Operator Representations in Geometry Processing

Omri Azencot

Operator Representations in Geometry Processing

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Omri Azencot

Submitted to the Senate
of the Technion — Israel Institute of Technology
Iyar, 5777 Haifa May, 2017

This research was carried out under the supervision of Asst. Prof. Mirela Ben-Chen, in the Computer Science Department.

Some results in this thesis have been published as articles by the author and research collaborators in conferences and journals during the course of the author's doctoral research period, the most up-to-date versions of which being:

Omri Azencot, Mirela Ben-Chen, Frédéric Chazal, and Maks Ovsjanikov. An operator approach to tangent vector field processing. In *Computer Graphics Forum*, volume 32, pages 73–82. Wiley Online Library, 2013.

Omri Azencot, Etienne Corman, Mirela Ben-Chen, and Maks Ovsjanikov. Consistent functional cross field design for mesh adrangulation. *ACM Transactions on Graphics (TOG)*, 36(4), 2017.

Omri Azencot, Maks Ovsjanikov, Frédéric Chazal, and Mirela Ben-Chen. Discrete derivatives of vector fields on surfaces–an operator approach. *ACM Transactions on Graphics (TOG)*, 34(3):29, 2015.

Omri Azencot, Orestis Vantzos, and Mirela Ben-Chen. Advection-based function matching on surfaces. In *Computer Graphics Forum*, volume 35, pages 55–64. Wiley Online Library, 2016.

Omri Azencot, Orestis Vantzos, Max Wardetzky, Martin Rumpf, and Mirela Ben-Chen. Functional thin films on surfaces. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 137–146. ACM, 2015.

Omri Azencot, Steffen Weißmann, Maks Ovsjanikov, Max Wardetzky, and Mirela Ben-Chen. Functional fluids on surfaces. In *Computer Graphics Forum*, volume 33, pages 237–246. Wiley Online Library, 2014.

Orestis Vantzos, Omri Azencot, Max Wardeztky, Martin Rumpf, and Mirela Ben-Chen. Functional thin films on surfaces. *IEEE transactions on visualization and computer graphics*, 23(3):1179–1192, 2017.

The generous financial help of the Technion, the Computer Science Department, the Jacobs–Qualcomm Fellowship, the Adams Fellowship of the Israel Academy of Sciences and Humanities, and Prof. Rahamimoff Travel Grant for Young Scientists of the US-Israel Binational Science Foundation (BSF) is gratefully acknowledged.

Contents

List of Figures

Abstract

This thesis introduces fundamental equations as well as discrete tools and numerical methods for carrying out various geometrical tasks on three-dimensional surfaces via *operators*. An example for an operator is the Laplacian which maps real-valued functions to their sum of second derivatives. More generally, many mathematical objects feature an operator interpretation, and in this work, we consider a few of them in the context of geometry processing and numerical simulation problems. The operator point of view is useful in applications since high-level algorithms can be devised for the problems at hand with operators serving as the main building blocks. While this approach has received some attention in the past, it has not reached its full potential, as the following thesis tries to hint.

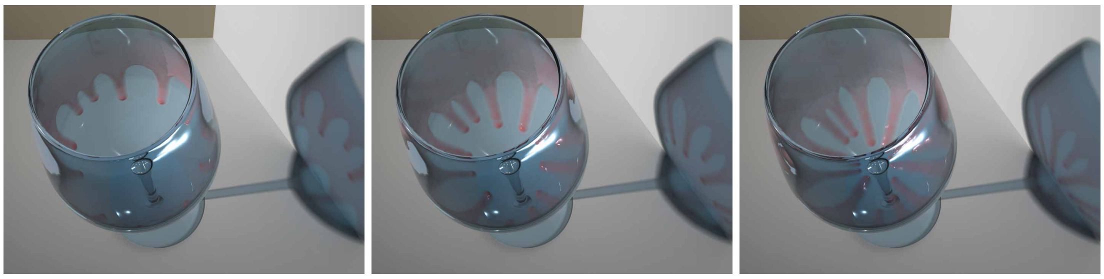
The contribution of this document is twofold. First, it describes the analysis and discretization of *derivations* and related operators such as covariant derivative, Lie bracket, pushforward and flow on triangulated surfaces. These operators play a fundamental role in numerous computational science and engineering problems, and thus enriching the readily available differential tools with these novel components offers multiple new avenues to explore. Second, these objects are then used to solve certain differential equations on curved domains such as the advection equation, the Navier–Stokes equations and the thin films equations. Unlike previous work, our numerical methods are *intrinsic* to the surface—that is, independent of a particular geometry flattening. In addition, the suggested machinery *preserves structure*—namely, a central quantity to the problem, as the total mass, is exactly preserved. These two properties typically provide a good balance between computation times and quality of results.

From a broader standpoint, recent years have brought an expected increase in computation power along with extraordinary advances in the theory and methodology of geometry acquisition and processing. Consequently, many approaches which were infeasible before, became viable nowadays. In this view, the operator perspective and its application to differential equations, as depicted in this work, provides an interesting alternative, among the other approaches, for working with complex problems on non-flat geometries. In the following chapters, we study in which cases operators are applicable, while providing a fair comparison to state-of-the-art methods.

Chapter 1

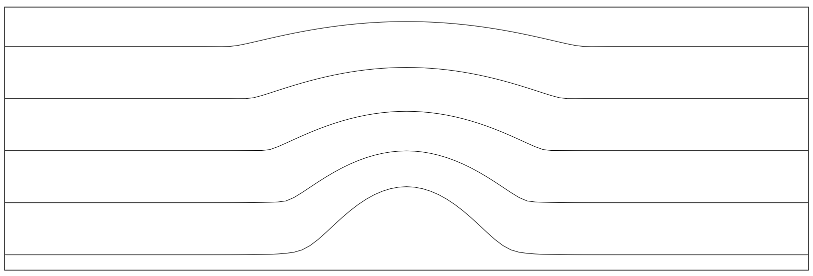
Introduction

This thesis presents practical methods for dealing with a few challenging problems which are considered on non-trivial curved domains. For instance, we would like to numerically estimate how a thin sheet of wine runs down a wine glass. How would one begin to solve this difficult problem?



The following aspects should be taken into consideration. First, the motion and its causes, i.e., the dynamics of the problem is analyzed into a concise mathematical representation given by a *differential equation*. In this work, we assume that the governing equations were already derived by previous work, e.g., our motivating example of wine could be modelled approximately with thin film equations [ODB97, CM09]. Second, the geometrical curved domain is represented in a way which allows to perform simple computations while maintaining the ability to encode complex geometries. To this end, the domain is approximated with a *triangle mesh*, namely, a set of vertices, edges and faces that are given by planar triangles glued over their edges. The third aspect and the main focus of this work, is the solution of the former equations on the discrete surface in terms of a numerical method which can be coded in some programming language.

Differential equations describe the change in time or space (or both) of an essential quantity to the problem. Thin films, for example, are characterized through the film's height, h, computed per point of the domain. The temporal and spatial evolution of h is considered a solution of the associated equations. Numerically solving differential equations is a challenging task, involving the construction of temporal integration rules along with a proper discretization of spatial differential operators to produce a reliable estimate of the underlying dynamics. With this classification in mind, the first half of the following work deals with the spatial aspects, whereas the second half mostly handles the temporal mechanisms. Below, we show the result of a numerical solver corresponding to the evolution of a thin film subject to surface tension forces in space (horizontal axis) and time (vertical axis).



We proceed by taking a top-down approach. We will first describe the partial differential equations (PDEs) we are set-out to solve, and then, we will focus on the spatial components that are required for composing effective numerical solvers. The temporal aspects will not be discussed in this introduction, but within the relevant chapters. As the above example hints, we will be interested in transport-dominated effects, and thus, we begin with presenting a PDE which mathematically captures how a certain quantity, u, moves due to a given velocity field, v. In fact, it is one of the most ubiquitous PDE, used to describe numerous real-life phenomena and it is known as *the transport equation*:

$$\partial_t u + \langle v, \operatorname{grad} u \rangle = 0 \ , \tag{1.1}$$

where ∂_t is a derivative in time, grad is the gradient operator and $\langle \cdot, \cdot \rangle$ is the standard scalar product. Indeed, Eq. (1.1) measures transport as it couples the change in u over time with its (minus) change in space, since the term $\langle v, \operatorname{grad} u \rangle$ is nothing else but a directional derivative of u with respect to v.

In Chapter 5, we consider the transport equation along with boundary conditions at the initial and final times, i.e., $u(0) = u_0$ and $u(1) = u_1$, and the main goal is to compute $u(t)$ and $v(t)$ for $t \in (0, 1)$. Naturally, it is an ill-posed problem and some regularization is needed to increase the chances of finding a solution. This particular setup is adapted to the task of improving an initial mapping φ between two surfaces $\mathcal{M}_1$ and $\mathcal{M}_2$. As we will show in Chapter 5, if the function space of $\mathcal{M}_1$ is put in correspondence with the function space of $\mathcal{M}_2$, the given mapping φ could be greatly improved. Fortunately, the above boundary value problem is tailored to this task, taking $u_0 = f \circ \varphi$ and $u_1 = g$, where $f \in L_2(\mathcal{M}_1)$ and $g \in L_2(\mathcal{M}_2)$, and using the resulting $v(t)$ to generate the new φ. We refer the reader to the relevant chapter for further details.

The next PDE we consider is related to flow of fluids, and it guarantees the preservation of momentum, being part of the famous *Navier–Stokes equations* [CMM90]:

$$\partial_t v + \nabla_v v - \mu \nabla^2 v + \operatorname{grad} p = 0 \ , \tag{1.2}$$

where v is the velocity, p is the pressure, ∇ is the covariant derivative and ∇^2 is a vector Laplacian with μ being the viscosity coefficient. Unfortunately, solving Eq. (1.2) on curved domains is challenging as we briefly mention below and discuss in length in Chapter 3. Nevertheless, the above vector PDE could be greatly simplified to a *scalar* PDE by defining the vorticity of the flow, $\omega = \operatorname{curl} v$, which always points in the direction of the surface normal, in the two-dimensional case. Thus, instead of solving Eq. (1.2) directly, we model almost inviscid flows in Chapter 6 by studying the *vorticity equation* [MB01], resulting from taking the curl of the above equation:

$$\partial_t \omega + \langle v, \operatorname{grad} \omega \rangle - \mu \Delta \omega = 0 \ . \tag{1.3}$$

We are back again with a transport-type equation, Eq. (1.1), but now it is non-linear due to the constraint which couples between ω and v and includes viscous effects.

The next and final set of PDEs we investigate in this work are the *thin films equations*, related to the problem we discussed at the beginning of this chapter. On curved domains, having u represent the mass density of the film, these equations read:

$$\partial_t u + \operatorname{div}\left(-M(u) \cdot \operatorname{grad} p(u)\right) = 0 \ , \tag{1.4a}$$

$$M(u) = \frac{1}{3} u^3 \operatorname{id} + \frac{\epsilon}{6} u^4 (H \operatorname{id} - S) \ , \tag{1.4b}$$

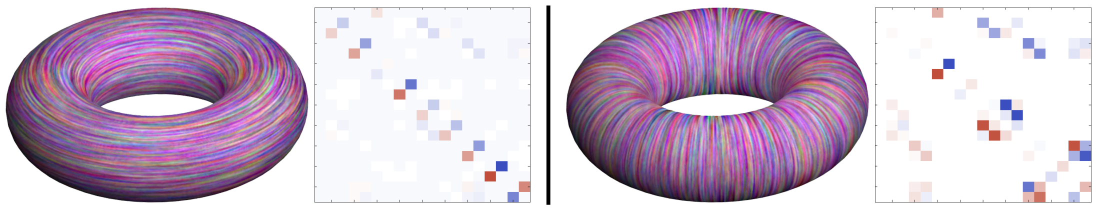
$$p = -H - \epsilon T u - \epsilon \Delta u \ , \tag{1.4c}$$

where H, T and S are curvature-related quantities, $\epsilon \ll 1$ is the fluid's height to length ratio, div is the divergence operator which measures the amount of flux associated with v, and Δ is the Laplace–Beltrami operator. In this case, the advection-type equation is non-linear and fourth order, making its discretization particularly challenging, as we discuss in Chapter 7. However, since thin films exhibit a gradient flow structure [RV13], we are able to discretize these equations efficiently, simulating various intricate motions of thin sheets of liquid on general domains, as shown, e.g., in the wine example above.

The equations presented in this introduction, except for Eq. (1.2), share a similar structure where the change in time is modelled by the differential change in space. In addition, recall that

$$\operatorname{div}(uv) = \langle v, \operatorname{grad} u \rangle + u \cdot \operatorname{div}(v) \ ,$$

and thus, we distinguish between differential operators as grad, div curl and Δ, whose discretization can be found in standard textbooks, such as [BKP$^+$10] to the *directional derivative*, $D(v) = \langle v, \operatorname{grad} \cdot \rangle$, which is the main focus of Chapter 2. Considering $D(v)$ as an operator which maps scalar functions to their derivatives has many advantages. For example, given a finite basis, $D(v)$ is simply a square matrix whose properties can be investigated or prescribed, as was done in Chapter 2. We show below the two harmonic vector fields—divergence- and curl-free vector fields—on the torus surface, along with the corresponding $D(v)$ operators, as computed in a reduced basis.



More importantly to this thesis, when solving transport-dominated differential equations, the use of $D(v)$ allows us to avoid the numerically sensitive process of *integrating the flow lines* which was done, e.g., in [SY04, ETK$^+$07]. Namely, instead of first computing the flow lines of v and then transporting u along them, we could directly facilitate $D(v)$ in our implicit or explicit time integrator. Encouraged by the usefulness of $D(v)$, we extended this idea to the case of *vector derivatives*, $\nabla_v v$, in Chapter 3 for the purpose of solving Eq. (1.2). Unfortunately, while $\nabla_v v$ is intrinsic to the surface, our discretization involves extrinsic differentiation with projection, and thus the obtained fluid simulator is inferior to the one we designed in Chapter 6. Nevertheless, the covariant derivative is useful in many other scenarios—vector field design being one of the main examples we focus on in that chapter.

Finally, another significant benefit of the operator point of view, is that it naturally leads to a *weak* formulation of the underlying problem. To contrast with a strong formulation where quantities are evaluated pointwise, weak formulations typically involve an integrated version of the quantity, in a small domain. Having a weak version of the problem helps when approximate or even noisy data is given. For instance, in Chapter 4, we study the problem of jointly remeshing a pair of input triangulated surfaces, $\mathcal{M}_1$ and $\mathcal{M}_2$, to *consistent* quadrangulations. The meshes can have different number of vertices and thus some mapping $\varphi : \mathcal{M}_1 \to \mathcal{M}_2$ is assumed to be given. To compute consistent quad meshes, we optimize for two rotationally symmetric vector fields x_1 and x_2 on $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively, and to enforce consistency we need the *map differential*, $\mathrm{d}\varphi$. That is, our basic consistency condition is that $\mathrm{d}\varphi(x_1) = x_2$ for each point in $\mathcal{M}_2$. In practice, φ can be noisy, leading to a noisy $\mathrm{d}\varphi$ and a non-robust algorithm. Instead, we design an efficient and robust algorithm in Chapter 4, by facilitating the *functional* version of the consistency condition:

$$C[\varphi] \cdot D(x_1) = D(x_2) \cdot C[\varphi] \ , \tag{1.5}$$

where $C[\varphi]$ is the operator which corresponds to the mapping φ. We refer the reader to the thorough discussion and further information in the relevant chapter.

To conclude, the operator approach is highly useful in several scenarios, as this thesis shows, allowing for a principled way to design algorithms which will be effective and robust, while remaining efficient to compute. It is the hope of the author, that the operator perspective will keep growing in the future, adding more formulations and applications to geometry processing literature and in other scientific domains.

Chapter 2

An Operator Approach to Tangent Vector Field Processing

In this chapter, we introduce a novel coordinate-free method for manipulating and analyzing vector fields on discrete surfaces. Unlike the commonly used representations of a vector field as an assignment of vectors to the faces of the mesh, or as real values on edges, we argue that vector fields can also be naturally viewed as *operators* whose domain and range are functions defined on the mesh. Although this point of view is common in differential geometry it has so far not been adopted in geometry processing applications. We recall the theoretical properties of vector fields represented as operators, and show that *composition* of vector fields with other functional operators is natural in this setup. This leads to the characterization of vector field properties through commutativity with other operators such as the Laplace-Beltrami and symmetry operators, as well as to a straight-forward definition of differential properties such as the Lie derivative. Finally, we demonstrate a range of applications, such as Killing vector field design, symmetric vector field estimation and joint design on multiple surfaces.

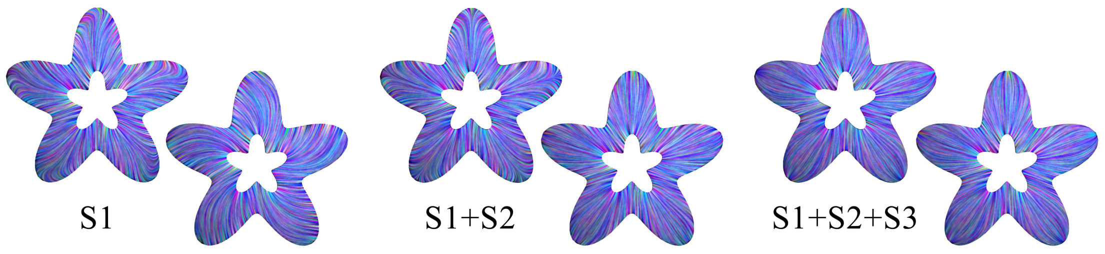


Figure 2.1: Using our framework various vector field design goals can be easily posed as linear constraints. Here, given three symmetry maps: rotational (S1), bilateral (S2) and front/back (S3), we can generate a symmetric vector field using only S1 (left), S1 + S2 (center) and S1 + S2 + S3 (right). The top row shows the front of the 3D model, and the bottom row its back.

2.1 Introduction

Manipulating and designing tangent vector fields on discrete domains is a fundamental operation in areas as diverse as dynamical systems, finite elements and geometry processing. The first question that needs to be addressed before designing a vector field processing toolbox, is how will the vector fields be represented in the discrete setting? The goal of this paper is to propose a representation, which is inspired by the point of view of vector fields in differential geometry as *operators* or *derivations*.

In the continuous setting, there are a few common ways of defining a tangent vector field on a surface. The first, is to consider a smooth assignment of a vector in the tangent space at each point on the surface. This is, perhaps, the most intuitive way to extend the definition of vector fields from the Euclidean space to manifolds. However, it comes with a price, since on a curved surface one must keep track of the relation between the tangent spaces at different points. A natural discretization corresponding to this point of view (used e.g. in [PP03]) is to assign a single Euclidean vector to each simplex of a polygonal mesh (either a vertex or a face), and to extend them through interpolation. While this representation is clearly useful in many applications, the non-trivial relationships between the tangent spaces complicate tasks such as vector field design and manipulation.

An alternative approach in the continuous case, is to work with *differential forms* (see e.g. [Mor01]), which are linear operators taking tangent vector fields to scalar functions. In the discrete setting this point of view leads to the famous Discrete Exterior Calculus [Hir03, FSDH07], where discrete 1-forms are represented as real-valued functions defined over the edges of the mesh. While this approach is coordinate-free (as no basis for the tangent space needs to be defined), and has many advantages over the previous method, there are still some operations which are natural in the continuous setting, and not easily representable in DEC. For example, the flow of a tangent vector field is a one parameter set of self-maps and various vector field properties can be defined by composition with its flow, an operation which is somewhat challenging to perform using DEC.

Finally, another point of view of tangent vector fields in the continuous case is to consider their *action* on scalar functions. Namely, for a given vector field, its *covariant derivative* is an operator that associates to any smooth function f on the manifold another function which equals the derivative of f in the direction given by the vector field. It is well known that a vector field can be recovered from its covariant derivative operator, and thus any vector field can be uniquely represented as a functional operator. We will refer to these operators as functional vector fields (FVFs). Note, that while this point of view is classical in differential geometry, it has so far received limited attention in geometry processing.

In this paper, we argue that the operator point of view yields a useful coordinate-free representation of vector fields on discrete surfaces that is complementary to existing representations and that can facilitate a number of novel applications. For example, we show that constructing a Killing vector field [Pet06] on a surface can be done by simply finding a functional vector field that commutes with the Laplace-Beltrami operator. Furthermore, we show that it is possible to transport vector fields across surfaces, find symmetric vector fields and even compute the flow of a vector field easily by employing the natural relationship between FVFs and functional maps [OBCS$^+$12]. Finally, the Lie derivative of two vector fields is given by the commutator of the two respective operators, and as a result the covariant derivative of a tangent vector field with respect to another can be computed through the Koszul formula [Pet06].

To employ this representation in practice, we show that for a suitable choice of basis, a functional vector field can be represented as a (possibly infinite) matrix. As not all such matrices represent vector fields, we show how to parameterize the space of vector fields using a *basis for the operators*. With these tools in hand, we propose a Finite Element-based discretization for functional vector fields, and demonstrate its consistency and empirical convergence. Finally, we apply our framework to various vector field processing tasks showing comparable results to existing methods, as well as novel applications which were challenging so far.

2.1.1 Related Work

The body of literature devoted to vector fields in graphics, visualization and geometry processing is vast and a full overview is beyond our scope. Thus, we will focus on the *representation* and discretization of vector fields, as these aspects of vector field processing are most related to our work.

One approach to discretization (e.g. [PP03, TLHD03]) is to use piecewise constant vector fields, where vectors are defined per face and represented in the standard basis in $\mathbb{R}^3$. This approach is simple and allows to define discrete versions of standard operators such as *div* and *curl*, which are consistent with their continuous counterparts (e.g. one can define a discrete Hodge decomposition [PP03]). However, since the representation is based on coordinate frames, it makes vector field design challenging as the relationship between tangent spaces is non-trivial, leading to difficult optimization problems.

An alternative discretization of vector fields was suggested as part of the formalism of Discrete Exterior Calculus (DEC) [Hir03], where vector fields are identified with discrete 1-forms, represented as a single scalar per edge. This approach is inherently coordinate-free, allowing to formulate vector field design as a linear system [FSDH07]. Unfortunately, computing the Lie derivative of vector fields remains a complex task using DEC (as shown in [MMP$^+$11]).

Vector field design and processing applications are also tightly connected to the analysis of rotationally symmetric (RoSy) fields, see e.g. [PZ07, RVAL09, CDS10]. In the latter work, for example, a vector field (or a symmetric direction field) is represented using an angle per edge (an angle valued dual 1-form), which represents how the vector changes between neighboring triangles. While these approaches are also coordinate-free and lead to linear optimization problems for direction field design, it is not clear how vector-field valued operators can be represented in such a setup.

In this paper, we argue that in addition to the existing discretization methods, it is often useful to represent vector fields through their covariant derivatives as linear functional operators. This representation is coordinate-free and, in addition, elucidates the intimate connection between vector fields and self maps of the surface, allowing us to extend the basic vector field processing toolbox to computational tasks which are challenging using existing discretization tools.

Note that the operator representation of vector fields has been used in the context of fluid simulation by Pavlov et al. [PMT$^+$11]. However, in that work, the authors were primarily interested in representing divergence-free vector fields and did not use this representation for tangent vector field analysis and design. In this paper, we consider general vector fields, demonstrate how this representation can be used for vector field processing, and show a deep connection with the functional map framework [OBCS$^+$12].

2.1.2 Contributions

Our main observation is that tangent vector fields can be represented in a coordinate-free way as functional operators. While this view is classical in differential geometry [Mor01], it has so far received limited attention in geometry processing. Using this perspective we:

- Show how functional vector fields can be naturally composed with other operators, and thus relate vector fields to other common operators such as maps between shapes and the Laplace-Beltrami operator (Section 2.2).
- Provide a novel coordinate-free discretization of tangent vector fields (Section 2.4).
- Describe various applications for vector field processing including Killing vector field design, design of symmetric vector fields and joint vector field design on multiple shapes, which are all easily solvable as linear systems in our framework (Section 2.5).

2.2 Vector Fields as Operators

In this section we define the coordinate-free view of vector fields as abstract *derivations* of functions in the continuous setting. This point of view is well-known in differential geometry (see e.g. [Mor01] for an excellent reference). Thus, we only recall the standard definition and its main properties.

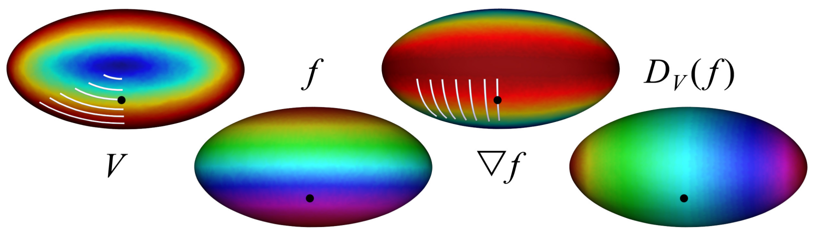


Figure 2.2: Given a vector field V (left) and a function f (center left), the inner product of ∇f (center right) with V is the covariant derivative $D_V(f)$ (right). For the marked point, for example, V is orthogonal to ∇f, yielding 0 for $D_V(f)$. Vector fields are visualized by color coding their norm, and showing a few flow lines for a fixed time t.

2.2.1 The Covariant Derivative of Functions

We first assume that we are given a compact smooth Riemannian manifold M and a tangent vector field V, which can be thought of as a smooth assignment of a tangent vector $V(p)$ to each point $p \in M$. The vector field defines a one-parameter family of maps, $\Phi_V^t : M \to M$ for $t \in \mathbb{R}$, called the *flow* of V. The flow is formally defined as the unique solution to the differential equation:

$$\frac{d}{dt}\Phi_V^t(p) = V(\Phi_V^t(p)), \qquad \Phi_V^0(p) = p. \tag{2.1}$$

Then, for a given function $f \in C^\infty(M)$, the *covariant derivative* $D_V(f)$ of f with respect to V is a function g, which intuitively measures the change in f with respect to the flow under V. Formally,

$$g(p) = D_V(f)(p) = \lim_{t\to 0}\frac{f(\Phi_V^t(p)) - f(p)}{t}.$$

A classical result in Riemannian geometry ([Mor01], p. 148) is that the covariant derivative can also be computed as :

$$D_V(f)(p) = g(p) = \langle(\nabla f)(p), V(p)\rangle_p\,, \tag{2.2}$$

where $\langle,\rangle_p$ denotes the inner product in the tangent space of p, and ∇f is the gradient of f (see Figure 2.2).

2.2.2 The Covariant Derivative as a Functional Operator

We stress that D_V is best viewed as an *operator*, which maps smooth functions on M to smooth functions on M. Moreover, one can show that D_V encodes V so that if V_1 and V_2 are vector fields such that $D_{V_1}f = D_{V_2}f$ for any $f \in C^\infty(M)$, then $V_1 = V_2$ (see [Mor01, pg. 38]). Said differently, there is no loss of information when moving from V to D_V.

The covariant derivative (viewed as a functional operator, i.e. an FVF) satisfies the following two key properties:

Linearity:

$$D(\alpha f + \beta g) = \alpha D(f) + \beta D(g), \tag{2.3}$$

and Leibnitz (product) rule:

$$D(fg) = fD(g) + gD(f). \tag{2.4}$$

Conversely, a functional operator D corresponds to a vector field, if and only if it satisfies (2.3) and (2.4) (see [Spi99, pg. 79]).

