a circle through three points a, b, c (*left*). A straightforward reproduction of the Euclidean construction fails because the intersection o of the two thin black lines is not equidistant from a, b, c (*center*). The intersection of curves obtained as isolines of the difference distance fields from pairs of points gives the correct center of the geodesic circle (*right*).

However, if the two bisectors are obtained as isolines of the difference distance field, as described above, then their intersection will indeed be equidistant from the three points, hence we can use it as the center for a geodesic circle through them. See Figure 6.12(right).

Note that this construction cannot be replicated while working in tangent space, because one would need to know the center o of the circle in advance.

6.5.4 Polygons

We already observed that the constructions in Section 6.4.4 and Section 6.4.5 do not ensure any property concerning the length of the sides and/or the amplitude of the internal angles of the resulting polygons. We now address some of those properties with direct constructions on S. To this aim, we rely on different Euclidean constructions, which do not work inside a circle.

Triangles

A triangle can be copied to another place with the same construction, both in the planar and in the manifold setting. Let $\bar{a}\bar{b}\bar{c}$ be a triangle, $\bar{\ell}$ a line and $\bar{a}'$ a point on $\bar{\ell}$. We want to copy the triangle in such a way that $\bar{a}$ goes to $\bar{a}'$, $\bar{b}$ goes to a point $\bar{b}'$ on $\bar{\ell}$, and $\bar{c}$ is placed at a point $\bar{c}'$ accordingly. We first draw a circle with amplitude $\bar{a}\bar{b}$ centered at $\bar{a}'$ and we select a point $\bar{b}'$ as one of the two intersections of the circle with line $\bar{\ell}$. Next we trace two more circles, one with amplitude $\bar{a}\bar{c}$ centered at $\bar{a}'$ and another with amplitude $\bar{b}\bar{c}$ centered at $\bar{b}'$; we select point $\bar{c}'$ as one of the intersections of such two circles. In the manifold setting, the result is a triangle with edges of the same length of abc, but nothing can be said about its angles. Moving a triangle while preserving the amplitude of its angles is inherently impossible in general, for consequences of the Gauss-Bonnet theorem.

Creating an equilateral triangle is among the simplest constructions: given an edge $\bar{a}\bar{b}$, intersect the two circles with radius $\bar{a}\bar{b}$ and centered at $\bar{a}$ and $\bar{b}$,

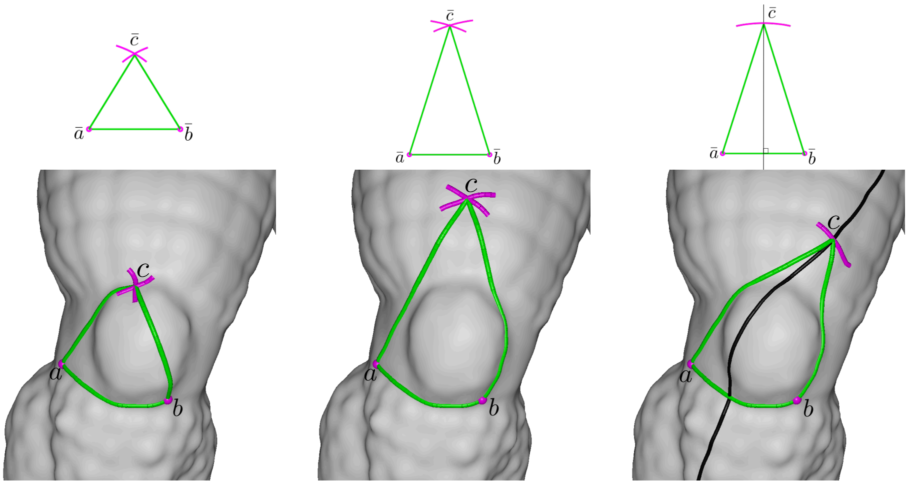


Figure 6.13: Straightedge and compass constructions of an equilateral (*left*) and isosceles triangle given the length of the sides (*center*) and the height (*right*) in the Euclidean (*top*) and manifold setting (*bottom*).

respectively. Any of their two intersections can be chosen as the third vertex $\bar{c}$ of the triangle. The same procedure works in the manifold setting too, if we aim at obtaining a triangle with three edges of the same length. This does not guarantee any other of the properties of the equilateral triangles, e.g., having three equal angles, having three equal heights that bisect the angles and bisect the edges, etc. Constructions fulfilling even one of such requirements seem not easy to obtain in the manifold setting.

Likewise, it is easy to build an isosceles triangle on a basis $\bar{a}\bar{b}$ with the diagonal edges of a given length (transferred with the compass from some given segment). Alternatively, one can build an isosceles triangle of a given height, by first constructing the perpendicular bisector of $\bar{a}\bar{b}$ and then transferring the height on it with the compass. Both such constructions work to some extent in the manifold setting, too. However, the first construction will not warrant anything about either equality of the angles at the basis, or the height from c to bisect ab. While the second construction will just warrant the latter property, but neither that the diagonal edges, nor that the angles at the basis are equal. In our system, we implemented a more practical, yet equivalent, variant of the first construction: we consider the *Isoline* of points equidistant from a and b, as in Section 6.5.2, and we let the user choose the length of the sides by dragging point c along such bisector. Figure 6.13 (bottom) shows Euclidean constructions for equilateral and isosceles triangles, together with their counterparts on a surface.

Squares and rectangles

A square can be built from one of its edges $\bar{a}\bar{b}$ as follows. A line perpendicular to $\bar{a}\bar{b}$ and through $\bar{a}$ is built first. Then the the length of $\bar{a}\bar{b}$ is transferred to segment ad on such a line by placing the needle point of the compass at $\bar{a}$.

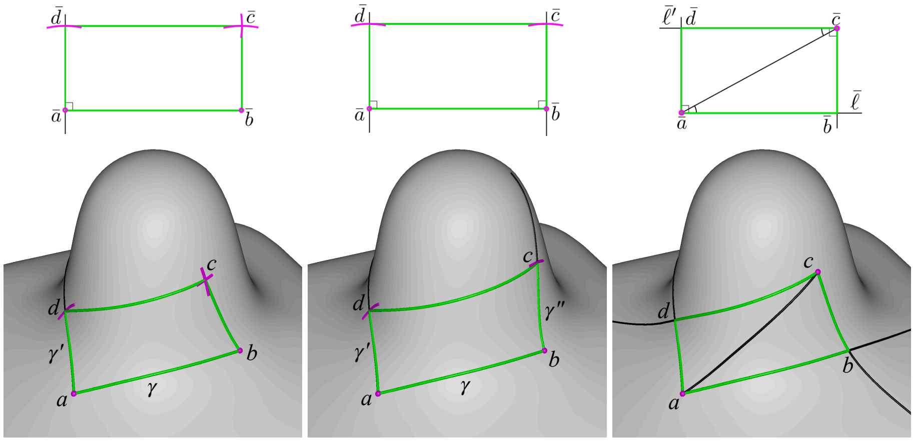


Figure 6.14: Rectangles obtained with different constructions: by tracing two perpendicular lines γ and γ' intersecting at a and tracing opposite sides of the same length (*left*); by tracing two lines γ' and γ'' perpendicular to γ at a and b and setting points d and c on γ' and γ'' at equal distance from a and b, respectively (*center*); by tracing the diagonal ac, transferring angle bac to acd and tracing two lines perpendicular to ab and cd at a and c, respectively (*right*). The constructions are shown both in the Euclidean (*top*) and in the manifold (*bottom*) setting.

Finally, the needle point of the compass is placed at $\bar{b}$ and at $\bar{d}$ with the same aperture $\bar{a}\bar{b}$, and the intersection $\bar{c}$ of the two circles gives the last vertex of square $\bar{a}\bar{b}\bar{c}\bar{d}$.

This same construction works in the manifold setting too. However, the resulting polygon will have four edges of equal length, but only angle $d\hat{a}b$ is guaranteed to be a square angle. An alternative construction consists of tracing perpendicular lines at both a and b, by means of the *Square-set*, transferring the length of ab on both of them, and connecting the points c and d obtained in this way. In this case, in the manifold setting we obtain a quadrilateral with three edges of the same length, namely ab, ad and bc, and two right angles $d\hat{a}b$ and $a\hat{b}c$; but nothing can be said about the length of edge cd and the amplitude of angles at c and d.

The same constructions apply to draw a rectangle, except that the aperture of the compass to obtain the vertical edges can be different than the length of ab. The outcome in the manifold setting has the analogous (lack of) properties.

We describe a third construction, which is more appropriate to the GUI of drawing systems. Given a basis line $\bar{\ell}$ and a point $\bar{a}$ lying on it, a diagonal segment $\bar{a}\bar{c}$ is traced first. Then the angle between such segment and line $\bar{\ell}$ is transferred at $\bar{c}$, to obtain a line $\bar{\ell}'$ parallel to $\bar{\ell}$. Finally, two lines are traced through $\bar{a}$ and $\bar{c}$, which are perpendicular to $\bar{\ell}$ and $\bar{\ell}'$, respectively. The intersections of such lines with the first two lines give the other two vertices $\bar{b}$ and $\bar{d}$ of the rectangle. This construction applies to the manifold setting, too, by copying the angle in the tangent planes, as described in Section 6.5.1, and using the square set to trace perpendicular lines. However, the resulting quadrilateral has two square angles at a and c, but nothing can be said on

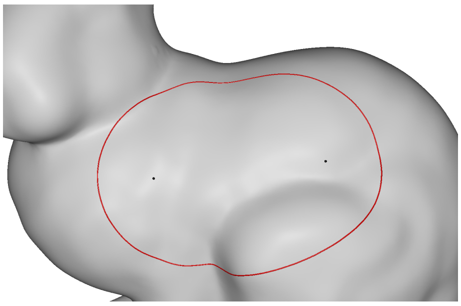


Figure 6.15: Ellipse (red curve) computed as an isoline of the sum of the distance fields from its foci (black dots).

the amplitude of the other two angles, and opposite edges are not congruent in general. A number of other constructions can be devised, which are all equivalent in the Euclidean setting, while none of them can warrant congruent opposite edges and four right angles. Each such construction privileges some of the properties of rectangles, at the expense of others.

Figure 6.14 shows examples of rectangles obtained with the three constructions described above.

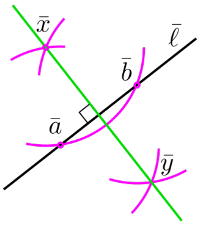
6.5.5 Ellipse

An ellipse cannot be constructed using the straightedge and compass. The best one can do is to compute the position of a point on the ellipse, using the so called *de La Hire*'s construction. However, since the ellipse can be defined as an isoline of the sum of the distance fields from its foci a and b. In details, once the foci have been chosen, we use the *Isoline* operator to compute the isoline of the field $f = d_a + d_b$ equal to $\alpha\ell$, where ℓ is the distance between the two foci and α is a scaling factor. We therefore added this primitive to our drawing system for practical reasons. An example is shown in Fig.6.15.

6.6 Unresolved Contructions

We could not find a straightforward way to port further constructions to a surface by relying just on the basic tools available in our arsenal. For completeness, we briefly discuss some such constructions, which may be relevant in the applications, leaving their investigation to future work.

Perpendicular to a line through a point not on the line



This is a basic construction, which is also useful in the context of more complex constructions in the Euclidean plane. Given a line $\bar{\ell}$ and a point $\bar{x}$ not on the line, find a line through $\bar{x}$ and perpendicular to $\bar{\ell}$. In the plane, we trace a circle centered at $\bar{x}$, with an aperture larger than its distance from $\bar{\ell}$; we find the intersection points $\bar{a}, \bar{b}$ of this circle with $\bar{\ell}$; and we trace another two circles centered at $\bar{a}$ and $\bar{b}$ with the same aperture. The result is the line through $\bar{x}$ and $\bar{y}$.

In the geodesic setting, the orthogonal projection of a point x onto a geodesic γ in general will not be the midpoint of the segment intercepted on γ with a circle centered at x. We rather have to define the problem in terms of distances: if point z on γ minimizes the distance from x, then the geodesic path γ_{xz} meets γ orthogonally at z, because it is a radial path of the circle centered at x and tangent to γ.

This problem could be tackled by computing the distance field from γ (which is not part of our arsenal, though) and evaluating it at x: the geodesic circle centered at x with radius $d_\gamma(x)$ is tangent to γ at z. Alternatively, one could restrict the distance field d_x to γ and find its minimum along it. Notice that both such solutions take measures, thus violating the rules of the straightedge and compass framework. A possible workaround consists of growing a geodesic circle centered at x until it becomes tangent to γ. The radius of the circle can be halved or doubled by relying on the basic constructions and a bisection technique can be followed. A similar problem is mirroring a point x about a line γ not containing it. Once we have found the projection z of x on γ, it is sufficient to trace a circle centered at z and through x and then find the intersection between such circle and the geodesic line through x and z.

Parallel lines

A number of constructions in the plane deal with parallel lines. In the manifold setting, the concept itself of parallel lines is ill-defined. Given a geodesic line γ_x and point on $x \in \gamma_x$, the tangent t_x of γ_x in x belongs to the tangent plane T_xS, and it is well defined its *parallel transport* to the tangent plane T_yS of another point $y \in S$. The parallel transport is a fairly complex operation that we have not considered in our preliminaries. Once the parallel transported direction t_y is given, we could trace the geodesic through y tangent to t_y and consider it "parallel" to γ_x. The trouble here is, that the direction t_y will be different depending both on the starting point x on γ_x, and on the trajectory that we choose to transport t_x to T_yS. Therefore, the result is not unique and it is somehow arbitrary. Another straightforward possibility it to take a reference line γ and define a bundle of "parallel" lines as all those lines that intersect γ with a given angle. Given points $x_0, \ldots, x_n$ along γ and the reference angle, it is possible to use the construction in Section 6.5.1 to trace such parallel lines through the points x_i. Note that, even if we were allowed to take distances, the locus of points that have a given distance from a geodesic line γ consists of two curves that in general are *not* geodesic lines. Addressing this problem thus requires first a robust notion of parallelism on a manifold.

	Tangent Space	**Geodesic Tools**	
Angle Bisection	is a geodesic ✓ bisects the angle ✓ is equidistant from sides ✗	is a geodesic ✓ bisects the angle ✗ is equidistant from sides ✗	
Segment Bisector	is a geodesic ✓ bisects the segment✓ is orthogonal ✓ is equidistant from endpoints ✗	is a geodesic ✗ bisects the segment ✓ is orthogonal ✓ is equidistant from endpoints ✓	
Circle through 3 points	✗	✓	
Isoscele Triangle	✗	2 equal sides ✓ apex belonging to the perpendicular bisector of the base ✗	2 equal sides ✗ apex belonging to the perpendicular bisector of the base ✓
Equilateral Triangle	equal sides ✗ equal angles ✗ radial geodesics of the same length ✓ angles at the center of 120° ✓	equal sides ✓ equal angles ✗ radial geodesics of the same length ✗ angles at the center of 120° ✗	
Square	equal sides ✗ equal angles ✗ radial geodesics of the same length ✓ angles at the center of 90° ✓ diagonals intersect at their midpoints ✓	4 equal sides ✓ 1 right angle ✓ radial geodesics of the same length ✗ angles at the center of 90° ✗ diagonals intersect at their midpoints ✗	2 equal sides ✓ 2 right angles ✓ radial geodesics of the same length ✗ angles at the center of 90° ✗ diagonals intersect at their midpoints ✗
Rectangles	equal sides ✗ equal angles ✗ radial geodesics of the same length ✓ diagonals intersect at their midpoints ✓	4 equal sides ✓ 1 rigth angle ✓ radial geodesics of the same length ✗ diagonals intersect at their midpoints ✗	2 equal sides ✓ 2 right angles ✓ radial geodesics of the same length ✗ diagonals intersect at their midpoints ✗
Polygons	equal sides ✗ equal angles ✗ radial geodesics of the same length ✓ equal angles at center (only regular ones) ✓	✗	

Table 6.1: Summary of the supported constructions, with the properties preserved by the algorithms described in Sec. 6.4 (Tangent Space) and in Sec. 6.5 (Geodesic Tools). Where more than one construction is available, we report the main differences between them splitting the corresponding columns. In most cases, the properties that fail with one approach are preserved by the other.

More constructions

Several other constructions exploit relations between angles and distances, which do not hold in the manifold case. For this reason, we could not find a straightforward way to reproduce such constructions in terms of our geodesic arsenal:

- Tangents to a circle through an external point: the construction in the plane is based on the fact that an angle at the circumference in a half circle measures $\pi/2$. This is no longer true in the manifold case.
- Circle inscribed in a triangle: the construction in the plane is based on the fact that all points in the bisectors of angles are equidistant from the edges. This is no longer true in the manifold case. It is not clear how the locus of points that are equidistant from two edges can be constructed, unless the distance fields from the edges can be computed (see also the discussion in Sec. 6.5.1).

Note also that constructions like the trisection of an angle or doubling the volume of a cube are not possible using the straightedge and compass while they can be achieved by using a marked ruler. This suggests that in the manifold case, too, more constructions could be supported be allowing the explicit computation of distances from curves.

6.7 Implementation

All the constructions described in the previous sections, which apply to the manifold setting, have been implemented by means of the primitives defined in Section 6.3 and included as an extension of an existing library [PNC19]. We have developed a prototype system that supports their interactive usage on meshes up to the size of millions of triangles. We refer to Chapter 3 for the definition of basic concepts such as tangent spaces and parallel transport, as well as the description of the algorithm for geodesic paths and distances computation.

DATA STRUCTURES The surface S is represented with a piecewise flat triangular mesh M, which is represented with an indexed data structure – i.e., encoding a list of vertices V and a list of triangles F – augmented with triangle-to-triangle adjacencies to support mesh navigation.

A mesh point p is encoded as a triple (t, α, β) where t is the index of the triangle containing p, and α, β are two barycentric coordinates of p in t (while the third barycentric coordinate is computed by difference to the unit).

A curve γ on S is discretized as a polyline having vertices at all intersections with edges of M. A curve connecting points p and q is encoded with a strip of triangles $(t_0, \dots, t_h)$ of M, where t_0 and t_h contain p and q, respectively, and an array of scalars $(l_0, \dots, l_{h-1})$, where l_i encodes the intercept of the polyline with the edge common to t_i, t_{i+1} parametrized along such edge.