Why are these the necessary properties for operators that represent vector fields? Intuitively, this is because vector fields can be thought of as first order directional derivatives, which have two essential properties. First, that *constant functions are mapped to the zero function.* And second, that $D_V(f)$ depends on f *only to first order.*

One of the advantages of considering vector fields as abstract derivations is that this point of view can be generalized to settings where differential quantities are not well defined. For example, on a discrete surface there is no well defined normal direction at vertices and edges. By working with purely algebraic constructs, such as linear operators, we can define differentiation without requiring the concept of a limit, which is useful when the underlying surface is not continuous and such a limit does not exist. Moreover, as we will see, the operator point of view makes it easier to manipulate vector fields and relate them to other functional operators.

2.2.3 Properties

While the operator point of view is equivalent to the standard notion of a vector field as a smooth assignment of tangent vectors, certain operations are more natural in this representation. Below we list such operations, which we will use in our applications in Section 2.5. The proofs of all lemmas are provided in the supplemental material.

Operator composition. By using the operator point of view of vector fields, it becomes easy to define their *composition* both with other vector fields and other more general functional operators. Unfortunately, given two vector fields D_{V_1} and D_{V_2}, the operator $D_{V_1} \circ D_{V_2}$ does not necessarily correspond to a vector field. However, one can modify this operator to obtain a fundamental notion:

Lie derivative of a vector field. Given two vector fields V_1 and V_2, the Lie derivative (or Lie bracket) of V_2 with respect to V_1 is a vector field V_3 defined as:

$$\mathcal{L}_{V_1}(V_2) = [V_1, V_2] = D_{V_3} = D_{V_1} \circ D_{V_2} - D_{V_2} \circ D_{V_1}. \tag{2.5}$$

It is easy to see that D_{V_3} thus defined is both linear and satisfies the product rule. Hence, D_{V_3} corresponds to a unique vector field V_3. Intuitively, the Lie derivative captures the

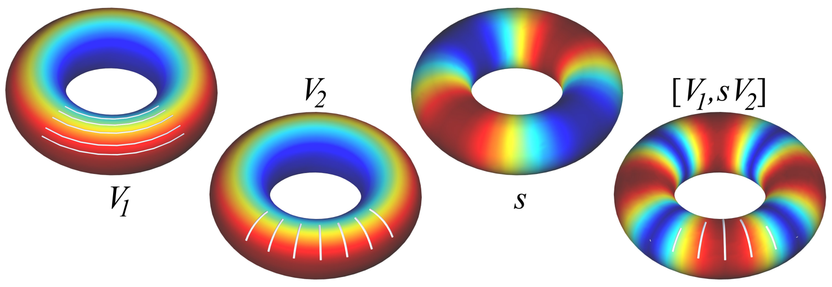


Figure 2.3: Two orthogonal vector fields on the torus V_1, V_2, whose Lie derivative is 0. Modifying the norm of V_2 using a function s yields a lie derivative which is parallel to V_2.

commutativity of the flows of V_1 and V_2. In particular, the Lie derivative is zero if and only if the flows defined by V_1 and V_2 commute (see [Spi99, pg. 157]):

$$\Phi_{V_2}^{-s} \circ \Phi_{V_1}^{-t} \circ \Phi_{V_2}^{s} \circ \Phi_{V_1}^{t} = Id \quad \forall s, t \in \mathbb{R} \tag{2.6}$$

Figure 2.3 illustrates the computation of the Lie derivative on a torus. We consider two vector fields V_1 and V_2 whose flows commute. The average norm of $[V_1, V_2]$ computed using the discrete operators we describe in Section 2.4 is on the order of 1e-8, close to 0 as expected. In general, if $[V_1, V_2] = 0$, it can be shown that for any scalar function $s : M \to \mathbb{R}$, $[V_1, sV_2]$ must be parallel to V_2. In Figure 2.3, we show a scaling function s, and the computed vector field $V_3 = [V_1, sV_2]$, which is parallel to V_2, as expected.

Composition with other operators. Of course, it is possible to consider the composition of the FVF operator D_V with other functional operators. Interestingly, the commutativity of D_V with a differential operator D is closely related to the commutativity of its flow with D.

Lemma 2.2.1. Let T_F^t, $t \in \mathbb{R}$ be the functional operator representations of the flow diffeomorphisms $\Phi_V^t : M \to M$ of V, defined by $T_F^t(f) = f \circ \Phi_V^t$ for any function $f \in C^\infty(M)$. If D is a linear partial differential operator then $D_V \circ D = D \circ D_V$ if and only if for any $t \in \mathbb{R}$, $T_F^t \circ D = D \circ T_F^t$.

For example, on a Riemannian manifold, we can consider composition with the Laplace-Beltrami operator L. The commutativity of D_V with L is then closely related to the metric distortion imposed by the flow of V. In particular, recall that a vector field is called a Killing vector field (KVF) if Φ_V^t is an isometry for all t (see [Pet06], Chapter 7). Then:

Lemma 2.2.2. A vector field V is a Killing vector field if and only if $D_V \circ L = L \circ D_V$.

From this lemma, it is easy to see that KVFs form a group under the Lie derivative. Indeed, the following lemma, which follows directly from the definition of the Lie derivative, is useful in general:

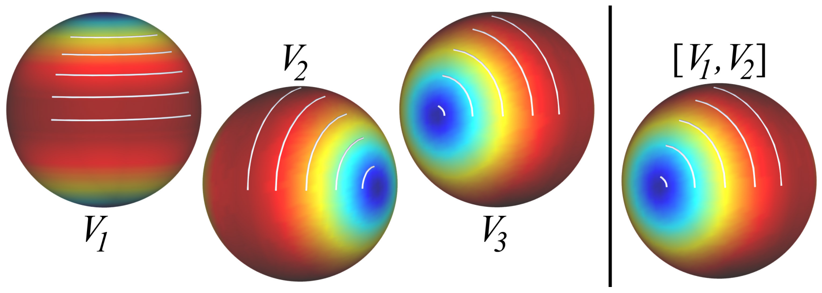


Figure 2.4: Using commutativity with L, we compute the KVFs on the sphere (V_1, V_2, V_3). Alternatively, we compute $V_4 = [V_1, V_2]$, note the similarity of V_3 and V_4.

Lemma 2.2.3. Given two vector fields D_{V_1} and D_{V_2} that both commute with some operator D, the Lie derivative $\mathcal{L}_{V_1}(V_2)$ will also commute with D.

Figure 2.4 demonstrates these properties on the sphere. On the left, we show V_1, V_2, V_3, the three KVFs of the sphere, computed using Lemma 2.2.2 by constructing a linear system (as explained in Section 2.5). On the right, we show $V_4 = [V_1, V_2]$, which was computed as the Lie bracket of the first two KVFs. Note the similarity between V_3 and V_4. We will use these results for designing approximate KVFs in Section 2.5.

Composition with mappings. In many settings we are also interested in the relation between vector fields on multiple surfaces related by mappings. In particular, given a vector field V_1 on surface M and a *diffeomorphism* $T : M \to N$, one can define the vector field V_2 on surface N via the push forward: $V_2(q) = dT(V_1(T^{-1}(q)))$. Note that in the discrete case, it is often difficult to compute the differential dT of a map T between shapes with different discretizations. At the same time, one can equivalently define the vector field V_2 using the operator approach, without relying on dT, by using the functional representation of the map T.

As mentioned in [OBCS+12], the functional representation T_F of a map T is a linear operator on the space of functions, taking functions on N to functions on M defined by $T_F(g) = g \circ T$ for any function $g \in C^\infty(N)$. This means that the functional vector field D_{V_2}, and thus V_2 itself can be obtained by simple composition of three linear functional operators without the need to estimate the differential dT, using:

Lemma 2.2.4. $D_{V_2} = (T_F)^{-1} \circ D_{V_1} \circ T_F$.

Figure 2.5 illustrates vector field transportation using this approach (vector fields are visualized using [PZ11]). Given V_1 on M, and a map $T : M \to N$, we generate V_2 on N using Lemma 2.2.4. V_3 is computed using the differential of the map, given by the affine map between corresponding triangles. Note that V_3 tends to be noisy, due to the locality of the transport procedure, as opposed to the smoother V_2. Furthermore, this approach is applicable to shapes with different connectivity, where computing dT is challenging. In Section 2.5 we use a similar approach to *design* vector fields which are consistent with the map $T : M \to N$.

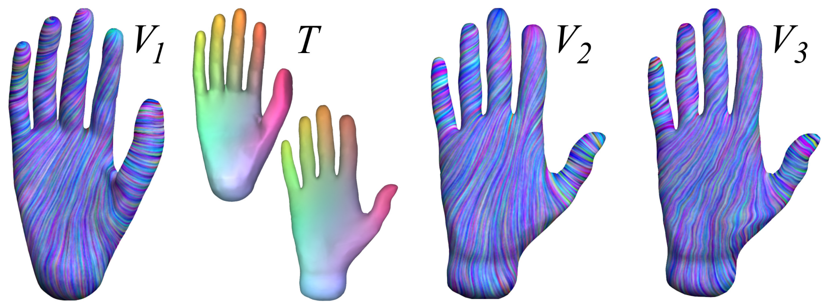


Figure 2.5: Given a vector field V_1 on M and a map $T : M \to N$, we generate a vector field V_2 on N using Lemma 2.2.4. Compare with directly transporting V_1 using the differential of the map, yielding V_3. Note the ringing artifacts in V_3.

Vector field flow. The FVF D_V representing a vector field is also closely related to the functional representation of the flow Φ_V^t. In particular:

Lemma 2.2.5. Let $T^t = \Phi_V^t$ be the self-map associated with the flow of V at time t. Then if T_F^t is the functional representation of T^t, for any real analytic function f (see [DFN92], p. 210):

$$T_F^t f = \exp{(t\ D_V)} f = \sum_{k=0}^{\infty} \frac{(tD_V)^k f}{k!}.$$

This lemma is particularly useful since it allows to avoid the potentially costly solution of the system of equations (2.1) and directly estimate the functional representation of the map Φ_V^t through operator exponentiation. Note that D_V is a moderately sized matrix when represented in a basis, and therefore its exponent can be computed efficiently. Figure 2.6 shows an example of function flow using this method.

Covariant derivative of a tangent vector field. Some PDEs can be described using the *covariant derivative* [Mor01] of a vector field V_1 with respect to another vector field V_2, denoted $\nabla_{V_2} V_1$. For planar vector fields, for example, $\nabla_{V_2} V_1 = J(V_1)V_2$, where $J(V_1)$ is the Jacobian matrix of V_1.

On a surface, however, this representation requires a basis for the tangent space at every point, and a suitable *connection* that allows to transport a vector $V(p)$ to a neighboring point q, which makes $\nabla_{V_2} V_1$ elusive to compute in a coordinate-free way. Fortunately, there is an intimate connection between the Lie and covariant derivatives of vector fields, through the Koszul formula, ([Pet06], p. 25):

$$\begin{aligned} 2g(\nabla_{V_1} V_2, Z) &= D_{V_1}(g(V_2, Z)) - g(V_1, [V_2, Z]) \\ &\quad + D_{V_2}(g(V_1, Z)) - g(V_2, [V_1, Z]) \\ &\quad - D_Z(g(V_1, V_2)) + g(Z, [V_1, V_2]). \end{aligned} \tag{2.7}$$

Here, Z is an arbitrary vector field, $g(\cdot, \cdot) = \langle \cdot, \cdot \rangle_p$ is the inner product in the tangent space of p, and $[\cdot, \cdot]$ is the Lie bracket (Eq. (2.5)). Hence, given an operator representation

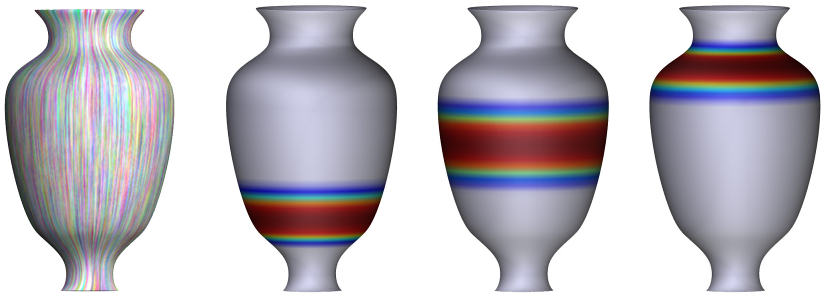


Figure 2.6: Applying the flow of a vector field (left) to a function (center left) using Lemma 2.2.5. (center right, right) The function after the flow, for two sample t values.

of D_{V_1} and D_{V_2}, we can use Equation (2.7) to compute $\nabla_{V_1} V_2$. We leave further investigation of this direction, and possible applications for future work.

2.3 Representation in a Basis

The properties mentioned above suggest that representing and analyzing tangent vector fields through their functional representation can enable a number of applications which are challenging using standard methods. Our goal, therefore, will be to represent this operator such that it can be easily analyzed and manipulated in practice.

2.3.1 Basis for the Function Space

As mentioned in Section 2.2.2, an FVF is a linear operator acting on smooth functions defined on the manifold. In practice, we will assume that the functional space of interest can be endowed with a (possibly infinite) basis, so that any function can be represented as a linear combination of some basis functions $\{\phi_i\}$. Then, for any given function $f = \sum_i a_i \phi_i$, we have that $g = D_V(f) = D_V(\sum_i a_i \phi_i) = \sum_i a_i D_V(\phi_i)$. Since $D_V(\phi_i)$ is also a function, it can be represented in the basis as $D_V(\phi_i) = \sum_j D_{ij} \phi_j$. Therefore, $g = \sum_j (\sum_i D_{ij} a_i) \phi_j = \sum_j b_j \phi_j$. In other words, if one thinks of the coefficients a_i, b_i as vectors $\mathbf{a}, \mathbf{b}$ and $D = (D_{ij})$ as a matrix, then the transformation between the basis representations of f and $g = D_V(f)$ is given by: $\mathbf{b} = D\mathbf{a}$.

When the basis functions ϕ_i are orthonormal with respect to the standard functional inner product on M, i.e. $\int_M \phi_i \phi_j d\mu = 1$ if $i = j$ and 0 otherwise, then the $(i,j)^{\text{th}}$ element D_{ij} of the FVF corresponding to V is given by:

$$D_{ij} = \int_M \phi_i D_V(\phi_j) d\mu(p) = \int_M \phi_i(p) \left\langle V(p), \nabla \phi_j \right\rangle_p d\mu(p), \tag{2.8}$$

where $\langle , \rangle_p$ denotes the inner product in the tangent space of the point p, and $d\mu(p)$ represents the volume measure at p.

The Laplace-Beltrami basis. A useful basis for the space of smooth functions on a compact manifold, which we will often use in practice, is the basis given by the eigenfunctions of the Laplace-Beltrami operator (note that on a compact manifold the space $L^2(M)$ is strictly larger than the space of smooth functions). Since each eigenfunction of the Laplace-Beltrami operator is smooth, Equation (2.8) is well defined. One advantage of this basis is that the basis functions are ordered and can be attributed a notion of *scale*, given by the corresponding eigenvalue. This has been exploited in a number of scenarios including the work on functional maps [OBCS$^+$12] where a mapping between two shapes is compactly encoded using a sub-matrix of a possibly infinite functional map matrix. This choice of basis yields a compact representation of the FVF operator as an $N_f \times N_f$ matrix, where N_f is the number of basis functions we use.

2.3.2 Parameterization with Basis Operators

As mentioned in Section 2.2.2, the space of linear functional operators is strictly larger than the space of vector fields. Therefore, in order to work with this representation in practice, it is useful to have a *parametrization* of the space of FVFs, which is easy to manipulate.

One possible such parameterization, is to consider a basis for the space of tangent vector fields ψ_i, and to represent an operator D_V as a linear combination of the functional vector field operators D_{ψ_i}. In our work, we consider the eigenfunctions of the 1-form Laplace-de Rham operator to generate a basis for the 1-forms on a surface, and use these as a basis for the tangent vectors, by duality [Mor01].

Given such basis operators D_{ψ_i}, the FVF operator D_V that represents a vector field $V = \sum_i a_i \psi_i$ is given by: $D_V = \sum_i a_i D_{\psi_i}$. Note, that this basis is also ordered, so that smoother vector fields can be represented using fewer basis operators. In practice, we truncate the basis, and limit the number of basis operators to a fixed value N_D.

With this parameterization, it is straightforward to use the properties we mentioned in Section 2.2 to design a vector field that has various desirable characteristics, simply by solving a linear system for the coefficients a_i. Figure 2.7 shows a vector field designed

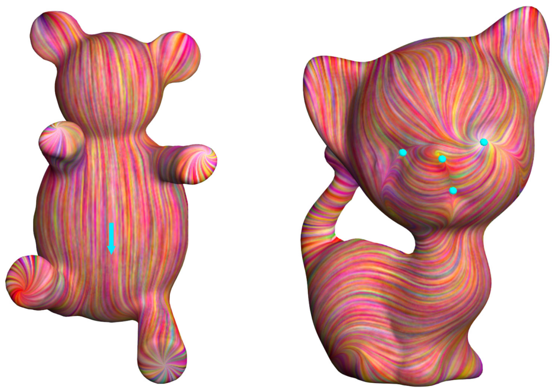


Figure 2.7: Prescribing directional constraints (left) or singularities (right).

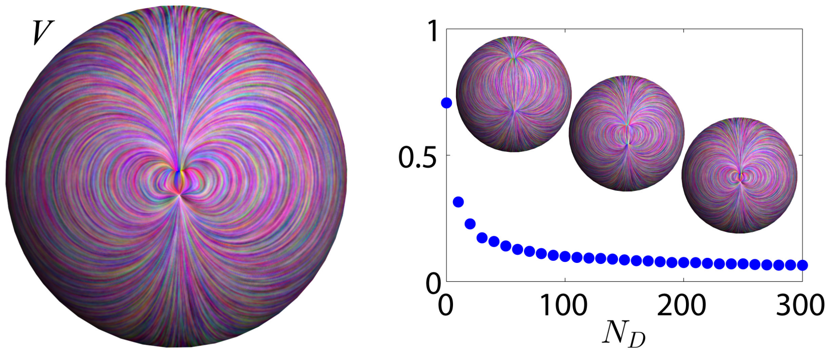


Figure 2.8: Given a vector field (left), we reconstruct it with growing accuracy by increasing the number of basis operators N_D (right). Note that the index 2 singularity is accurately reconstructed given enough basis operators.

by posing a small number of directional constraints (one direction for the teddy (left) and 4 zero valued vectors for the kitten (right)), and solving for the coefficients as explained in Section 2.5.

Figure 2.8 demonstrates the effect of using a varying number of basis operators. Given a direction field (left), we project it on a growing number of basis operators and show the reconstruction error as a function of N_D (right). We additionally show the reconstructed vector field, for a few choices of N_D. Note, that although the direction field is smooth, due to the jump from unit length norm to zero norm at the singular point, it is difficult to reconstruct this vector field exactly. However, using a growing number of basis operators we can approximate better this discontinuity in scale.

2.4 Discretization

So far we have described the properties of tangent vector fields as functional operators in the continuous case. In this section we will focus on the discretization of these concepts to surfaces which are represented as triangle meshes. We propose a finite-element based discretization, and discuss its consistency and experimental convergence properties.

2.4.1 Representation

We will first address the following problem: given a triangle mesh $M = (X, F, N)$, where X are the vertices, F the faces and N the normals to the faces, and a piecewise constant tangent vector field $V = \{v_r \in \mathbb{R}^3 | r \in F, v_r \perp N_r\}$, how do we represent the functional vector field D_V?

The answer is in fact straightforward, when we consider the representation of D_V in the functional basis given by the standard hat functions. On a triangle mesh we can represent functions in a piecewise linear basis, namely $f(p) = \sum_{j=1}^{|X|} b_j \gamma_j(p)$, where γ_j are the standard hat functions (valued 1 at vertex i and 0 at all other vertices), and $b_j \in \mathbb{R}$ are the coefficients. Now, given the function $f(p) = \sum_j b_j \gamma_j(p)$, and a piecewise constant vector field V, we wish to compute $g = D_V(f)$. We set $g(p) = \sum_j a_j \gamma_j(p)$, and solve (2.2) in the weak sense, as is standard in Finite Element Analysis (see [AFW06] for a complete discussion of this approach):

$$\int_M \gamma_i g d\mu = \int_M \gamma_i D_V(f) d\mu, \quad \forall i.$$

Plugging in the expressions for f, g and D_V we get $\forall i$:

$$\sum_j a_j \int_M \gamma_i \gamma_j d\mu = \sum_j b_j \int_M \gamma_i \langle \nabla \gamma_j, V \rangle d\mu. \tag{2.9}$$

The integrands in (2.9) vanish everywhere, except on the set of triangles $R_{ij} \subset F$, for which both γ_i and γ_j are non-zero. For $i = j$, these are the triangles neighboring the vertex i. For $i \neq j$, we have that (i, j) must be an edge, and R_{ij} contains only the two triangles which share that edge.

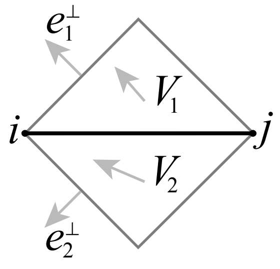
This leads to $\sum_j a_j B_{ij} = \sum_j b_j S_{ij}$, where:

$$B_{ij} = \sum_{t_r \in R_{ij}} \int_{t_r} \gamma_i \gamma_j d\mu, \quad S_{ij} = \sum_{t_r \in R_{ij}} \int_{t_r} \gamma_i \langle \nabla \gamma_j, V \rangle d\mu.$$

Computing the elements B_{ij} yields the standard mass matrix used in the solution of Laplacian systems, whereas S_{ij} is given by (see the inset figure for the notations):

$$S_{ij} = \frac{1}{6} \left(\left\langle V_1, e_1^\perp \right\rangle + \left\langle V_2, e_2^\perp \right\rangle \right)$$

$$S_{ii} = - \sum_{j \in N(i)} S_{ij}.$$



Here, r_1 and r_2 are the two faces that share the edge (i, j), V_1 is the value of V on the face r_1, $e_1^\perp$ is the rotation by $\pi/2$ of the edge opposite to the vertex j in the face r_1 (similarly for V_2 and $e_2^\perp$), and $N(i)$ are the neighboring vertices of vertex i. The derivation is given in the supplemental material.

We further replace B with a diagonal lumped mass matrix W of the Voronoi areas w_i of the vertices [BKP+10], and get:

$$\mathbf{a} = \hat{D}_V \mathbf{b}, \quad \hat{D}_V = W^{-1} S. \tag{2.10}$$

Note, that the size of $\hat{D}_V$ is $|X| \times |X|$, but it is sparse, as only the diagonal and entries of adjacent vertices are non-zero.

It is sometimes useful to decompose $\hat{D}_V$ as a product of two operators: $\hat{D}_V = P_{|X| \times |F|} (\hat{D}_V^F)_{|F| \times |X|}$, where P is independent of V and depends only on the mesh. We take:

$$(P)_{ir} = \frac{1}{3w_i}\mathcal{A}_r, \quad (\hat{D}_V^F)_{ri} = \langle \nabla\gamma_i, V\rangle_r\,, \tag{2.11}$$

where $\mathcal{A}_r$ is the area of the triangle t_r. In fact, the operator $\hat{D}_V^F$ is simply the smooth operator D_V per triangle, where V is fixed. Therefore, it preserves most of the properties of its smooth counterpart. However, to get an operator which commutes with other operators, we need to apply P, averaging values from faces to vertices. This introduces a discretization error into our formulation, due to the discontinuity of the vector field near the vertices.

Alternatively, we can use the first N_f eigenvectors $\hat{\phi}_i$ of the discrete Laplace-Beltrami operator as the basis for the function space, and then D_V will be represented using an $N_f \times N_f$ matrix, which we will denote by $\hat{D}_V^{LB}$. We compute $\hat{D}_V^{LB}$ by using a change of basis:

$$\hat{D}_V^{LB} = B^+\hat{D}_V B, \tag{2.12}$$

where B is a matrix whose columns are $\hat{\phi}_i$ and B^+ is its pseudo-inverse. This representation introduces some additional error, due to the truncation of the basis, and there exists a trade-off between the complexity of the representation (in terms of N_f) and the amount of detail the functions we work with can represent.

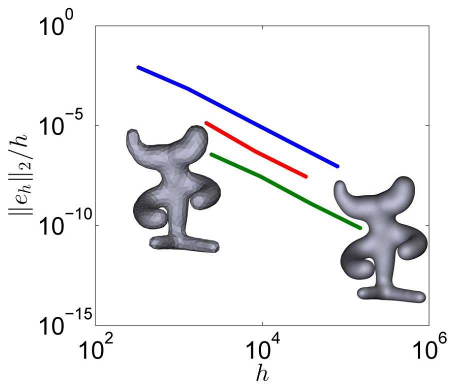
2.4.2 Properties

It is interesting to investigate which properties of D_V are preserved from the smooth case, and which are not but converge under refinement of the mesh.

Constant functions. We have that $D_V(c) = 0$, for any constant function c. It is easy to see this property is preserved in the discrete case, since the rows of $\hat{D}_V$ sum to zero, hence the constant functions are in its kernel.

Product rule. The continuous D_V fulfills two defining properties: linearity (Equation (2.3)) and the Leibnitz product rule (Equation (2.4)). Since $\hat{D}_V$ is a matrix, linearity is clearly satisfied. However, as we work in a limited subspace of functions, the product rule is no longer valid: given two PL functions f, g, their pointwise product fg is no longer PL, and therefore we cannot apply $\hat{D}_V$ to it. However, we can show empirically that when applying increasingly finer discretizations of D_V to increasingly finer discretizations of continuous functions f, g, the product rule error decreases.

Let f_h, g_h, be the two random smooth piecewise linear functions defined on a mesh with h vertices, and take V to be a smooth tangential vector field. Now, for every h, compute the error $e_h = D_V(f_h g_h) - (g_h D_V(f_h) + f_h D_V(g_h))$, where the multiplication is done vertex-wise. The inset figure shows the graph of $\|e_h\|_2/h$ as a function of h, in *loglog* scale, for a few choices of models. Note that the graph is linear, implying



exponential convergence under refinement.

Uniqueness. The correspondence between a vector field V and its FVF operator D_V is one-to-one and onto in the continuous case, implying that given an operator D_V we can uniquely reconstruct the vector field V. This property, unfortunately, may not hold in the discrete case. We do, however have the following weaker result:

Lemma 2.4.1. Let $M = (X, F, N)$ and let V_1, V_2 be two piecewise constant vector fields on M. Then: $\hat{D}^F_{V_1} = \hat{D}^F_{V_2}$ if and only if $V_1 = V_2$.

In practice, given an operator $\hat{D}_V$ we reconstruct the corresponding vector field V by projecting on the operator basis, as described in Section 2.3.2.

Metric invariance. The continuous functional vector field operator D_V commutes with the pushforward under a map. Specifically, given a bijective diffeomorphism $T : M \to N$, a vector field V_1 on M and a function $f : M \to \mathbb{R}$, we have that $D_{V_1}(f)(p) = D_{V_2}(f \circ T^{-1})(T(p))$, where $V_2 = dT(V_1(p))$, and dT is the differential of T. As a consequence, D_V does not depend on the embedding of the shape M.