TANGENT Let γ be the shortest path connecting two points q_0, q_1, represented as described above. Given the representation of polylines described above, the tangent vector w_p at a point p on γ is computed as follows. If p lies in a triangle t_i then w_p belongs to the plane containing t_i, and it is computed as $w_p = p_i - p$, with $p_i := (1-l_i)v_0 + l_i v_1$ where v_0, v_1 are the endpoints of the edge shared by t_i and t_{i+1}. If p belongs to an edge, we proceed in a similar way: the only difference is that w_p will be of the form $p_{i+1} - p_i$, with obvious meaning of the notations. If p is a vertex, we first compute w_p as before, obtaining a vector defined in the plane containing a triangle t_i in the one-ring of p. Then we map w_p to the tangent space of p in the same way we have mapped its neighbors.

ISOLINE We linearly interpolate a field inside each triangle of M. For each triangle t, which crosses a given isovalue, the segment of isoline crossing t is computed independently. While linear interpolation is good enough on high resolution meshes, it might be too rough on coarse meshes. Better results can be achieved by supersampling the polyline while using a more accurate estimate of the distance field inside t. For instance, isolines can be approximated as arcs of circle in the plane containing t, where the center of the circle is estimated on the basis of the values of the distance field at the vertices of t.

INTERSECT Lines on M are encoded as paths, as described before. Intersections between a pair of lines are found in linear time in the total number of triangles in the corresponding paths. Each triangle intersecting one of the paths is assigned a unique tag; next the triangles forming the other path are scanned, and intersections are computed just at tagged triangles.

CONVEXITY BALLS As already remarked, we assume that our constructions occur within a convex set. We thus provide an algorithm to test the radius of convexity about a given point p. This is computed by considering the largest ball B centered at p within which the Hessian of $d_p^2(x)$ is definite-positive for every $x \in B$. This boils down to test the positive-definiteness of $\left(\mathrm{Hess} d_p^2)_{ij}\right)$ by checking its eigenvalues while applying a growing-region procedure from p. Note that, while a test of convexity guarantees correctness, in practice many constructions may work well also on larger neighborhoods. Since all constructions are interactive, we leave freedom to the user to apply them over arbitrarily large regions.

ROTATION, TRANSLATION, SCALING In order to support interaction, we allow the user to edit a drawing by translating, rotating and scaling geometric objects over the surface. Geometric transformations are applied to the control points that define our constructions, while the objects are generated each time from the updated points. Given an anchor point p, we use the log map to represent all control points of the object at hand in the tangent plane of p. This is implemented point-wise by evaluating the shortest paths between p and each such point, and finding the tangent of each path at p. Rotation and scaling are implemented trivially, by changing one of the polar coordinates of the points in tangent space: the angle for rotation and the distance for scaling. Then we map the updated points to the surface with the exp map, which is implemented point-wise by tracing geodesic lines from p either in the updated directions, or with updated lengths. Translation consists of dragging the anchor point while parallel transporting the reference frame of its tangent space along the trajectory. Upon dragging, the control points are regenerated likewise from the transported frame.

For most constructions described in Section 6.4, a natural choice for the anchor point is the center c where we locate the tangent space for the construction. For the remaining constructions, we select as anchor point one of the control points participating in the construction.

MACROS We provide some macro-operations, which combine different primitive constructions to obtain complex decorations at once. Some examples of macros are shown in the decorations in Figure 6.1. For instance: a *wreath* is obtained by multiple instances of a polygon rotated about the same center; similarly, we allow the user to draw *nested shapes* like circles or polygons; a *flower* is built by drawing arcs of circles centered at the vertices of a polygon and trimming them at their intersections. The *cross* and the *spider net* on the skull are also generated by macros that intersect circles. Macros are controlled interactively with simple parameters that tune, e.g., the number of polygons forming a wreath, the number of nested shapes, the number of petals in a flower, etc.

6.8 Concluding remarks

We have presented two approaches – namely, constructions in tangent space and direct constructions on the surface – to port straightedge and compass constructions to the manifold setting. It follows from our analysis that not all constructions can be ported successfully, and also those that can be ported may guarantee only *some* of the properties they have in the Euclidean case. We extended the scope of basic constructions by exploiting our *Distance-field* operator beyond the limitations of the straightedge and compass framework, yet remaining compatible with it, since we neither take explicit measures nor do arithmetic computations. The constructions we propose already support several operations in the context of interactive vector graphics on surfaces.

A few relevant constructions are still not supported. Such operations may require explicit measures, which are forbidden in the straightedge and compass framework and may require further tools beyond our geodesic arsenal, such as computing the distance field from a curve.

A further challenge is extending our primitives to work over larger regions. However, even basic properties of lines and circles can be lost outside strongly convex regions. See Figure 2.8 for some examples. Some operations have been addressed already in the literature, including primitives that can be computed with distance fields [NPP22], and Bézier splines [MNPP22].

A relevant limitation, stemming from the intrinsic curvature of surfaces, is the impossibility to warrant the congruence of both lengths and angles together. Regular tilings, which are hard to apply because of this limitation, can be addressed by relaxing some conditions on angles and/or lengths, but they remain challenging to extend over large regions. This problem is tightly related to the design of N-RoSy fields [VCD+17], in particular to the presence of field singularities, which cannot be avoided, as a consequence of the Gauss-Bonnet theorem.

A possible avenue is to relax the constraint of lines to be straight, in geodesic terms, trading some straightness for other properties. This leads to the concept of *as-straight-as-possible* lines under given constraints, e.g., joining their endpoints with a prescribed length or with given tangent directions. This approach entails investigating Jacobi fields [PHD+10, Le 19] and related optimization problems.

We plan to address the above challenges in future work.

7

b/Surf: Interactive Bézier Splines on Surface Meshes

This chapter includes contents from a co-authored paper [MNPP22] that has been re-formatted for this thesis.

7.1 Introduction

Bézier curves are the building blocks of most vector graphics packages, since most other primitives can be converted into *Bézier splines* (chains of Bézier curves) and edited as such [Far01].

In many design applications, it would be beneficial to edit vector graphics directly on surfaces, instead of relying on parametrization or projections that have inherent distortions [NPP22, PSOA18]. Yet, bringing vector graphics to surfaces is all but trivial, since basic rules of Euclidean geometry do not hold under the geodesic metric on manifolds; and distances, shortest and straightest paths cannot be computed in closed form, as seen in the previous chapter. In particular, in spite of several attempts to define curves under the geodesic metric, a complete computational framework that supports their practical usage in an interactive design setting is still missing.

In this work, we present the first *practical* method, which supports the *interactive* and *robust* design and editing of Bézier splines on high resolution meshes, *without any limitation on the position of their control points.* For the sake of simplicity, we restrict our study to cubic Bézier curves. Our contributions tackle different aspects of the problem, as outlined in the following.

7.1.1 Curve schemes in the manifold setting

To the best of our knowledge, all existing extensions of Bézier curves to the manifold setting have been proven to work just “in the small”, i.e., when control points are sufficiently close to one another. However, such limitation is incompatible with a practical usage. We show that, indeed, the manifold extensions of the de Casteljau and Bernstein evaluation algorithms may fail and lead to discontinuous curves for general sets of control points. On the other hand, we show that some existing subdivision schemes may be generalized to manifolds “in the large”, too. We show that the recursive de Casteljau (RDC) subdivision scheme proposed by Noakes [Noa98] indeed works for any set

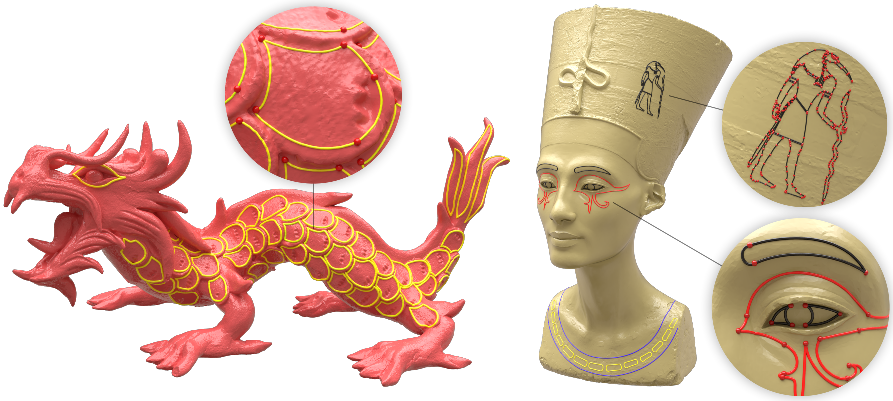


Figure 7.1: We propose algorithms to interactively edit Bézier splines on large meshes, including curve editing, curve transformations and import and editing complex SVG drawings. All computations occur in the intrinsic geodesic metric of the surface. All splines in this figure have been drawn interactively. Control points and tangents of curves under editing are shown in the zoomed insets. Asian Dragon ~7.2M triangles; Nefertiti ~500K triangles.

of control points, always producing C^1 curves. And we propose a manifold extension of an open-uniform Lane Riesenfeld (OLR) subdivision scheme; we elaborate on results of Duchamp et al. on the manifold extension of the standard Lane Riesenfeld scheme for B-splines [DXY18], to show that our OLR scheme is the first one to produce C^2 Bézier segments in the manifold setting. Curves from both the RDC and the OLR schemes can be joined with C^1 continuity to form Bézier splines.

7.1.2 Algorithms

We provide the basic tools for curve design and rendering with the RDC and OLR subdivision schemes. To the best of our knowledge, all previous proposals in the manifold setting provided only algorithms for curve tracing, i.e., to produce a discrete approximation of the curve. Besides such algorithms, we give the algorithms for evaluating the curve at a given parameter; and for splitting a curve at a given point, by approximating a single Bézier segment with a spline of two segments.

7.1.3 Computational framework and system

While the curve schemes and related algorithms are defined on smooth manifolds, our implementation addresses triangulated surfaces. As remarked by Wallner and Pottmann [WP06], *"after discretization the question of smoothness does no longer make sense in the strict mathematical sense. Even so, it is important to know that ... the ideal geometric object one tries to approximate is smooth."* All our algorithms rely on the computation of repeated manifold averages,

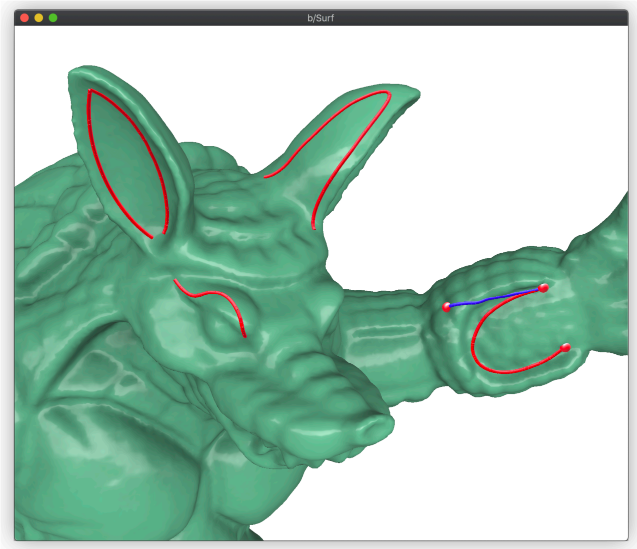


Figure 7.2: The GUI of our system. Curves on the eye and on the arm consist of a single cubic segment each, while the two splines on the ears consist of two cubic segments each, with a sharp corner to the left and smooth junction to the right. A control tangent (in blue) is depicted on the curve under editing.

which involve finding geodesic shortest paths. Such computations are known to converge to the equivalent measures in the smooth setting as the geometric mesh is refined, e.g., through subdivision [DGDT16]. In order to target meshes with millions of triangles, we need a very efficient framework for geodesic computations. We develop an algorithm for computing locally shortest paths that is robust and beats the performances of the other methods at the state of the art. In particular, we greatly improve the step to find an initial guess, which is the bottleneck of all local methods.

We integrate our algorithms in a user interface, thus providing the first prototype system that supports the robust interactive design of Bézier splines on manifold meshes for any choice of control points. We support all basic operations that 2D editors have, including: click-and-drag of control points and tangents; point insertion and deletion; and translation, rotation and scaling of curves. We also support mapping of 2D SVG drawings onto the surface. Fig. 7.2 shows the interface of our system with some simple curves traced on a model. The supplemental video shows a full editing session. Our system remains interactive on meshes made of millions of triangles, such as the ones shown in Fig. 7.1.

7.1.4 Assessment and comparisons

To assess the robustness and performance of our algorithms, we trace curves on the more than five-thousands watertight, manifold, meshes of the Thingi10k repository [ZJ16] with one hundred randomly generated control polygons for each mesh. Our algorithms handle all cases well. We discuss the sensitivity

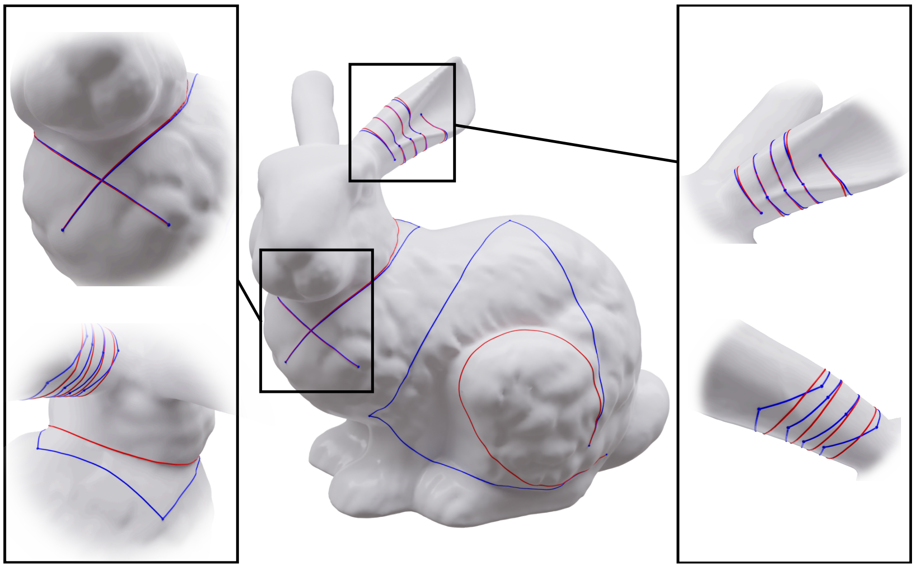


Figure 7.3: Curves that wind about the object or require large control polygons may be challenging to draw with an approach based on parametrization. The collar and the curl consist each of a single cubic segment, while the spiral is a spline of four segments joined with smooth (C^1) continuity. Control polygons are depicted in blue.

of our algorithms to the input mesh, and how to deal with critical meshes containing nearly degenerate triangles. We run an extensive comparison to the methods at the state of the art, in terms of both robustness and time performance, consistently beating their results.

7.2 Related work

The design of spline curves on manifolds has been addressed by several authors, both from a mathematical and from a computational perspective. We review only methods addressing general surfaces.

A traditional approach to circumvent the problems of the Riemannian metric consists of linearizing the manifold domain via parametrization, designing curves in the parametric plane, and mapping the result to the surface. Parametrization introduces seams, and drawing lines across them becomes problematic. Moreover, distortions induced by parametrizations are hard to predict and control. The exponential map can provide a local parametrization on the fly for the region of interest [BMBZ02, HA19, Sch13, SZZ+13, SGW06]. However, its radius of injectivity can be small (e.g., in regions of high curvature), while control polygons and curves may extend over large regions. Even curves as simple as the ones depicted in Fig. 7.3 may be hard to control using either local or global parametrizations.

As reported in [WP06], another common approach consists of relaxing the manifold constraint, resolving the problem in Euclidean space, and projecting the result back to the surface. Panozzo et al. [PBDSH13] use an embedding in a higher-dimensional Euclidean space, followed by Phong projection. These methods may support user interaction, but they provide only approximate results, are prone to artifacts, and are hard to scale to large meshes.

The design of curves can also be addressed as an optimization problem in a variational setting. Noakes et al. [NHP89] and Camarinha et al. [CSLC95] provide the basic variational theory of splines on manifolds. This approach is adopted in several other papers [AGS$^+$15, GSA14, GMA18, HP04, JSW$^+$19, PH05, SASK11]. While most such works do not address implementation and performance, [HP04] and [JSW$^+$19] eventually resort to projection methods. Overall, the variational approach is too computationally expensive to support user interaction on large meshes. Moreover, these curves are harder to control interactively than traditional Bézier splines.

Concerning the specific case of Bézier curves, Park and Ravani [PR95] first extended the de Casteljau algorithm to Riemannian manifolds, without developing the computational details. Later on, the de Casteljau algorithm on surfaces has been explored by several other authors [GMA18, LW01, MCV08, NYP13, PN07]. Among these, Morera et al. [MCV08] extend the recursive de Casteljau bisection, and Sharp et al. [SC$^+$19] achieve interactive performance on the same algorithm, by using a fast method for evaluating locally shortest geodesic paths [SC20a]. We adopt the same structure of [MCV08] for curve tracing with the recursive de Casteljau (RDC) subdivision. In Section 7.6.3, we further discuss the method of [SC$^+$19, SC20a] and compare their results and performances with our method. Absil et al. [AGSW16] define Bézier curves both with the de Casteljau algorithm and with the Riemannian center of mass (RCM), and show that they may produce different results. A method for the direct computation of the RCM through gradient descent has been proposed in [SSC19b], which is computationally intensive, though. Conversely, the method proposed in [PBDSH13] is very efficient (after pre-processing), but provides just an approximation of the RCM. In Section 7.6.3, we compare to both such methods in terms of robustness and performance.

Several authors have investigated the theoretical aspects of the subdivision approach to splines in the manifold setting. We refer to Wallner [Wal20] for a detailed analysis, reporting just the results most relevant to our work. Noakes [Noa98] proves that the recursive de Casteljau subdivision converges and produces a C^1 curve in the cubic case, subject to strong constraints on the control polygon. Most recent results [DXY18, DS17, DGL19] focus on Lane-Riesenfeld schemes and show that a scheme of order k is convergent and C^k in the manifold and functional settings. These latter approaches motivate our approach to the open-uniform Lane-Riesenfeld (OLR) subdivision. We exploit observations reported in [Wal20] to show that such schemes can be generalized to deal with any control polygon.

7.3 Bézier Curves on Manifolds

We consider different constructions of Bézier curves, all of which produce the same curves in the Euclidean setting, and we analyze their extensions to the manifold setting. We provide just the basics of each construction, referring the reader to [Far01, Sal06] for further details. All definitions are given in general, while results are given just for cubic Bézier curves, for the sake of simplicity; extensions to curves of a different order are just outlined. In the following, we denote by $\mathcal{M}$ a smooth, compact and connected surface embedded in $\mathbb{R}^3$, endowed with the Riemannian metric induced by the embedding.

7.3.1 Preliminaries and notations

In the Euclidean setting, a Bézier curve is the image of a polynomial parametric function of degree k

$$\mathbf{b}^k : [0,1] \longrightarrow \mathbb{R}^d,$$

which is defined by means of a *control polygon* $\Pi = (P_0, \dots, P_k)$, where all $P_i \in \mathbb{R}^d$. Curve $\mathbf{b}^k$ interpolates points P_0 and P_k, and it is tangent to Π at them. All constructions of Bézier curves in the Euclidean setting rely on the computation of *affine averages* of points of the form

$$\bar{P} = \sum_{i=0}^{h} w_i P_i \tag{7.1}$$

where the w_i are non-negative weights satisfying the partition of unity. For $h = 1$, the affine average reduces to linear interpolation

$$\bar{P} = (1-w)P + wQ. \tag{7.2}$$

By analogy with the Euclidean setting, a control polygon Π in the manifold setting consists of a polyline of shortest geodesic paths, connecting the control points that lie on $\mathcal{M}$.

Affine averages are not available on manifolds, since they lack the structure of a vector space, but they can be substituted with the Riemannian center of mass [GK73, Kar77]. Given points $P_0, \dots, P_h \in \mathcal{M}$ and weights $w_0, \dots, w_h$, their *Riemanninan Center of Mass* (RCM) on $\mathcal{M}$ is defined

$$RCM(P_0, \dots, P_h; w_0, \dots, w_h) = \operatorname*{argmin}_{P \in \mathcal{M}} \sum_{i=0}^{h} w_i d(P, P_i)^2 \tag{7.3}$$

where $d(\cdot,\cdot)$ is the geodesic distance on $\mathcal{M}$. If $\mathcal{M}$ is a Euclidean space, then the solution to Eq.(7.3) is the usual affine average of Eq. (7.1).

The RCM requires that Eq. (7.3) has a unique minimizer. Karcher [Kar77] provides a condition of existence and uniqueness of the solution, which requires all points P_i to be contained inside a strongly convex ball, whose maximum radius depends on the curvature of $\mathcal{M}$. In the following, we will refer to this condition as the *Karcher condition.* If such condition is satisfied, then the RCM is smooth in both the P_i's and the w_i's [Afs09]. Unfortunately, the

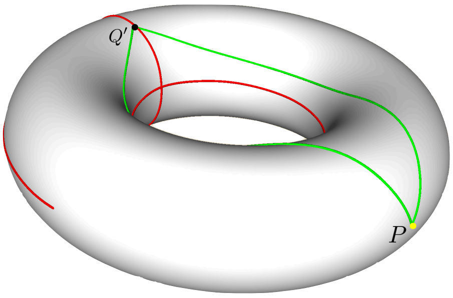


Figure 7.4: The cut locus $C(P)$ of a point P on a torus (red). For a point Q' on $C(P)$ there exist two different shortest geodesics joining P to Q' (green).

Karcher condition restricts the applicability of the RCM to relatively small neighborhoods in the general case.

For *any* two points $P, Q \in \mathcal{M}$, which are connected with a *unique* shortest path $\gamma_{P,Q}$ with $\gamma(0) = P$ and $\gamma(1) = Q$, their RCM with weights $(1 - w)$ and w is always defined and lies at $\gamma_{P,Q}(w)$. This provides the analogous of the affine average of Eq. (7.2) for pairs of points that do not lie on each other's cut locus [WP06].

7.3.2 Extension of the weighted average

Figure 7.4 shows an example of cut locus of a point P lying on a torus. If point Q lies on the cut locus of P, then there is ambiguity on which shortest path should be taken to compute their average. We extend the pairwise average to the cut locus, too, by picking one *arbitrary,* but *deterministically selected,* shortest path connecting P to Q. We thus define the *manifold average between two points*

$$\mathcal{A} : \mathcal{M} \times \mathcal{M} \times [0, 1] \longrightarrow \mathcal{M}; \quad (P, Q; w) \mapsto \gamma_{P,Q}(w) \tag{7.4}$$

where $\gamma_{P,Q}$ is a (deterministically selected) shortest geodesic path joining P to Q. We have that $\mathcal{A}(P, Q; w) = RCM(P, Q; (1 - w), w)$ as long as P and Q do not lie on each other's cut locus; while at the cut locus it returns a point, which depends on the selected shortest path $\gamma_{P,Q}$. The averaging operator of Eq. (7.4) provides the analogous of Eq. (7.2) in the manifold setting for *any* pair of points.

Notice that, the operator $\mathcal{A}$ remains smooth everywhere in its w parameter, but it fails to be continuous at pairs $(P, Q) \in \mathcal{M} \times \mathcal{M}$ that lie on each other's cut locus. Such a discontinuity may affect the manifold constructions, as we will see in the following.

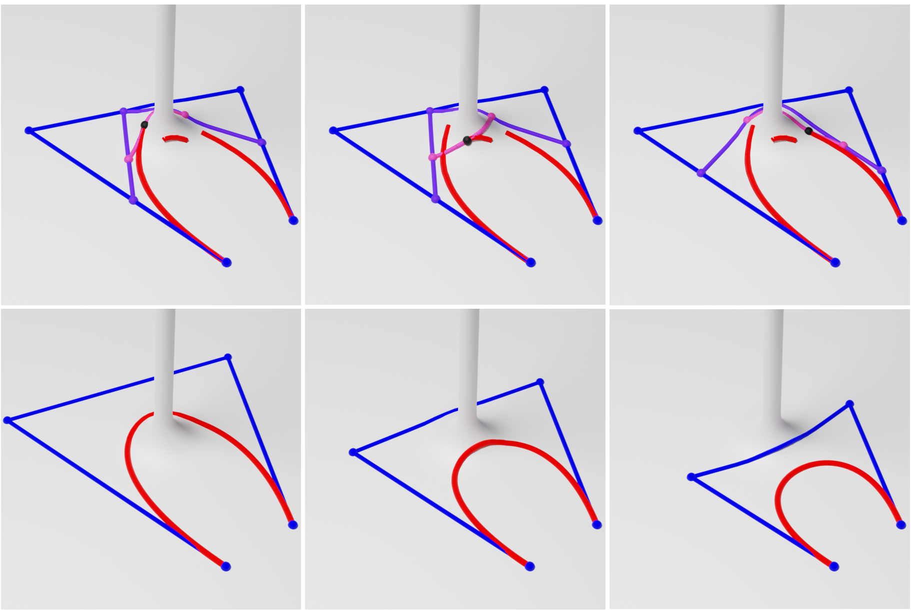


Figure 7.5: Top: example of a failure case of direct de Casteljau evaluation. The black bullets at the discontinuities correspond to consecutive parameter values near a critical value, and the blue/purple/pink lines provide the de Casteljau construction. Note how the pink line jumps from one side of the pole to the other as t passes critical values, causing discontinuities. Bottom: our method always produces a smooth curve regardless of the positioning of the control points. The same control polygon of the top figure generates the curve in the center; dragging the handles we may force the curve to pass behind the pole (left) or further shrink (right). Note how the control polygon to the right also switches to the front of the pole, while leaving the smoothness of the curve unaffected.

7.3.3 de Casteljau point evaluation

The de Casteljau construction provides a recursive definition, which evaluates a Bézier curve as $\mathbf{b}^k(t) = \mathbf{b}_0^k(t)$, where

$$\begin{aligned} \mathbf{b}_i^0(t) &= P_i \\ \mathbf{b}_i^r(t) &= (1-t)\mathbf{b}_i^{r-1}(t) + t\mathbf{b}_{i+1}^{r-1}(t) \end{aligned} \tag{7.5}$$

for $r = 1, \ldots, k$ and $i = 0, \ldots, k - r$. A curve is traced by computing Eq. (7.5) for t varying in $[0, 1]$.

This construction can be extended to the manifold setting in a straightforward way by substituting the affine averages between pairs of points with the manifold average $\mathcal{A}$ defined above. This extension was proposed first by Park and Ravani [PR95]. As shown by Popiel and Noakes [PN07], if all consecutive pairs of points in the control polygon Π lie in a totally normal ball, then the resulting curve is smooth. However, if the constraint is violated, the resulting curve can be discontinuous. In fact, even if all shortest geodesics paths in Π are unique, some pairs of intermediate points involved in the construction may lie on each other's cut locus, for some value of the parameter t. As t passes such critical value, the manifold average $\mathcal{A}$ returns a discontinuous result, thus causing a discontinuity in the curve. Fig. 7.5(top) illustrates the construction near failure points; Fig. 7.6(a) provides another example of failure on a more complex shape.

7.3.4 Bernstein point evaluation with the RCM

A Bézier curve can be evaluated in closed form as an affine sum of all its control points:

$$\mathfrak{b}^k(t) = \sum_{i=0}^{k} B_i^k(t) P_i \tag{7.6}$$

where the $B_i^k(t)$ are the Bernstein basis polynomials of degree k

$$B_i^k(t) = \binom{k}{i} t^i (1-t)^{n-i}.$$

This expression can be rewritten for the manifold case as

$$\mathfrak{b}^k(t) = RCM(P_0, \ldots, P_k; B_0^k(t), \ldots, B_k^k(t)) \tag{7.7}$$

where the Riemannian center of mass RCM has been defined in Eq. (7.3). Again, a curve is traced by computing Eq. (7.7) for t varying in $[0, 1]$. This construction was addressed in [PBDSH13], where an approximation of the RCM is proposed, which is based on an embedding in a higher dimension and Phong projection (see Sec. 7.6.3 for further details). A direct evaluation of the RCM is also possible through gradient descent on the energy of Eq. (7.3). A method has been proposed in [SSC19b], which requires computing a log map at each iteration, though.

If the control points are close enough to fulfill the Karcher condition, then the resulting curve is smooth, since both the RCM and the Bernstein polynomials are smooth. However, if the Karcher condition is not fulfilled, then the energy in Eq. (7.3) is no longer guaranteed to be convex, and it might even have infinitely many minima. In this case, the curve may be undetermined at some intervals. Fig. 7.6(b) provides an example of failure, where the RCM has been computed directly by gradient descent. Note that, the failure is independent of the method used to implement the RCM, being intrinsic to the non-convexity of the energy for some values of the weights. More examples of failure for the method of [SSC19b] and for the approximation of [PBDSH13] are demonstrated in Section 7.6.3.

7.3.5 Recursive de Casteljau subdivision (RDC)

One step of the de Casteljau construction subdivides polygon Π into two control polygons Π_L and Π_R. See Fig. 7.7 (RDC) for an example. The junction point of Π_L and Π_R lies on the curve. The recursive application of this procedure for $t = 1/2$ defines a sequence of subdivision polygons Π_{DC}^n, which converges to the Bézier curve.

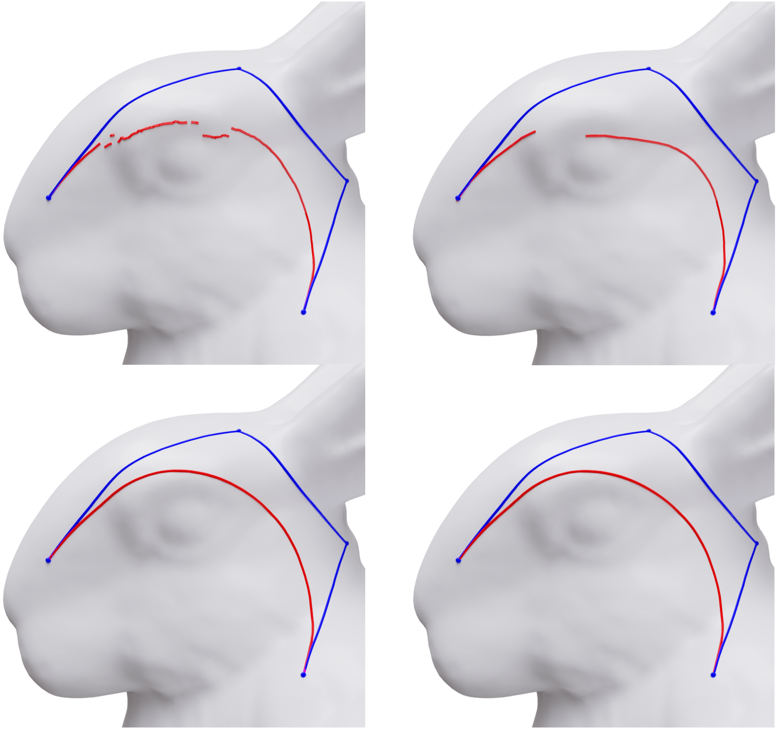


Figure 7.6: An example of failure in tracing a curve with the direct de Casteljau (*top-left*), and the RCM evaluation (*top-right*). The same control polygon gives two smooth and nearly identical curves with the Recursive de Casteljau (*bottom-left*) and the Open-uniform Lane-Riesenfeld schemes (*bottom-right*) described in Sections 7.3.5 and 7.3.6, respectively.

In the manifold setting, this algorithm produces a curve, which is *different* from the one obtained with the direct evaluation by varying the value of parameter t, as reviewed in Sec.7.3.3. This scheme in the manifold setting was studied first by Noakes [Noa98], and implementations were proposed in [MCV08, SC+19]. Noakes proved that this subdivision converges to a C^1 limit curve, provided that the initial control points lie in a convex set [Noa98]. We extend this result to any set of control points.

PROPOSITION 7.1 : For any given control polygon $\Pi = (P_0, P_1, P_2, P_3)$, the RDC subdivision implemented with the $\mathcal{A}$ operator converges to a limit curve that is C^1 continuous.

Proof: As observed by Wallner [Wal20] (Sec. 2.4), a result "in the small" can be generalized to any control polygon, if the control points in the sequence of subdivided polygons become sufficiently close after a finite number of subdivisions. This is straightforward from the following two facts:

- the RDC scheme always satisfies the *contractivity property*, in particular, the greatest distance between two consecutive points of Π_{DC}^{n+1} is not greater than half the greatest distance between two consecutive points in Π^n_{DC}. This property depends only on the triangular inequality of the geodesic distance function, which holds everywhere, including at the cut locus.

- On a compact manifold of bounded curvature there exists $\delta > 0$ s.t. every ball of radius $r \leq \delta$ is convex (Section 2.4).

Therefore, in a finite number of subdivision steps, we obtain a sequence of control polygons such that each of them is contained in a convex set, hence undergoes the hypothesis of [Noa98]. Moreover, by construction, every two consecutive segments have the same tangent at their junction point, thus their limit curves join with C^1 continuity. □

Note that the constraints imposed by Noakes [Noa98] have the purpose of warranting the uniqueness of a geodesic and its smooth dependence from its endpoints. In our case, if such constraints are violated, the limit curve is just one of the possible curves, which one obtains by the arbitrary, but deterministic, choices made by operator $\mathcal{A}$ at the cut locus. Once the choice is made, the resulting curve is guaranteed to be C^1. However, the result may not be continuous in the *space of curves* while varying the control points. The consequences of this fact will be discussed in Sec. 7.3.7.

Concerning curves of different order, Noakes [Noa99] proved that the C^1 continuity holds also for quadratic curves. It remains an open question whether the RDC scheme produces curves with higher smoothness.

7.3.6 Open-uniform Lane-Riesenfeld Subdivision (OLR)

In [DXY18], a uniform subdivision scheme has been proposed, which ports to the manifold setting the well known Lane-Riesenfeld (LR) scheme [LR80]. Such a scheme converges to B-splines and cannot be used directly to design curves with fixed endpoints. We generalize their result to a scheme with end conditions, which defines Bézier curves, and we show that, in the cubic case, it converges to segments that are C^2 everywhere, possibly except at the endpoints, where they are at least C^1.

We briefly review the Euclidean scheme [CDS07, Sal06]. A cubic Bézier can be represented with an open-uniform B-spline[1] of order 4, having the same control polygon Π, and knot vector $(0, 0, 0, 0, 1, 1, 1, 1)$. Repeated knot insertion at the midpoint of all non-zero intervals produces a sequence of open uniform B-splines, all describing the same curve; and the sequence of control polygons Π^n_{LR} converges to the curve itself.

The corresponding subdivision requires four special stencils at each end of

[1] A B-spline is said to be open-uniform, or uniform with end conditions, if it is uniform, except at its endpoints, where repeated knots are inserted to make the curve interpolate the endpoints of its control polygon.

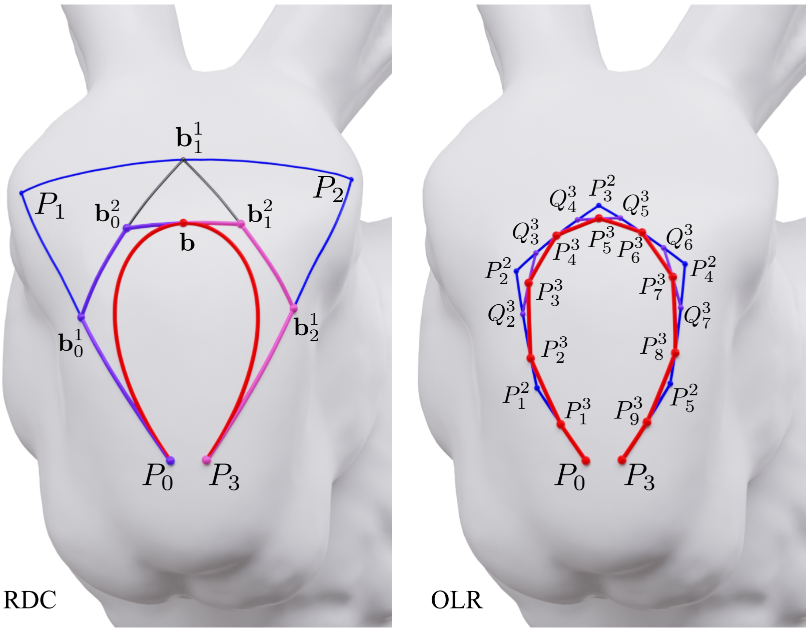


Figure 7.7: The constructions at the basis of the RDC and OLR schemes for the same control polygon for the cubic case. *RDC (left)*: The control polygon (blue) is split into a chain of two control polygons (purple and pink) by computing three shortest geodesic paths. The limit curve is depicted in red. Here we show only the first subdivision. *OLR (right)*: One step of subdivision from Π^2 (blue) to Π^3 (red polygon). The even points P_{2j}^3, as well as the intermediate points Q_i^3 lie on segments of Π^2 and are evaluated first. The evaluation of each odd point P_{2j+1}^3 requires computing one shortest geodesic path (purple). This construction corresponds to one midpoint subdivision followed by two steps of smoothing by averaging consecutive points.

the polygon, and it is defined as follows:

$$\begin{aligned}
P_{2j}^{n+1} &= \tfrac{1}{2}P_j^n + \tfrac{1}{2}P_{j+1}^n & j &= 2 \ldots 2^n - 2 \\
P_{2j+1}^{n+1} &= \tfrac{1}{8}P_j^n + \tfrac{3}{4}P_{j+1}^n + \tfrac{1}{8}P_{j+2}^n & j &= 2 \ldots 2^n - 3 \\
P_0^{n+1} &= P_0 \\
P_1^{n+1} &= \tfrac{1}{2}P_0^n + \tfrac{1}{2}P_1^n \\
P_2^{n+1} &= \tfrac{3}{4}P_1^n + \tfrac{1}{4}P_2^n \\
P_3^{n+1} &= \tfrac{3}{16}P_1^n + \tfrac{11}{16}P_2^n + \tfrac{2}{16}P_3^n
\end{aligned} \tag{7.8}$$

where, for the sake of brevity, we have omitted the end conditions to the right end side, which are symmetric to the ones on the left. We also omit the special stencils that are needed at the first and second levels of subdivision, which can be derived easily, and treated analogously in the context of the following extension. Note that, the first two stencils in Eq. (7.8) give the uniform LR scheme with two smoothing steps, which is applied to all central points. Here, the stencils are written in compact form, instead of the usual sequence of one average step followed by two smoothing steps, because this leads to the same result in the linear scheme.