As we do not have the uniqueness property, the discrete metric invariance property is also limited to the $\hat{D}^F_V$ operator:

Lemma 2.4.2. Let $M_1 = (X_1, F, N_1)$ and $M_2 = (X_2, F, N_2)$ be two triangle meshes with the same connectivity but different metric (i.e. different embedding). Additionally, let V_1 be a piecewise constant vector field on M_1, then $\hat{D}^F_{V_1} = \hat{D}^F_{V_2}$.

Here $(V_2)_r = A(V_1)_r$, where A is the linear transformation that takes the triangle r in M_1 to the corresponding triangle in M_2. Note that $\hat{D}_{V_i}$ is computed using the embedding X_i.

Integration by parts. For a closed surface, we have that $\int_M f(\nabla \cdot V) = \int_M \langle \nabla f, V \rangle = \int_M D_V(f)$, for all $f : M \to \mathbb{R}$. This holds exactly in the discrete case, when using the standard vertex-based discrete divergence, defined as in [PP03]:

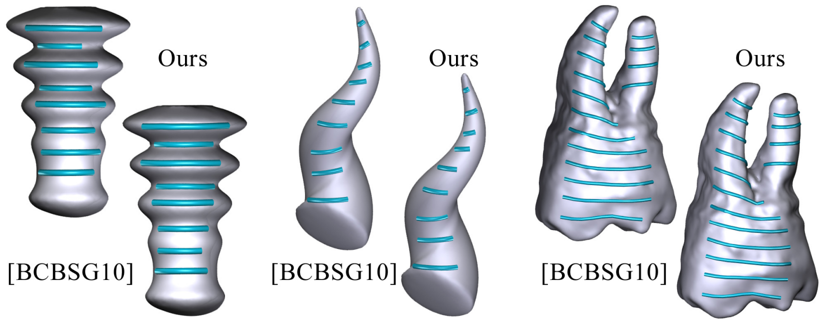


Figure 2.9: Geodesic distances between pairs of starting points are measured before and after the flow. Comparing the normalized average error for the models shown yields (left to right): 0.2,0.96, 2.47 for our method, and 0.23,1.15, 4.5 for [BCBSG10] (units are average edge length).

Lemma 2.4.3. Let $M = (X, F, N)$, V a piecewise constant vector field on M, $f = \sum_i f_i \gamma_i$ a PL function on M, and w_i the Voronoi area weights, then:

$$\sum_{i=1}^{|X|} w_i (\hat{D}_V f)_i = \sum_{i=1}^{|X|} w_i f_i (\nabla \cdot V)_i.$$

2.5 Applications

In this section, we describe how our representation can be used to compute vector fields which have various desirable properties. While some of the suggested applications have been attempted before (e.g. designing vector fields using direction and singularity constraints [FSDH07, CDS10], computing Killing vector fields [BCBSG10] and symmetric vector fields [PLPZ12], among others), our framework is unique in that it allows to combine any such constraints into a single optimization problem. In addition, we provide a proof-of-concept for more advanced tools, such as jointly designing vector fields on two or more surfaces.

2.5.1 Implementation Details

Given a mesh M, scalars N_f, N_D and a set of desired properties for a vector field, we propose the following algorithm:

1. Compute the first N_f eigenfunctions of the LB operator $\hat{\phi}_i$, using the area normalized cotangent scheme [BKP$^+$10].

2. Compute the first N_D 1-form eigenfunctions of the Laplace-de Rham operator, and convert those to piecewise constant vector fields $\hat{\psi}_i$. We used the definitions from [FSDH07] for both operations.

3. Convert $\hat{\psi}_i$ to $\hat{D}^{LB}_{\hat{\psi}_i}$ using Equation (2.12).

4. Optimize simultaneously for the vector field $V = \sum_i a_i \hat{\psi}_i$ and its functional representation $D_V = \sum_i a_i \hat{D}^{LB}_{\hat{\psi}_i}$, by solving a linear system for a_i. The joint formulation allows us to stack constraints which are best represented using the operator (e.g. commutativity constraints) together with constraints which require the vector field (e.g. prescribed directions at specified locations). This yields a linear system $\mathbf{Wa} = \mathbf{c}$, which we solve in the least squares sense.

5. Output the computed vector field $V = \sum_i a_i \hat{\psi}_i$.

Throughout our experiments we used meshes in the range of 5k-200k vertices, with N_f and N_D between 50 and 300, depending on the experiment. The computational time was dominated by the eigen-decompositions and took a few minutes on a standard laptop.

Figures 2.3, 2.4, 2.5 and 2.7 from the previous sections were generated using this framework. In addition, we describe a few examples of potential applications of our framework, related to the properties discussed in Section 2.2.

2.5.2 Approximate Killing Vector Fields

Lemma 2.2.2 provides a linear constraint on the FVF operator, which guarantees that a given vector field is a KVF. We can use this result, and optimize for the best KVF on a given surface, by optimizing for a set of coefficients **a** such that the resulting operator D_V will commute with the Laplace-Beltrami operator, i.e. $||D_V \circ L - L \circ D_V|| = 0$. Here we get a homogeneous system $\mathbf{Wa} = 0$, hence the AKVF is the singular vector corresponding to the lowest singular value.

Figure 2.9 shows a comparison of the resulting vector fields with the results of the state-of-the-art algorithm [BCBSG10]. The comparison is done using the same meshes, where on each mesh we pick a few vertices and show the flow lines for a fixed time t starting from these vertices. Note, that we achieve similar results, but in our framework we can easily combine the KVF constraint with other constraints such as commutativity with a symmetry operator.

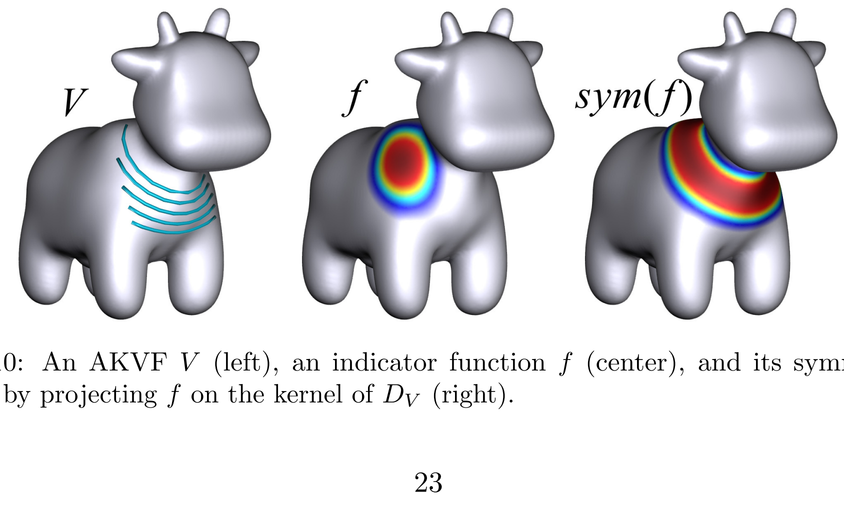


Figure 2.10: An AKVF V (left), an indicator function f (center), and its symmetrization computed by projecting f on the kernel of D_V (right).

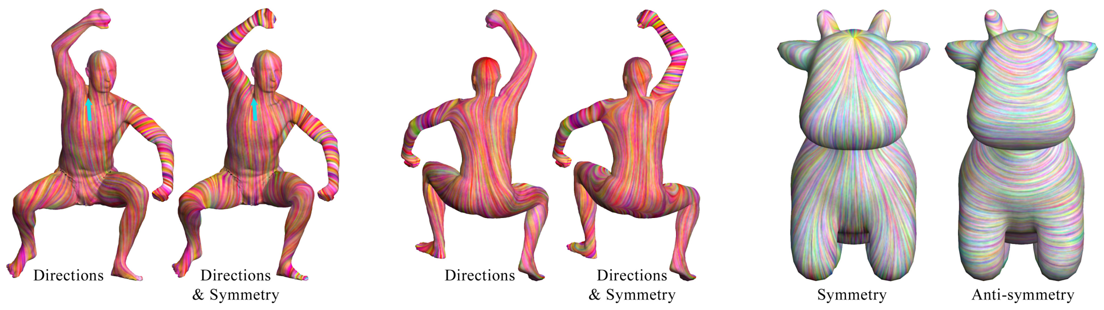


Figure 2.11: On the human model (left and center) we show design results with and without symmetry constraints - note the difference on the right hand. On the spot model (right) we show symmetric and anti-symmetric vector fields.

Interestingly, the spectral decomposition of the functional vector field operator is meaningful and potentially useful in applications. Specifically, functions are in the kernel of D_V if and only if they are fixed points of the flow Φ_V^t for all t (since $D_V f = 0$ if and only if $\exp(tD_V)f = f, \forall t$). Therefore, the kernel of an AKVF operator spans the linear subspace of symmetric functions under the corresponding symmetry. This implies, that given an arbitrary function f, we can symmetrize it by projecting it onto the kernel of such an operator. Figure 2.10 shows an example of an AKVF V, an indicator function f and its symmetrization $sym(f)$.

2.5.3 Composition with Mappings

Given a self-map S, we design a symmetric vector field by posing a constraint of the form $||D_V \circ S - S \circ D_V|| = 0$. Figure 2.11 (left and center) shows an example of a vector field designed with directional constraints and one designed with both directional and symmetry commutativity constraints. Note the difference on the hand of the model, as the symmetric constraints enforce similar behavior on both hands. Additionally, we can define an anti-symmetric vector field, by requiring $V(S(p)) = -V(p)$, where S is the symmetry map. To enforce this requirement, we use the constraint $||D_V \circ S + S \circ D_V|| = 0$. Figure 2.11 (right) shows an example of symmetric and anti-symmetric vector fields.

Given a collection of shapes, a desirable goal when designing vector fields is to have different constraints on each shape, yet generate compatible vector fields across the collection. In Figure 2.12 (right) we achieve this goal using the map composition property. We are given two shapes M_1 and M_2 and a functional map T_F between the corresponding function spaces. In addition, on each shape we are given a set of directional constraints c_1, c_2. We wish to generate vector fields V_i on the shapes M_i, such that V_i commute with T_F, and fulfill the constraints. A natural approach would be to transfer the constraints and solve separately for each mesh. However, as shown in Figure 2.12 (left), there is a large difference between the resulting fields - e.g in the locations of the singularities. Figure 2.12 (right), shows the result when solving jointly for both shapes. Note that the singularities on the back of the shape are consistent between the models. For evaluation, we transport V_1 to M_2 and measure the angle difference between the resulting vector field and V_2. Figure 2.12 (center) shows the resulting histogram, emphasizing that our joint design method preserves the directions better.

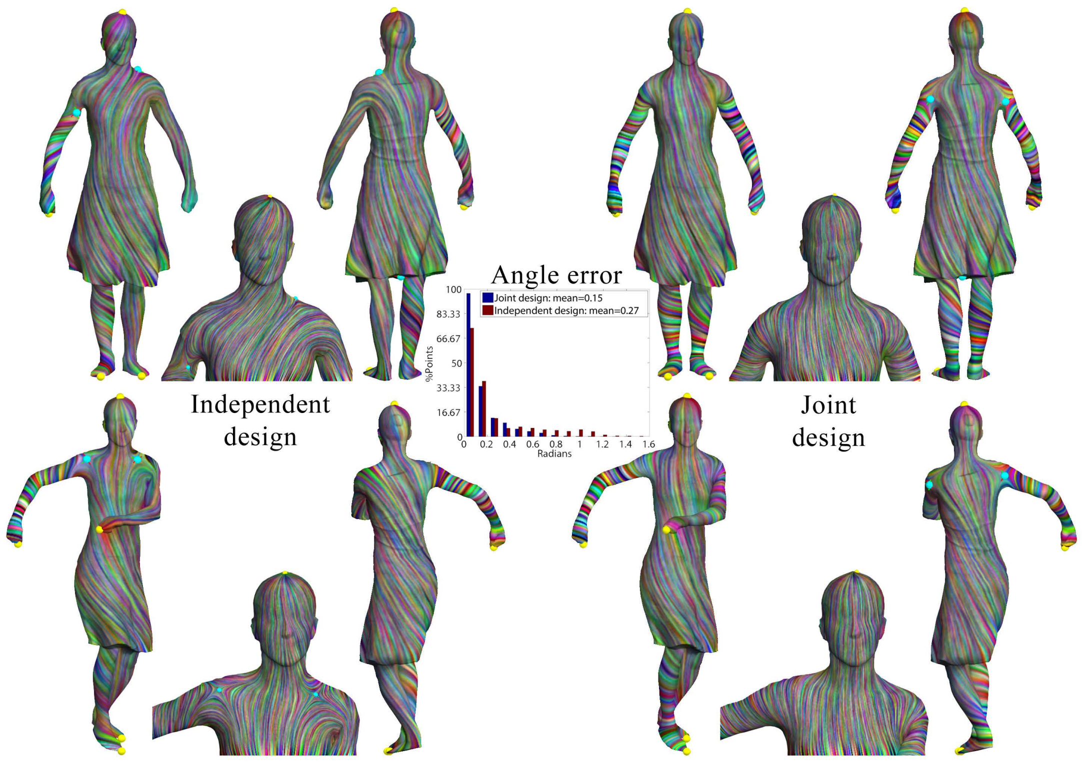


Figure 2.12: (left) Independent design on two shapes which are in correspondence does not yield a consistent vector field, even if compatible constraints are used. (right) Solving jointly using our framework yields consistent vector fields (note the corresponding locations of the singularities on the back of the shape). See the text for details.

2.6 Discussion

Tangent vector fields on surfaces are used in a myriad of applications in computer graphics and geometry processing. We propose to represent them as functional operators, thus enabling applications which were not easily attainable using standard representations. We have provided a discretization of the operator, and demonstrated it is consistent and experimentally convergent under refinement. Finally, we described some high level vector field design applications, such as Killing, symmetric and joint vector field design.

We believe the proposed representation opens the door for many additional applications. Specifically, the covariant derivative of one vector field with respect to another could potentially be useful for computing the Gaussian curvature, and for posing smoothness constraints for vector field design. Further applications include finding pairs of vector fields with zero Lie derivative for surface parameterization.

In general, we feel that we only uncovered the tip of the iceberg of possible applications and extensions of this framework. In an even broader context, considering both the operator representation of maps between surfaces, and the operator representation of vector fields, seems to imply that a lot is to gain by abstracting common notions in geometry processing, and viewing them more generally as operators. It remains to be seen whether this approach is applicable to additional concepts as well.

Chapter 3

Discrete Derivatives of Vector Fields on Surfaces – An Operator Approach

Vector fields on surfaces are fundamental in various applications in computer graphics and geometry processing. In many cases, in addition to representing vector fields, the need arises to compute their *derivatives*, for example for solving partial differential equations on surfaces, or for designing vector fields with prescribed smoothness properties. In this work, we consider the problem of computing the *Levi-Civita covariant derivative*, i.e., the tangential component of the standard directional derivative, on triangle meshes. This problem is challenging since formally, tangent vector fields on polygonal meshes are often viewed as being discontinuous, and hence it is not obvious what a good derivative formulation would be. We leverage the relationship between the Levi-Civita covariant derivative of a vector field and the directional derivative of its component functions to provide a simple, easy-to-implement discretization for which we demonstrate experimental convergence. In addition, we introduce two linear *operators*, which provide access to additional constructs in Riemannian geometry that are not easy to discretize otherwise, including the *parallel transport* operator, which can be seen simply as a certain matrix exponential. Finally, we show the applicability of our operator to various tasks, such as fluid simulation on curved surfaces, and vector field design by posing algebraic constraints on the covariant derivative operator.

3.1 Introduction

Tangent vector fields are ubiquitous in computer graphics. From fluid simulation to texture synthesis, the need to represent vectorial data arises in many applications. Often, it is necessary to compute the *covariant derivative* of a tangent vector field in an arbitrary tangent direction. For example, when simulating fluid flow using Euler equations, the covariant derivative of the fluid's velocity is the main ingredient in the computation of the time evolution of the flow [Tay96]. Furthermore, some vector fields are characterized by the properties of their derivatives: smooth vector fields [KCPS13] minimize the Dirichlet energy, while Geodesic vector fields [PHD$^+$10] are constant length and have symmetric covariant derivative operators. Although specific solutions have been tailored to various applications, there currently exists little work on discrete representations of derivatives of tangent vector fields on polygonal meshes, which are applicable to general scenarios.

There are two main challenges in deriving such a discretization. First, even on smooth surfaces, defining derivatives of tangent vector fields is more involved than defining derivatives of functions. Specifically, comparing the values of a function at two points on the surface is trivial, but it is not obvious how given two tangent vectors at different points one can determine if they are “the same”, since tangent vectors at different points are expressed with respect to different reference frames. Hence, one needs a way to transport vectors across tangent planes, a construct encoded by a notion of *parallel transport*. Unfortunately most theoretical treatments of these topics make heavy use of local coordinates, which makes defining discrete analogues for polygonal meshes difficult.

The second challenge is due to the nature of discrete surfaces, namely polygonal meshes, and the way tangent vector fields are represented. The simplest representation, which is the one we opt for, is piecewise constant vectors on the faces of the mesh. However, in such a representation vector fields are discontinuous across edges, which a priori can lead to difficulties in computing their derivatives. In this paper, we formalize this intuition by showing that for this choice of vector field representation, there exists no definition of a discrete vector field derivative which satisfies all the properties of the continuous Levi-Civita covariant derivative exactly. Faced with these challenges, we propose a novel approach to discretize the Levi-Civita covariant derivative. We compute the *directional derivatives* of the vector field’s *component functions* and take the tangential part of the resulting vector field. In the continuous case, it is well-known that such a definition yields the unique Levi-Civita covariant derivative [Mor01, pg. 181]. While being intuitive and easy to implement, our approach offers several conceptual benefits. First, by working with functions instead of vector fields, we overcome the difficulty of comparing vectors in different tangent planes. Second, by projecting the component functions on a multi-scale basis, we impose some smoothness on the underlying vector field, which allows us to obtain a stable discretization of the Levi-Civita covariant derivative for which we demonstrate experimental convergence. Finally, we derive a representation of the covariant derivative as an operator *acting on vector fields*. This allows us to design vector fields with various properties, and to define parallel transport without resorting to the computation of discrete flow lines, simply as a matrix exponential.

3.1.1 Related Work

Unlike the discretization of the directional derivatives of functions, which can be reduced to computing gradients and is thus well-established (e.g., [BKP$^+$10, ABCCO13]), there exists, to the best of our knowledge, no unified treatment of covariant derivatives of vector fields on meshes. Some derived quantities such as the divergence and the curl have received wide attention [PP03, War07, Hir03, MDS$^+$02], whereas the general case we are interested in—the *Levi-Civita covariant derivative of a tangent vector field*, has not been discretized directly. As a full review of the use of derivatives of vector fields in applications is beyond our scope, we mention a few representative examples.

Discrete calculus frameworks There exist several frameworks for geometry processing and graphics applications that provide discretizations of differential quantities. Discrete exterior calculus (DEC) [Hir03] is one of the most extensive and widely used, and provides discrete equivalents for vector field operators such as curl, divergence, gradient and Hodge Laplacian. In addition, DEC provides a strong theoretical foundation in the discrete setting with theorems which mimic the corresponding statements for smooth surfaces. However, not all operators are supported in DEC, and specifically there is currently no consistent discretization of the covariant derivative of vector fields. Other frameworks, such as surface Finite Element Methods (FEM) [DE13], and Finite Element Exterior Calculus [AFW06] have also been proposed, but their focus has traditionally been on solving boundary value problems for differential equations. While these approaches have been successfully used to discretize differential operators including the Laplace–Beltrami operator [War07, DE13], discretizing arbitrary differential quantities on unstructured meshes remains challenging.

Another approach is to use a global conformal parameterization to the plane [LWC05] together with standard FEM to solve a modified problem which takes into account the distortion introduced by the parameterization. Such methods, however, can be sensitive to the large area distortion induced by conformal maps, which may cause many triangles in the planar mesh to collapse, leading to unstable numerical systems.

Vector field design Vector derivatives are often required for vector field design applications. One of the most prominent requirements is that the resulting vector field is *sufficiently smooth*, and this calls for a way to relate vectors in nearby tangent spaces. On a triangle mesh, two classes of methods have been proposed to quantify smoothness of vector fields. The first is to use discrete 1-forms instead of vector fields, and rephrase the required operators in terms of DEC [FSDH07, BCBSG10], making use in particular of the Hodge Laplacian operator which provides a measure of smoothness for vector fields in a similar way as the Laplace–Beltrami operator does for functions. However, this limits the scope of applications, since, for example, it is not clear how to compute the *directional* derivatives of vector fields, and if various operators (e.g., the symmetric part of the covariant derivative operator) can be represented in DEC.

Another common method to measure smoothness of vector fields is by prescribing a rule on every edge of the mesh, which allows one to compare vectors on the faces across this edge. Perhaps the most natural instance of this approach is to relate vectors on a pair of neighboring triangles by "unfolding" them into a single plane. Indeed, it is customary to refer to this process as the *discrete Levi-Civita connection*, e.g., [CDS10], and various comparison rules have been proposed for different applications ([PS06, CDS10, PHD$^+$10, LJX$^+$10] among others).

However, this general approach has several significant drawbacks. First, these comparison rules only define directional derivatives in the direction of the dual edges of the mesh, and it is not obvious what the derivative should be in a general direction. If we extend this approach to a general direction by following the discrete geodesic in that direction, it is not clear what happens at a vertex. Furthermore, the resulting definition is not stable: a small change in the direction can change the following face on the geodesic path, yielding a different vector and potentially a large change in the derivative. Finally, in many cases the "unfolding" approach is used to define discrete parallel transport, namely a way to transfer a vector between faces on the mesh. Our method provides a more general definition of parallel transport, by allowing to transport a vector field on *the flow lines of another vector field.* Implementing this using the unfolding approach would require numerically integrating the direction vector field to generate the flow lines, and then unfolding the triangles along the flow lines, which are both algorithmically complicated and numerically sensitive operations. Using our method we can compute discrete parallel transport simply using a matrix-vector multiplication.

Fluid simulation The directional derivative of a vector field with respect to itself appears in various PDEs, one of them is given by the Euler equations for inviscid incompressible flow. Understanding the solutions to these equations is a research field in itself (see e.g., [Bat00]), thus we only mention some of the more relevant work in computer graphics, and specifically fluid simulation on surfaces. Existing solutions include parameterization-based techniques [LWC05], and methods which assume a particular structure on the mesh, e.g., by working with subdivision surfaces [Sta99]. These methods have the drawbacks of introducing unwanted errors due to the distortion of the parameterization, and the added complexity of converting a general triangle mesh to a subdivision surface. Note that on a two-dimensional surface, the Euler equations can be reformulated in terms of the *vorticity* of the flow [NVW12, ETK$^+$07], yielding a simpler representation of the velocity through the *stream function.* However, vortex methods have several limitations, e.g., it can be more difficult to set boundary conditions, and therefore in some cases it is preferable to use a velocity based method. Finally, a method which is tailored for inviscid and incompressible flows on triangle meshes is provided in [SY04]. This method is based on semi-Lagrangian velocity advection on a triangle mesh, which requires tracing velocity flow lines and triangle unfolding, that suffer from the drawbacks mentioned previously.

3.1.2 Contributions

Our main contribution is a simple yet efficient method for discretizing the Levi-Civita covariant derivative on triangle meshes. We focus on three aspects in our exposition: properties of the discretization, the novel perspective offered by the operator approach, and sample applications. Note, that since we provide a *tool* and not a specialized application, we focus on proof-of-concept scenarios to illustrate the possibilities associated with our discretization.

In the following sections we discuss our main contributions:

- The discrete formulation of the Levi-Civita covariant derivative, including experimental convergence results (Section 3.3).
- A representation of the derivative as a linear operator that takes vector fields to vector fields, whose algebraic properties have geometric meaning, e.g., exponentiation leads to an algebraic definition of parallel transport (Section 3.4).
- Several examples demonstrating the applicability of our discrete derivative: vector field design and fluid simulation on surfaces (Section 3.5).

3.2 Directional derivatives of vector fields

Our main goal is to discretize the directional derivative of a vector field on a surface, also known as the Levi-Civita covariant derivative. We will first discuss the definition of such a derivative and its properties in the continuous case. We provide a brief intuitive introduction to the required concepts in this section. Readers well versed in differential geometry can skim these and proceed to the discrete treatment in Section 3.3. As we focus mostly on the geometric intuition behind the definitions, we refer the interested readers to [Mor01, Chaps. 5.2, 5.3] and [Car94, Chap. 2] for the detailed treatment.

3.2.1 Notation

In the following we denote a surface by $M \subset \mathbb{R}^3$, upper case letters (e.g., U, V, W) denote tangent vector fields, and lower case letters (e.g., f, g) denote real-valued functions. We denote by $\|\cdot\|$ an operator which takes a tangent vector field and outputs a function of its pointwise norms.

3.2.2 The Levi-Civita covariant derivative

To gain some intuition, first consider the motion of a particle in the plane, $\mathbb{R}^2$. Its trajectory forms a path $\gamma(t) \in \mathbb{R}^2, t \in \mathbb{R}$, and its velocity $\gamma'(t) = U(t) \in \mathbb{R}^2$ is a vector

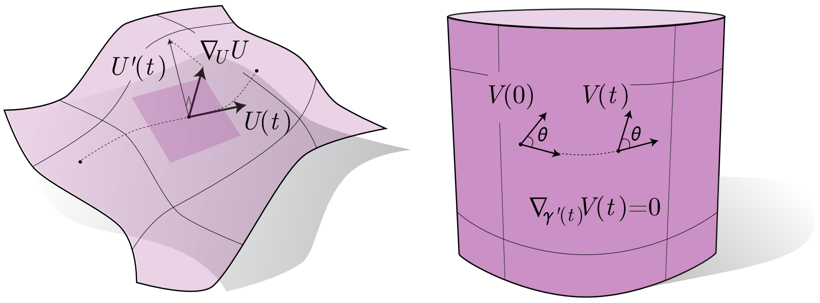


Figure 3.1: (left) The velocity $U(t)$ and acceleration $U'(t)$ of a particle traveling along a curve $\gamma(t)$ on a surface, and the tangential component of the acceleration $\nabla_U U$. (right) The parallel transport of V_0 along γ is the vector field $V(t)$ defined as the unique solution of the differential equation $\nabla_{\gamma'(t)}V(t) = 0$ with $V(0) = V_0$.

tangent to the path. Its acceleration is the vector:

$$U'(t) = \lim_{\Delta t \to 0} \frac{U(t+\Delta t) - U(t)}{\Delta t}. \tag{3.1}$$

For example, if the trajectory is a straight line and the velocity is not constant, then $U'(t)$ will point in the direction of travel. If the particle travels at constant speed, then the acceleration $U'(t)$ is in a direction orthogonal to the path, since $\langle U(t), U(t)\rangle' = 2\langle U'(t), U(t)\rangle = 0$. Like the velocity, the acceleration vector lies in $\mathbb{R}^2$.

Now, consider the same particle traveling on a curved surface $M \subset \mathbb{R}^3$. Again, its trajectory forms a path $\gamma(t) \in M, t \in \mathbb{R}$, and its velocity vector $U(t)$ is tangent to it. However, the acceleration vector $U'(t)$ is no longer tangent to M and it decomposes into a component normal to M, the *normal acceleration*, and a component tangent to M, the *tangential acceleration* (see Figure 3.1, left). Intuitively, since the particle is constrained to live on the surface M, we can take an intrinsic point of view by considering only the tangential part of the acceleration.