In order to port such a scheme to the manifold setting, we first observe that some of the stencils appearing in Eq. (7.8) involve more than two control points. Since, in general, we cannot rely on the RCM, we need to factorize such stencils with repeated averages, computed with operator $\mathcal{A}$. In the manifold setting, different factorizations may lead to different curves. We adopt a factorization that "in the middle" (i.e., for $j = 2 \dots 2^n - 2$) gives the same scheme of [DXY18]:

$$
\begin{aligned}
\widetilde{P}^{n+1}_{2j} &= P^n_j \quad \widetilde{P}^{n+1}_{2j+1} = \mathcal{A}(P^n_j, P^n_{j+1}, \tfrac{1}{2})\\
Q^{n+1}_{2j} &= \mathcal{A}(\widetilde{P}^{n+1}_{2j}, \widetilde{P}^{n+1}_{2j+1}, \tfrac{1}{2}) \quad Q^{n+1}_{2j+1} = \mathcal{A}(\widetilde{P}^{n+1}_{2j+1}, \widetilde{P}^{n+1}_{2j+2}, \tfrac{1}{2})\\
P^{n+1}_{2j} &= \mathcal{A}(Q^{n+1}_{2j}, Q^{n+1}_{2j+1}, \tfrac{1}{2}) \quad P^{n+1}_{2j+1} = \mathcal{A}(Q^{n+1}_{2j+1}, Q^{n+1}_{2j+2}, \tfrac{1}{2}).
\end{aligned}
$$

This factorization indeed consists of one average step followed by two smoothing steps. Note, however, that Q^{n+1}_{2j} and Q^{n+1}_{2j+1} lie on the shortest geodesic path γ_j connecting P^n_j to P^n_{j+1}, and that averages between points lying on γ_j are in fact linear with respect to its arc length. Consequently, we have that $P^{n+1}_{2j} = \widetilde{P}^{n+1}_{2j+1}$, and we can rewrite the above formulas more compactly as follows:

$$
\begin{aligned}
Q^{n+1}_{2j} &= \mathcal{A}(P^n_j, P^n_{j+1}, \tfrac{3}{4}), \quad Q^{n+1}_{2j+1} = \mathcal{A}(P^n_{j+1} P^n_{j+2}, \tfrac{1}{4})\\
P^{n+1}_{2j} &= \mathcal{A}(P^n_j, P^n_{j+1}, \tfrac{1}{2}), \quad P^{n+1}_{2j+1} = \mathcal{A}(Q^n_{2j}, Q^n_{2j+1}, \tfrac{1}{2}).
\end{aligned}
\tag{7.9}
$$

We factorize the end stencil for P^{n+1}_3 (and its symmetric point to the other end of the polygon) in a similar way. We require that this point is again obtained with one averaging step followed by two smoothing steps, and we apply the considerations above to express repeated averages along the same geodesic in a compact way. In order to accommodate for the other end conditions, the averaging step between P^n_1 and P^n_2 must be unbalanced, while all other steps can be maintained balanced. The only factorization fulfilling these constraints is given by the following equations:

$$
\begin{aligned}
P^{n+1}_0 &= P_0\\
P^{n+1}_1 &= \mathcal{A}(P^n_0, P^n_1, \tfrac{1}{2})\\
P^{n+1}_2 &= \mathcal{A}(P^n_1, P^n_2, \tfrac{1}{4})\\
Q^{n+1}_2 = \mathcal{A}(P^n_1, P^n_2, \tfrac{5}{8}), &\quad Q^{n+1}_3 = \mathcal{A}(P^n_2, P^n_3, \tfrac{1}{4})\\
P^{n+1}_3 &= \mathcal{A}(Q^{n+1}_2, Q^{n+1}_3, \tfrac{1}{2})
\end{aligned}
\tag{7.10}
$$

where, as above, the expressions of Q^{n+1}_2 and Q^{n+1}_3 incorporate the averaging step and the first smoothing step.

In summary, Equations 7.9 and 7.10 provide the stencils that generalize Eq. (7.8) to the manifold case. One step of subdivision for $n = 3$ is exemplified in Figure 7.7 (OLR).

We generalize the results of [DXY18] to the above scheme, as follows:

PROPOSITION 7.2 : For any given control polygon $\Pi = (P_0, P_1, P_2, P_3)$, the manifold OLR subdivision converges to a limit curve that is C^2 continuous, possibly except at its endpoints. The limit curve interpolates the endpoints of polygon Π and it is tangent to it.

Proof: We show that everywhere, except at the endpoints, the results of [DXY18] apply after a finite number of subdivision steps. To this aim, we recall that our scheme is the result of repeated knot insertion, which bisects all non-null intervals at each iteration. We exploit the relation between knots and points of the subdivision polygon to show that for every $t \in (0,1)$ the limit curve exists and is C^2. For any given value of t, after $\bar{n}$ iterations, we have that $t \in (2^{-\bar{n}}(j-1), 2^{-\bar{n}}(j+1))$ for some $j \in \mathbb{N}$. We can always choose $\bar{n}$ large enough that $j > 7$ and $j < 2^{-\bar{n}} - 3$. In this case, the five consecutive control points $P^{\bar{n}}_{j-4}, \ldots, P^{\bar{n}}_{j}$, will undergo the uniform LR stencils at all subsequent levels of subdivision. Next we proceed as in the proof of Proposition 7.1. By triangular inequality, it is easy to show that the manifold OLR scheme is contractive. Thus, in a finite number of subdivision steps, say $\tilde{n}$, all 5-tuples of consecutive points in $\Pi^{\tilde{n}}_{LR}$, are contained in a totally normal neighborhood. If we take $n = \max(\tilde{n}, \bar{n})$ then, by [DXY18], it follows that the polygon $P^n_{j-4}, P^n_{j-3}, P^n_{j-2}, P^n_{j-1}, P^n_j$ converges to a C^2 limit curve, corresponding to interval $(2^{-n}(j-1), 2^{-n}(j+1))$, which contains t. The end conditions in Eq. (7.10) trivially guarantee that the limit curve interpolates the initial control polygon Π at P_0, and it will be tangent at P_0 to the geodesic connecting P_0P_1. □

It follows from the proposition above that a single cubic Bézier segment has C^2 continuity, while different segments can be joined to form splines with C^1 continuity. It remains an open problem how to build splines with C^2 continuity at junction points.

7.3.7 Limitations

Since the operator $\mathcal{A}$ is deterministic, the curve obtained with either the RDC or the OLR scheme is uniquely defined by its control points. However, the curve may jump to a different configuration for small displacements of control points, which make some of the paths in the construction cross a cut locus. In Fig. 7.8 (left, center), a tiny displacement of one control point takes one of the shortest paths in the control polygon to a drastically different route, resulting in a different curve; see the bottom part of Fig. 7.5 for another example. In the accompanying video we provide more dynamic examples of jumps. Note that jumps occur quite rarely, as the cut locus of each point covers a set of zero measure. This fact is intrinsic to the discontinuity of the manifold metrics and constitutes an essential limitation to the design of splines in the manifold setting, independently of the approach adopted.

This limitation can be circumvented easily, by means of splines containing more control points, instead of single Bézier segments. See Fig. 7.8 (right). Point insertion can be used to constrain the curve to a desired path, as is customarily done in curve design, and motivates the algorithms we present in Sections 7.4.1 and 7.4.2. Note that, in general, the spline obtained with point insertion is just an approximation of the original curve. This is also an intrinsic

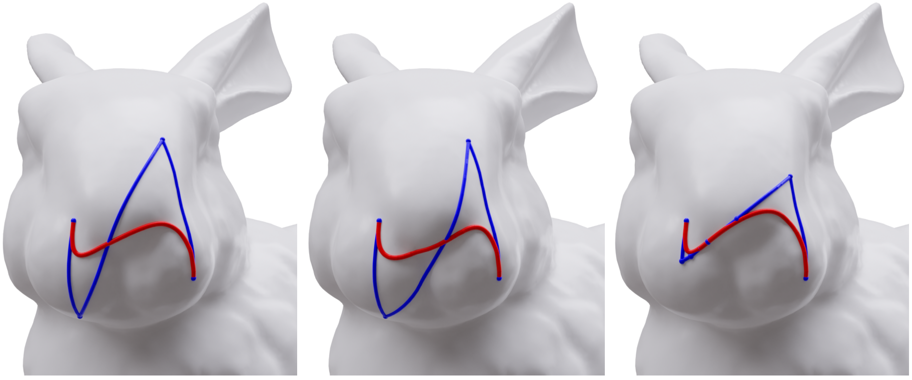


Figure 7.8: Left, center: the geodesic line corresponding to the central segment of the control polygon can take two different routes at the cut locus of its endpoints, thus producing two different curves; in practice, the curve will jump between the two configurations when dragging a control point across a cut locus. Right: splitting the curve by point insertion makes the selected configuration stable upon dragging.

limitation of the manifold setting, where a sub-segment of a Bézier curve is not necessarily a Bézier curve itself. Automatic solutions would be possible. It is easy to check when a curve “jumps” while dragging a control point; in this case, the control polygon may be split, e.g., by adding a point on the curve before displacement as a new control point. In our user interface, we decided to avoid using automatic methods to warrant maximum flexibility to the user.

7.4 Practical Algorithms

We now focus on the RDC and OLR schemes. We provide algorithms for: approximating the curve with a geodesic polyline (curve tracing); evaluating a point on the curve for a given parameter value (point evaluation); and splitting a curve at a given point into a spline approximating it with two segments (point insertion). The algorithm for curve tracing with the RDC scheme is equivalent to the one proposed in [MCV08], while the other five algorithms are novel.

To develop our algorithms, we assume to have procedures for (1) computing the point-to-point shortest path between pairs of points of $\mathcal{M}$; (2) evaluating a point on a geodesic path at a given parameter value; and (3) casting a geodesic path from a point in a given direction. The computational details of such procedures, as well as additional algorithms to support interactive control, are provided in Section 6.7.

7.4.1 Algorithms for the RDC scheme

Curve tracing

The tracing algorithm recursively subdivides a geodesic polygon Π into two sub-polygons Π_L and Π_R. Recursion is initialized by computing the three shortest paths that constitute the polygon connecting the initial control points P_0, P_1, P_2, P_3. Referring to Fig. 7.7 (RDC), one step of subdivision entails computing three geodesic paths, and evaluating six midpoints of existing geodesics. The polygons Π_L and Π_R are built by collecting the sub-paths depicted in violet and in pink, respectively.

For uniform subdivision, a maximum level of recursion is either chosen by the user, or computed on the basis of the total length $L(\Pi)$ of the initial polygon Π, and a threshold δ. Since the paths in the subdivided polygon are shrinking through recursion, then after $\lceil \log_2(L(\Pi)/\delta) \rceil$ recursion levels, the length of a geodesic path in the output will be bounded by δ. For adaptive subdivision, we stop recursion as soon as the angles between tangents of consecutive segments of Π differ for less than a given threshold θ. For a small value of θ, this suggests that the curve can be approximated with a geodesic polyline connecting the points of Π. Note that angles are computed in tangent space, hence accounting just for the geodesic curvature of the curve while disregarding the normal curvature induced from the embedding. This approach works even when a curve crosses sharp creases on polyhedral objects. Like in the Euclidean case, cusps may appear at the transition between a simple and a self-intersecting configuration of a curve. We resolve cusps by stopping the recursion after a maximum number of levels.

Point evaluation

We support the evaluation of the point at a value $\bar{t}$ on the curve. This requires traversing the recursion tree with a bisection algorithm. We split the control polygon at each level as described above, but only compute the sub-polygon that contains $\bar{t}$. We stop recursion with the same criteria listed above.

The point at $\bar{t}$ is computed by direct de Casteljau evaluation on the leaf control polygon. Here we are assuming that the control polygon in the leaf node is short enough to support direct de Casteljau evaluation, and we use it to approximate the limit point on the subdivided curve. By using arguments of proximity, as in [Wal06, Wal20], it can be shown that this approximation converges to the limit curve, as the subdivision polygon is subdivided further.

Point insertion

With point insertion, a user can split a curve at a given point $P_{\bar{t}}$, and obtain a spline consisting of two Bézier curves, from P_0 to $P_{\bar{t}}$ and from $P_{\bar{t}}$ to P_k, which substitutes the input curve. This is used to add detail during editing. While this computation is exact in the Euclidean setting, the identity of curves before and after point insertion cannot be guaranteed in the manifold setting. Here we provide a solution that approximates the input curve by interpolating its endpoints, as well as point $P_{\bar{t}}$, and preserving the tangents at such points. We place the unconstrained points by mimicking the algorithm in the Euclidean setting, trying to obtain a spline that closely approximates the input curve.

We refer to the construction illustrated in Fig. 7.9. We descend the recursion

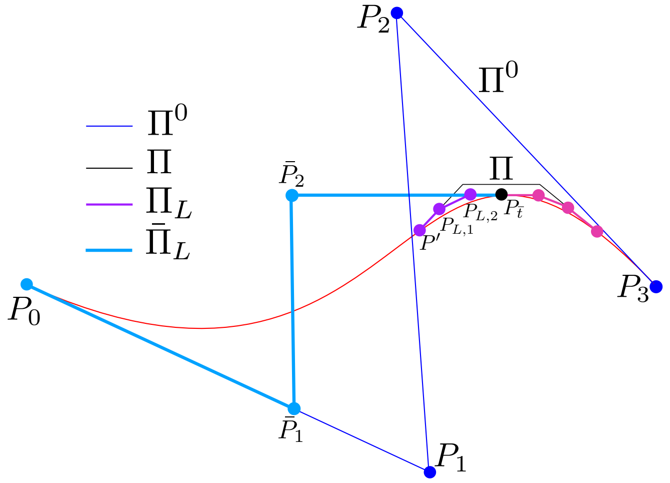


Figure 7.9: The point insertion algorithm for a cubic curve (left side only). The control polygon $\bar{\Pi}_L$ (light blue) defining the left side of the curve upon split at $P_{\bar{t}}$ is built by shortening the first segment of Π^0 (dark blue) and extending the last segment of Π_L (purple).

tree as in the previous algorithm, in order to find the leaf Π containing the splitting point $P_{\bar{t}}$. Assuming, as above, that Π is small enough, we split it at value $\bar{t}$ by direct de Casteljau evaluation, thus obtaining the two polygons Π_L and Π_R, in purple and magenta in the figure. We process the two halves independently. Here we show the algorithm for finding the polygon $\bar{\Pi}_L$ defining the curve between P_0 and $P_{\bar{t}}$. The construction of the other half is symmetric.

Let us denote

$$\Pi_L = (P', P_{L,1}, P_{L,2}, P_{\bar{t}})$$

the polygon in purple in Fig. 7.9, and let t' be the the parameter corresponding to P' on the input curve. In order to interpolate the endpoints and the related tangent directions, we must have

$$\bar{\Pi}_L = (P_0, \bar{P}_1, \bar{P}_2, P_{\bar{t}})$$

where $\bar{P}_1$ must lie along the geodesic line connecting P_0 and P_1, and $\bar{P}_2$ must lie along the extension of the geodesic line connecting $P_{L,2}$ and $P_{\bar{t}}$. The remaining degrees of freedom concern where to place $\bar{P}_1$ and $\bar{P}_2$ along such lines. We place such two points by maintaining the same proportions that would be used in the Euclidean case.

Since $P_{\bar{t}}$ is an endpoint of the curve defined by $\bar{\Pi}_L$, and it lies at parameter $\bar{t}$ on the input polygon Π, then $\bar{P}_1$ is found at distance $\bar{t} \cdot d(P_0, P_1)$ from P_0. Now we place $\bar{P}_2$ in such a way that, once $\bar{\Pi}_L$ is split at the parameter corresponding to P', the polygon Π_L is generated as its right sub-polygon. The parameter of P' with respect to the sub-curve from P_0 to $P_{\bar{t}}$ is $t'/\bar{t}$, thus we have

$$d(P_{\bar{t}}, P_{L.2}) = (1 - \frac{t'}{\bar{t}}) \cdot d(P_{\bar{t}}, \bar{P}_2).$$

Therefore, we conclude that $\bar{P}_2$ is obtained by extending the geodesic line from $P_{\bar{t}}$ to $P_{L,2}$ for a length $d(P_{L,2}, P_{\bar{t}}) \cdot \frac{t'}{(\bar{t} - t')}$.

While we do not provide any bound on the approximation of the input curve, we tested this algorithm on many curves and the results were mostly very close to the input. Except, as all other editing operations (e.g., dragging control points) a split may cause a jump of one sub-curve, discussed in Sec. 7.3.7. Jumps can be easily recovered either with further splits or by dragging the handle points that control tangents. Notice that, for practical applications, point insertion is aimed at adding more degrees of freedom to the spline: interpolation of pinned points and control of tangents at them are in fact all a designer requires.

7.4.2 Algorithms for the OLR scheme

Curve tracing

For uniform subdivision, our OLR scheme can be easily expanded up to a certain level $\bar{n}$, and the curve approximated with the geodesic polygon $\Pi^{\bar{n}}$. The maximum expansion level $\bar{n}$ is set as in the corresponding RDC algorithm. At each level of subdivision, we obtain the vertices of the refined polygon by applying the subdivision stencils of Equations 7.9 and 7.10. The construction of the third level of subdivision is shown in Fig. 7.7 (OLR).

Notice that the uniform subdivision, as described above, defines a (virtual and infinite) binary tree of intervals, that we call the *expansion tree*: the root of the expansion tree corresponds to the whole interval $[0, 1]$, while a generic node $[t^i_j, t^i_{j+1}]$ at level i is split in the middle into two intervals at level $i + 1$. The node $[t^i_j, t^i_{j+1}]$ encodes a segment of B-spline, defining the curve in the corresponding interval, with control points $(P^i_{j-3}, \ldots, P^i_j)$. One more level of subdivision splits this interval into two sub-intervals $[t^{i+1}_{2j}, t^{i+1}_{2j+1}]$ and $[t^{i+1}_{2j+1}, t^{i+1}_{2j+2}]$ and generates 5 new control points, which depend just on $(P^i_{j-3}, \ldots, P^i_j)$: the first 4 points are associated to the interval to the left, and the last 4 to the interval to the right, with an overlap of 3 control points between the two sets. The expansion tree is defined implicitly and it needs not being encoded.

We exploit the structure of the expansion tree to design an algorithm for adaptive subdivision, which is controlled by the same stopping criterion used for the RDC scheme, i.e., we stop the expansion of a node as soon as the angle between consecutive segments of the polygon is small enough. The algorithm corresponds to visiting a subtree of the expansion tree in depth-first order; a leaf of the subtree is a node of the expansion tree where we stop recursion. During the visit, at each internal node, we split the interval as described above, and generate the control points for its two children to continue the expansion; while at each leaf, we generate the nodes of the output polygon.

Depth-first traversal guarantees that leaves are visited left to right: the leftmost leaf in the expansion tree is the first one to produce an output, adding all its control points; all other leaves add just their rightmost control point to the output. The final approximation of the curve is obtained by connecting the output points pairwise with shortest geodesic paths.

Note that, it is not necessary to encode the subtree visited by the algorithm. It is just sufficient to encode the path in the expansion tree connecting the root to the current node, storing at each node its corresponding interval, and its control polygon.

Point evaluation

The point evaluation algorithm is analogous to the one for the RDC scheme, by descending a path in the expansion tree described above. Given interval $[t^i_j, t^i_{j+1}]$ containing $\bar{t}$ at subdivision level i, we only need to compute, with the proper stencils, the 4 points corresponding to its sub-interval containing $\bar{t}$ at the next level.

Once recursion stops, we assume that all pairs of consecutive control points in the current interval lie in a totally normal ball. Here we evaluate the curve directly with a manifold version of the de Boor algorithm [Far01], which works on repeated averages and can be obtained by substituting the affine averages with the manifold average $\mathcal{A}$, just like the direct de Casteljau evaluation.

The same remarks we made for the RDC scheme about approximation and convergence in the limit apply here, too.

Point insertion

This algorithm is analogous to the one described for the RDC scheme. We descend the recursion tree as in the point evaluation algorithm. When reaching the leaf containing the splitting point $P_{\bar{t}}$, we convert the control polygon of the uniform B-spline in that leaf into the corresponding control polygon of the Bézier curve, by applying the well known conversion formula [Sal06](Sec.7.5), where affine averages are substituted with the manifold average $\mathcal{A}$. Then we proceed as described for the RDC scheme.

7.5 Implementation and User Interface

We implemented the algorithms described in the previous section for discrete surfaces represented as triangle meshes, targeting interactivity for long curves and meshes of millions of triangles. Our algorithms are implemented on top of a few geodesic operations that we describe in this section together with operations required to support the user interface. All operations are implemented in C++ and released as open source in [PNC19]. The triangle mesh M discretizing $\mathcal{M}$ is encoded as describe in Section 6.7, as well as the geodesic paths. The shortest and straightest paths are computed as described in Sections 4.3.1 and 4.2.5, respectively.

7.5.1 User interface

Leveraging the proposed algorithms, we developed a graphical application, demonstrated in the supplemental video, which allow users to interactively

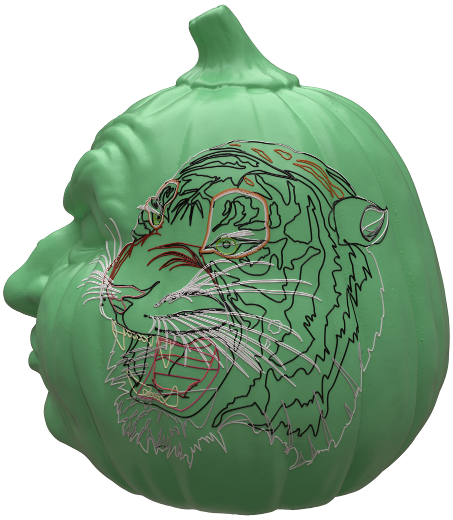


Figure 7.10: Example of importing a large SVG, made of 2056 curves, onto the pumpkin model, consisting of 394k triangles. Our algorithm takes 289 milliseconds to trace all curves.

edit cubic splines on meshes, with the same interaction metaphors used in 2D vector graphics. Our application supports moving, adding, and deleting control points, and by translating, scaling and rotating whole splines on the surface domain. Here we describe the main editing feature, referring the reader to the supplemental video for a demonstration.

Curve editing

Borrowing the editing semantic from 2D tools, control points are distinguished in *anchor points* and *handle points*. Anchors are those points where two Bézier curves are joined, while handle points are the ones preceding and following the anchor points. We connect each handle point with its corresponding anchor with a geodetic segment. A spline is tangent to those segments at the anchor points. In the 2D setting, when an anchor is dragged, the two tangent segments move with it and so do the associated handle points. To obtain the same behavior on the surface, when moving an anchor point from P to P', we find the two tangent directions of the tangent segments at P. Then, for each such segment, we trace a straightest geodesics starting at P' and for the same length of the segment, in the direction of its tangent, rotated by the parallel transport from P to P'. The endpoint of each segment is the new position of the corresponding handle point. In the 2D setting the user can impose an anchor to be "smooth", i.e. the two associated tangent segments are always colinear, which automatically ensure C^1 continuity at the anchor point. To provide the same functionality on the surface, whenever the handle point Q_1 is moved, the opposite handle point Q_2 is recomputed by tracing a straightest geodesic from the anchor P along the tangent direction defined from segment Q_1P to find the new position of handle Q_2.

algorithm	percent of trials		times at percentile	
	< 0.001s	< 0.1s	90%	99%
RDC Uniform	43.1%	99.0%	< 0.0122	< 0.097
OLR Uniform	44.7%	98.9%	< 0.0123	< 0.105
RDC Adaptive	43.9%	99.0%	< 0.0120	< 0.095
OLR Adaptive	30.0%	98.1%	< 0.0215	< 0.185

Table 7.1: Time performances of our algorithms in 556,700 trials. We report the percentage of trials in which tracing a curve takes less than 0.001 and 0.1 seconds, and the running times at the 90th and 99th percentiles.

Rotation, Scaling and Translation

We support translation, rotation and scaling of a whole spline. In the 2D settings these operations are obtained by just applying the same affine transform to all control points. In the surface setting, we define the transformation about the mesh point C under the mouse pointer. The normal coordinates of the control points are computed with respect to C, in a sort of discrete exponential map. Then, the linear transformation is applied on these 2D coordinates, which are finally converted back into mesh points by tracing straightest geodesic paths outward from C. Translation needs special handling, as the center of the transformation C is dragged to a new position C'. To compensate for the change of reference frame, the normal coordinates are rotated by the opposite angle of the parallel transport given by the tangent vector from C to C'.

Importing SVG drawings

Sometimes, it is helpful to map a collection of 2D splines onto the surface. To this aim, we map the 2D control points onto surface, and then trace the splines on the manifold. See Figures 7.1 and 7.10 and the accompanying video for examples. We map control points with a method analogous to Biermann et al. [BMBZ02] that is based on the conversion between polar coordinated in 2D and normal coordinates on the manifold. Each control point of the SVG drawing is converted into a mesh point by taking its polar coordinates, and tracing a geodesic from a center point in the given tangent direction, for the given distance.

Notice that this is intended just as a rough initialization. Since we map only the control points, the intersections between lines will be preserved only at the interpolated points of the splines, while they may differ elsewhere. The advantage here is that all distortions and artifacts that might arise in this phase can be adjusted by editing the result, which is provided in vector format directly on the surface. While this would be impossible with parametrization approaches that just map the discretized splines.

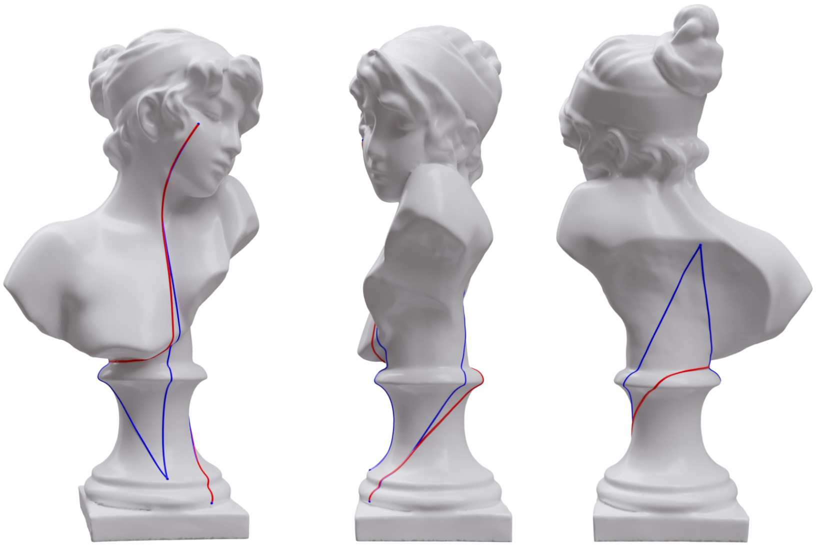


Figure 7.11: Three views of a random curve generated during trials on a model from the Thingi10k repository: in all experiments the control points of each curve were randomly selected over the surface of the object.

7.6 Results and Validation

We validate our work by tracing curves over a large number of meshes, by comparing it with state-of-the-art solutions, and by performing interactive editing sessions. Our algorithms always produce a valid output, in a time compatible with interactive usage in over 99% of the trials (Table 7.1). We overcome the limitations of state-of-the-art methods, producing valid results with any control polygon on any surface (Fig. 7.17); and our running times are comparable or faster than state-of-the-art methods (Table 7.2 and Fig. 7.21). Our system supports editing for meshes of the order of one million triangles on a laptop computer. Interaction is still supported on meshes with several millions of triangles, provided that single curves do not span too large a fraction of the model (see Figures 7.1 and 7.23, Table 7.3, and the accompanying video). Very long segments are rare in actual editing sessions, as real designs are usually made of many splines, each consisting of several small segments.

7.6.1 Robustness and performance

We tested our algorithms for robustness by running a large experiment on the Thingi10k repository [ZJ16]. We extracted the subset of meshes that are manifold and watertight, for a total of 5567 models. The models are used as is, without any pre-processing. For each model, we consider 100 random cubic curves. For each curve, we take the model in its standard pose, and pick points on it by casting random rays orthogonal to the view plane, until we find four points that lie on the surface. These become the control points of the curve.

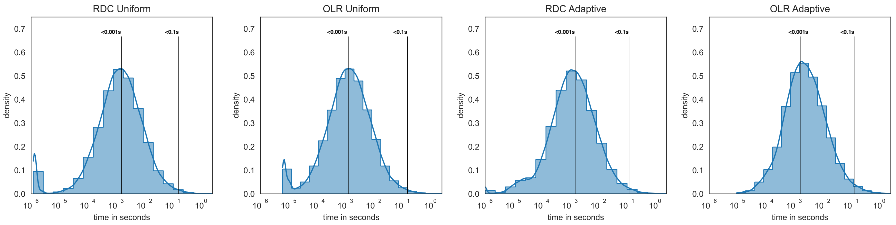


Figure 7.12: The distributions of running times of our four algorithms for curve tracing in 556,700 trials on 5,567 models from the Thingi10k repository, tracing 100 random curves on each model. All algorithms provide a valid output in all trials. The different algorithms have similar behavior and are compliant with interaction (< 0.1 seconds/curve) in about 99% of the trials. For uniform subdivision, the OLR algorithm is slightly faster than the RDC algorithm; while for adaptive subdivision, the RDC algorithm performs slightly better than the OLR algorithm. Adaptive algorithms have slightly narrower distributions than uniform algorithms.

We place no restriction on the arrangement of the control points. This gives us a total of more than half million control polygons. Fig. 7.11 shows a random curve generated on one of the objects during trials.

For each test, we run both the RDC and the OLR tracing algorithms, in their uniform and adaptive configurations. The uniform RDC algorithm is expanded to 4 levels of recursion, which generates a geodesic polyline consisting of 48 geodesic segments. The uniform OLR algorithm is expanded to 6 levels of recursion, which generates a geodesic polyline consisting of 66 geodesic segments. In fact, because of the different subdivision rules, we cannot generate the same number of segments for both schemes. For the adaptive variants, we set a threshold $\theta = 5°$ for the maximum angle between consecutive geodesic segments along the polyline. In this case, the number of geodesic segments in output is variable, depending on the curve and on the method. Since all algorithms generate very similar curves, the final tessellated paths that approximate the curve on the mesh have about the same number of segments in all four cases. Since we have no ground truth to compare with, we indirectly assess the correctness of the results with the following tests:

- Termination: the algorithm must complete;
- Continuity: all pairs of consecutive points along the output polyline must lie either inside or on the edges of the same triangle (notice that points on edges are forced by the algorithm to go across adjacent triangles).
- Smoothness: the angle between consecutive segments of the polyline, measured in tangent space, must be lower than a given threshold (5 degrees).

All our algorithms passed all the tests in all experiments.

Trials were executed on a Linux PC with an AMD Ryzen 5 2600x and 32GB memory, running on a single core. In Table 7.1 and Fig. 7.12, we compare the timing performance of the four algorithms. All algorithms perform quite similarly, and remain interactive in all cases, with roughly 40% of trials running at less than 1 millisecond per curve, and 99% of the trials running faster than 0.1 second/curve. The few trials in which they take more time are concerned, with very few exceptions, either with very long curves on large meshes (>1M triangles), or with meshes containing many topological holes, in which finding shortest paths between points is more expensive.

There are small differences in the performances of the different algorithms. For uniform subdivision, the OLR algorithm generates results as fast as the RDC algorithm, beside generating a more refined geodesic polyline. For adaptive subdivision, the RDC algorithm runs slightly faster than the OLR algorithm. These differences are probably due to the simpler structure of the OLR uniform algorithm in one case, and to the more involved structure of the OLR adaptive algorithm in the other. In fact, both variants of the RDC algorithm follow the same recursive pattern. On the contrary, the OLR uniform algorithm expands the curve level by level, following a simpler pattern; while the OLR adaptive algorithm requires a recursive pattern, with a slightly more involuted structure than the RDC algorithms.

In the previous experiments, the cost of computing a curve depends on both the length of the curve and the size of the mesh, with trends that are not linear. Roughly speaking, the cost of finding the initial path depends on both the length of the curve and the size of the mesh, while the subsequent cost of finding the shortest path depends just on the length of the curve. As the relative length of the curve grows, the cost of finding the initial path prevails, since it may requires exploring most of the mesh. Statistics on the relative costs of the two phases are shown in Fig. 7.22.

For the sake of brevity, we do not present here results on the algorithms for curve tracing and point insertion, which run much faster than the tracing algorithms.

7.6.2 Sensitivity to the input mesh

All the algorithms presented in Sections 4.3.1 and 4.2.5 are driven by the connectivity of the underlying mesh. In particular, all intersections between the traced lines and the mesh are computed locally to each triangle and forced to lie on its edges, so that each traced line consistently crosses a strip of triangles. With this approach, we could process even meshes containing nearly degenerate triangles, with angles near to zero and edge lengths near to the machine precision, by relying just on floating point operations, without incurring in numerical issues. While this is usually not the case with models used in a production environment, such meshes are common in the Thingi10k repository and provide stress tests for the robustness of our algorithms. On the other hand, our algorithm for point-to-point shortest path assumes the initial guess obtained during Phase (i) to be homotopic to the result. This assumption is common to all algorithms for computing locally shortest paths, and it is reasonable as long as the mesh is sufficiently dense and uniform with respect to the underlying surface [SC20a]. If, conversely, the mesh is

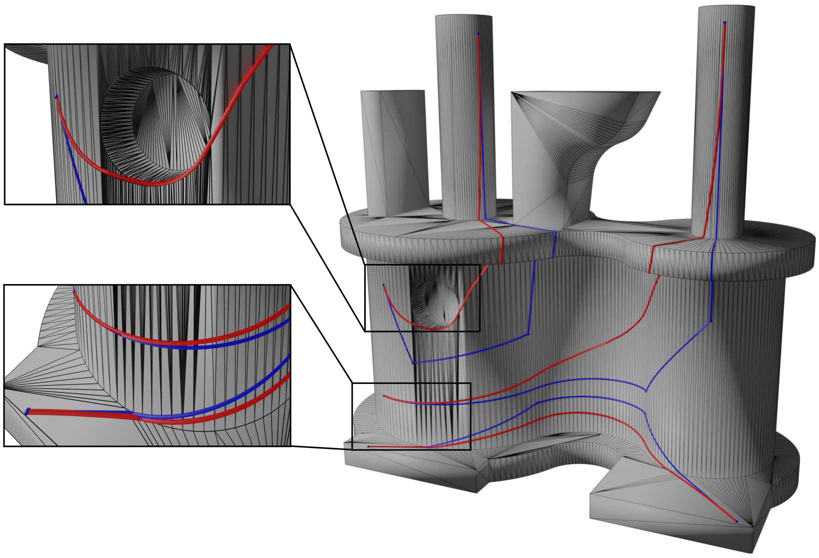


Figure 7.13: The shortest path algorithm is driven by mesh connectivity and uses a refined dual graph, providing correct results even on highly anisotropic meshes containing long edges and narrow angles.

too coarse and anisotropic, then Phase (i) may provide an initial guess, which cannot be homotopically shortened to the correct solution. In this case, a naive application of the algorithm may get stuck in local minima of the space of shortest paths, leading to a wrong curve.

This limitation is quite rare in practice for meshes used in design applications, which is our target, but did happen for some meshes in the Thingi10k dataset. We overcome this limitation simply by creating a more accurate graph for computing the initial guess when dealing with meshes with long edges.

When we build the dual graph to be used in Phase (i), we split mesh edges that are too long at their midpoint, until all edges are shorter than a given threshold, and we symbolically subdivide their incident triangles accordingly. We chose the 5% of the diagonal of the bounding box of the model as threshold. Note that this subdivision is done just for the purpose of building the graph, without changing the underlying mesh. In this augmented graph, a single triangle may be represented by multiples nodes, giving us a more accurate approximation of paths. This approach has the effect of densifying the graph without changing the mesh upon which we run Phase (ii). Once the strip is computed on the augmented graph, we reconstruct the strip on the mesh using the graph's node provenance, i.e. the mesh triangle corresponding to each node, which we store during initialization. Fig.7.13 shows examples on highly anisotropic meshes from the Thingi10k repository.