We can similarly compute the tangential component of the derivative of *any* vector field V defined along a curve, and not necessarily tangent to it, by considering the tangential component of $\lim_{\Delta t \to 0} \frac{V(\gamma(t+\Delta t)) - V(\gamma(t))}{\Delta t}$ along the curve γ. Finally, using the standard x, y, z coordinates in $\mathbb{R}^3$, this definition can be further extended to define the covariant derivative of a tangent vector field $V = (v_x, v_y, v_z)$ on M in a specific direction given by a vector field U on M:

$$\nabla_U V(p) = P_p((D_U v_x, D_U v_y, D_U v_z)(p)), \; p \in M, \tag{3.2}$$

where P_p is the orthogonal projection on the tangent plane to M at p and, for any function f, $D_U f = \langle \nabla f, U\rangle$ denotes the derivative of f in the direction of U. Notice that $(D_U v_x, D_U v_y, D_U v_z)(p)$ is a vector in $\mathbb{R}^3$, while $\nabla_U V(p)$ is a tangent vector. The vector field $\nabla_U V$ is known as the *Levi-Civita covariant derivative* of V with respect to U [Mor01, pg. 181].

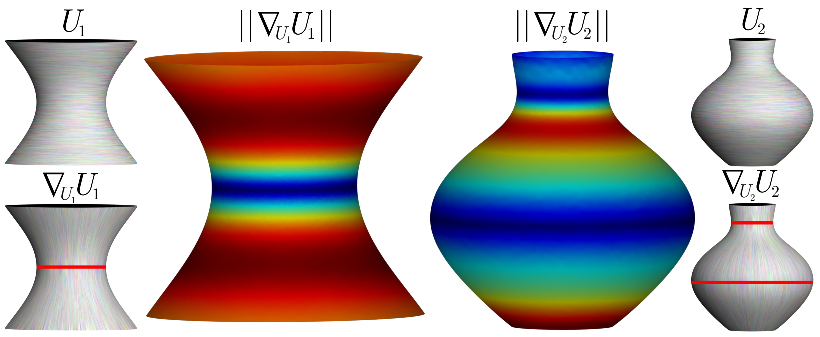


Figure 3.2: Constant norm vector fields U_i on a surface of revolution, and their norm $\|\nabla_{U_i}U_i\|$ and flow lines. Note that the norm is zero on the geodesics (marked red), and that the flow lines are orthogonal to U_i, since they are constant norm.

3.2.3 Parallel transport

The definition of the covariant derivative is closely related to the notion of parallel transport. Intuitively, parallel transport allows us to "carry" a vector along a curve, such that it remains "parallel" to itself. For example, the norm of a parallel-transported vector remains fixed, and if the curve is a geodesic then the angle the vector forms with the tangent to the curve also remains fixed. This is formalized using the idea that parallel transport should be the *integral of the covariant derivative*. Formally, given a curve $\gamma(t)$ in M and a tangent vector V_0 at $\gamma(0)$, the parallel transport of V_0 along γ is defined as the unique solution of the differential equation $\nabla_{\gamma'(t)}V(t) = 0$ with initial condition $V(0) = V_0$ [Car94, pg. 52], see Figure 3.1 (right).

Before we dive into the properties and the proposed discretization of $\nabla_U V$ we would like to give some intuition as to the quantity we are computing. Consider a surface of revolution, like the ones shown in Figure 3.2, and a constant norm vector field U which is orthogonal to the rotation axis (i.e., it "goes around" the surface). Now consider a particle traveling on the flow lines of U at constant speed. If the flow line is a geodesic, e.g., as the curves marked in red, then traveling at constant speed would yield 0 tangential acceleration. This is seen in the center figures, which show the color coding of $\|\nabla_U U\|$. If the particle is not traveling on a geodesic, it has to accelerate to keep "turning". However, since the speed is constant, the acceleration $U'(t)$ would be orthogonal to the direction of travel, as is seen in the figures showing the flow lines of $\nabla_U U$.

3.2.4 Properties

As we aim for a generic discretization of $\nabla_U V$, which works well in various applications, we would like to assess the properties that are required from such an object. For example, it has been shown in [War07] that for the Laplace–Beltrami operator and under mild conditions, there is no discretization which fulfills all the defining properties of the continuous operator. In our case, the Fundamental Theorem of Riemannian

Geometry guarantees that if an operator fulfills the following five properties, then it is the *unique* Levi-Civita covariant derivative [Car94, pgs. 50–55]. Hence, it is of interest to understand these properties, and see whether they are achievable in the discrete case. To make the discussion more concrete, we also denote for each property the application in which it will be required.

Linearity. As any derivative it is a linear operator:

$$\nabla_U(V + W) = \nabla_U V + \nabla_U W. \tag{3.3}$$

Linearity allows us to represent the operator ∇V in a basis, and construct various energies for vector field design.

Product rule.

$$\nabla_U(fV) = f\nabla_U V + V D_U f. \tag{3.4}$$

Although we do not use this property directly in our applications, the product rule is a fundamental characteristic of any derivative.

Locality. The derivative operator is "local" in the direction argument, namely it depends on the value of U at a point, and not on its neighborhood. In other words, if U_1 and U_2 are vector fields such that $U_1(p) = U_2(p)$ for some point p, then $(\nabla_{U_1} V)(p) = (\nabla_{U_2} V)(p)$ for any smooth vector field V. This means that there are no derivatives of U involved, and therefore this requirement can be rephrased as linearity with respect to functions in the direction argument:

$$\nabla_{fU+gW}(V) = f\nabla_U V + g\nabla_W V. \tag{3.5}$$

This allows us to represent the operator ∇_U in a basis, which we use for computing parallel transport.

Metric compatibility. This property relates the derivative of a vector field to the derivative of its norm. Similar to the case of a particle in $\mathbb{R}^2$, where we had $\langle V(t), V(t)\rangle' = 2\langle V'(t), V(t)\rangle$, in general, $D_U\langle V, V\rangle = 2\langle \nabla_U V, V\rangle$. Note that together with linearity, this implies for any pair of vector fields, V and W:

$$D_U\langle V, W\rangle = \langle \nabla_U V, W\rangle + \langle V, \nabla_U W\rangle. \tag{3.6}$$

Symmetric Hessian. Finally, the last property relates to the second derivatives of functions. In the Euclidean case, the Hessian matrix is symmetric since partial derivatives commute. The generalization of the Hessian to the surface is the bilinear operator: $H(f)(U, V) = \langle \nabla_U \nabla f, V\rangle$ [Car94, pg. 142]. The last property requires that this operator is symmetric:

$$\langle \nabla_U \nabla f, V\rangle = \langle \nabla_V \nabla f, U\rangle. \tag{3.7}$$

A consequence of this property is that $[U, V] = \nabla_U V - \nabla_V U$ for any vector fields U and V, where $[\cdot]$ represents the *Lie bracket* operator [Car94, p. 27]. We use this operator to design local parameterizations.

In the following section we investigate the discretization of the covariant derivative. We first address the question of how vector fields are represented on a mesh, and discuss our choices. Then we consider the challenges for our choice of representation in the discrete setting. We show that for piecewise constant vector fields, under some mild conditions, it is not possible to define a discrete version of the covariant derivative operator which is both linear and fulfills the metric compatibility property. Finally, we propose a simple approach that is based on the recently introduced multi-scale discretization of the directional derivative of functions [ABCCO13], and we demonstrate experimental convergence of the previously mentioned properties under mesh refinement, when both the vector fields and the functions are smooth.

3.3 Discretization

3.3.1 Vector field representation

The definition of a derivative of a vector field is closely linked with the way vector fields are represented in the discrete setting. One option is to use discrete 1-forms [Hir03], which would require using the flat and sharp operators for converting from vector fields to 1-forms and back. Another option is to define a smooth atlas on the mesh through a parameterization of the 1-ring of each vertex (e.g., as in [ZMT06, KCPS13]), effectively turning the mesh into a smooth manifold. If a vector field is continuous and piecewise smooth in the atlas, it is possible to define first weak derivatives. Further, recent work by [RS14, MPZ14] showed how a combinatorial data structure can be used to represent vector fields while ensuring that field flow lines do not merge.

While these options can be a potential starting point for discretizing the covariant derivative, they require a somewhat complicated definition of a discrete vector field. We, on the other hand, choose the most simple discretization of a tangent vector field, namely *piecewise constant on faces.* Such vector fields occur often in applications. For example, scalar functions are often discretized as piecewise linear on the vertices of the mesh, and their gradients are piecewise constant vector fields. Furthermore, in mesh parameterization and mesh quadrangulation applications [KNP07, BZK09, CBK12, BCE$^+$13, MPZ14, MPZ14, PPTSH14] piecewise constant vector fields are often given as constraints for controlling the alignment of the result. Hence, as we work directly with piecewise constant vector fields, without requiring additional conversions to 1-forms or atlas-based representations, our approach is simpler, more intuitive and easier to implement.

3.3.2 Notation

We represent surfaces with triangle meshes, given by $\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$, which denote the vertices, edges and faces, respectively. Functions are represented as *piecewise constant on the faces*, namely $f : \mathcal{F} \to \mathbb{R}, f = \{f^i, i \in \mathcal{F}\}$. Tangent vector fields are given as piecewise constant on triangles, namely $U : \mathcal{F} \to \mathbb{R}^3, U = \{U^i = (u^i_x, u^i_y, u^i_z), i \in \mathcal{F}\}$, such that U^i is parallel to the plane containing the i-th face. Discrete operators are represented with a "tilde", e.g., $\tilde{D}_U : (\mathcal{F} \to \mathbb{R}) \to (\mathcal{F} \to \mathbb{R})$ is the discrete directional derivative for functions and $\tilde{\nabla}_U : (\mathcal{F} \to \mathbb{R}^3) \to (\mathcal{F} \to \mathbb{R}^3)$ is the discrete covariant derivative for vector fields. In what follows, we assume to be given a function f and tangent vector fields U, V.

3.3.3 Challenges in the discrete setting

As mentioned, we choose to represent vector fields as piecewise constant on the faces. Such a representation, while simple and intuitive, leads to an inherent difficulty in defining a meaningful notion of covariant derivatives, since intuitively the derivatives of piecewise constant vector fields should be zero at the faces.

Indeed, inside a triangle, taking derivatives of piecewise constant vector fields is futile. Thus, a bigger patch must be taken into account. This, however, would require constructing a mechanism for transporting vectors across triangles. Moreover, it is easy to see that given the above discretization of vector fields and functions, the product rule (Eq. (3.4)) cannot hold exactly for every pair of functions and vector fields. This, however, is true for many notions of discrete derivatives.

Unfortunately, there exists a more fundamental difficulty in discretizing the Levi-Civita covariant derivative, which holds not only for our discretization, but even if functions do not "live on the same domain" as the vector fields, e.g., functions that are piecewise linear. In particular, even in this case, two of the defining properties of the covariant derivative, namely linearity and metric compatibility, cannot be both satisfied exactly in the discrete setting, under some mild conditions. To state this precisely, since the inner product $\langle U, V \rangle$ produces a function on the faces of the triangle mesh, to allow discrete functions to live on a different domain we can use an averaging operator $\mathcal{A}$ that takes functions on faces and produces functions on vertices, edges or faces. We will assume that $\mathcal{A}$ is linear, non-negative and maps constant functions to constant functions. This leads to the following formulation of the metric compatibility condition:

$$\tilde{D}_X \mathcal{A}(\langle U, V \rangle) = \mathcal{A}(\langle \tilde{\nabla}_X U, V \rangle + \langle \tilde{\nabla}_X V, U \rangle). \tag{3.8}$$

Here $\tilde{D}_X$ is a directional derivative for functions with respect to the vector field X. I.e., $\tilde{D}_X$ takes a function defined on some domain (e.g., vertices, edges or faces) and produces a function defined on the same domain. $\tilde{\nabla}_X U$ is the covariant derivative for vector fields, and the inner product is the standard inner product of vector fields in

$\mathbb{R}^3$. Under these conditions, we have the following result (proved in the supplemental material):

Lemma 3.3.1. *If $\tilde{D}_X$ is a linear operator such that $\tilde{D}_X f = 0$ if f is a constant function, and the covariant derivative for vector fields is linear: $\tilde{\nabla}_X(U_1 + U_2) = \tilde{\nabla}_X U_1 + \tilde{\nabla}_X U_2$, then the metric compatibility condition (Eq. (3.8)) implies that $\tilde{D}_X f = 0$ for all f in the range of $\mathcal{A}$. I.e., $\tilde{D}_X \mathcal{A}(h) = 0$ for any h.*

We note that although this lemma is stated for vector fields that are constant on the faces, the proof is actually quite general and can be adapted to other settings as well. Hence, as we cannot hope to achieve the exact properties of the smooth covariant derivative, we opt for a simple discretization which is based on the directional derivative of the *component functions*, as given by equation (3.2). Using this definition, it is possible to show that all the properties of the Levi-Civita covariant derivative (except the symmetry of the Hessian) are all consequences of the product rule for functions [Mor01, pg. 181]. Therefore, if the operator $\tilde{D}_U f$ provides a better approximation to the product rule as the mesh resolution increases, so we can expect that the operator $\tilde{\nabla}_U V$ will give a better approximation to properties (3.3)–(3.6) under mesh refinement, although the metric compatibility condition will never be satisfied exactly.

It has recently been shown in [ABCCO13] that it is possible to discretize the directional derivative of functions $\tilde{D}_U f$ using a *multi-scale* basis, such that the error in the product rule property experimentally decreases with the increase in the mesh resolution. We choose a similar discretization for the directional derivative of functions defined on *the faces* of the mesh, and thus get experimental convergence of the product rule for the component functions of the vector field. This in turn, in the convergence experiments we performed, leads to experimental convergence of the covariant derivative properties.

3.3.4 Directional derivative of functions

In the discrete differential geometry literature, functions are commonly discretized either as scalars on the vertices or as scalars on edge midpoints, which are then linearly interpolated to the faces. These are known as conforming and non-conforming linear elements, respectively. In both cases, the gradient operator is well-defined as piecewise constant on the faces (see [War07, Chap. 2] for a full discussion).

Contrary to the common setting, our functions are defined on faces, thus we need to extend the notion of a discrete gradient. Given a function f, we define an averaging operator $\mathcal{A}$, and define $\tilde{\nabla} f = \nabla \mathcal{A} f$, where $\mathcal{A}$ averages the values of f to the edges, and ∇ is the discrete gradient for non-conforming elements. Potentially, it is possible to define $\mathcal{A}$ such that it averages values to the vertices instead of edges. However, then $\mathcal{A}$ will be of size $|\mathcal{V}| \times |\mathcal{F}|$, and therefore, its range will be smaller than its domain. Thus, there will necessarily be two functions on the faces which are mapped to the same

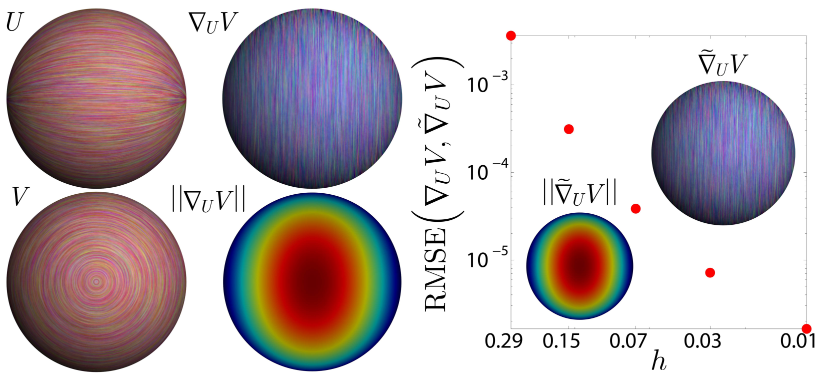


Figure 3.3: Comparison of our discretization $\tilde{\nabla}_U V$ with the analytic solution for specific U, V on the sphere. We show the convergence graph for the RMSE error for decreasing mean edge length, as well as a visualization of the flow lines and norm of the computed $\tilde{\nabla}_U V$ for the densest mesh.

function on the vertices. This will lead to difficulties, as it can introduce non-zero vector fields, whose interpolation to the vertices leads to a zero vector field. If, on the other hand, $\mathcal{A}$ averages to the edges, its size is $|\mathcal{E}| \times |\mathcal{F}|$, and therefore the range is larger than the domain, and this problem is potentially avoided. It is easy to see that a positive local averaging operator $\mathcal{A}$ will have an empty kernel in general, and in particular for any mesh that has at least one odd degree vertex (see the proof in the supplemental material).

Formally, we define the directional derivative for functions as:

$$\tilde{D}_U f = \langle \nabla \mathcal{A} f, U \rangle, \tag{3.9}$$

where $\mathcal{A}_{ij} = w_j / \sum w_k$ if i is an edge in face j, and $\mathcal{A}_{ij} = 0$, otherwise. w_j is the area of face j and the sum runs over the faces which share the edge i. Now, as $\mathcal{A}f$ is a function on edges, its gradient is piecewise constant per face, and has a standard definition (see [Pol05, Sec. 2.3]).

As mentioned previously, we represent the operator $\tilde{D}_U$ in a reduced multi-scale basis (the eigenfunctions of the Laplace–Beltrami operator), as this enforces some smoothness on our vector fields.

3.3.5 Covariant derivative of vector fields

Our covariant derivative operator is based on the *extrinsic* definition presented in Eq. (3.2). Given the discretization for the directional derivative of functions on the faces, the covariant derivative for vector fields follows easily:

$$\tilde{\nabla}_U V(p) = P_p\left((\tilde{D}_U v_x, \tilde{D}_U v_y, \tilde{D}_U v_z)(p) \right), p \in \mathcal{M} \tag{3.10}$$

where $V = (v_x, v_y, v_z)$ and P_p is the projection operator onto the tangent plane of $\mathcal{M}$ at p. As the directional derivatives of the components of V are given on the faces, P_p is well defined.

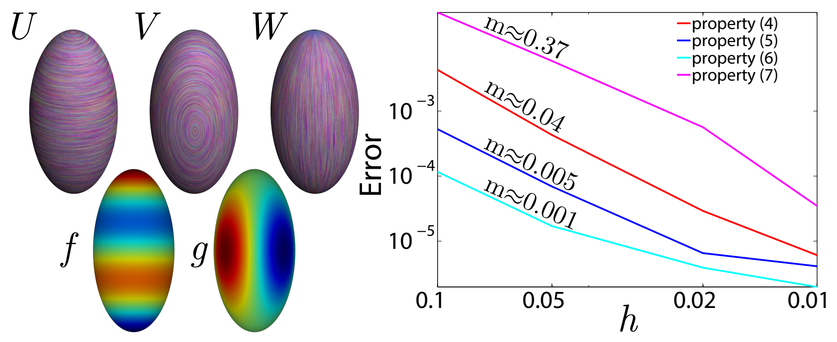


Figure 3.4: The behavior of our discretization of the covariant derivative on the properties (3.4)–(3.7) under mesh refinement, for the ellipsoid model. We show the RMSE error of the left hand side vs. the right hand side of the equation for decreasing mean edge length h. Note that the plot suggests a polynomial convergence rate in h, where we denote by m the respective slope estimate. We additionally show the functions and vector fields that were used for the highest mesh resolution. See the text for further details.

To summarise, given two piecewise constant vector fields, U and V, we first take the component coordinate functions of V, average them onto the edges, and compute the corresponding gradients. These are piecewise constant on the faces, therefore their inner products with U give us three real-valued functions on the faces. We use those functions to construct a vector field in $\mathbb{R}^3$, and project this vector field onto the faces.

To validate our discretization, we experiment with known vector fields U, V on the unit sphere and compare our result with the expected result in the continuous setting. Figure 3.3 shows the result of this comparison, for meshes with decreasing average edge length h. We show U, V, the analytic result $\nabla_U V$, and the result of our computation $\tilde{\nabla}_U V$. Note that the convergence is polynomial in h, and that for the most dense mesh the figures of the flow lines and norm are almost indistinguishable from the ground truth. We further demonstrate the convergence results in Figure 3.4, which shows the log log plot of the RMSE error of properties (3.4)–(3.7), for ellipsoid meshes with decreasing average edge length h. We additionally show the vector fields U, V, W and the functions f, g which were used for the mesh with smallest edge length. The functions f, g are the eighth and tenth eigenfunctions of the area weighted cotangent Laplace–Beltrami operator and the vector fields U, V, W correspond to eigen 1-forms 4, 3 and 1 of the Hodge Laplacian. Note that the plot suggests a polynomial convergence rate in h, where we denote by m the respective slope estimate. Furthermore, given eq. (3.10), it is easy to verify that property (3.3) holds exactly.

3.4 Geometry from linear operators

In addition to computing the quantity $\tilde{\nabla}_U V$, it is often advantageous to fix one of the vector fields, and consider the corresponding operator on all possible inputs. For example, we can omit the direction U and consider the operator $\tilde{\nabla} V$, which will provide some information on the derivatives of V in *all possible directions*. This point of view is useful, because it can uncover some hidden structure of V, in a global way. As a simple example, the singular vector of $\tilde{\nabla} V$ which corresponds to the smallest singular value, will provide the directions in which V changes as little as possible.

This interplay between the algebraic properties of the operators and the geometry of the vector fields they represent is quite useful in practice, because it allows us to do *global* operations which are traditionally local. For example, manipulating $\tilde{\nabla} V$ is instrumental for vector field design, and $\tilde{\nabla}_U$ allows us to easily compute parallel transport.

3.4.1 Preliminaries

Matrix representation While it is possible to analyze these operators directly as abstract linear operators, it is more intuitive to consider their *matrix representation.* Specifically, we assume that we have a finite orthonormal basis of vector fields $\{\Psi_i, i \in 1, .., k\}$, i.e., $\int_M \langle \Psi_i, \Psi_j \rangle = 1$ if $i = j$ and 0 otherwise, and such that the vector fields we are interested in can be represented as $V = \sum_{i=1}^{k} a_i \Psi_i$ (in Section 3.5.1 we will elaborate more on our choice of basis). Now, any linear operator $\mathcal{R}$ from tangent vector fields to tangent vector fields can be represented using a $k \times k$ matrix R, whose (i, j) entry is: $R_{i,j} = \int_M \langle \mathcal{R}(\Psi_i), \Psi_j \rangle$. In the following we will discuss the properties of the operators using their matrix representations. For example, when we mention the operator transpose, we refer to the corresponding matrix transpose.

Flow of a vector field We will need the following definition. The *flow* of a vector field U is a one-parameter family of maps $\Phi_U^t : M \to M$ for $t \in \mathbb{R}$, such that the following holds:

$$\frac{d}{dt} \Phi_U^t(p) = U(\Phi_U^t(p)), \qquad \Phi_U^0(p) = p.$$

Intuitively, the flow of a vector field encodes what happens to a particle which starts at a point $p \in M$, and its velocity is dictated by the vector field at every point. Hence, it provides a way to recover the trajectory of a particle from its velocity, and thus computing the flow is also known as *integrating* the vector field.

3.4.2 The operator ∇V

Operator action: $(\nabla V)(U) = \nabla_U V$.

Here V is fixed, and we compute its derivative in some direction given as input. This operator is the extension to surfaces of the Jacobian operator of vector fields in Euclidean space, which is simply the matrix of partial derivatives. Its algebraic structure provides us with information about the nature of the derivatives of V in various directions. For example, as any linear operator, it can be decomposed into

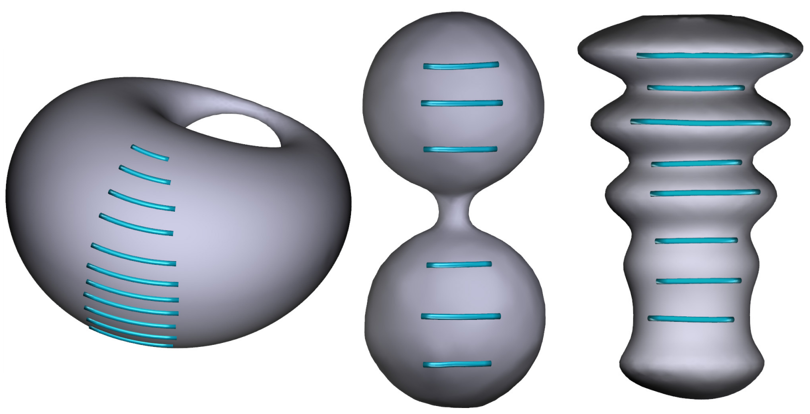


Figure 3.5: Approximate Killing vector fields computed by minimizing the symmetric part of ∇V.

symmetric and anti-symmetric parts:

$$\nabla V = \frac{1}{2}\left(\nabla V + (\nabla V)^T\right) + \frac{1}{2}\left(\nabla V - (\nabla V)^T\right) = K_V + G_V,$$

where as discussed previously, we consider the operator as a $k \times k$ matrix representation and thus can compute its transpose. The symmetric and anti-symmetric parts are also linear operators which take tangent vector fields to tangent vector fields and have geometric meaning.

Symmetric Part. The operator $K_V = \frac{1}{2}\left(\nabla V + (\nabla V)^T\right)$ is related to how much the flow Φ_V^t distorts the metric. Specifically, if $K_V = 0$, then V is called a *Killing vector field* (KVF), and its flow Φ_V^t is an isometry for all t ([Pet06, Chap. 7.1]). One such example in the plane is $V = (-y, x)$, whose flow is simply a global rotation. Such vector fields are quite rare, and exist only on very specific surfaces, however we can try to minimize $\|K_V\|^2$ for any surface, yielding vector fields whose flow is close to an isometry. Such vector fields are useful in geometry processing applications, as they allow to generate texture and geometric patterns [BCBSG10].

We use this property to design vector fields which are approximate KVFs, by solving a linear system of equations. Note, that as opposed to previous work, we can pose the constraints *directly* on the derivative operator, without requiring an indirect approach through commutativity with the Laplace–Beltrami operator [ABCCO13], or reformulation using DEC [BCBSG10]. Figure 3.5 shows a few approximate Killing vector fields computed this way. Interestingly, KVFs are also related to fluid flow on surfaces, as they provide a steady state solution to the Euler equations (see Section 3.5). Furthermore, the Killing operator K_V plays a role in the behavior of viscous fluids [NVW12], which we would like to investigate in future work.