An alternative approach to cope with the same problem would be to precompute an intrinsic Delaunay triangulation in the sense of [SSC19a] and do all computations by using intrinsic triangulations.

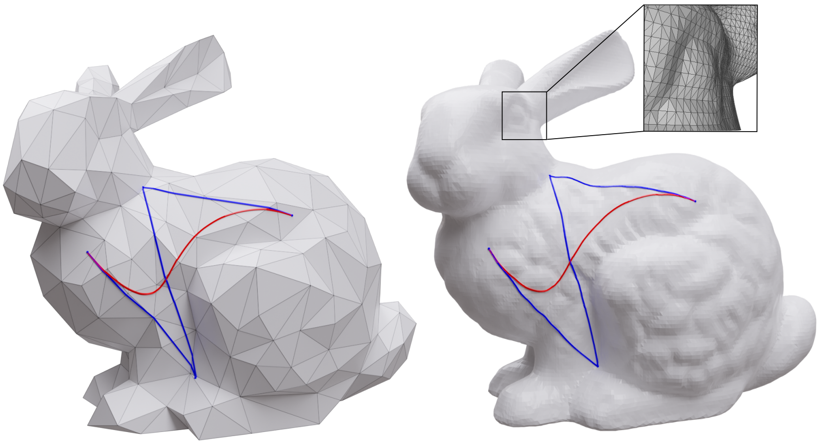


Figure 7.14: Curves traced by positioning the control points in a similar way on two meshes representing the same object, one consisting of ~ 700 triangles (left) and the other consisting of ~ 70k triangles (right). Our method produces similar curves in both cases.

model		WA		ours (OLR)
name	triangles	pre-proc. (s)	tracing (ms)	tracing (ms)
cylinder	10k	54	2–2	1–1
kitten	37k	234	3–3	3–3
bunny	140k	665	2–2	10–12
lion	400k	2316	3–3	4–24
nefertiti	496k	2571	6–64	25–67

Table 7.2: Compared time performances of curve tracing with the WA method and our OLR, on the curves shown in Fig. 7.17 and Fig. 7.18. Each curve is sampled at 67 points; curve tracing times are averaged on each curve repeating tracing 1000 times per curve; we report minimum and maximum times over the different curves shown in the images.

Coarse meshes, boundaries, bumps and creases

Our algorithms are insensitive to the resolution of the mesh and work equally well on coarse as well as refined meshes. Fig. 7.14 shows similar curves obtained on two meshes representing the same shape at two very different resolutions. We can draw curves on meshes with boundaries, too, as shown in Fig. 7.15. In this case, some shortest paths, hence the curves they generate, may be constrained to follow convex portions of the boundary. Since our algorithms work in the intrinsic metric of the surface, they are insensitive to creases and bumps, as shown in Fig. 7.16.

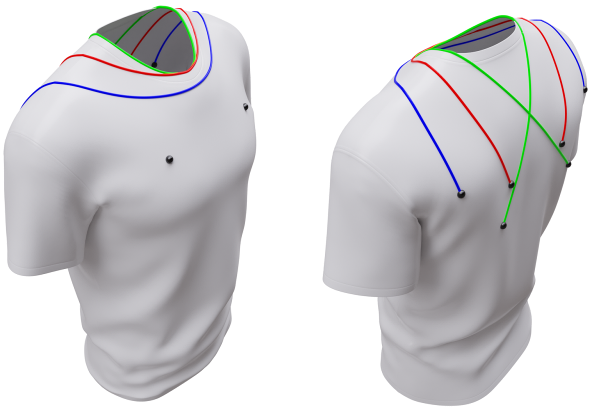


Figure 7.15: Curves traced on a mesh with boundary. The three curves have different anchor points and the same handle points on the front of the shirt (black bullets). The green and the red curves are constrained to partially follow the boundary at the neck.

7.6.3 Comparison with the state-of-the-art

Weighted Averages (WA) [PBDSH13]

Panozzo et al. presented a method to estimate the RCM on a surface, by approximating the geodesic distance on the input mesh M with the Euclidean distance on a higher-dimensional embedding of M [PBDSH13]. Given a set of control points and weights, instead of resolving our Eq. 7.3 on M, they compute the standard affine average of Eq. 7.1 in the embedding space. Then they use a special technique, called *Phong projection*, to bring the resulting space curve to the embedded mesh. Finally they recover the corresponding points on M. We compare with this technique by using the implementation provided by the authors, with the same sampling used in our experiments.

The embedding and the data structures to support Phong projection are computed in a pre-processing step, which is quite heavy in terms of both time and space, and can hardly scale to large datasets (see Table 7.2). We managed to pre-process datasets up to about 500K triangles, but we could not process some of the larger datasets we use in our work, because memory limits were exceeded. The embedding is built by sampling a small subset of the vertices first (fixed to 1000 by the authors), computing all-vs-all geodesic distances on M for such subset, and embedding such vertices in a 8D Euclidean space by keeping their mutual Euclidean distances as close as possible to their geodesic distances on M. The remaining vertices are embedded next, by using the positions of the first embedded vertices as constraints. The connectivity of M is preserved, and the positions of vertices are optimized, so that the distances between adjacent vertices remain as close as possible to their distances on M.

The online phase of WA is very fast, and it is insensitive to the size of the input and the length of the curve (see Table 7.2). However, we experienced a case that took one order of magnitude more time than the others. We conjecture

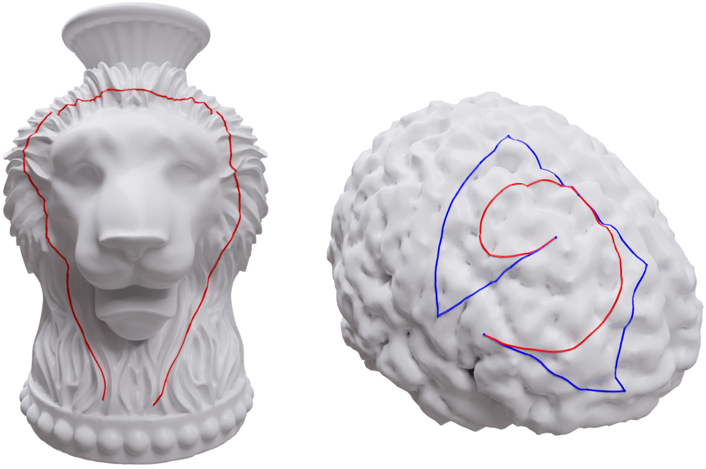


Figure 7.16: Splines traced on meshes with many creases or bumps. Our algorithms work in the intrinsic metric of the surface and are oblivious of normal curvature caused from the embedding.

this is due to some unlucky configuration for the Phong projection, slowing its convergence. On the contrary, the performance of our method is dependent on both the size of the dataset and the length of the curve, being faster than WA on small datasets and shorter curves, and slower on large datasets and long curves. In terms of speed, both methods are equally compatible with interaction on the tested models.

Concerning the quality of the result, the smoothness of the WA embedding, which is necessary to guarantee the smoothness of the Phong projection, cannot be guaranteed, hence the WA method suffers of limitations similar to the RCM method analyzed in Sec. 7.3.4. As soon as the segments of the control polygon become long, relevant artifacts arise, and the curve may even break into several disconnected segments. Some results obtained with the WA method, compared with our results, are shown in Figures 7.17 and 7.18. In particular, Fig. 7.18 exemplifies the behaviors of the two methods as a control polygon becomes larger. While our curve remains smooth and stable throughout, except for the necessary jump between the "reversed S" and the spire, the WA curve becomes unstable and breaks in most configurations where the control points are far apart.

RMC based on vector heat [SSC19b]

Sharp et al. presented the vector heat method, which supports the efficient computation of the log map at an arbitrary point on the surface [SSC19b]. Algorithm 3 in the same paper uses such a log map to estimate the RCM by gradient descent, starting at an initial guess and iteratively converging to the point that minimizes the same energy of our Eq. 7.3. We have implemented the algorithm for tracing a Bézier curve defined with the RCM, by using the authors' implementation of the vector heat, and plugging their Algorithm 3

model		control	subdivided	time (ms)	
name	triangles	polygons	segments	total	per curve
veil	132k	2	402	2.3	1.1
arm	145k	2	856	35.6	17.8
boot	175k	2	755	21.1	10.5
deer	227k	4	1511	21.8	5.4
lady	281k	9	1917	11.4	1.2
car	282k	2	670	28.0	14.0
pumpkin	394k	5	1750	30.0	6.0
rhino	502k	7	2395	39.8	5.6
owls	641k	14	3224	20.8	1.4
alexander	699k	5	1560	20.5	4.1
vase	754k	8	1677	9.0	1.1
nike	5672k	7	4147	253.8	36.2
nefertiti	496k	463	64110	73.4	0.2
dragon	7218k	221	60656	761.7	3.4

Table 7.3: Time performances for curve tracing on the models in Fig. 7.23 and in the teaser, using the uniform OLR algorithm with 5 levels of subdivisions. We report the total time of computing all the curves and the average time of computing a single curve. For all the reported models, our algorithm achieves performance compatible with real-time editing, since the time per curve is at most in the order of tens of milliseconds.

into a loop, where parameter t of Eq. 7.7 varies between 0 and 1, and the point returned at each iteration is taken as initial guess to the next. Figures 7.19 and 7.20 show examples of failure, where curves are either discontinuous or highly perturbed, because the energy has more than one local minimum for certain values of t. The zoom-ins of Fig. 7.20 show the gradient fields before and after the jump. We remark again that such failures stem from an intrinsic limitation of the RCM and are independent of the method for computing it. Concerning efficiency, this algorithm requires computing a log map at each iteration during gradient descent, thus becoming rather slow when applied to curve tracing. We do not report the detailed time performance of this method, which takes minutes to evaluate a few tenths points on a curve in the reported examples.

RDC based on flipOut [SC+19, SC20a]

Sharp and Crane proposed recently the *flipOut* algorithm as a fast solution to the computation of locally shortest geodesic paths [SC20a]. On the basis of the *flipOut* algorithm, the same authors have implemented the algorithm of [MCV08], which uses the same recursive scheme of our RDC algorithm for curve tracing [SC+19].

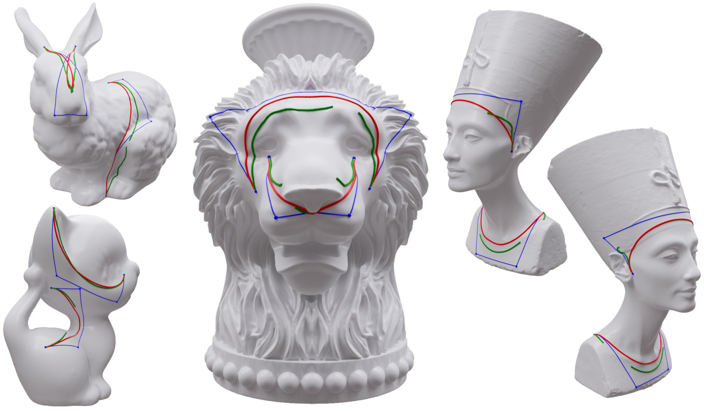


Figure 7.17: Comparisons between the WA method [PBDSH13] (green curves) on ours (red curves); control polygons in blue. The WA curves may contain heavy artifacts (bunny), lose tangency at the endpoints (bunny, nefertiti), or be broken (kitten, lion, nefertiti).

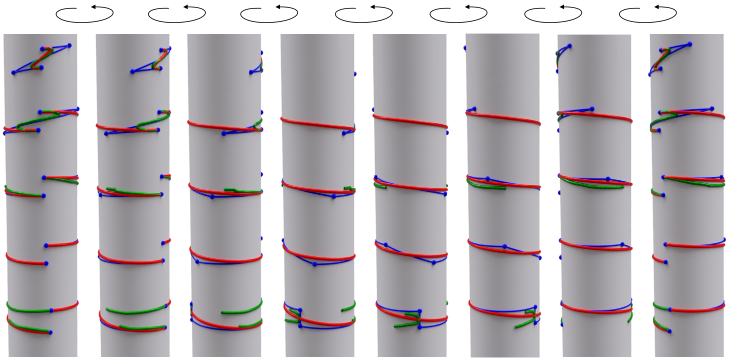


Figure 7.18: Evolution of a curve while dragging handle points about a cylinder (top to bottom, rotated views left to right) with the WA method [PBDSH13] (green curves) and ours (red curves). Our curve jumps from the "reversed S" configuration to the spire and remains stable throughout. The WA curve is stable only in the "reversed S" configuration, next it breaks, then it forms a spire, and eventually it breaks again.

While our algorithms have no limitations, and could provide a valid output in all 556,700 trials, the algorithm in [SC+19] requires that the control polygon does not contain self-intersections, a case which is pretty common with cubic curves, and happens in 33% of the randomly generated polygons. This is due to an intrinsic limitation of the *flipOut* algorithm, which was discussed in [SC20a].

We have used the authors' implementation [SC+19] to run the same experiments of Sec. 7.6.1, with the same parameter used for our RDC algorithm with uniform expansion. Because of the above limitation, we excluded from the comparison all the trials for which their algorithm could not provide an output, keeping a total of 78,854 out of 556,700 trials. From a visual inspection of random samples of the results, it seems that both their algorithm and ours generate the same curves. In Fig. 7.21, we present a comparison between the performances of the two algorithms. Our RDC uniform algorithm exhibits a

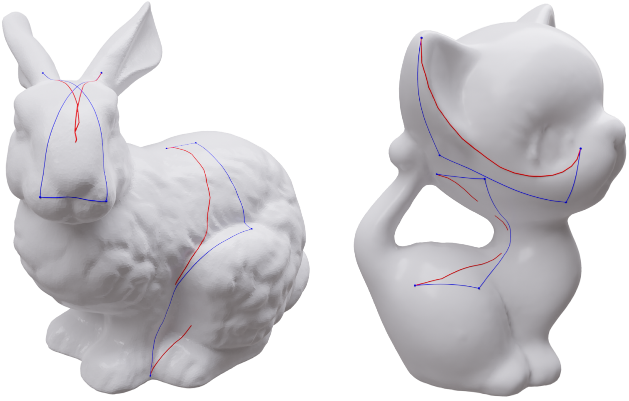


Figure 7.19: The same curves of Fig. 7.17 on the kitten and bunny models have been traced with the RMC based on vector heat [SSC19b]. Some results are either discontinuous or highly perturbed because of non-convex configurations with multiple local minima.

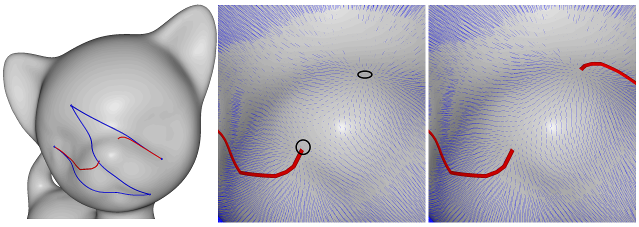


Figure 7.20: Left: A curve traced with the RCM based on vector heat [SSC19b] has a big jump. Center (zoom-in): the gradient of the energy (blue needles) corresponding to the last point of the left branch, where two minima are present (black circles). Right (zoom-in): gradient field corresponding to the first point of the right branch, just one minimum has remained, which is found by gradient descent, causing the jump. Curve tracing occurs from left to right.

speedup of more than 10x on average.

Shortest paths: comparison with flipOut [SC20a]

The speedup in the previous experiment is totally due to our shortest path algorithm described in Section 4.3.1. Note that the flipOut algorithm is one of the fastest available at the state of the art for computing locally shortest paths [SC20a]. We compared the two algorithms by substituting the authors' implementation of flipOut [SC+19] in our curve tracing algorithm in all the trials above, and measuring the times necessary just for the shortest path computations in the two cases. The comparison is shown in Fig. 7.22. Indeed, on average, our algorithm is one order of magnitude faster than flipOut. More precisely, while the times for path shortening (Phases (ii) and (iii) of our algorithm) are comparable with those of flipOut, our speedup is mostly due to the computation of the initial guess (Phase (i)), which is a well known bottleneck for all this class of algorithms.

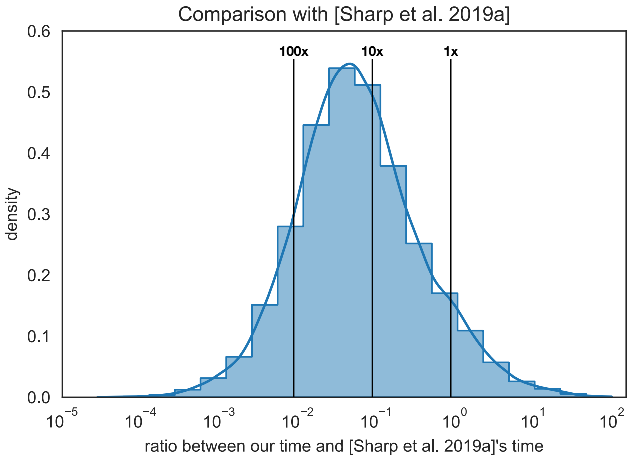


Figure 7.21: The graph shows the distribution of the ratio of the running times between our RDC uniform algorithm and the implementation from [SC⁺19], which is based on the *flipOut* method for computing geodesics [SC20a]. Here we report only the 78,854 trials, out of 556,700, for which [SC⁺19] could provide an output. On average, our RDC algorithm implementation provides more than a 10x speedup over [SC⁺19].

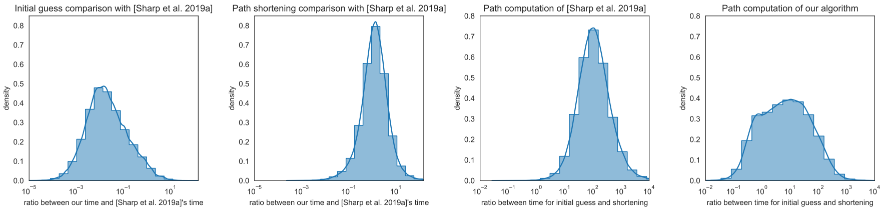


Figure 7.22: Left (2 charts): On average, our algorithm for shortest paths beats by one order of magnitude the flipOut algorithm: the speedup is related to the computation of the initial guess, while the two algorithms have comparable speed for the path shortening stages. Right (2 charts): The initial guess is a bottleneck for both algorithms, but the ratio between the different stages is much more favorable for our algorithm.

7.6.4 Interactive use

We have used extensively our system on a variety of models. All editing sessions where performed on a MacBook laptop with a 2.9GHz Quad-Core Intel Core i7 with 16GB memory, running on a single core.