Anti-symmetric part. The operator $G_V = \frac{1}{2}\left(\nabla V - (\nabla V)^T\right)$ encodes the failure of ∇V to be symmetric. We know from property (3.7) that if $V = \nabla f$ for some function f then ∇V is symmetric, hence it is possible to consider G_V as the failure of V to be the

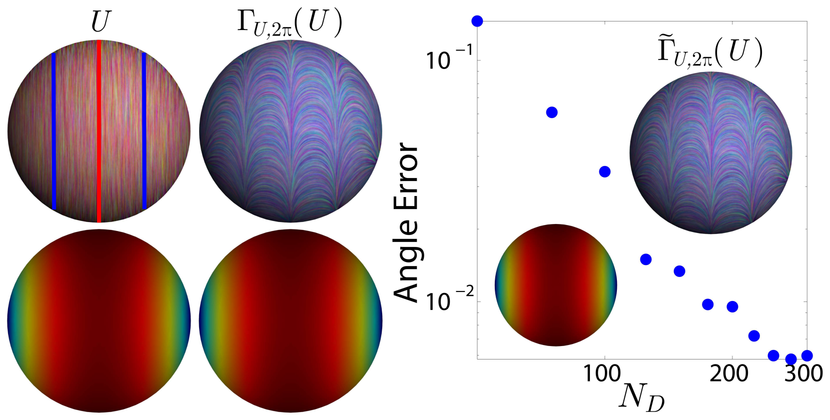


Figure 3.6: Parallel transport of a vector field U (left) along its own flow lines, comparison to the ground truth on the sphere (middle). Note the 3 marked singularity curves: the red curve is a geodesic, so vectors transported on it preserve their orientation. The blue curves are two symmetric singularity curves. The vectors transported on them rotate by π, so they reverse their orientation. The transition between these singularity curves is smooth. (right) Convergence graph of the error in the computed angle, and the final result of our computation for the largest number of basis functions.

gradient of a function. Specifically, minimizing $\|G_V\|^2$ with some additional conditions would provide vector fields which are "as gradient as possible". For example, if we require that $\|V\| = const$ it is possible to show that the flow lines of V are geodesics and V is a *geodesic vector field* (GVF) if and only if $G_V = 0$ [PHD+10], which can be useful in architectural geometry. In the applications section we demonstrate how by constraining ∇V to be symmetric, in addition to the smoothness induced by our framework, we can, using a much simpler setup, achieve similar results, even without adding the constraint on the norm of V. Furthermore, our approach allows us to combine various constraints, e.g., that the resulting vector field is symmetric with respect to some symmetry map of the surface.

Uniqueness. As we discussed, we can design vector fields V which have certain properties, by posing constraints (e.g., symmetry or anti-symmetry) on ∇V. This raises the question whether a given ∇V completely encodes V, or there can be multiple vector fields with the same ∇V. We have the following:

Lemma 3.4.1. *For a closed oriented surface M, $\nabla_U V = 0$ for every smooth U if and only if $V = 0$ or M is a flat torus.*

Hence, if $\nabla V_1 = \nabla V_2$, then $\nabla_U(V_1 - V_2) = 0 \ \forall\ U$, which by the lemma implies that $V_1 = V_2$, yielding the uniqueness we required.

3.4.3 The operator ∇_U

Operator action: $(\nabla_U)(V) = \nabla_U V$.

Here the direction of the derivative is given by a fixed U, and we compute the derivative of some vector field V given as input. This operator is closely related to the

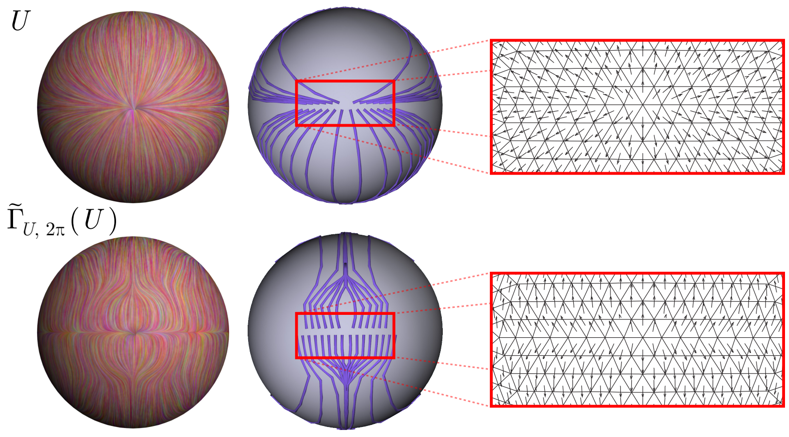


Figure 3.7: Parallel translation of U (top row) along the flow lines of U. Our discrete parallel transport is robust to merging flow lines as is shown in the result, $\tilde{\Gamma}_{U,2\pi}$, (bottom row).

directional derivative of functions, which we denoted as D_U. The scalar directional derivative operator was recently used by Azencot and colleagues [ABCCO13] to represent, analyze and design discrete vector fields. While this approach is useful in certain applications, it is also limited, since the scalar directional derivative operator D_U does not depend on the metric of the surface, making the computation of metric-dependent operations such as the parallel transport of vector fields impossible without additional structure. As we show below, the Levi-Civita covariant derivative, *acting on vector fields* shares many useful properties with the functional operator, such as uniqueness and decomposition, but also enables more applications including parallel transport in a very compact and convenient manner.

Uniqueness. The operator ∇_U encodes the vector field U uniquely. Hence we can design a vector field U by defining constraints on ∇_U. We have:

Lemma 3.4.2. *Two smooth vector fields U and V are equal if and only if $\nabla_U W = \nabla_V W$ for all smooth vector fields W.*

Symmetric part. The operator ∇_U allows us to easily distinguish divergence-free vector fields, as those whose symmetric part of ∇_U is zero:

Lemma 3.4.3. *Let M be a closed surface. A smooth vector field U is divergence-free if and only if ∇_U is anti-symmetric with respect to the inner product on the surface. I.e., if and only if $\int_M \langle \nabla_U V, W \rangle dx = -\int_M \langle \nabla_U W, V \rangle dx$ for all smooth vector fields V and W.*

Parallel transport. The Levi-Civita covariant derivative, represented as an operator ∇_U is intimately related to parallel translation *along the flow lines of U*. Suppose we have a vector field V and let $\Phi_U^t(p)$ be the flow of U. Now, consider the operator $\Gamma_{U,t}$, which takes a vector field on M and returns a vector field on M, which is defined as follows: $\Gamma_{U,t}(V)(p)$ is the vector obtained by parallel transporting the vector $V(\Phi_U^t(p))$

along the flow line from $\Phi_U^t(p)$ to p. It is well-known (e.g., [Car94, pg. 57]) that the following relation between the operators ∇_U and $\Gamma_{U,t}$ holds:

$$\nabla_U(V)(p) = \frac{d}{dt}\Big(\Gamma_{U,t}(V)(p)\Big)\Big|_{t=0}. \tag{3.11}$$

Hence, the ∇_U operator is the derivative of the *backward* parallel transport operator at the point p. Now, if we consider the discrete version of (3.11), i.e., replace ∇_U and $\Gamma_{U,t}$ with their discrete matrix-based representations, $\tilde{\nabla}_U$ and $\tilde{\Gamma}_{U,t}$, respectively, it is easy to check (see supplemental material) that $\tilde{\Gamma}$ given by:

$$\tilde{\Gamma}_{U,t} = \exp(t\tilde{\nabla}_U), \tag{3.12}$$

where exp is the matrix exponentiation, is a solution. By *defining* $\tilde{\Gamma}_{U,t}$ as in (3.12) we maintain the relation between the discrete parallel transport and covariant derivative operators which exists in the continuous case, and gain an easy to implement matrix-based operator.

This observation allows us to compute the parallel transport of vector fields along the flow lines of other vector fields simply by using the matrix exponential of $\tilde{\nabla}_U$. This is somewhat remarkable since computing discrete parallel transport on discrete flow lines directly would require us to numerically integrate the field U to generate the flow lines, and then compute the discrete geodesic curvature of these flow lines for the transport, e.g., as was done in [PS06]. This procedure can be cumbersome, computationally heavy and potentially numerically unstable. For example, the result may not even be a well-defined vector field with multiple vectors in a single face, and some faces not containing any vectors.

On the other hand, when considering the Levi-Civita covariant derivative as an operator acting on vector fields, and representing it as a matrix in a basis, computing parallel transport becomes a standard linear algebra operation involving only matrix exponent and matrix vector multiplication. Note, that parallel transporting a vector field U along its own flow lines is closely related to the numerical scheme known as "semi-Lagrangian advection" in fluid simulation [SY04]. It is therefore possible that our parallel transport matrix operator could be used in such a setup. We leave further investigation of this direction as future work.

In Figure 3.6 we compare the result of parallel transport done using our approach to the ground truth on the sphere. We take a vector field $U = (0, z, -y)$, which rotates around the sphere, and compute $\tilde{\Gamma}_{U,2\pi}(U)$, the parallel transport of U over itself for time $t = 2\pi$, by taking $\exp(2\pi\tilde{\nabla}_U)U$. In this case, the flow lines are constant latitude lines, and the result of the parallel transport has an analytic expression [dCV92, pg. 243].

Figure 3.6 shows the vector field U (left) and the ground truth result (center). Our parallel transport operator uses a fixed number of basis vectors, and the parallel

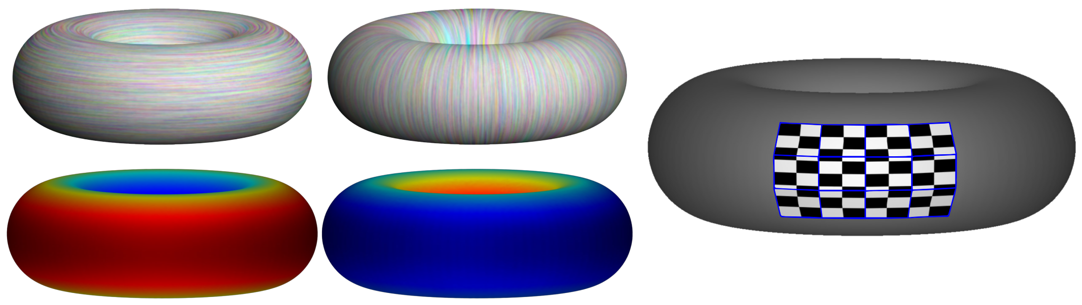


Figure 3.8: Given a vector field U (left), we construct local parameterization by optimizing for V (middle) which minimizes the energy $\int_M \|[U,V]\|^2 + \lambda \int_M \| \langle U,V \rangle \|^2$. The local coordinates are computed by flowing on U and V, resulting in a texture mapped grid marked in blue (right).

transported vector field is non-smooth, therefore we expect the result to improve with an increasing number of basis vectors. This is indeed demonstrated in the graph on the right. The graph shows the error in our computation of the angle of the parallel transported vector field $\tilde{\Gamma}_{U,2\pi}(U)$ with U, with respect to using a growing number of basis vectors N_D. The two figures in the graph show the flow lines and the norm of $\tilde{\Gamma}_{U,2\pi}(U)$ for the largest number of basis functions. Interestingly, the norm of the parallel transported vector field can be flown separately using the flow of the operator for functions $\tilde{D}_U$, which leads to more accurate results. Note that the resulting norm and angles are almost indistinguishable from the ground truth.

We provide further evaluation of our discrete parallel transport. It is known that discrete flow lines of vector fields can in some cases merge or split (e.g., [SZ12, Fig. 4]). In Figure 3.7 we demonstrate the result of parallel translation of U (top row) along U. Notice that although the flow lines of U might split (see the zoomed area, top, right), our result, $\tilde{\Gamma}_{U,2\pi}(U)$, preserves its smooth behavior.

While matrix exponentiation is itself a difficult problem, and the result can be inaccurate for large matrices [MVL03], note that in our case the matrices are relatively small (on the order of 300), as the vector field is represented in a multi-scale basis. In our implementation we used Matlab's *expm* function, and did not encounter any issues. Furthermore, to compute the parallel transport there is no need to compute the full matrix exponent, but only the matrix vector product $\exp(2\pi\tilde{\nabla}_U)U$, for which more stable and efficient methods exist [AMH11]. It is possible that more basis vector fields would be required to represent complex vector fields with a large number of singularities, which are common in parameterization and quadrangular remeshing applications. In such cases, it might be instrumental to investigate our operator in the hat basis, which will lead to a sparse representation, for which methods such as [AMH11] are still applicable. We leave further study in this direction for future work.

3.4.4 The operator $[U,\cdot]$

Operator action: $[U,\cdot](V) = \nabla_U V - \nabla_V U$. Given two vector fields U, V, consider the

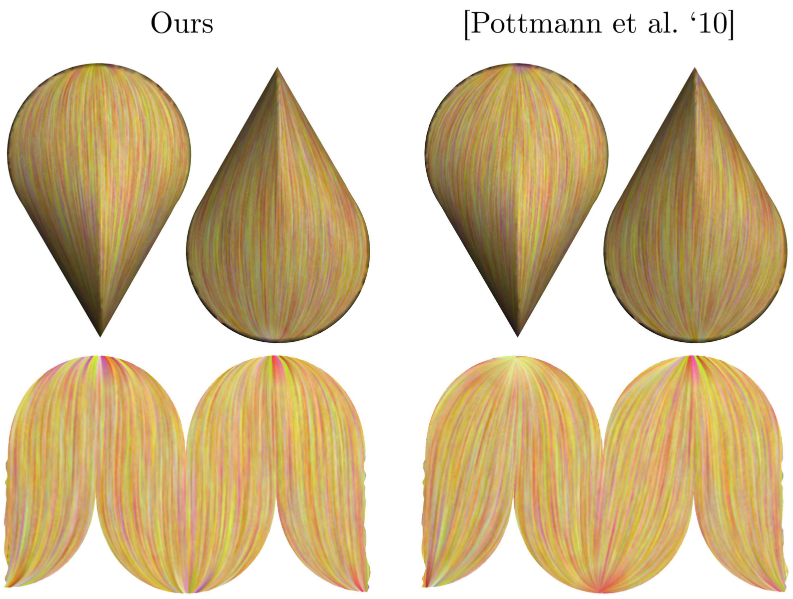


Figure 3.9: Approximate geodesic vector field design. We seek a vector field V which minimizes the energy $\|\nabla V - (\nabla V)^T\|^2$, which yields V that is close to a geodesic vector field (top left). The Oloid model has zero Gaussian curvature everywhere except on the creases, hence when it is flattened the flow lines should yield straight lines (bottom left). Compare with the result of [PHD+10](right). Our results are comparable, while our setup is considerably simpler, and allows for combination of constraints.

problem of constructing local texture coordinates (u, v) such that the iso-v and iso-u lines align with U and V, respectively. Given $p \in M$, one naïve approach would be to flow along U from p and sample the flow line at fixed constant intervals. Then, starting from the resulting sampled points, flow along V and sample again. The union of the sampled points forms a grid. Of course, we could reverse the order and flow first on V and then on U, however, we expect to obtain the same set of sampled points. Formally, this requirement means that the flows of U and V should commute.

The operator $[U, V]$, which is known as the *Lie bracket* or *Lie derivative* of U and V, computes exactly this property—the lack of commutativity of the flows of U and V. Specifically, it is possible to construct a local parameterization as described previously around a point $p \in M$ if and only if $U(p), V(p)$ form a basis for the tangent plane and $[U, V] = 0$ (see e.g., [Kol93, thm. 3.17]).

Using the operators ∇_U and ∇U we can represent $[U, \cdot]$, and use it for vector field design. For example, given a vector field U, we can construct a matrix representation of $[U, \cdot]$, and compute its singular vectors. Since $[U, U] = 0$, U is always the singular vector corresponding to the 0 singular value. However, the next singular vector V minimizes $\int_M \|[U, V]\|^2$, and would give us the best vector to couple with U to get a parameterization. Note, that we can easily add additional terms to the energy, e.g., $\int_M \| \langle U, V \rangle \|^2$, if we want U and V to be orthogonal.

Figure 3.8 demonstrates this for the computation of a local parameterization. We are

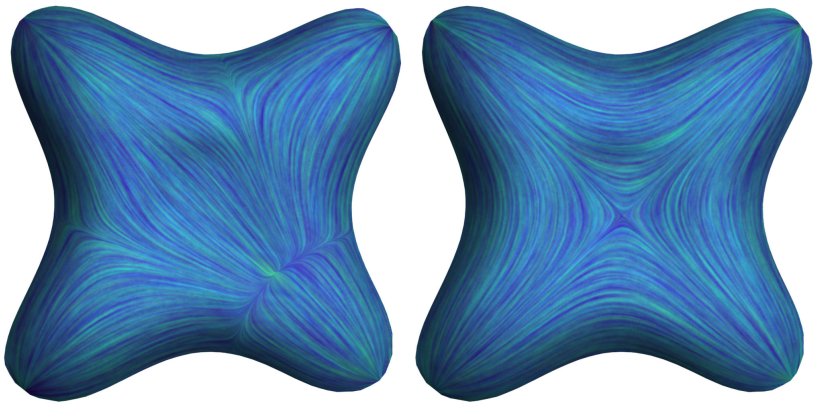


Figure 3.10: Trade-off between as-gradient-as-possible vector field constraints and symmetric vector field constraints, with the symmetry constraints weighted higher in the image on the right.

given U (left), and we minimize the energy $E_{[U,\cdot]}(V) = \int_M \|[U,V]\|^2 + \lambda \int_M \| \langle U,V \rangle \|^2$. The resulting vector field V (middle) together with U is used to build the local coordinates using the flow method described previously. This yields a textured mapped grid (right, shown in blue). Note, that the vector fields U, V are orthogonal but do not have the same norm. Hence, simply rotating U by $\pi/2$ would not have given the same texture coordinates, as the flows would not necessarily commute.

3.5 Applications

Until now we have concentrated on the properties of the various operators we can derive from the Levi-Civita covariant derivative, and provided some proof-of-concept applications for the geometric quantities it allows us to compute. In this section, we first discuss some implementation details and limitations, and then discuss two concrete applications of this machinery: designing tangent vector fields and simulating fluid flow on surfaces.

3.5.1 Implementation details

Choice of basis For our basis for $\tilde{D}_U$, we chose the first N_f eigenvectors of the DEC based 2-form Hodge Laplacian [Hir03]. For $\tilde{\nabla}_U$, $\tilde{\nabla} V$ and all operators acting on vector fields, the basis is given by the first N_D eigenvectors of the DEC based 1-form Hodge Laplacian [FSDH07]. To represent our operators as matrices in the basis, we first convert the 1-forms to piecewise constant vector fields (as in [FSDH07, eq. 4], where we sample at the barycenter of the triangle), then apply the operator to the basis elements, and project the result back onto the basis.

Limitations We define the covariant derivative using the embedding in $\mathbb{R}^3$, however, a classical and fundamental property of the covariant derivative in the continuous case is that it is intrinsic, i.e., it does not depend on this embedding ([Mor01, pg. 181]). In the discrete case, we no longer maintain this property. For rigid deformations, there exists a trade-off between invariance and discretization error. If we use a small number of basis functions, the component functions are smooth, but we lose invariance to rigid transformations. However, the error introduced by the rigid transformation decreases polynomially in the number of basis functions. If, on the other hand we use the full basis in equation (3.9), the operator will be invariant to rigid transformations (see supplemental material for the proof). For isometric deformations the averaging operator $\mathcal{A}$ introduces some error even when using the full basis (as it causes averaging of vectors on faces which undergo different rotations), and for a truncated basis we again have an error which decreases polynomially. Despite this limitation, we believe that the additional simplicity we gained by using the embedding is worthwhile, especially in applications which use a single non-eforming mesh.

3.5.2 Vector field design

As discussed in the previous sections, by using the covariant derivative operators, we can pose various constraints to design tangent vector fields with some prescribed differential properties. Since the operators $\tilde{\nabla}_U$ and $\tilde{\nabla}V$ are linear, each of the optimization problems that we formulate can be solved efficiently by solving a linear system, or by computing a singular value decomposition.

As-Gradient-As-Possible vector fields We first consider minimizing the energy $\|\tilde{\nabla}V - (\tilde{\nabla}V)^T\|^2$, which quantifies the anti-symmetric part of $\tilde{\nabla}V$. As mentioned in Section 3.4.2, this energy will be zero if V is a gradient field. Furthermore, [PHD$^+$10] showed that if additionally the norm of V is constant, then the energy will be zero only if V is a vector field whose flow lines are geodesics, also known as a *geodesic vector field* (GVFs).

While we do not impose the additional constraint, our results on the Oloid model, as shown in Figure 3.9, are comparable to the results of [PHD$^+$10], when weighing the edges according to their mean curvature is not taken into account.

Finally, as we work in the generic framework of functional operators, it is straightforward to combine this energy with additional constraints in a similar manner to [ABCCO13]. For example, we can require the vector field to be symmetric with respect to some symmetry map provided for the surface. By weighing differently the constraints we can allow the user to explore multiple solutions (see Figure 3.10) which may be difficult to achieve using other frameworks.

As-Killing-As-Possible vector fields As mentioned previously, vector fields V for which $\tilde{\nabla}V$ is anti-symmetric are vector fields whose flow preserves the metric, also known as Killing vector fields (KVFs). These are useful for pattern generation, as shown e.g., in [BCBSG10]. By minimizing the energy $\|\tilde{\nabla} V + (\tilde{\nabla} V)^T\|^2$, we can construct vector fields that are as close as possible to KVFs, as we demonstrate in Figure 3.5.

Smooth vector fields As our last design goal we consider the task of computing as smooth as possible vector fields, similarly to what was done in [KCPS13]. One way to characterize such vector fields, is by minimizing the Dirichlet energy $\|\tilde{\nabla} V\|^2$. Figure 3.11 shows an example of two vector fields computed this way, and Figure 3.12 compares the vector field computed using our method (left), with the one computed by the approach of [KCPS13] (right). Note that the resulting vector fields are comparable in terms of smoothness. Compared to the ground truth on the unit sphere, the Dirichlet energy obtained by [KCPS13] is more accurate than ours (1.0017 vs. 0.9515, where the analytic solution is 1), potentially due to energy loss incurred by our projection on the basis of vector fields. Furthermore, the method by [KCPS13] is more general than ours, as it can handle N-RoSy fields in addition to vector fields.

To conclude, while there exist other specialized methods for posing many of the design constraints mentioned here, e.g., [ABCCO13, PHD$^+$10, KCPS13, BCBSG10] our setup is unique in that it is simple, it allows us to pose *all* of these constraints, and generate a large variety of vector fields, since we have direct access to the ∇V and ∇_U operators.

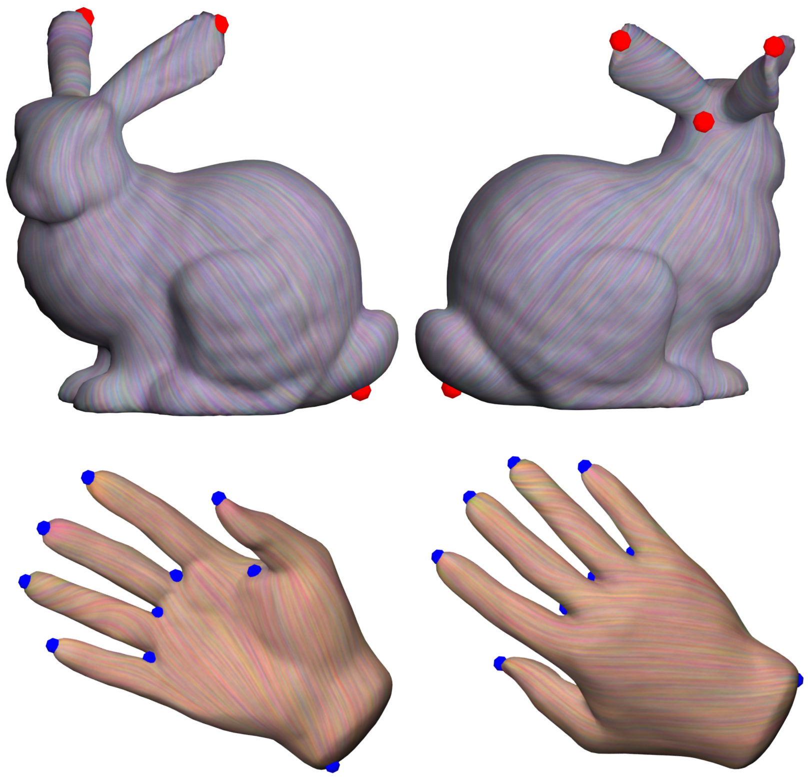


Figure 3.11: Designing smooth vector fields by finding vector fields which minimize the energy $\|\tilde{\nabla} V\|^2$.

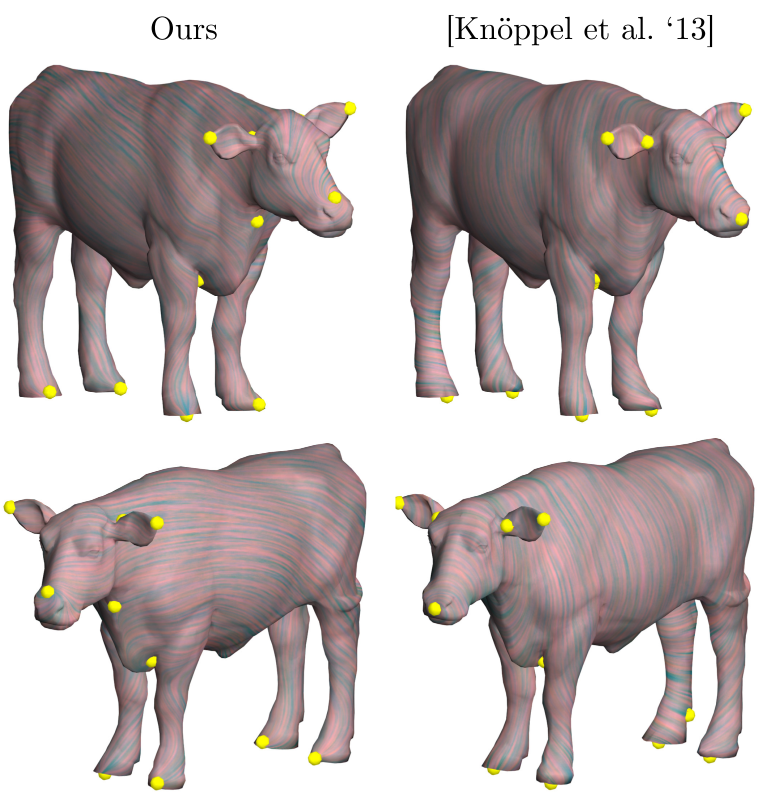


Figure 3.12: Our smooth vector field (left), compared to the one obtained by the method of [KCPS13] (right).

3.5.3 Fluid simulation on surfaces

As our last application, we consider the problem of simulating the behavior of an incompressible flow on a curved surface. A fluid can be described as a time varying velocity field $U(t)$, whose behavior is governed by the Navier–Stokes equations [Tay96]. We discuss here only incompressible (divergence-free) inviscid (viscosity-free) flows, for which the defining equations are known as the Euler equations [Tay96, Eq. 1.10]:

$$\frac{\partial U}{\partial t} = -P_{curl}(\nabla_U U), \tag{3.13}$$

where P_{curl} is the orthogonal projection onto the space of divergence free vector fields.

Using our discrete definition of the covariant derivative, it is straightforward to compute the time-varying velocity $U(t)$ of a flow, given some initial conditions. We implemented a very simple pipeline, using a black box time integrator (Matlab's ode45 [DP80]). One iteration consists of computing $\tilde{\nabla}_U U$ using our operator, followed by projection onto the space divergence free vector fields by solving the Poisson equation $\Delta s = -\omega$, where ω is the vorticity function given by the curl of U, projected onto the space of functions spanned by our basis. The change in U is now given by the gradient of s rotated by $\pi/2$ in each face. We use the operator from [PP03] for computing the

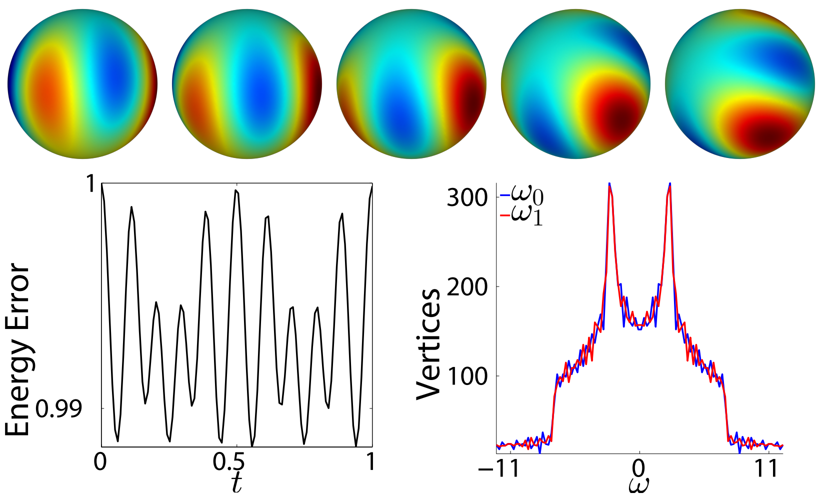


Figure 3.13: (top) A few frames from a periodic solution of the Euler equations on the sphere. Note that the vorticity (color coded) is globally rotated, as expected. See the text for details. (bottom, left) The relative kinetic energy $\int_M \|U(t)\|/\int_M \|U(0)\|$ during the simulation. Note that it is periodic, and remains within 98% of the original energy. (bottom, right) A histogram of the vorticity, for the first (blue) and last (red) frames. Note, that the histogram is preserved as expected.

curl of a vector field.

Despite the simplicity of this approach, we found that in most cases it was enough to simulate interesting flows, for which we know the analytic solution or expected behavior. We demonstrate some example in the accompanying video for the simulation of the flows. We stress that this is a proof-of-concept of the applicability of our operator to fluid simulation on surfaces. We leave further tuning, as well as incorporating a more sophisticated time integrator as future work.

Steady state solutions If U is a Killing vector field, or $U = J\nabla\phi_i$, where ϕ_i is an eigenfunction of the Laplace–Beltrami operator, then $U(t) = U$ is a steady state solution to equation (3.13) (see [MB01, pg. 46, eq. 2.13], and also the supplemental material for a simple proof). Hence, as a sanity check, we compute the average of $\|P_{curl}(\tilde{\nabla}_U U)\|/\|U\|$ for such a vector field U. The result can be considered an indicator to the stability of our method, and was on the order of 10^{-4} for the unit sphere.

Periodic solution on the sphere On the sphere there exists a periodic time varying solution, given by: $U(t) = U_0 + \sum_i a_i(t)J\nabla\phi_i$, where U_0 is a Killing vector field, and ϕ_i are eigenfunctions of the Laplace-Beltrami operator corresponding to *the same* eigenvalue. Furthermore, the curl of the velocity field (its vorticity) $\omega(t)$ is advected by this flow isometrically, namely a pure rotation. We are not aware of a reference for this solution in the literature, and thus provide the proof in the supplemental material. Figure 3.13 (top) shows a few frames from such a simulation on the unit sphere, where we took ϕ_i to be an eigenfunction in the third group of spherical harmonics. We show the color coding

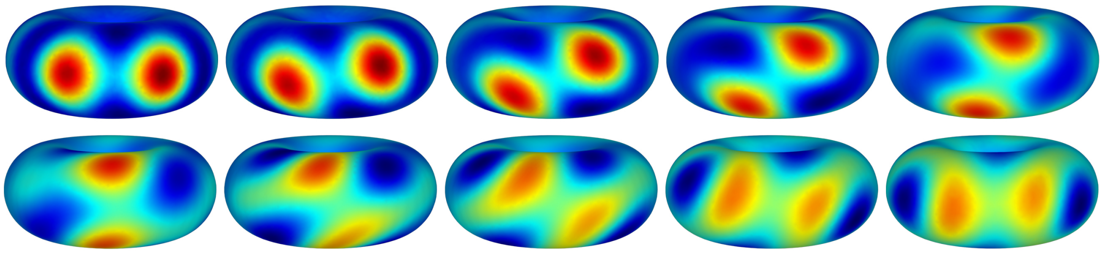


Figure 3.14: A few frames from a solution of the Euler equations on the torus for a co-rotating vortex pair.

of the vorticity function, which is indeed advected as an isometry. Figure 3.13 (bottom right) shows the relative kinetic energy $\int_M \|U(t)\|/\int_M \|U(0)\|$ during the simulation. Note, that the energy itself exhibits periodic behavior, and remains within 98% of the original energy. This indicates the stability of our method, especially since we used a straightforward black box time integrator for all simulations. Finally, Figure 3.13 (bottom left) shows a histogram of the vorticity values, for the first and last frames of the simulation. Note that the histogram remains fixed, as expected.

Co-rotating vortex pair On a plane, a pair of *point vortices* (namely singular points where all the vorticity is concentrated) spinning in the same direction should rotate around each other ([Saf92, pg. 117]). We generate a similar configuration on a torus, where we take the initial vorticity ω_0 to be constant at all vertices except two vertices v_i, v_j, where we take ω_0 to be 1. The constant is set such that $\int \omega_0 = 0$, and then ω is projected onto the span of our basis functions. Figure 3.14 shows a few frames from this simulation (see also the accompanying video). Note that the vortices rotate as expected. One limitation of our method is that it is not circulation preserving, as is for example the method in [ETK$^+$07]. This is visible in the torus simulation, as some of the vorticity is lost due to numerical dissipation. We leave the exploration of efficient methods to overcome this limitation as future work.

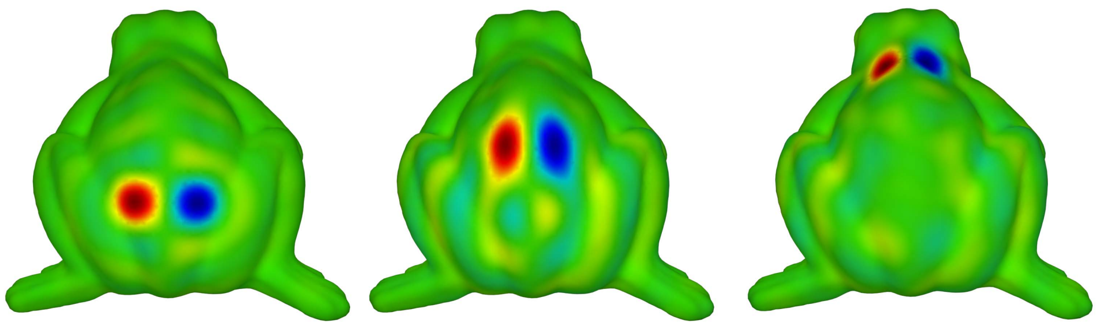


Figure 3.15: Three frames from a fluid flow simulation showing a positive/negative vortex pair on a surface.

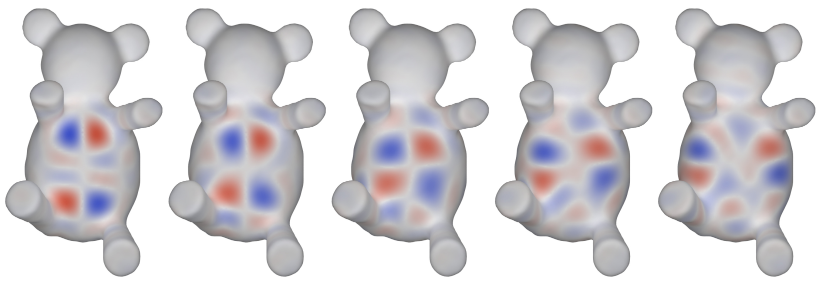


Figure 3.16: A few frames from a solution of the Euler equations on the teddy for two colliding pairs of counter-rotating vortices.

Counter-rotating vortex pair Similarly to the previous experiment we take two point vortices rotating in opposite directions. In the plane such a configuration translates in a straight line ([Saf92, pg. 117]), and a similar behavior is demonstrated on the back of the frog model, in Figure 3.15 and in the accompanying video. The stability of our method is exhibited by the fact that the vortex pair travels intact the whole length of the frog model.

N-vortex structures Here we take a more complicated configuration of vortices. The first includes two pairs of counter-rotating vortices which collide, where the expected behavior is that they continue in a direction orthogonal to the original direction after collision. This is shown in Figure 3.16 and in the accompanying video on the teddy bear model. The second configuration includes 3 co-rotating vortices forming an equilateral triangle, where the flow should rotate the three vortices as a single unit ([New01, pg. 78]). We reproduce this behavior as can be seen in the video. Note that while two of the vortices merge during the process, they separate again at the end of the flow, returning to a configuration similar to the original one.

3.6 Conclusions and Future Work

In this paper, we proposed a novel discretization for the Levi-Civita covariant derivative of vector fields on discrete surfaces, which has various appealing properties. First, it exhibits experimental convergence of the five defining properties of the derivative in the continuous case. Second, it can be represented as a linear operator acting on tangent vector fields, thus allowing us to harness tools from linear algebra, such as matrix exponential, to perform geometric operations which were otherwise harder to achieve, e.g., parallel transport of a vector field along the flow lines of another vector field. Finally, we demonstrated the applicability of our discretization to various geometry processing tasks, such as local parameterization, vector field design and fluid simulation.

We believe there is much more left to explore, as we only gave a taste of the possible applications of our formulation. First, the covariant derivative appears in many PDEs on surfaces, and it is interesting to apply our discretization to additional problems. For example, it is possible to compute the covariant derivative of the normal vector field, thus yielding a novel discretization of the shape operator. Second, our parallel transport approach can potentially be applied to fluid flow simulation, to yield a more stable exponential integrator, and the Killing operator can be used for simulating viscous flow. Furthermore, we would like to investigate additional operators derived from the covariant derivative, such as the connection Laplacian, which can potentially be used for vector field smoothing. To conclude, we believe that our discrete covariant derivative will inspire future work that tackles additional challenges in vector field processing, thus providing a stepping stone towards a complete framework for vector calculus on discrete surfaces.

Chapter 4

Consistent Functional Cross Field Design for Mesh Quadrangulation

We propose a novel technique for computing consistent cross fields on a pair of triangle meshes given an input correspondence, which we use as guiding fields for approximately consistent quadrangulations. Unlike the majority of existing methods our approach does not assume that the meshes share the same connectivity or even have the same number of vertices, and furthermore does not place any restrictions on the topology (genus) of the shapes. Importantly, our method is robust with respect to small perturbations of the given correspondence, as it only relies on the transportation of real-valued functions and thus avoids the costly and error-prone estimation of the map differential. Key to this robustness is a novel formulation, which relies on the previously-proposed notion of *power vectors*, and we show how consistency can be enforced without pre-alignment of local basis frames, in which these power vectors are computed. We demonstrate that using the same formulation we can both compute a quadrangulation that would respect a given *symmetry* on the same shape or a map across a pair of shapes. We provide quantitative and qualitative comparison of our method with several baselines and show that it both provides more accurate results and allows to handle more general cases than existing techniques.

4.1 Introduction

Remeshing of triangle meshes to quad meshes is a fundamental task in geometry processing and related domains with applications in shape modeling, texture synthesis and numerical simulation, to name a few. In many cases, quad remeshing is jointly applied to several shapes and when their correspondences are given, the results are frequently required to be *consistent* with respect to those mappings. For instance, the quad mesh which models an animated character should be aligned to the underlying

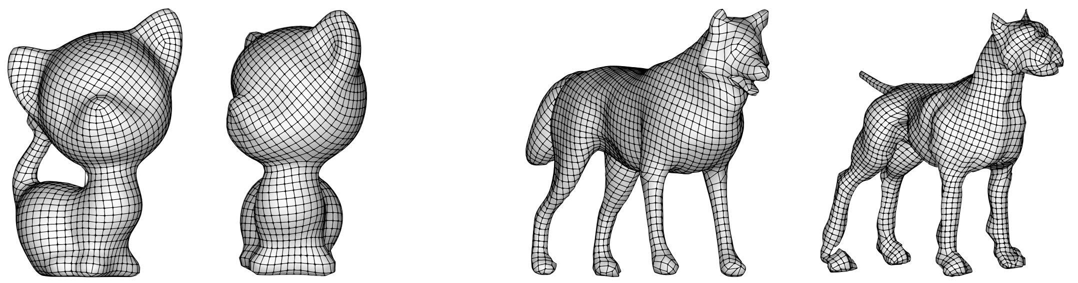


Figure 4.1: Our method computes guiding fields on triangle meshes which respect either the underlying symmetry of a single surface (left) or the related correspondence between a pair of shapes (right), while being able to handle arbitrary topology such as the genus one surface on the left. We use these fields to compute approximately consistent quad meshes with off-the-shelf quadrangulation methods.

deformation modes [MPP$^+$13]. Similarly, on a single shape which exhibits symmetry, a symmetric quadrangular mesh is often preferred [PLPZ12]. The goal of this paper is to propose a robust, unified framework for approximately consistent quad remeshing which is applicable to a single shape or a pair of shapes, without assumptions on the mesh connectivity or shape topology.

To date, there exist several automatic methods for generating quadrangular surfaces from triangle meshes. A common approach, which we will also follow in our paper, uses a guiding field within a parametrization-based method. Namely, remeshing is achieved by designing a smooth *cross field* that accounts for local features, followed by an optimization part which seeks a parametrization whose gradients are aligned with the computed field. Quadrangulation is then performed in the parameter domain, where correct stitching of isolines is maintained along cut graphs [BLP$^+$13]. In this context, our algorithm produces a set of consistent cross fields, which are used as input to previous remeshing machinery [BZK09, EBCK13]. Namely, quadrangulation is computed on each mesh separately, and thus we obtain only *approximate* consistency of quads.

One option for designing smooth cross fields is to encode the angle with respect to a local basis per triangle. The goal is then to minimize the squared difference of these angles along edges, while allowing for integer period jumps [RVLL08]. Unfortunately, the resulting mixed-integer problem is non-convex and achieving a global optimum is challenging in practice. To rectify this, using trigonometric periodic functions on the angles multiplied by 4 allows to avoid integer variables altogether, see e.g., [RVAL09], at the cost of introducing pointwise unit-length constraints. Equivalently, in the complex-valued representation [KCPS13] each cross is encoded using the unique *power vector* obtained by representing the cross directions as complex numbers and taking the 4-th power. Further, dropping the pointwise unit-length constraints yields a convex quadratic problem whose global minimum is attained with a single linear solve.

All of the above commonly-used cross field design approaches depend on a choice of local basis (frame) per triangle. In many cases, this basis dependency does not pose any practical challenges. However, when consistency is needed, computing transformations which align these basis vectors across shapes is essential in order to faithfully compare the measured angles. For instance, for meshes with different connectivities, a triangle is not necessarily mapped to a single triangle, and thus several basis vectors must be taken into account. One of the main advantages of our approach is that we formulate the consistency constraints in terms which are *invariant* to the local basis. This novel change greatly simplifies the problem since the basis vectors can be chosen arbitrarily on each shape.

To enforce consistency of cross fields between two shapes, scalars or vectors need to be mapped and compared using the input map. Therefore, the quality of the map and map differential are of crucial importance to achieve good quadrangulation results. However, computing acceptable approximations of these objects is a hard problem in itself, making the entire remeshing pipeline highly dependent and potentially sensitive to high frequency noise in the given correspondences. In our framework, we relax this constraint by assuming that only functional correspondences are given. Functional maps [OBCS+12] provide robust means to encode mappings between surfaces by putting in correspondence their function spaces. We pose the consistency requirements solely in the functional language, which allows us to apply our machinery to any shapes for which functional mappings are available. This includes both functional correspondences obtained via a pull-back with respect to a given pointwise map (thus represented in the full basis), and functional maps computed automatically and represented in a reduced spectral basis. An advantage of our formulation is that it allows a separation of the involved components. Namely, the smoothness and alignment constraints are high-dimensional but sparse, whereas the consistency terms are either high-dimensional and sparse or low-dimensional and dense. In both cases, this separation leads to a structured Hessian of the minimized energy, allowing us to employ efficient optimization techniques.

In this paper, we suggest an effective methodology to design consistent cross fields for the purpose of compatible mesh quadrangulation. Thanks to our functional approach, the obtained machinery is similar regardless of whether a single symmetric shape or two shapes are used. Moreover, unlike most previous techniques, such as [MPP+13], we place no restriction on the connectivity of the triangle meshes, and further can handle shapes with arbitrary topology. To summarize, our main contributions include: the invariance of the proposed method with respect to a local basis, the ability to design fine details separately on each mesh while requiring consistency only in a low-dimensional space, and the ability to handle arbitrary meshes. We also demonstrate that our method is simple and robust, in large part due to its ability to avoid the potentially difficult and error-prone step of computing a map differential, and at the same time scalable, as it can accommodate functional correspondences represented in a reduced basis. To achieve these goals, we formulate consistent cross field design via a simple, global quadratic energy minimization problem which we efficiently solve by evaluating the action of the

Hessian on a general vector.

4.1.1 Related Work

Quadrangular remeshing is a challenging problem, and in the last few years there has been a surge of research in this direction. We refer the reader to recent reviews for a general overview of quadrangulation methods [BLP+13] and direction field design [VCD+16] on a single shape, and focus our literature review on *joint* design of cross fields and quadrangular meshes.

Perhaps closest to our approach is the Functional Vector Field work [ABCCO13] where joint design of smooth *vector fields* is formulated in the functional framework. The optimization there is convex, yet the vector fields need to be represented in a low dimensional basis, which is computed using the eigenfunctions of the Hodge Laplacian. We generalize this approach to cross fields, by representing vector fields in a local frame per face, thus avoiding the need for a low dimensional basis, and formulating a functional consistency constraint which is invariant to this choice of frame.

One of the first approaches to joint quad mesh design was presented by Yao et al. [YCJL09]. There, the user sketched compatible skeletons which were used to generate compatible base meshes, from which compatible quadrangulations were extracted. Unfortunately, this approach requires extensive manual input, and affords little control on the quality and smoothness of the resulting quads. Later approaches to interactive design of quadrangular meshes were based on learning quad templates from examples [TIN+11, MTP+15], with the goal of computing quad meshes which are *approximately* consistent with quadrangular meshes designed by artists. These methods are local, as they rely on segmenting the input into disk-like patches, which may yield sub-optimal results.

The more general problem of computing consistent or approximately consistent quad meshes jointly on a pair or a collection of shapes with respect to an input correspondence has only been addressed by a few methods so far. Given a collection of shapes in correspondence, Meng et al. [MH16] co-extract compatible feature lines, and then design cross fields independently for each shape, using the feature lines as alignment constraints. In the following step, they co-design a compatible cut graph, and then align all the shapes in a common parameter domain. However, since the cross fields are designed independently, the correspondence of feature-less regions is not taken into account. Alternatively, Marcias et al. [MPP+13] take as input a set of shapes with compatible triangulations, use the principal directions of the deformation gradient as alignment constraints for designing a single cross field on one of the shapes, extract from it a quad mesh and then propagate it to the rest of the sequence. The case where the connectivities of the triangle meshes are different, is not handled in that work.

These two approaches highlight the main challenge of cross field-guided compatible quad remeshing: transporting the cross fields across meshes. The first approach avoids this issue by designing each cross field separately, whereas the second approach uses a high quality triangle-to-triangle map to transport vectorial information. In an attempt to address this challenge in the context of symmetry-aware cross field design, Panozzo et al. [PLPZ12] use a *fuzzy* symmetry map, which averages the contribution of the transported cross field from the neighborhood of a few triangles. While achieving excellent results in some cases, this approach has some important limitations. First, it requires the computation of a high-quality symmetry map, especially tailored to their approach. As is shown in [PLPZ12], when using other symmetry maps, the results can be suboptimal. This is a practical limitation, as their proposed symmetry computation method does not handle, for example, high genus intrinsic symmetries. Second, their algorithm uses hard constraints to align the cross field with the symmetry line. This constraint prevents singularities from appearing on the symmetry line, unnecessarily limiting the space of feasible cross fields. Furthermore, a high quality symmetry line, which might be challenging to compute, is required for this constraint. Finally, the formulation provided there is not in the form of a global optimization problem, and the optimality of the solution under their proposed error metric is not guaranteed.

Our method overcomes the limitations exhibited by previous approaches, as it is robust to the input map, can be applied to meshes with different triangulations and to symmetric meshes without requiring the computation of the symmetry line, and is formulated as a convex quadratic optimization problem whose global optimum is efficient to compute.

4.2 Overview and Background

Given a shape along with a self-map (e.g., associated with a symmetry) or a pair of shapes with maps between them our goal is to produce quadrangulations that would be consistent with respect to the input map. To this end, we first design consistent *cross fields*, and then use existing methods to extract quadrangulations from them. Thus, the main focus of our work is to devise a robust method for consistent cross-field design, and we present quad-remeshing as the main target application, among many possible others.

Our main contribution is a novel cross field *consistency energy* (Section 4.3.3) which we combine with existing smoothness and curvature directions alignment energies into a global convex quadratic optimization problem. For the method to be widely applicable, the consistency energy should be robust to imperfections in the input map. Thus, instead of transporting cross fields with estimated map differentials, which are often noisy for imperfect input maps, we formulate consistency in terms of *scalar functions* and use *composition* with the input map for transport.

Cross fields can be represented discretely in a few ways (see e.g., [VCD$^+$16, Sec. 5]), which affects the way the difference between two crosses is measured. Since each cross is composed of a set of 4 indistinguishable vectors, any comparison between two crosses should be invariant to reordering of the sets. A common way to handle this is to define a local basis, represent the cross as a complex number in this basis, and then compute its *power vector*, namely the 4-th complex power of this vector [KCPS13, Fig. 5]. As the power vector is unique for each cross, power vectors can be compared directly in order to compare crosses.

This link between cross fields and their corresponding power vector fields hints at the possibility to leverage techniques used for robust vector field transport [ABCCO13] for cross field transport. However, one caveat is that smoothness of power vector fields is measured differently than smoothness of vector fields, therefore a low dimensional basis of smooth vector fields can no longer be used for the representation. Furthermore, the power vector fields are dependent on the local basis in which they were computed. We show how this dependence can be eliminated, and use this insight to formulate the consistency energy.

We efficiently solve the resulting optimization problem, then normalize the output power fields and convert them to their associated cross fields. To extract the quadrangulations, we feed the resulting cross fields to a Mixed Integer Quadrangulation (MIQ) [BZK09] implementation that computes parametrization functions whose gradients align with the directions of the cross fields. Finally, the meshes and the parametrizations are given as input to a Quad Extraction (QEx) [EBCK13] implementation which robustly extracts the quad meshes associated with the parametrizations. We use the implementations of MIQ and QEx directly and thus we omit further discussion on these methods, and refer the interested reader to the respective papers for additional information. Technical details such as the actual code packages and the parameters we used are described in Section 4.5.

We emphasize that we optimize for a consistent cross field and thus the quadrangulation is only approximately compatible in practice. Indeed, we concentrate specifically on the design of consistent cross-fields and our method is not intended to provide guarantees about the quality of the quad final meshes. Nevertheless, in practice, we achieve highly-consistent quad meshes, as can be seen in Fig. 4.2. The bunny model exhibits an intrinsic symmetry (the head is rotated), which makes the stationary line non-trivial, and yet, our method produces a visually appealing symmetric quad mesh.

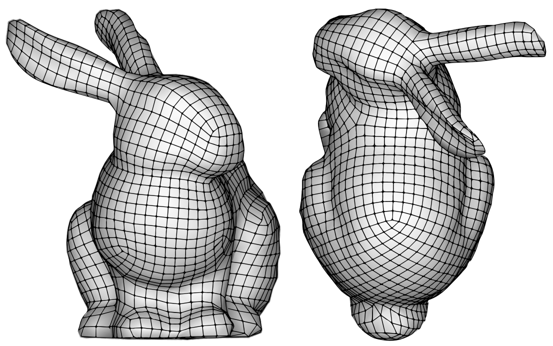


Figure 4.2: A quad mesh generated with our method using $k = 100$ eigenfunctions on the intrinsically symmetric bunny model.

In the following two sections, we describe the design of consistent cross-fields first with respect to a symmetry on a single mesh (Section 4.3) and then with respect to a pair of meshes with a (functional) map between them (Section 4.4). We then present results obtained using our approach, by focusing on joint quadrangulation as a principal potential application.

4.3 Self-consistent Cross Field Design

We assume to be given an orientable manifold triangle mesh M with vertex set $\mathcal{V}$, edge set $\mathcal{E}$ and face set $\mathcal{F}$. Vector fields as well as cross fields are piecewise-constant on faces in our setup. Namely, per triangle, a vector is encoded using 2 numbers with respect to a local basis $(b, b^{\perp})$ and thus both the cross field x and its power field y can be expressed as vectors in $\mathbb{R}^{2|\mathcal{F}|}$, where $|\mathcal{F}|$ is the number of faces. To compute the power vector corresponding to a cross in a given face, we take an arbitrary vector of the given 4, compute the angle θ it makes with b, and the resulting power vector in this face is the unit length vector in the 4θ direction. Next, we describe the energy terms we use to design the power field y.

4.3.1 Smoothness

Following previous work, we use *Dirichlet's energy* which is defined via the *covariant derivative* of power fields, to quantify how much y changes across the edges of the mesh. Integrating the squared norm of this measure over the surface leads to the following smoothness energy term:

$$E_s = \frac{1}{2}\| \operatorname{grad}_p y\|_M^2 = \frac{1}{2} y^T \operatorname{grad}_p^T G_{\mathcal{E}} \operatorname{grad}_p y \ , \tag{4.1}$$

where $G_{\mathcal{E}} \in \mathbb{R}^{2|\mathcal{E}|\times 2|\mathcal{E}|}$ is a diagonal matrix which encodes the barycentric mass of edges, and $\operatorname{grad}_p \in \mathbb{R}^{2|\mathcal{E}|\times 2|\mathcal{F}|}$ is the covariant derivative, also referred to as the discrete Levi-Civita connection, modified to account for taking the 4-th power, whose construction is given in e.g., [DVPSH14, Eq. (3)].

4.3.2 Alignment to input directions

In many situations, the designed field will be required to align with certain directions, where the principal curvature directions are a natural choice for quad remeshing. Given an input cross field, we compute its associated power vector field $w \in \mathbb{R}^{2|\mathcal{F}|}$, and arrive at the straightforward alignment term:

$$E_l = \frac{1}{2}\|S(y-w)\|_M^2 = \frac{1}{2}(y-w)^T S^T G_{\mathcal{F}}\, S(y-w) \ , \tag{4.2}$$

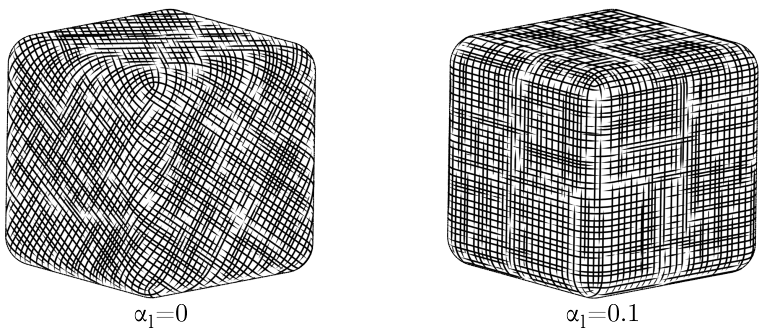


Figure 4.3: Optimizing for smooth cross fields which are not aligned (left) or aligned (right) to curvature directions produces equally smooth cross fields, where the right field better respects the underlying geometry.

where $G_{\mathcal{F}} \in \mathbb{R}^{2|\mathcal{F}| \times 2|\mathcal{F}|}$ is the diagonal mass matrix for the faces, and $S \in \mathbb{R}^{2|\mathcal{F}| \times 2|\mathcal{F}|}$ is a diagonal matrix of weights given by the user, indicating the relative importance of the alignment constraints. For instance, when the principal curvature directions are used for alignment, S is usually a measure of the anisotropy of the curvature. In Fig. 4.3, we show that without alignment constraints (left), the smoothest cross field may not necessarily follow the curvature directions, whereas even a modest alignment requirement yields a smooth field which is parallel to the cube's edges (right).