Fig. 7.23 presents a gallery of curves drawn interactively on objects picked from the Thingi10k collection. Statistics for each example are summarized in Table 7.3. Interaction is quite intuitive, being supported with a GUI that mimics the drawing of spline curves in standard 2D systems, as described in Sec. 7.5.1. The most tricky aspects, with respect to the standard 2D case, are concerned with using tangents that consist of geodesic lines instead of straight lines. In our experience, the use of geodesic tangents, which is intrinsic to the manifold metric, becomes intuitive quickly.

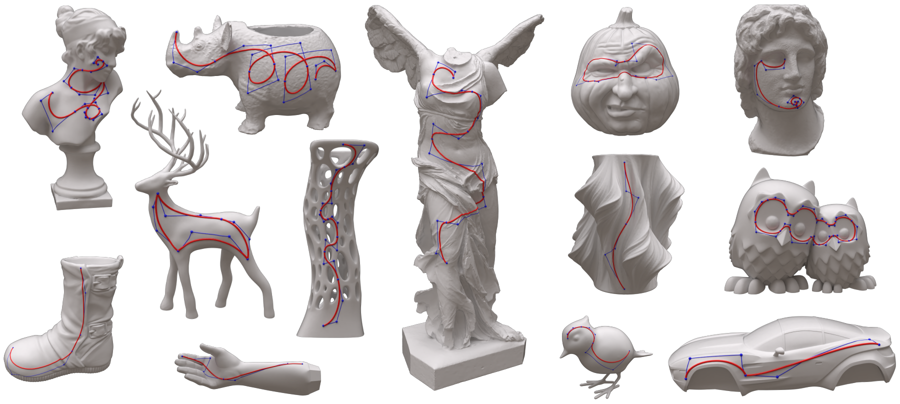


Figure 7.23: A gallery of models and splines drawn with our method. Both smooth (C^1) and corner (C^0) continuity at junction points are exemplified. The selected models span a wide range of shapes and the sizes of meshes vary between about 130k and 5.7M triangles.

We have stressed our system by working on very large meshes as shown in Fig. 7.1. Even on meshes of a few million triangles, our implementation remains interactive, as shown in Table 7.3.

7.7 Concluding remarks

In this chapter, we described methods for interactively drawing and editing of Bézier curves on meshes of millions of triangles, without any limitation on the curve shape and extension of control polygons. Our algorithms are robust, having been tested on over five thousands shapes with over half a million randomly generated control polygons, and they are compatible with interactive usage even on large meshes. Both subdivision schemes we have presented are simple to implement and produce C^1 splines. The Open-uniform Lane-Riesenfeld scheme provides the smoothest practical solution so far for Bézier segments in the manifold setting. It remains an open question how such segments could be chained to obtain C^2 Bézier splines.

The main limitation of these methods lie in the discontinuities of the space of curves with respect to their control points: curves are always smooth, but they may jump between different configurations during editing. Such a discontinuity is inherent of the geodesic metric, and it can be overcome by using a spline with shorter control polygons, instead of a single large polygon, to define the curve. Our algorithms for point insertion greatly help in this task.

In the future, we want to consider other types of splines. An extension of our approach to B-splines is straightforward. An extension to interpolating splines seems easy, but it requires manifold extrapolation, which may become unstable. The most complex extension would be to handle NURBS, which at this point remains unclear how to do. More generally, the smoothness analysis in the non-uniform case needs a thorough investigation.

8

Concluding Remarks

The algorithms proposed in this thesis fulfill the requests formulated in Chapter 1: they generate geometric primitives on highly-tessellated meshes in real-time, and all of them are defined without resorting to any kind of local/global parametrization or projection. Summarizing, we have described algorithms for the tracing of: straighest and shortest geodesic paths, geodesic circles, geodesic polygons, cubic Bézier curves. Moreover, we proposed efficient and robust methods for the estimation of differential quantities and the computations of normal and strongly convex balls. All of these methods have been integrated in a prototype drawing system endowed with a GUI which can be used to interactively draw on highly-tessellated meshes.

Nevertheless, with respect to Euclidean vector graphics, we are missing two very important features. The first one, is a global coordinate system in which all the primitives of a given mesh can be defined. The fact that, the Euclidean case, geometric all the geometric primitives are defined in $\mathbb{R}^2$ (or $\mathbb{R}^3$), allows the user to copy a given shape from one "paper" to another. The second one, is the possibility of considering regions bounded by certain number of primitives, with the purpose of, e.g., coloring it or performing some boolean operation with it. We think both of these features could greatly improve the usability of our drawing system, and we are currently work on how to solve the first of these two problems, as described below.

8.1 Future works

Concerning the definition of a global system of coordinates in which the primitives defined on two different meshes can be defined, we are currently trying to solve a simpler instance of such problem, nonetheless far from being trivial. We aim at mapping geometric primitives from one mesh M to another mesh N representing the same shape of M. To fix ideas, we are addressing the problem of understanding how to map the primitives on the skull of Figure 6.1 to another mesh representing the same skull, but with a different tessellation. The idea we are exploring consists in defining an intrinsic system of coordinates on reference mesh, which is assumed to be good enough representation of the continuous shape S, and then find a way of expressing every point on a given mesh M representing S in such system of coordinates. This in fact would allow the mapping of geometric primitives between two meshes representing the same shape, since such primitives could be expressed in a common reference frame.

Another problem we are currently addressing, is the computation of Riemannian weighted averages. In fact, we did not manage to endow our drawing system with a tool that compute such averages, which would extremely widen the range of possible operations. For example, one may think to all the spline curves defines through weighted averages such as rational Bézier curves, NURBS, etc. The main problem encountered at this regard is the fact the Riemannian center of mass has a unique and well defined minimum if the control points are contained in a strongly convex ball, which doesn't allow the user to pick the points wherever on the mesh. Since we are able to compute strongly convex balls, we can assess whether such an average is well defined, but this is not satisfactory, since we would like to produce a result for an arbitrary positioning of the control points, as done in the case of Bézier curves and as is in the 2D setting. Solving this problem essentially means find a way of approximating the RCM outside strongly convex balls. We are currently investigating an idea similar to the one proposed by Grohs in [Gro13], which basically consists in subdividing the space of the weights into simplicial cells and compute *RCM* through repeated geodesic averages.

Publications

[1] **Gradient Field Estimation on Triangle Meshes**
Claudio Mancinelli, Marco Livesu, Enrico Puppo
STAG-Smart Tools and Apps for Graphics, 2018 (Brescia, Italy)

[2] **A Comparison of Methods for Gradient Field Estimation on Simplicial Meshes**
Claudio Mancinelli, Marco Livesu, Enrico Puppo
Computer & Graphics 2019 (Journal extension of [1])

[3] **Practical Computation of the Cut Locus on Discrete Surfaces**
Claudio Mancinelli, Marco Livesu, Enrico Puppo
Computer Graphics Forum, 2021

[4] **Straightedge and Compass Constructions on Surfaces**
Claudio Mancinelli, Enrico Puppo
STAG-Smart Tools and Apps for Graphics, 2021 (Rome, Italy)

[5] **Vector Graphics on Surfaces Using Straightedge and Compass Constructions**
Claudio Mancinelli, Enrico Puppo
Computers & Graphics, 2022 (Journal extension of [4])

[6] **b/Surf: Interactive Bézier Splines on Surfaces**
Claudio Mancinelli*, Giacomo Nazzaro*, Fabio Pellacini, Enrico Puppo
IEEE Transactions on Visualization and Computer Graphics, 2022, presented at SGP 2022.
*: joint first author

[7] **Non-uniform interpolatory subdivision schemes with improved smoothness**
Nira Dyn, Kai Hormann, Claudio Mancinelli
Computer Aided Design, 2022

Bibliography

[Ado21] Adobe illustrator, 2021.

[Afs09] B. Afsari. *Means and Averaging on Riemannian Manifolds*. PhD thesis, University of Maryland, 2009.

[AGS$^+$15] A. Arnould, P.-Y. Gousenbourger, C. Samir, P.-A. Absil, and M. Canis. Fitting Smooth Paths on Riemannian Manifolds: Endometrial Surface Reconstruction and Preoperative MRI-Based Navigation. In *Geometric Science of Information*, pages 491–498. Springer, Cham, 2015.

[AGSW16] P.A. Absil, P.-Y. Gousenbourger, P. Striewski, and B. Wirth. Differentiable Piecewise-Bézier Surfaces on Riemannian Manifolds. *SIAM Jou. on Imaging Sciences*, 9(4):1788–1828, 2016.

[AKP19] S. Alexander, V. Kapovitch, and A. Petrunin. Alexandrov geometry: preliminary version no. 1, 2019.

[Ale48] A. D. Alexandrov. Intrinsic geometry of convex surfaces. *OGIZ, Moscow-Leningrad*, 1948.

[ALM00] Lyudmil Aleksandrov, Mark Lanthier, and Anil Maheshwari. An epsilon-approximation algorithm for weighted shortest paths on polyhedral surfaces. *Lecture Notes in Computer Science*, 1432, 06 2000.

[AMP08] P A Absil, R Mahoney, and R Sepulchre Press. Optimization Algorithms on Matrix Manifolds. Princeton University Press, 2008.

[AMS00] Lyudmil Aleksandrov, Anil Maheshwari, and Jörg-Rüdiger Sack. Approximation algorithms for geometric shortest path problems. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 286–295, New York, NY, USA, 2000. Association for Computing Machinery.

[AMS05] L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *J. ACM*, 52(1):25–53, jan 2005.

[AOCBC15] Omri Azencot, Maks Ovsjanikov, Frédéric Chazal, and Mirela Ben-Chen. Discrete derivatives of vector fields on surfaces – an operator approach. *ACM Trans. Graph.*, 34(3), May 2015.

[Aut21] Autodesk. Mudbox, 2021.

[BBA21] Astrid Bunge, Mario Botsch, and Marc Alexa. The diamond laplace for polygonal and polyhedral meshes. *Eurographics Symposium on Geometry Processing*, 40, 2021.

[Ber98] D.P. Bertsekas. *Network optimization: continuous and discrete models.* Athena Scientific, Belmont, Massachusetts, 1998.

[Ber07] M. Berger. *A Panoramic View of Riemannian Geometry.* Springer Berlin Heidelberg, 2007.

[Bis77] Richard L. Bishop. Decomposition of cut loci. *Proceedings of the American Mathematical Society*, 65(1):133–133, 1977.

[BMBZ02] Henning Biermann, Ioana Martin, Fausto Bernardini, and Denis Zorin. Cut-and-paste editing of multiresolution surfaces. *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '02*, pages 312–321, 2002.

[BMSW11] Prosenjit Bose, Anil Maheshwari, Chang Shu, and Stefanie Wuhrer. A survey of geodesic paths on 3d surfaces. *Computational Geometry*, 44(9):486–498, 2011.

[Buc77] Michael A. Buchner. Simplicial structure of the real analytic cut locus. *Proc. Amer. Math. Soc.*, 64(1):118–121, 1977.

[BvdPPH11] Nicolas Bonneel, Michiel van de Panne, Sylvain Paris, and Wolfgang Heidrich. Displacement interpolation using lagrangian mass transport. *ACM Trans. Graph.*, 30(6):1–12, 2011.

[BYF$^+$19] Max Budninskiy, Gloria Yin, Leman Feng, Yiying Tong, and Mathieu Desbrun. Parallel transport unfolding: A connection-based manifold learning approach. *SIAM Jou. Appl. Algebra and Geo.*, 3(2):266–291, 2019.

[Cai35] S. S. Cairns. Triangulation of the manifold of class one. *Bulletin of the American Mathematical Society*, 41(8):549 – 552, 1935.

[CC89] S.S. Chern and L. Cesari. *Global Differential Geometry.* Number v. 27 in Global Differential Geometry. Mathematical Association of America, 1989.

[CCC$^+$08] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In *EG Ital. Chap. Conf.*, pages 129–136, 2008.

[CDHR08] Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. 35(3), 2008.

[CDS07] T.J. Cashman, N.A. Dodgson, and M.A. Sabin. Non-uniform B-spline subdivision using refine and smooth. In R. Martin, M. Sabin, and J. Winkler, editors, *Mathematics of Surfaces XII - LNCS*, volume 4647. Springer, Berlin Heidelberg, 2007.

[CE75] J. Cheeger and D.G. Ebin. *Comparison Theorems in Riemannian Geometry*. North-Holland mathematical library. North-Holland Publishing Company, 1975.

[CEMS01] Dario Cordero-Erausquin, Robert J McCann, and Michael Schmuckenschläger. A Riemannian interpolation inequality à la Borell, Brascamp and Lieb. *Inventiones Mathematicae*, 146(2):219–257, 2001.

[CH90] Jindong Chen and Yijie Han. Shortest paths on a polyhedron. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, SCG '90, pages 360–369, New York, NY, USA, 1990. Association for Computing Machinery.

[Cha06] I. Chavel. *Riemannian Geometry: A Modern Introduction*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2006.

[CHM09] Carlos D Correa, Robert Hero, and Kwan-Liu Ma. A comparison of gradient estimation methods for volume rendering on unstructured meshes. *IEEE Transactions on Visualization & Computer Graphics*, (3):305–319, 2009.

[CL05] Frédéric Chazal and André Lieutier. The "λ-medial axis". *Graphical Models*, 67(4):304–331, 2005.

[CLO98] David A. Cox, John B. Little, and Donal O'Shea. *Using Algebraic Geometry*, volume 185 of *Graduate Texts in Mathematics*. Springer, first edition, 1998.

[CLPQ20] K. Crane, M. Livesu, E. Puppo, and Y. Qin. A survey of algorithms for geodesic paths and distances, 2020.

[CSLC95] M. Camarinha, F. Silva Leite, and P. Crouch. Splines of class C^k on non-euclidean spaces . *IMA Jou. of Math. Control and Information*, 12(4):399–410, 1995.

[CWW13] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Trans. Graph.*, 32(5), 2013.

[dC76] Manfredo P. do Carmo. *Differential geometry of curves and surfaces*. Prentice Hall, 1976.

[dC92] M.P. do Carmo. *Riemannian Geometry*. Mathematics (Boston, Mass.). Birkhäuser, 1992.

[DGDT16] Fernando De Goes, Mathieu Desbrun, and Yiying Tong. Vector field processing on triangle meshes. In *ACM SIGGRAPH 2016 Courses, SIGGRAPH 2016*, pages 1–49, New York, New York, USA, July 2016. Pixar Animation Studios, United States, ACM Press.

[DGL19] N. Dyn, R. Goldman, and D. Levin. High order smoothness of non-linear Lane-Riesenfeld algorithms in the functional setting. *Computer Aided Geometric Design*, 71:119–129, 2019.

[Dib17] James Dibble. The convexity radius of a riemannian manifold. *Asian Journal of Mathematics*, 21:169–174, 2017.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[DL09] Tamal K. Dey and Kuiyu Li. Cut locus and topology from surface point data. *Proce. Symp. on Comp. Geo.*, pages 125–134, 2009.

[DS17] N. Dyn and N. Sharon. Manifold-valued subdivision schemes based on geodesic inductive averaging. *Journal of Computational and Applied Mathematics*, 311:54–67, 2017.

[DXY18] T. Duchamp, G. Xie, and T. Yu. Smoothing nonlinear subdivision schemes by averaging. *Numerical Algorithms*, 77(2):361–379, 2018.

[Epp03] David Eppstein. Dynamic generators of topologically embedded graphs. In *Proc. ACM-SIAM Symp. Disc. Alg.*, pages 599–608, 2003.

[EW05] Jeff Erickson and Kim Whittlesey. Greedy optimal homotopy and homology generators. In *Proc. 16th ACM-SIAM Symp. Disc. Alg.*, volume 5, pages 1038–1046, 2005.

[Far01] G. Farin. *Curves and Surfaces for CAGD: A Practical Guide.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2001.

[Ful97] W. Fulton. *Algebraic Topology: A First Course.* Graduate Texts in Mathematics. Springer New York, 1997.

[Gén20] François Générau. *On a stable variational approximation of the cut locus, and a non-local isoperimetric problem.* PhD thesis, Université Grenoble Alpes, June 2020.

[GHL04] S. Gallot, D. Hulin, and J. Lafontaine. *Riemannian Geometry.* Universitext. Springer Berlin Heidelberg, 2004.

[GK73] K. Grove and H. Karcher. How to conjugate c1-close group actions. *Mathematische Zeitschrift*, 132:11–20, 1973.

[GMA18] P.-Y. Gousenbourger, E. Massart, and P.A. Absil. Data Fitting on Manifolds with Composite Bézier-Like Curves and Blended Cubic Splines. *Jou. of Math. Imaging and Vision*, 61:1–27, 2018.

[GMST05] J. Gravensen, S. Markvorsen, R. Sinclair, and M. Tanaka. The Cut Locus of a Torus of Revolution. *Asian Journal of Mathematics*, 9(1):103 – 120, 2005.

[GOV22] Générau, François, Oudet, Edouard, and Velichkov, Bozhidar. Numerical computation of the cut locus via a variational approximation of the distance function. *ESAIM: M2AN*, 56(1):105–120, 2022.

[Gro13] P Grohs. Quasi-interpolation in Riemannian manifolds. *IMA Journal of Numerical Analysis*, 33(3):849–874, July 2013.

[GSA14] P.-Y. Gousenbourger, C. Samir, and P.A. Absil. Piecewise-Bézier C1 Interpolation on Riemannian Manifolds with Application to 2D Shape Morphing. In *Proc. 22nd Int. Conf. on Pattern Recognition*, pages 4086–4091, 2014.

[HA19] Philipp Herholz and Marc Alexa. Efficient Computation of Smoothed Exponential Maps. *Computer Graphics Forum*, January 2019.

[HP04] M. Hofer and H. Pottmann. Energy-minimizing splines in manifolds. *ACM Trans, Graph.*, 23:284–293, 2004.

[Ink21] Inkscape's Contributors. Inkscape, 2021.

[IS04] Jin-ichi Itoh and Robert Sinclair. Thaw: A Tool for Approximating Cut Loci on a Triangulation of a Surface. *Experimental Mathematics*, 13(3):309 – 325, 2004.

[JP+18] Alec Jacobson, Daniele Panozzo, et al. libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/.

[JSW+19] Y. Jin, D. Song, T. Wang, J. Huang, Y. Song, and L. He. A shell space constrained approach for curve design on surface meshes. *Computer-Aided Design*, 113:24–34, 2019.

[Kar77] H Karcher. Riemannian center of mass and mollifier smoothing. *Communications on Pure and Applied Mathematics*, 30(5):509–541, 1977.

[KCPS13] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Globally optimal direction fields. *ACM Trans. Graph.*, 32(4), 2013.