4.3.3 Consistency

Vector fields. A vector field is *consistent* with respect to a self-map $\phi : M \to M$, if for any point $q \in M$, the following equation holds:

$$\mathrm{d}\phi(v(q)) = v(\phi(q)) \ . \tag{4.3}$$

Namely, points which match under the mapping should be equipped with identical vectors, via the transformation of the tangent spaces given by the *map differential* $\mathrm{d}\phi$. Two major challenges are related to enforcing the above equation in practice. Firstly, when ϕ is approximate, enforcing Eq. (4.3) to too many outliers may erroneously affect the result. Secondly, computing the map differential $\mathrm{d}\phi$ is a non-trivial and potentially unstable task, especially in the presence of noisy maps. See Section 4.6 for further details and comparisons.

Instead of working directly with Eq. (4.3), vector fields can also be seen as *derivations* [Mor01, pg. 37]. That is, we can apply Eq. (4.3) to a real-valued function $f : M \to \mathbb{R}$ and obtain the following consistency constraint:

$$v(f) \circ \phi = v(f \circ \phi) \ , \tag{4.4}$$

where $v(f) = \langle v, \operatorname{grad} f \rangle$ is the pointwise directional derivative, and $\circ$ denotes composition with a map. It is a well-known direct consequence of the chain rule [Mor01, Eq. (1.14)] that for a fixed tangent vector field v, Eq. (4.4) is satisfied for all smooth functions f, if and only Eq. (4.3) holds. Note that for a fixed f and ϕ, Eq. (4.4) is *linear* in v, meaning that it can be optimized, for example by solving a linear least squares system.

Power vector fields. Cross fields are only smooth up to rotation by integer multiples of $\pi/2$, and thus Eq. (4.4) cannot be applied directly without incorporating integer constraints. With a local basis $(b, b^{\perp})$ per face, the four vectors of the cross can be mapped to a single vector per face, by taking the 4-th complex power of the vector with respect to the basis b. This *power vector field* is smooth, if the change of basis between neighboring faces is taken into account. However, power vector fields are not canonical, as they depend on the choice of local basis $(b, b^{\perp})$ per face. Thus, comparing two power vectors at a given point is only meaningful if they were defined *with respect to the same local basis*. In other words, it is not enough for the power vector field to satisfy Eq. (4.3) or, equivalently Eq. (4.4) for all f, to guarantee a consistent cross field. Instead, the transformation between the *basis vectors* (at every pair of points q and $\phi(q)$) should also be accounted for.

To overcome this difficulty, one can try to treat the basis b as a smooth vector field, and design it simultaneously with the power vector field y, such that b also fulfills the consistency condition in Eq. (4.3). However, this yields an optimization problem which is twice as large in the number of variables and constraints, and which is non-linear because of the dependency between the basis vector field and the power field y. Instead, we show how to remove the basis dependency altogether using the following observation:

Lemma 4.3.1. Given a cross field x and an arbitrary point $q \in M$, we compute the associated power vectors y_1 and y_2 at q using two different basis vectors b_1 and b_2, respectively. Then, for any real-valued function f, the following relation holds

$$\langle y_1, (\operatorname{grad} f)_{1,p} \rangle = \langle y_2, (\operatorname{grad} f)_{2,p} \rangle \ ,$$

where $(\operatorname{grad} f)_{i,p}$ is the power vector of $(\operatorname{grad} f)$ at q in the basis b_i.

In other words, the inner product of two power vectors defined with respect to the same local basis is invariant to the choice of basis. Intuitively, the inner product between two power vectors encodes the angle between *the underlying crosses* and is thus basis independent. See Appendix C.1 for the straightforward proof. Thus, in the case of power fields, we modify Eq. (4.4) and consider instead:

$$\langle y, (\operatorname{grad} f)_p \rangle \circ \phi = \langle y, (\operatorname{grad}(f \circ \phi))_p \rangle \ . \tag{4.5}$$

Note that the two sides of the equation are computed at different tangent spaces, of the symmetric points q and $\phi(q)$, with respect to arbitrary basis vectors. The comparison between these values is meaningful due to the proposition above.

Intuitively, for the constraint to hold, the function $\langle y, (\operatorname{grad} f)_p \rangle$ should be symmetric under the map ϕ. Fig. 4.4 visualizes this function on the surface during our iterative optimization process, described below. As the optimization proceeds the function becomes more symmetric, and thus the consistency error is reduced. We note that unlike Eq. (4.4), Eq. (4.5) is *non-linear* in the function f. However, both of these equations are linear in y, which allows us to use this equation directly to enforce consistency of a cross field with respect to a given map ϕ, with an arbitrary local basis.

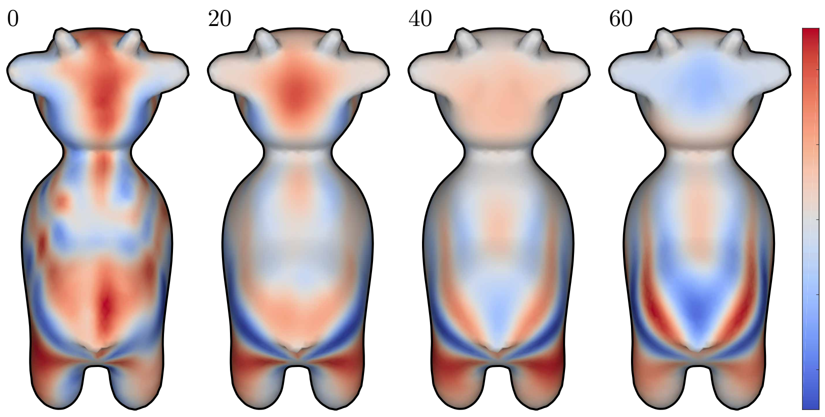


Figure 4.4: We iteratively optimize for a consistent cross field whose action on a fixed function produces a symmetric result. In this example, y's action at iteration 0 is not highly symmetric, but it quickly improves during the iterations $20, 40$ and 60.

Discretization. In practice we work with functions represented in a chosen functional basis B, with the two most commonly used bases in our setting being either the indicator (hat) basis at the vertices, or a multiscale low-dimensional basis $B \in \mathbb{R}^{|\mathcal{V}| \times k}$ such as the Laplace–Beltrami eigenfunctions. In that case, functions are represented as vectors of size k, where $k < 300$ in all our experiments. We are given as input a functional map, which maps real-valued functions represented in the basis B to other such functions, and we represent it as a matrix C of size $k \times k$.

The operator grad is the standard gradient operator for functions in the piecewise linear hat basis, and thus we use the transformations $\tilde{f} = Bf$ and $f = B^{+}\tilde{f}$ between functions $f \in \mathbb{R}^{k}$ in the basis B and functions $\tilde{f} \in \mathbb{R}^{|\mathcal{V}|}$ in the hat basis, where B^{+} is the pseudo-inverse of B. We further use the matrix $I_{\mathcal{V}}^{\mathcal{F}} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{F}|}$ to interpolate face-wise values to vertex-wise values.

The function $\langle y, (\operatorname{grad} \tilde{f})_p \rangle \in \mathbb{R}^{|\mathcal{F}|}$ is linear in y, and thus its computation can be encoded as a matrix-vector product. We interpolate the face-wise values of the inner product to the vertices and define $\tilde{D}(\tilde{f}) \in \mathbb{R}^{|\mathcal{V}| \times 2|\mathcal{F}|}$ such that $\tilde{D}(\tilde{f}) \cdot y = I_{\mathcal{V}}^{\mathcal{F}} \langle y, (\operatorname{grad} \tilde{f})_p \rangle$. To use functions f given in a reduced basis B we define the reduced operator $D(f) \in \mathbb{R}^{k \times 2|\mathcal{F}|}$ as: $D(f) = B^{+} \tilde{D}(Bf)$.

Finally, using these operators we enforce our consistency rule (4.5) on a subset of m

functions f_i given in the basis B, and arrive at the following novel constraint:

$$E_c = \frac{1}{2} \sum_{i=1}^{m} \| C\, D(f_i) \cdot y - D(C \cdot f_i) \cdot y \|^2 \ . \tag{4.6}$$

Discussion. While in the above formulation we describe the use of functional maps encoded in a reduced basis of size k, we stress that our framework can be easily applied in the particular case when a dense (vertex to vertex) or precise (vertex to point on face) map is known. In this setting, B is the identity matrix of size $|V|$ and $C \in \mathbb{R}^{|V| \times |V|}$ is a *sparse* matrix encoding the dense correspondence or using three values per row for the precise (inside the face) map. Similarly, $D(f) = \tilde{D}(f)$. For all the figures we provide in this paper, we show the results obtained using the *reduced* functional map, unless noted otherwise (see e.g., Figs. 4.10a, 4.10b and 4.14).

4.3.4 Energy minimization

We combine the above design constraints into a single minimization problem. Linear blending of energy terms is controlled via two parameters α_c and α_l, both in the range $[0, 1]$. Overall, the power field that we use to generate the quad mesh with is the minimizer of the problem:

$$\arg\min_{y} \, (1 - \alpha_l)[(1 - \alpha_c)\, E_s + \alpha_c\, E_c] + \alpha_l\, E_l \ . \tag{4.7}$$

Notice that the above problem is *quadratic* in y since we omit the unit length constraint on y as was done in [KCPS13]. Consequently, we have a linear gradient and a constant Hessian. In Appendix C.2, we discuss how to efficiently solve the above problem using a standard optimization toolbox. In particular, we show that although the Hessian of our energy is large and dense, its product with a given vector can be computed efficiently by decomposing it into a large and sparse and small and dense parts. In Fig. 4.5, we present a few examples of models with intrinsic symmetries and the quadrangulation

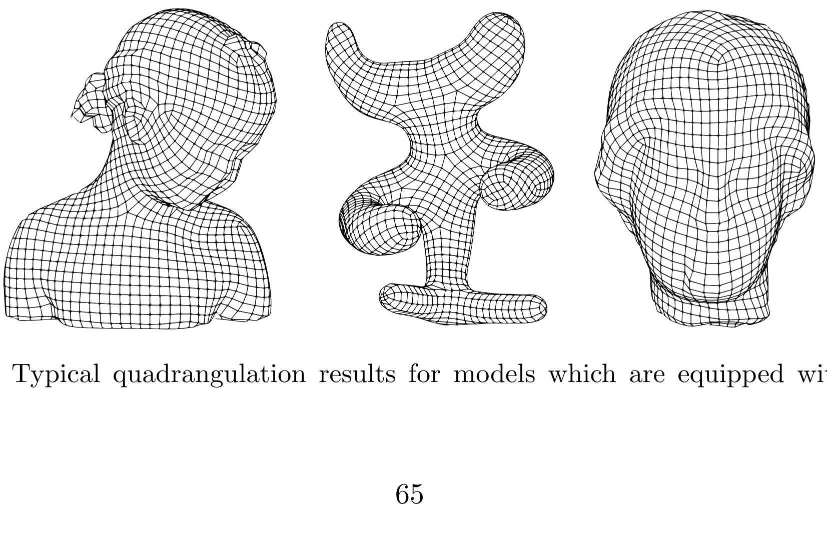


Figure 4.5: Typical quadrangulation results for models which are equipped with intrinsic symmetries.

result we obtained using only $k = 100$ eigenfunctions in all three cases.

4.3.5 Relation to other functional approaches.

Functional maps [OBCS+12]. Given a map ϕ, the linear operator that maps any real-valued function f to the pull-back $f \circ \phi$ was denoted by Ovsjanikov et al. [OBCS+12] as the *functional map* representation of ϕ. That work, and follow up works [OCB+16], have observed that it is often easier to frame problems using the functional map rather than the pointwise map. Our work follows this theme, as in order to enforce Eq. (4.5), it is only necessary to have access to the functional map $f \circ \phi$. As we show below, this greatly simplifies the computations and at the same time extends the applicability of the resulting algorithm. This is because our formulation avoids not only the estimation of the map differential, but does not require even the knowledge of a precise point-to-point map, the estimation of which from a functional map can be challenging [RMC15], and is required by some state-of-the-art mapping methods [LRB+16, OCB+16].

Functional vector fields [ABCCO13]. Joint design of smooth vector fields has been done in [ABCCO13] by leveraging Eq. (4.4). There, to avoid working with a local basis per face, vector fields have been represented as functional operators, namely matrices, and Eq. (4.4) was implemented as a commutativity constraint. However, to reconstruct the face-wise vector field from its functional representation, and to enforce smoothness on the resulting vector field, Azencot et al. worked in a *low dimensional basis* of tangent vector fields computed as the eigenfunctions of the Hodge Laplacian. To generalize their approach to cross fields, one would need to modify the basis to be able to represent smooth cross fields, and in addition modify the consistency term to take into account the local basis in which the cross-field was computed. We avoid the first issue by working directly with the face-based vector fields as the variables, and the second issue by working with a basis-invariant formulation. Note, that simply designing a power vector field using the functional vector field machinery would not yield the smoothest cross-field, as the singularities that arise are different (see Fig. 4.6, and also [RLL+06, Fig. 8]).

4.4 Consistent Field Design on Two Shapes

To extend the model we proposed in Section 4.3, we consider the following scenario. Given a pair of triangle meshes M_1 and M_2, possibly with *different* vertex and face sets, our pipeline requires as input the functional maps C_{12} and C_{21}, which map functions on M_1 to functions on M_2 and vice versa. One of the advantages of our approach to consistent cross field design, is that it naturally generalizes from the case of a single shape to a pair of shapes. Indeed, the new smoothness and alignment components are extremely similar to the former case, whereas the main change is in the consistency term

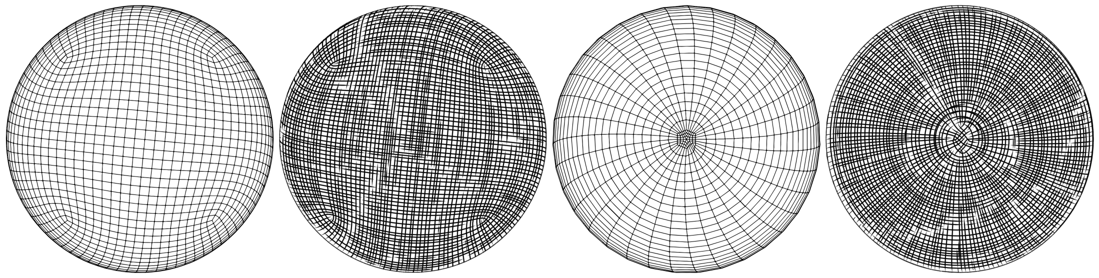


Figure 4.6: Using the method of Azencot et al. [ABCCO13] for designing power fields would not yield the smoothest cross fields, but would produce fields with different singularity structures. For comparison, we show the quad mesh computed from the smoothest cross field (left and middle left), and from the smoothest vector field treated as a power field with a smooth local basis (middle right and right).

where we now optimize for two power fields instead of one. Our objective is to optimize for fields y_1 on M_1 and y_2 on M_2 such that the following energy terms are minimized. For example, we show in Fig. 4.7 the different results we obtain with (bottom) and without (top) our consistency condition.

In this setting, given two shapes, we simply add together the smoothness and alignment constraints for each y_i. Formally,

$$E_s = \frac{1}{2}\| \operatorname{grad}_p y_1\|^2_{M_1} + \frac{1}{2}\| \operatorname{grad}_p y_2\|^2_{M_2} \ , \tag{4.8}$$

$$E_l = \frac{1}{2}\|S_1(y_1 - w_1)\|^2_{M_1} + \frac{1}{2}\|S_2(y_2 - w_2)\|^2_{M_2} \ , \tag{4.9}$$

where w_i and S_i are typically the curvature directions and their weights on M_i. To avoid clutter of notation, we uniformly use grad_p for the covariant derivatives on both of the meshes, in cases where no confusion might arise. Notice that while being stacked jointly, Eqs. (4.8) and (4.9) are independent of the relations between M_1 and M_2, i.e., the associated Hessians are *block-diagonal*.

To develop the consistency rule for a pair of shapes, we recall the geometric meaning of our constraint on a single mesh. Namely, in the former case we required that for a given function, taking the appropriate inner product with y and the functional map (pull-back) should commute (Eq. (4.5)). For two shapes, we have a similar scenario, being different in that the mapped versions are on the other mesh, where before we had only one surface. In addition, to avoid favoring a particular mapping direction, we symmetrize our constraint by adding an analogous term in the other direction, and thus we need both $C_{12} \in \mathbb{R}^{k_2 \times k_1}$ and $C_{21} \in \mathbb{R}^{k_1 \times k_2}$. We obtain the following consistency condition:

$$\begin{aligned} E_c = \frac{1}{2}\sum_{i=1}^{m} \|C_{21}\, D(f_{i,2}) \cdot y_2 - D(C_{21} \cdot f_{i,2}) \cdot y_1\|^2 \\ + \frac{1}{2}\sum_{i=1}^{m} \|C_{12}\, D(f_{i,1}) \cdot y_1 - D(C_{12} \cdot f_{i,1}) \cdot y_2\|^2 \ , \end{aligned} \tag{4.10}$$

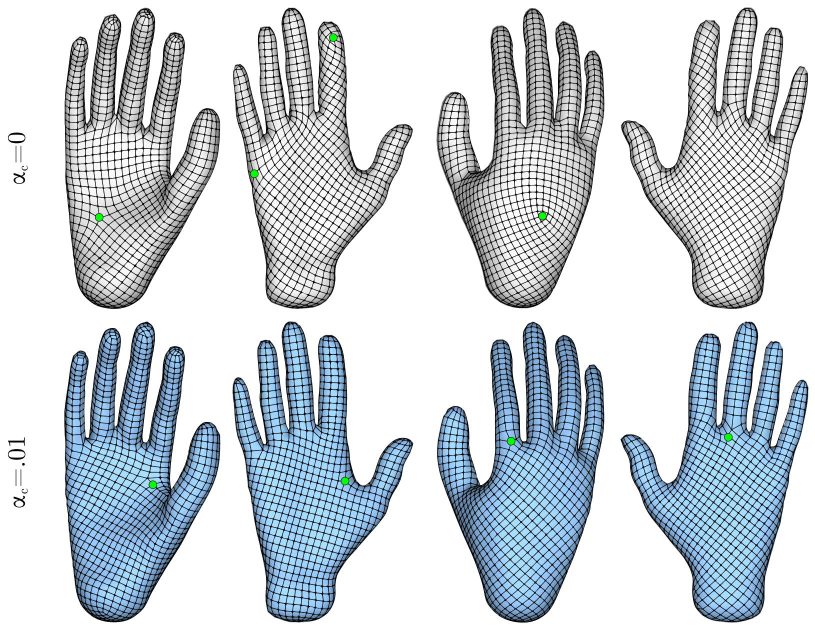


Figure 4.7: Curvature information can sometimes lead to quasi-consistent results even without consistency $\alpha_c = 0$ (top row). However, we show that facilitating our compatibility condition $\alpha_c = .01$ with the precise mapping from BIM represented using a functional map of size $k = 50$, produces more consistent quad meshes (bottom row).

where $\{f_{i,1}\}$ and $\{f_{i,2}\}$ are sets of m functions chosen arbitrarily on M_1 and M_2, respectively. Again, we remind that our operators are given in some pre-calculated functional basis. For instance, C_{12} maps a function f_1 represented in the basis $B_1 \in \mathbb{R}^{|\mathcal{V}_1| \times k_1}$ to a function $f_2 = C_{12} \cdot f_1$ given in the basis $B_2 \in \mathbb{R}^{|\mathcal{V}_2| \times k_2}$.

Finally, we gather the above energy terms into a single problem, where we optimize for power fields y_1 and y_2. Notice that, as in the case of a single shape, while y_i are encoded in a specific *local* basis in every tangent plane of every point on shape $i \in \{1, 2\}$, our formulation is invariant to the choice of these bases. We employ the same weighting parameters as before, and arrive at our final optimization energy:

$$\arg\min_{y_1, y_2} (1 - \alpha_l)[(1 - \alpha_c)\, E_s + \alpha_c\, E_c] + \alpha_l\, E_l \ . \tag{4.11}$$

Discussion. We point out that our approach can be extended to the case of shape *collections* in a straightforward way. That is, smoothness and alignment constraints are simply stacked as in Eqs. (4.8) and (4.9), and consistency could be achieved by either enforcing Eq. (4.10) between each of the shapes and a template mesh or by exhaustively enforcing it between all possible pairs. One challenge involved in taking this approach is that it might be not practical to solve the obtained problem when the collection is large. As we were focused on developing the cases of a single shape and a pair of shapes in this paper, we leave further investigation of consistent quadrangulation of shape sets for future work.

4.5 Implementation Details

We implemented our method using MATLAB and tested it on a Intel Xeon 3.20GHz processor with 32GB RAM. The optimization problems we consider in Eqs. (4.7) and (4.11) could be re-arranged as standard quadratic programming problems, composed of sparse components, E_s and E_l, and a dense element, E_c (see Appendix C.2). Thus, we were able to use MATLAB's `quadprog` optimization tool with a user-handle to compute Hv, where H is the Hessian and v is a vector, in order to avoid storing the full dense Hessian. The initial solution was the smoothest power field ($\alpha_c = 0$ and $\alpha_l = 0$) in all our tests. For problems with 5k/11k/20k/27k/50k vertices, the power field design part converges in 5/7/13/34/90 seconds with a point to point mapping or in 10/18/68/90/169 seconds using a reduced basis of size $k = 100$, respectively.

For the functional basis B, we took the first k eigenfunctions ordered by their eigenvalues, starting with the smallest one. Similarly, we use the first $m = \min(k_1 - 1, k_2 - 1)$ eigenfunctions excluding the constant one for the test functions $\{f_i\}$ which appear in Eqs. (4.6) and (4.10). In practice, we test against the power of the gradient $(\operatorname{grad} f_i)_p$ weighted by λ_i^{-1}, where λ_i is the associated eigenvalue. To generate point to point mappings, we used implementations of Blended Intrinsic Maps (BIM) [KLF11] without landmark correspondences, seamless surface mappings [APL15] and the descriptor based pipeline from [OBCS$^+$12] with landmark constraints. For the computation of functional maps in a reduced basis we used a pipeline that combined descriptor and sparse landmark correspondences by adapting the approaches of [OBCS$^+$12] and [PBB$^+$13].

In all of our experiments, the power version of the principal curvature directions is used for the alignment constraints (Eq. (4.2)). To this end, we implemented the method proposed in [Rus04], where the weights are computed per triangle j by $S(j) = |\kappa_1 - \kappa_2|^2$ with κ_i the extremal curvature values, and we clamp values below .1 to zero. Once the cross fields are computed, we use it as input for the implementation of MIQ provided in libigl [JPS$^+$13], and we then feed the resulting parametrization to the implementation of QEx provided by the authors [EBCK13] to obtain a quadrangular mesh. Both, MIQ and QEx, were used with default parameters in our tests.

4.6 Evaluation and Results

4.6.1 Comparison with FSS

We compare our symmetric quadrangulation results with the state-of-the-art method of Fields on Symmetric Surfaces (FSS) by Panozzo et al. [PLPZ12]. In all of the following experiments, for computing the FSS results we used the cross field output data provided by the authors. For generating our cross field, we used as input the symmetry-map generated by their intrinsic symmetry computation method (using code provided by the authors), which results in a vertex-to-point in face mapping, and used the full hat basis unless otherwise noted. For both methods, we generated a quad mesh from the cross field using MIQ and QEx, using the same parameters for both approaches.

Behavior near the symmetry line. As discussed in Proposition 6 in [PLPZ12], at the symmetry line the cross field should either have a singularity, be aligned with the symmetry line, or form $\pi/4$ angle with the symmetry line. In their approach, the field is forced to align with the symmetry line using hard constraints. However, allowing singularities on the symmetry line, and therefore allowing the cross field to switch between the two configurations (aligned and rotated by $\pi/4$), may increase the overall consistency and smoothness. Instead, we omit this constraint, and solve for the global minimizer of Eq. (4.7), allowing us to compute quadrangular meshes which are more consistent with respect to singularity point locations and error metrics. In Fig. 4.8 we show the quad mesh generated by FSS (yellow) and by our approach (blue) using the full map. Note how forcing the quad directions to align with the symmetry line generates noisy quads in the FSS approach, e.g., along the chest and nose of the gargoyle as shown in the zoomed-in figures, whereas our method generates a smoother edge-flow.

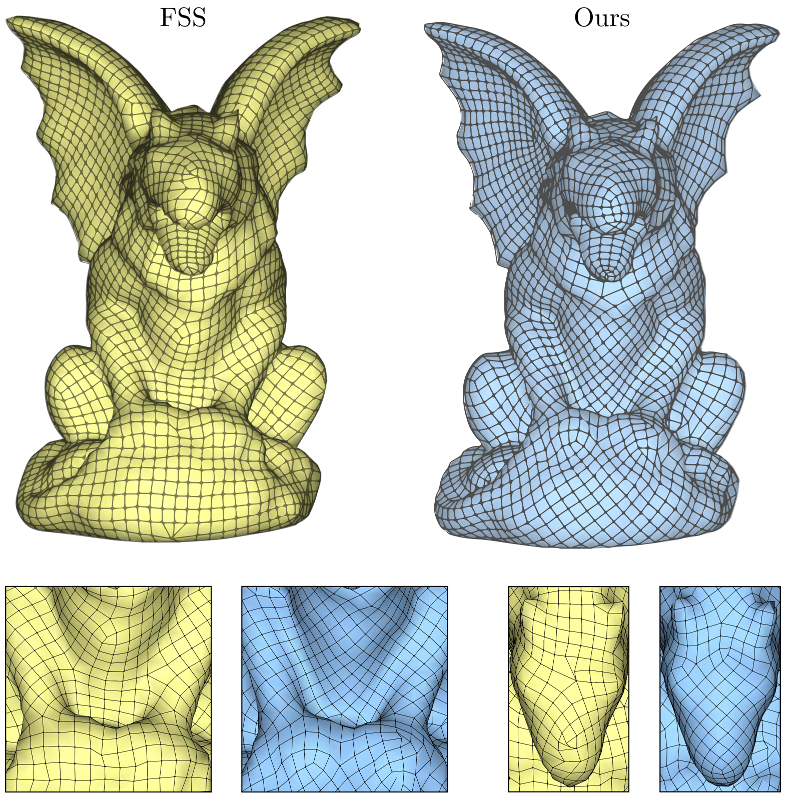


Figure 4.8: The approach of Panozzo et al. [PLPZ12] constrains the field to be aligned with the stationary line (yellow). Thus, the space of possible minimizers is significantly smaller, yielding sub-optimal results on the chest and nose of the shape (zoomed-in areas). In contrast, our method allows for general cross fields which exhibit intricate behavior along the symmetry line (blue). Consequently, our output better respects the involved geometry, while achieving lower error values (see rightmost column in Figs. 4.10a and 4.10b).