[KCPS15] F Knöppel, K Crane, U Pinkall, and P Schröder. Stripe patterns on surfaces. *ACM Trans. Graph.*, 34(4):1–11, 2015.

[KRG03] Peter Kipfer, Frank Reck, and Günther Greiner. Local Exact Particle Tracing on Unstructured Grids. *Computer Graphics Forum*, 22(2):133–142, June 2003.

[KS98] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences of the United States of America*, 95:8431–8435, 1998.

[KSH⁺03] G. Kumar, Prabha Srinivasan, V. Holla, K. Shastry, and B. Prakash. Geodesic curve computations on surfaces. *Computer Aided Geometric Design*, 20:119–133, 05 2003.

[Lan97] Mark Anthony Lanthier. Approximating weighted shortest paths on polyhedral surfaces. In *Proceedings of the thirteenth annual symposium on Computational geometry - SCG '97*. ACM Press, 1997.

[Lan00] Mark Anthony Lanthier. *Shortest Path Problems on Polyhedral Surfaces*. PhD thesis, CAN, 2000. AAINQ48348.

[LBS05] T. Langer, A.G. Belyaev, and H.-P. Seidel. Asymptotic analysis of discrete normals and curvatures of polylines. In *Proc. 21st Spring Conf. on Comp. Graph.*, pages 229–232, 2005.

[LCT11] Y J Liu, Z Chen, and K Tang. Construction of iso-contours, bisectors, and Voronoi diagrams on triangulated surfaces. *IEEE Trans. Pattern Analysis Machine Intelligence*, 33(8):1502–1517, 2011.

[LD11] Yaron Lipman and Ingrid Daubechies. Conformal wasserstein distances: Comparing surfaces in polynomial time. *Advances in Mathematics*, 227(3):1047–1077, 2011.

[Le 19] A. Le Brigant. A Discrete Framework to Find the Optimal Matching Between Manifold-Valued Curves. *Jou. of Mathematical Imaging and Vision*, 61(1):40–70, 2019.

[Lee19] J.M. Lee. *Introduction to Riemannian Manifolds*. Graduate Texts in Mathematics. Springer International Publishing, 2019.

[LGS12] Marco Livesu, Fabio Guggeri, and Riccardo Scateni. Reconstructing the curve-skeletons of 3d shapes using the visual hull. *IEEE Trans. Vis. Comp. Graph.*, 18(11):1891–1901, 2012.

[Liv19] Marco Livesu. cinolib: a generic programming header only c++ library for processing polygonal and polyhedral meshes. *Trans. Comp. Sci. XXXIV*, 2019. https://github.com/mlivesu/cinolib/.

[LMRS01] M. Lanthier, A. Maheshwari, and J. R. Sack. Approximating shortest paths on weighted polyhedral surfaces. *Algorithmica*, 30(4):527–562, 2001.

[LP84] D.T. Lee and F.P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

[LR80] J.M. Lane and R.F. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2(1):35–46, 1980.

[LTGD16] Beibei Liu, Yiying Tong, Fernando De Goes, and Mathieu Desbrun. Discrete connection and covariant derivative for vector field analysis and design. *ACM Transactions on Graphics*, 35:1–17, 2016.

[LW01] A. Lin and M. Walker. CAGD techniques for differentiable manifolds. In J. Levesley, I.J. Anderson, and J.C. Mason, editors, *Algorithms for Approximation IV - Proc. Int. Symp. on Algorithms for Approximation*, pages 36–43. Huddlersfield University, UK, 2001.

[LZX+08] Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman, and Steven J Gortler. A local/global approach to mesh parameterization. In *Computer Graphics Forum*, volume 27, pages 1495–1504. Wiley Online Library, 2008.

[Mac49] Richard MacNeal. *The Solution of Partial Differential Equations by Means of Electrical Networks*. PhD thesis, Caltech, 1949.

[MBAM11] Marek Krzysztof Misztal, Jakob Andreas Bærentzen, François Anton, and Steen Markvorsen. Cut locus construction using deformable simplicial complexes. *Proc. Int. Symp. Vor. Diag. Sci. Eng.*, 2:134–141, 2011.

[MBBV15] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *IEEE ICCV Workshop*, pages 832–840, 2015.

[MCSK+17] Manish Mandad, David Cohen-Steiner, Leif Kobbelt, Pierre Alliez, and Mathieu Desbrun. Variance-minimizing transport plans for inter-surface mapping. volume 36. Association for Computing Machinery, 2017.

[MCV08] D.M. Morera, P.C. Carvalho, and L. Velho. Modeling on triangulations with geodesic curves. *The Visual Computer*, 24:1025–1037, 2008.

[MDSB03] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H Barr. Discrete Differential-Geometry Operators for Triangulated 2-Manifolds. In *Visualization and Mathematics III*, pages 35–57. Springer, Berlin, Heidelberg, Berlin, Heidelberg, 2003.

[MKK21] Thomas W. Mitchel, Vladimir G. Kim, and Michael Kazhdan. Field Convolutions for Surface CNNs, 2021.

[MLP19] Claudio Mancinelli, Marco Livesu, and Enrico Puppo. A comparison of methods for gradient field estimation on simplicial meshes. *Computers & Graphics*, 80:37 – 50, 2019.

[MLP21] Claudio Mancinelli, Marco Livesu, and Enrico Puppo. Practical computation of the cut locus on discrete surfaces. *Computer Graphics Forum*, 40, 2021.

[MM97] Christian S. Mata and Joseph S. B. Mitchell. A new algorithm for computing shortest paths in weighted planar subdivisions (extended abstract). In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, SCG '97, pages 264–273, New York, NY, USA, 1997. Association for Computing Machinery.

[MM02] Carlo Mantegazza and Andrea Carlo Mennucci. Hamilton—Jacobi Equations and Distance Functions on Riemannian Manifolds. *Applied Mathematics & Optimization*, 47(1):1–25, 2002.

[MMP87] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16(4):647–668, August 1987.

[MMU14] Carlo Mantegazza, Giovanni Mascellani, and Gennady Uraltsev. On the Distributional Hessian of the Distance Function. *Pacific J. of Math.*, 270:151–166, 2014.

[MNPP22] C. Mancinelli, G. Nazzaro, F. Pellacini, and E. Puppo. b/surf: Interactive bézier splines on surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[MP22] Claudio Mancinelli and Enrico Puppo. Vector graphics on surfaces using straightedge and compass constructions. *Computers & Graphics*, 2022.

[MRCK21] Thomas W Mitchel, Szymon Rusinkiewicz, Gregory S Chirikjian, and Michael Kazhdan. Echo: Extended convolution histogram of orientations for local surface description. *Comp. Graph. Forum*, 40(1):180–194, 2021.

[MVdC04] Dimas Martínez, Luiz Velho, and Paulo de Carvalho. *Geodesic Paths on Triangular Meshes.* 01 2004.

[Mye35] Sumner Byron Myers. Connections between differential geometry and topology I,II. *Duke Mathematical Journal*, 1(3):276 – 391, 1935.

[Nee07] Robert Neel. The small-time asymptotics of the heat kernel at the cut locus. *Comm. in Analysis and Geometry*, 15(4):845–890, 2007.

[NHP89] L. Noakes, G. Heinzinger, and B. Paden. Cubic splines on curved spaces. *IMA Jou. of Math. Control and Information*, 6:465–473, 1989.

[NJ99] G M Nielson and Il-Hong Jung. Tools for computing tangent curves for linearly varying vector fields over tetrahedral domains. *IEEE Transactions on Visualization and Computer Graphics*, 1999.

[Noa98] L. Noakes. Nonlinear corner-cutting. *Adv. in Comp. Math.*, 8(3):165–177, 1998.

[Noa99] L. Noakes. Accelerations of Riemannian quadratics. *Proc. of the American Math. Soc.*, 127(6):1827–1836, 1999.

[NPP20] Giacomo Nazzaro, Enrico Puppo, and Fabio Pellacini. Decosurf: Recursive geodesic patterns on triangle meshes, 2020.

[NPP22] G. Nazzaro, E. Puppo, and F. Pellacini. geoTangle: interactive design of geodesic tangle patterns on surfaces. *ACM Trans. Graph.*, 41(2):12:1–12:17, 2022.

[NYP13] E. Nava-Yazdani and K. Polthier. De Casteljau's algorithm on manifolds. *Computer Aided Geometric Design*, 30(7):722–732, 2013.

[PBDSH13] D. Panozzo, I. Baran, O. Diamanti, and O. Sorkine-Hornung. Weighted averages on surfaces. *ACM Trans. Graph.*, 32(4):60:1–12, 2013.

[PH05] H. Pottmann and M. Hofer. A variational approach to spline curves on surfaces. *Computer aided geometric design*, 22(7):693–709, 2005.

[PHD+10] H. Pottmann, Q. Huang, B. Deng, A. Schiftner, M. Kilian, L. Guibas, and J. Wallner. Geodesic patterns. *ACM Trans. Graph.*, 29(4):43:1–43:10, 2010.

[Pil21] Pilgway. 3d-coat, 2021.

[Pix22] Pixologic. Zbrush, 2022.

[PN07] T. Popiel and L. Noakes. Bézier curves and C2 interpolation in Riemannian manifolds. *Jou. of Approximation Theory*, 148(2):111–127, 2007.

[PNC19] Fabio Pellacini, Giacomo Nazzaro, and Edoardo Carra. Yocto/GL: A Data-Oriented Library For Physically-Based Graphics. In *EG Ital. Chap. Conf.*, 2019. https://github.com/xelatihy/yocto-gl.

[Poi05] Henri Poincare. Sur les lignes geodesiques des surfaces convexes. *Trans. American Math. Soc.*, 6(3):237–274, 1905.

[PR95] F.C. Park and B. Ravani. Be´ zier curves on Riemannian manifolds and Lie groups with kinematics applications. *Jou. of Mechanical Design*, 117(1):36–40, 1995.

[PS98] Konrad Polthier and Markus Schmies. Mathematical visualization. pages 135–150. Springer, 1998.

[PSOA18] M. Poerner, J. Suessmuth, D. Ohadi, and V. Amann. adidas TAPE: 3-d footwear concept design . In *ACM SIGGRAPH 2018 Talks*, pages 1–2, New York, 2018. ACM Press.

[QHY+16] Yipeng Qin, Xiaoguang Han, Hongchuan Yu, Yizhou Yu, and Jianjun Zhang. Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation. *ACM Trans. Graph.*, 35(4):125:1–125:13, 2016.

[RGA+20] Marie-Julie Rakotosaona, Paul Guerrero, Noam Aigerman, Niloy Mitra, and Maks Ovsjanikov. Learning delaunay surface elements for mesh reconstruction, 2020.

[RKS00] C Rössl, L Kobbelt, HP Seidel, and 2000. Extraction of feature lines on triangulated surfaces using morphological operators. In *AAAI Symp. on Smart Graphics*, 2000.

[Sak97] Takashi Sakai. *Riemannian Geometry*, volume 149. American Mathematical Society, 1997.

[Sal06] D Salomon. *Curves and surfaces for computer graphics.* Springer, New York, 2006.

[SASK11] C. Samir, P.A. Absil, A. Srivastava, and E. Klassen. A Gradient-Descent Method for Curve Fitting on Riemannian Manifolds. *Foundations of Comp. Math.*, 12(1):49–73, 2011.

[SC+19] Nicholas Sharp, Keenan Crane, et al. geometry-central, 2019. www.geometry-central.net.

[SC20a] N. Sharp and K. Crane. You can find geodesic paths in triangle meshes by just flipping edges. *ACM Trans. Graph.*, 39(6):249:1–15, 2020.

[SC20b] Nicholas Sharp and Keenan Crane. A Laplacian for Nonmanifold Triangle Meshes. *Computer Graphics Forum (SGP)*, 39(5), 2020.

[Sch13] R Schmidt. Stroke Parameterization. *Comp. Graph. Forum*, 32(2pt2):255–263, May 2013.

[SDGP+15] Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Trans. Graph.*, 34(4):1–11, 2015.

[SGW06] R. Schmidt, C. Grimm, and B. Wyvill. Interactive decal compositing with discrete exponential maps. *ACM Trans. Graph.*, 25(3):605–613, 2006.

[SH02] Alla Sheffer and John C Hart. Seamster: inconspicuous low-distortion texture seam layout. In *IEEE Vis.*, pages 291–298, 2002.

[Sol18] Justin Solomon. Optimal transport on discrete domains. *AMS Short Course on Discrete Differential Geometry*, 2018.

[SRC+20] Zhiyu Sun, Ethan Rooke, Jerome Charton, Yusen He, Jia Lu, and Stephen Baek. Zernet: Convolutional neural networks on arbitrary surfaces via zernike local tangent space estimation. *Comp. Graph. Forum*, 39(6):204–216, 2020.

[SRGB14] Justin Solomon, Raif Rustamov, Leonidas Guibas, and Adrian Butscher. Earth mover's distances on discrete surfaces. *ACM Trans. Graph.*, 33(4):1–12, 2014.

[SS84] Micha Sharir and Amir Schorr. On shortest paths in polyhedral spaces. STOC '84, pages 144–153, New York, NY, USA, 1984. Association for Computing Machinery.

[SSC19a] N Sharp, Y Soliman, and K Crane. Navigating intrinsic triangulations. *ACM Transactions on Graphics*, 38(4), 2019.

[SSC19b] N Sharp, Y Soliman, and K Crane. The vector heat method. *ACM Trans. Graph.*, 38(3):1–19, June 2019.

[SSK+05] Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J. Gortler, and Hugues Hoppe. Fast exact and approximate geodesics on meshes. *ACM Transactions on Graphics*, 24:553, 2005.

[ST02] Robert Sinclair and Minoru Tanaka. Loki: Software for Computing Cut Loci. *Experimental Math.*, 11(1):1 – 25, 2002.

[SZZ+13] Q. Sun, L. Zhang, M. Zhang, X. Ying, S.-Q. Xin, J. Xia, and Y. He. Texture brush: An interactive surface texturing interface. In *Proc. ACM SIGGRAPH Symp. Interactive 3D Graphics and Games*, pages 153–160, New York, NY, USA, 2013. ACM.

[TDS+16] Andrea Tagliasacchi, Thomas Delame, Michela Spagnuolo, Nina Amenta, and Alexandru Telea. 3d skeletons: A state-of-the-art report. In *Comp. Graph. Forum*, volume 35, pages 573–597, 2016.

[TW97] Grit Thürmer and Charles A. Wüthrich. Normal computation for discrete surfaces in 3d space. *Computer Graphics Forum*, 16(3):C15–C26, 1997.

[Var67] S. R.S. Varadhan. On the behavior of the fundamental solution of the heat equation with variable coefficients. *Communications on Pure and Applied Mathematics*, 20(2):431–455, May 1967.

[VCD$^+$17] A. Vaxman, M. Campen, O. Diamanti, D. Bommes, K. Hildebrandt, M. Ben-Chen, and D. Panozzo. Directional field synthesis, design, and processing. In *ACM SIGGRAPH 2017 Courses*. ACM, 2017.

[Vil08] Cédric Villani. *Optimal Transport Old and New*. Springer, 2008.

[Vil11] Cédric Villani. Regularity of optimal transport and cut locus: From nonsmooth analysis to geometry to smooth analysis. *Discrete and Continuous Dynamical Systems*, 30(2):559–571, 2011.

[W3C10] Scalable vector graphics, 2010.

[Wal06] J. Wallner. Smoothness Analysis of Subdivision Schemes by Proximity. *Constructive Approximation*, 24(3):289–318, November 2006.

[Wal20] Johannes Wallner. Geometric subdivision and multiscale transforms. In Philipp Grohs, Martin Holler, and Andreas Weinmann, editors, *Handbook of Variational Methods for Nonlinear Geometric Data*, pages 121–152. Springer, 2020.

[Wan04] Charlie Wang. Cybertape: An interactive measurement tool on polyhedral surface. *Computers & Graphics*, 28:731–745, 10 2004.

[WFW$^+$17] Xiaoning Wang, Zheng Fang, Jiajun Wu, Shi Qing Xin, and Ying He. Discrete geodesic graph (dgg) for computing geodesic distances on polyhedral surfaces. *Computer Aided Geometric Design*, 52-53:262–284, 2017.

[Whi33] J. H.C. Whitehead. Convex regions in the geometry of paths-addendum. *Quarterly Journal of Mathematics*, os-4:226–227, 1933.

[Whi40] J.H.C. Whitehead. On c^1-complexes. *Annals of Mathematics*, 41:809–824, 1940.

[Whi57] Hassler Whitney. *Geometric Integration Theory*. Princeton University Press, 1957.

[WMKG07] Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun. Discrete laplace operators: No free lunch. *Eurographics Symposium on Geometry Processing*, 2007.

[WP06] J. Wallner and H. Pottmann. Intrinsic subdivision with smooth limits for graphics and animation. *ACM Trans. Graph.*, 25(2):356–374, 2006.

[Xu13] Guoliang Xu. Consistent approximations of several geometric differential operators and their convergence. *Applied Numerical Mathematics*, 69:1–12, 2013.

[Xu18] Shicheng Xu. Local estimate on convexity radius and decay of injectivity radius in a riemannian manifold. *Communications in Contemporary Mathematics*, 20, 2018.

[XW07] Shi Qing Xin and Guo Jin Wang. Efficiently determining a locally exact shortest path on polyhedral surfaces. *CAD Computer Aided Design*, 39:1081–1090, 2007.

[XW09] Shi-Qing Xin and Guo-Jin Wang. Improving chen and han's algorithm on the discrete geodesic problem. *ACM Transactions on Graphics*, 28:1–8, 2009.

[XWL$^+$15] Chunxu Xu, Tuanfeng Y. Wang, Yong Jin Liu, Ligang Liu, and Ying He. Fast wavefront propagation (fwp) for computing exact geodesic distances on meshes. *IEEE Transactions on Visualization and Computer Graphics*, 21:822–834, 2015.

[YCS00] Peter J Yim, Peter L Choyke, and Ronald M Summers. Grayscale skeletonization of small vessels in magnetic resonance angiography. *IEEE Trans. Medical Imaging*, 19(6):568–576, 2000.

[YLT19] C. Yuksel, S. Lefebvre, and M. Tarini. Rethinking Texture Mapping. *Comp. Graph. Forum*, 38(2):535–551, 2019.

[YWH13] Xiang Ying, Xiaoning Wang, and Ying He. Saddle vertex graph (svg): A novel solution to the discrete geodesic problem. *ACM Trans. Graph.*, 32(6), nov 2013.

[ZJ16] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. 2016.

[ZMT07] Eugene Zhang, Konstantin Mischaikow, and Greg Turk. Vector field design on surfaces. *ACM Transactions on Graphics*, 25:1294–1326, 2007.