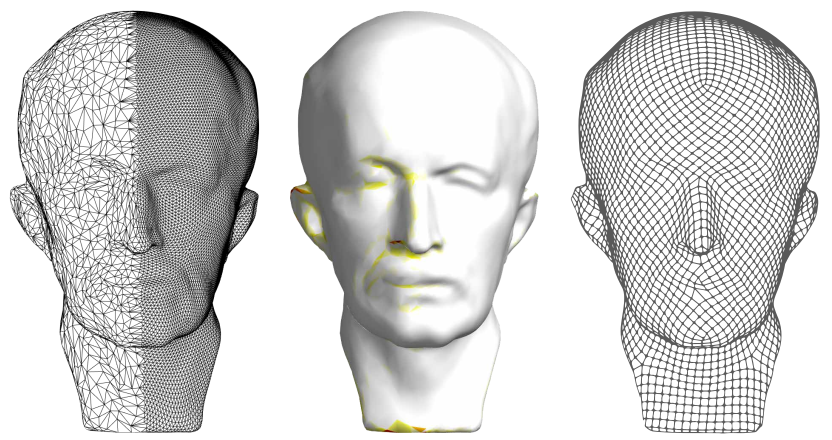


Figure 4.9: Robustness to triangulation. (left) We extensively decimated %85 of the vertices in the left part of Max Planck's model, leading to non-symmetric curvature alignment constraints (middle) due to the difference in triangle areas. Nevertheless, our method produces a symmetric cross field whose associated quad mesh is highly consistent (right). Notice that this example is particularly challenging for methods which employ the map differential.

Applicability. As mentioned in their paper (see Figure 8 there), FSS requires a high-quality symmetry map, and a corresponding symmetry line. In addition, for computing the map differential, they use the gradients of two functions (one symmetric and one anti-symmetric), which should also be extracted from the map and be of high quality. Our approach, on the other hand, requires only a functional correspondence, which can be given in a reduced or full basis, and can potentially be noisy. Therefore, our approach is applicable to more general shapes and less robust correspondences and as different mapping methods work better in different scenarios, ours general applicability is a clear advantage. For instance, in Fig. 4.9, we employ the mapping obtained using BIM on a mesh which is particularly challenging as there is a significantly different density of triangles along the stationary line (left). Nevertheless, our method produces a reasonable quad mesh (right) using only $k = 100$ eigenfunctions.

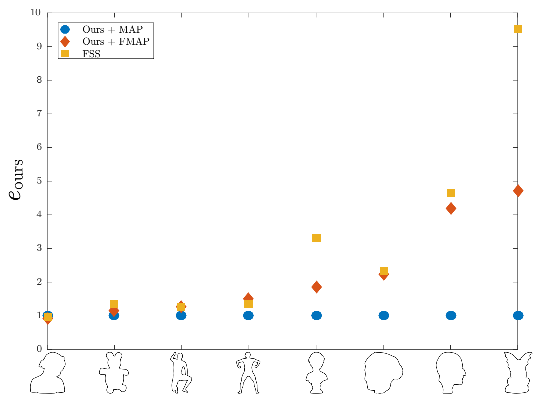
Quantitative comparison. We ran our method on all the models shown in [PLPZ12], for which the FSS intrinsic map computation could be used. We measured the consistency error of the resulting cross fields using two metrics: e_{ours} which is closely related to our consistency condition (Fig. 4.10a) and e_{FSS} which is guiding the FSS approach (Fig. 4.10b). The first metric, e_{ours}, is given by

$$e_{\text{ours}} = E_c(m, f_i) + E_c(n, g_i) \ ,$$

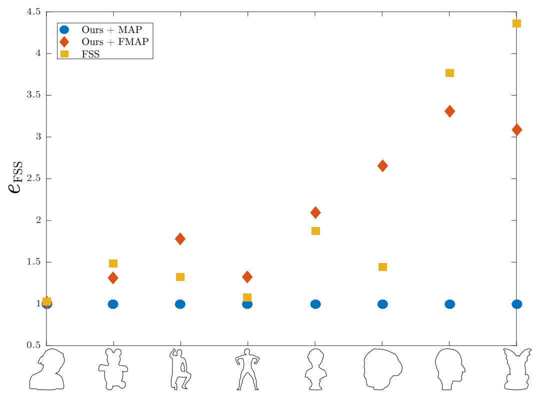
where both of the terms are computed using the functional map constructed from the known precise mapping, i.e., $C \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$. In the left term $E_c(m, f_i)$, we use m and f_i as defined in Section 4.5, and we randomly generated $n = 1000$ vertex indices for which we created hat functions g_i that are used to compute $E_c(n, g_i)$. The second metric is defined as follows.

$$e_{\mathrm{FSS}} = \|x_p - (\mathrm{sym}(x))_p\|_M^2 \ ,$$

where x is the cross field generated by FSS, and $\mathrm{sym}(x)$ is the symmetrized version of x, computed by applying the "symmetrization by field transport" step of the FSS algorithm to x. We compare x with its symmetrized version by comparing their power fields, and weigh the errors by the face area, namely $\|y\|_M^2 = y^T G_{\mathcal{F}} y$. If $e_{\mathrm{FSS}} = 0$, it would imply that symmetrizing the cross field x has no effect, and thus x is already exactly symmetric. For both metrics, we show the results for FSS (yellow squares) and for our approach when using the FSS map in the full basis (blue circles) and in the reduced basis (red diamonds). To evaluate the error results together, we consider the *relative* error as it is measured with respect to the value we obtained with our method when using the full basis.



(a) We compared our method using functional maps given in a full basis (blue circles) or a reduced basis (red diamonds) to FSS [PLPZ12]. Using the full basis, our approach achieves superior results on all the models, and in some cases by a large margin, as can be seen in the relative error above. See the text for additional details.



(b) In addition to the comparison we show in Fig. 4.10a that uses a modified version of our consistency condition, e_{ours}, we also compute the relative error metric e_{FSS}, which is optimized in FSS. For all the models, we obtain improved error metrics when using the full basis and comparable results for the reduced case. For more details, see the text.

As can be seen in Figs. 4.10a and 4.10b, our method with the full basis achieves better error results on all of the meshes except for bimba, where the results of the two methods are similar. Note that in these experiments we used the same input including the symmetry map computed by the method in [PLPZ12], and only the computation of the cross field is different. Thus, these quantitative results highlight the robustness and accuracy of our functional formulation. In particular, on models where our cross fields switched their behavior along the symmetry line, we gained significant improvement: e.g., of factors $3.3, 4.65$, and 9.53 on the models busto, Max Planck and gargoyle, respectively, in our metric e_{ours}. The improvement in e_{FSS} for these models was $1.87, 3.75$ and 4.35, respectively. Moreover, we note that our method with a reduced functional basis of size $k < 300$ produced comparable or better results when compared to

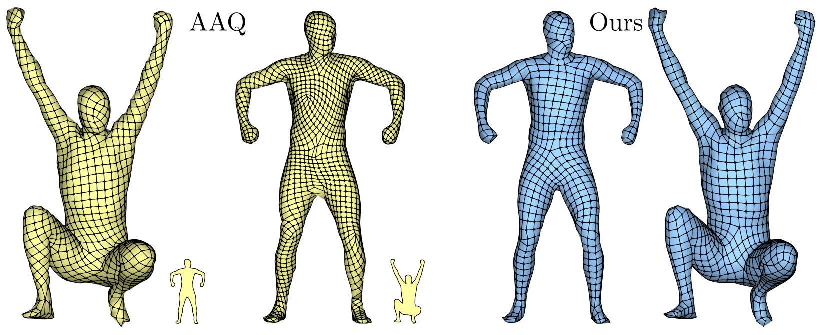


Figure 4.11: We compare our consistent quadrangulation of a pair of meshes with AAQ [MPP+13] using code supplied by the authors. We computed a quad mesh on each mesh using it as the base mesh (yellow), which should then be transported to the second mesh to yield exactly consistent quadrangulations. For our approach (blue), we used the point-to-point correspondence also given to AAQ. Note that our results are both smooth and consistent, where as while the AAQ results are exactly consistent, they are dependent on the base mesh, and considerably less smooth. See the text for more details.

FSS with our metric e_{ours}. However, when measured in e_{FSS}, these cross fields typically generated inferior error results, while being visually plausible as can be seen in Fig. 4.5 (left and right).

4.6.2 Comparison with AAQ

We compare our consistent quadrangulation of a pair of meshes with the state-of-the-art method of Animation Aware Quadrangulation (AAQ) by Marcias et al. [MPP+13]. For computing the AAQ results we used the code provided by the authors, and applied it to the two human meshes shown in Fig. 4.11, using the default parameters. We generated two cross fields, by choosing first the kneeling human as the base mesh (yellow left) and then the standing human as the base mesh (yellow right). Each of these quad meshes should be transported to the second frame to generate exactly consistent quad meshes (we do not show the transported quads). Note, that AAQ can only be applied to meshes with the same triangulation, thus we also used this point-to-point map (in the full basis) as our input and applied our pipeline to generate consistent cross fields (blue). We then generated a quad mesh from the cross fields using MIQ and QEx, using the same parameters for both approaches.

Note, that while the output of AAQ is exactly consistent (as they transport the quads directly), the resulting quad mesh pairs would be very different depending on which mesh is used as the base mesh. Furthermore, our result is qualitatively better for both meshes (as it is both smooth and consistent), even though we only jointly design the cross-fields and quadrangulate separately. In general, the deformation of the triangles between these two meshes is quite large, which, as noted in the AAQ paper, is a challenge for the their method. Further, since AAQ only considers the deformation between the meshes, and not the curvature directions explicitly, the results for a pair of meshes are less aligned with the geometry than our results. We do note that AAQ is

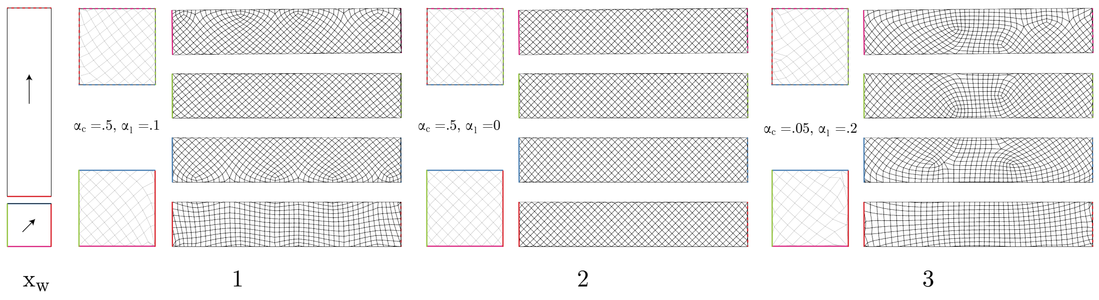

α_c =.5, α_l =.1
α_c =.5, α_l =0
α_c =.05, α_l =.2
x_w
1
2
3

Figure 4.12: Alignment vs. symmetry on a 3D bar shape. (x_w) The directional constraints are placed on the bottom face and and on one of the sides as marked by the arrows. (1) Due to the misaligned constraints, requiring solid-to-dashed line consistency as well as directional alignment yields a rather complex quad structure. (2) Relaxing the alignment constraint leads to a highly symmetric quadrangulation. Notice that the resulting cross field in this case is in fact the minimizer of the Dirichlet's energy. (3) Conversely, low values for consistency with high values for alignment produces mildly symmetric field which is better aligned in a least squares sense.

geared towards the more complex scenario of a collection of meshes, which we do not currently support, however their approach is specifically designed for triangle meshes with the same connectivity, thus is less general than ours in this respect.

4.6.3 Parameters exploration

Effect of consistency and alignment constraints. To motivate the use of our compatibility constraints, we demonstrate in Fig. 4.7 the quad meshes we obtain with and without this constraint on a pair of surfaces. Specifically, in the top row, we show that when requiring zero consistency, i.e., $\alpha_c = 0$, the resulting quads are somewhat related, mainly due to curvature information, but the singularities are in different locations (green points). Increasing this parameter to $\alpha_c = .01$ yields a compelling result, where the isolines and singularity locations are very consistent (bottom row). In addition, we show in Fig. 4.12 a more thorough evaluation of consistency vs. alignment weights on a 3D bar model. Our results show that requiring high alignment (x_w) in this case produces relatively complex quad meshes (1 and 3), while low alignment with high solid-to-dashed consistency yields smooth and symmetric result (2). The colored edges encode the parametrization cuts.

Size of the functional basis. In theory, our consistency rules in Eqs. (4.6) and (4.10) should hold for any function. However, these constraints are based on mapping functions between surfaces using a functional map, which is potentially given in a low-dimensional basis. Thus, transferring functions with high-frequencies in this case may result in significant errors due to projection onto the basis. Nevertheless, as our basis is given in terms of a multiscale eigen-decomposition of the Laplace–Beltrami operator, we hope to use as few as possible basis elements in our application. In practice, the basis size k determines which functions are well-represented, and in Fig. 4.13 we try to quantify which k's allow to produce high-quality quad meshes. (left) Using only $k = 10$ eigenfunctions is clearly insufficient as the resulting mesh is only quasi-symmetric, mainly due to curvature information, whereas increasing the basis to size $k = 100$ (middle) produces a highly-consistent mesh. With $k = 5161$, we obtain optimal results.

4.6.4 Robustness

Noisy point-to-point mappings. A key advantage to working in the functional setup is that it allows to gracefully handle scenarios where approximate or noisy correspondences are given. To evaluate the robustness of our method to inexact mappings in the context of approximately symmetric quad remeshing, we propose the following experiment. The model we use in Fig. 4.14 is equipped with a *compatible* triangulation, and thus we have the ground-truth mapping ϕ. Using this map, we generate two additional noisy correspondences, ϕ_2 and ϕ_4, where each point is randomly mapped with gaussian weights to the 2-ring and 4-ring neighborhood of its matching point, respectively,

Equipped with this data, we generate symmetric quadrangulations with our method using ϕ, ϕ_2 and ϕ_4 shown in the left, middle and right columns, respectively, where the top row is computed with the full basis and the bottom row uses a reduced basis of size $k = 100$. As can be seen in Fig 4.14, with perfect information ϕ, the results we obtain are outstanding, exhibiting complex quad structures (top left). However, our outputs are of lesser quality when noisy maps are used, with bent isolines on the head for ϕ_2 (top middle) and non-symmetric stationary line around the chest for ϕ_4 (top right). In contrast, when we use the associated functional maps in a reduced basis, the results we obtain and show at the bottom row reveal comparable consistency quality, regardless of the underlying noise in the mappings. While this result might seem non-intuitive in light of the error values we achieved in graphs 4.10a and 4.10b, we stress that the intrinsic mappings produced with FSS [PLPZ12] are of extremely high quality. However,

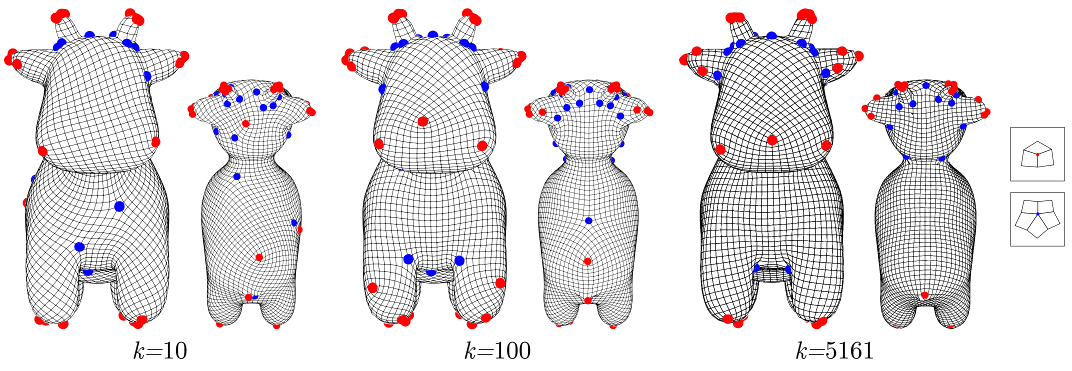


Figure 4.13: Effect of changing the functional basis' size. (left) When using only 10 eigenfunctions in B, we show that the resulting quadrangulation is hardly consistent with respect to the existing bilateral symmetry. (middle) Increasing the basis to include 100 elements significantly improves the result. (right) Finally, using the whole spectrum yields nearly perfect results.

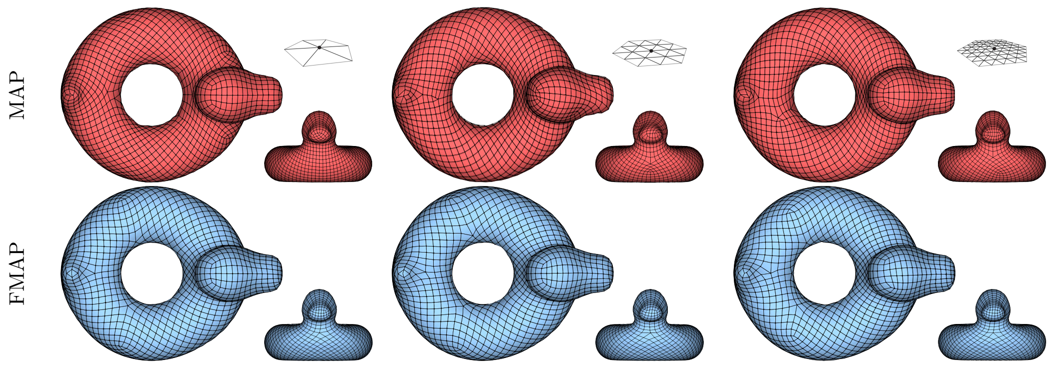


Figure 4.14: In this example, we demonstrate the robustness of our method in the presence of imperfect correspondences. We compared the quad meshes we obtain when using maps with deteriorating quality (left to right) in the full basis (top row) and reduced basis (bottom row). While we achieve very good results with the exact mapping (top left), the quad meshes produced with the noisy maps display only quasi-symmetry (top middle and top right). For comparison, employing a small functional map of size $k = 100$, yields consistent quadrangulations in all cases (bottom row). See the text for additional details.

computing good mappings is a hard problem in the general case, and thus we advocate the use of a reduced basis in cases where inexact data is given.

Pushforward error evaluation. When given a pair of shapes with the same connectivity, we can provide a more accurate measurement of the consistency error related to our computed cross fields. To this end, we facilitate a decimated version (752 vertices) of the SCAPE dataset [ASK$^+$05]. We compute cross fields y_1 and y_2 on the template pose paired with each of the first 50 poses. Then, using the ground-truth map differential, we push the associated x_1 to M_2 and calculate the error $G_{\mathcal{F}2}(q)\|x_2(q) - (\mathrm{d}\phi(x_1))(q)\|^2$, for each face $q \in \mathcal{F}_2$. In Fig. 4.15 we show the resulting sorted error distribution as computed for all of the pairs. The obtained results are consistently within the 10^{-5} range for all pairs, which is reasonable for such coarse triangulations.

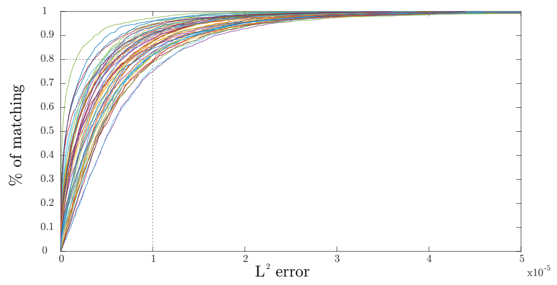


Figure 4.15: We design consistent cross fields x_1 and x_2, on pairs of shapes from the SCAPE dataset ($\alpha_c = .1, \alpha_l = 0$), and we measure the pointwise L^2 error of the computed x_2 compared to $\mathrm{d}\phi(x_1)$ which is the pushforward of x_1 using the ground-truth map differential. In the above plot, we show the distribution of the error for all of the pairs. Notice that in most of the cases, 80% of the points have an error of at most 10^{-5}.

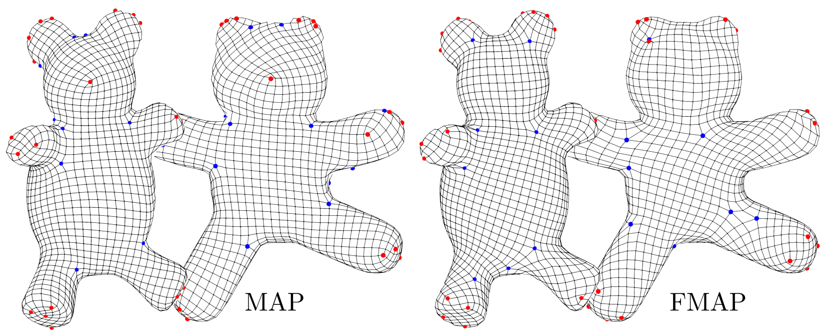


Figure 4.16: We generated a precise mapping using sparse landmark correspondences given as input to the seamless method [APL15]. With the resulting map, we compute consistent cross fields on both meshes with the full (left) and reduced (right) basis.

Applicability to various mapping methods. In our tests, we use different mapping methods and functional maps. For example, the results in Figs. 4.2, 4.5 (left and right), 4.8, 4.10a and 4.10b, are based on the intrinsic correspondences generated with FSS. Moreover, we utilized BIM in Figs. 4.5 (middle), 4.9, 4.7 and 4.13. Example 4.16 is particularly challenging as it involves non-isometric meshes for which the current state-of-the-art methods produce only approximate maps. Specifically, we used the seamless mapping method [APL15], and we generated approximately consistent quad meshes using the full basis (left) and the reduced basis (right). Notice that the resulting quadrangulations are qualitatively similar being slightly more consistent for the full map case.

Genus 1 examples. In Fig. 4.1 (left), we show an example on a genus 1 model with intrinsic symmetry. Notice that since the tail is attached to the head of the kitten, it is unclear in this model where exactly the symmetry line goes through, which makes it a stress test for many mapping techniques. Nevertheless, we were able to compute a high quality functional map, which allows us to generate an approximately consistent quad mesh which mostly respects the underlying symmetry. In addition, we tried a similar experiment on a pair of meshes from the FAUST dataset, where we "glued" the hands of one of the persons. Using a reduced functional map of size $k = 50$, we obtain compatible quad meshes, as can be seen in Fig. 4.17. Notice that the right leg is somewhat less consistent and it is due to the map which is only approximate. We show in the zoomed-in figures that the same singularity structure is maintained on both meshes, but it is twisted on the left person. We validated that the mapping is wrong in this area by mapping a function from the left person to the right, and, indeed, the colors in the zoomed-in area are inconsistent between the meshes.

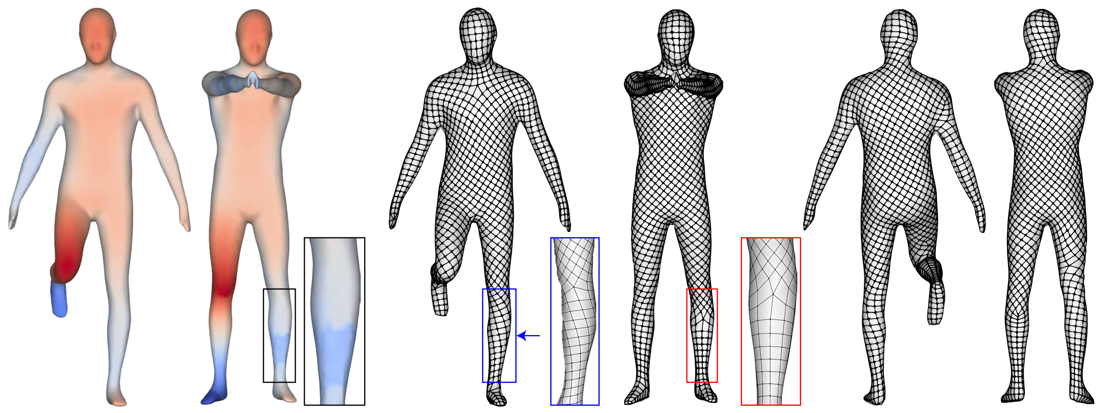


Figure 4.17: Unfortunately, methods for mapping surfaces with different genus are scarce. Nevertheless, the robustness of our machinery to different mapping methods allows us to compute consistent quadrangulations even in the difficult case of genus 0 and genus 1 surfaces.

4.7 Limitations

One limitation of our method is that in some cases, we achieve poor consistency results on certain areas of the mesh, even though the general quadrangulation is relatively consistent. We believe it is due to the fact we omitted pointwise unit length constraint, allowing the optimization to reduce energy by scaling vectors in problematic regions. Related to this issue, is that we generate *uniform* quadrangulations, regardless of the underlying geometry. In this context, additionally optimizing for a consistent sizing field might be beneficial. Finally, as we mentioned in Sec. 4.2, our method produces only approximate consistent quad meshes, since we optimize for a guiding field and not directly for the quads. As a result, our method does not provide guarantees about the exact quality of the resulting quad meshes. All of these shortcoming offer interesting directions for further consideration and future work.

4.8 Conclusion and Future Work

In this paper, we presented a novel unified technique for computing consistent quadrangulations of individual and pairs of shapes, with respect to a given symmetry and correspondence respectively. Our method does not require the input shapes to have the same triangulation and can handle shapes with arbitrary topology, while at the same time placing special emphasis on robustness and efficiency. Key to the success of our technique is a novel formulation that only requires a functional (rather than pointwise) correspondence across shapes and allows us to avoid the difficult estimation of the map differential, while being able to accommodate functional maps given in a reduced basis. Our formulation results in a simple and easy to implement method that produces more accurate results compared existing baselines and allows to handle more general difficult cases.

In the future we plan to extend our method to handle entire *collections* of shapes, and also to use our functional formulation to enable cross field design with other (possibly user-guided) novel constraints, which are difficult to enforce locally. In addition, our formulation is applicable to any N-RoSy fields, and not necessarily cross fields, and we wish to further investigate its applicability to joint design of PolyVector Fields [DVPSH14].

Chapter 5

Advection-Based Function Matching on Surfaces

A tangent vector field on a surface is the generator of a smooth family of maps from the surface to itself, known as the *flow*. Given a scalar function on the surface, it can be transported, or *advected*, by composing it with a vector field's flow. Such transport is exhibited by many physical phenomena, e.g., in fluid dynamics. In this paper, we are interested in the inverse problem: given source and target functions, compute a vector field whose flow advects the source to the target. We propose a method for addressing this problem, by minimizing an energy given by the advection constraint together with a regularizing term for the vector field. Our approach is inspired by a similar method in computational anatomy, known as LDDMM, yet leverages the recent framework of *functional vector fields* for discretizing the advection and the flow as operators on scalar functions. The latter allows us to efficiently generalize LDDMM to curved surfaces, without explicitly computing the flow lines of the vector field we are optimizing for. We show two approaches for the solution: using linear advection with multiple vector fields, and using non-linear advection with a single vector field. We additionally derive an approximated gradient of the corresponding energy, which is based on a novel *vector field transport* operator. Finally, we demonstrate applications of our machinery to intrinsic symmetry analysis, function interpolation and map improvement.

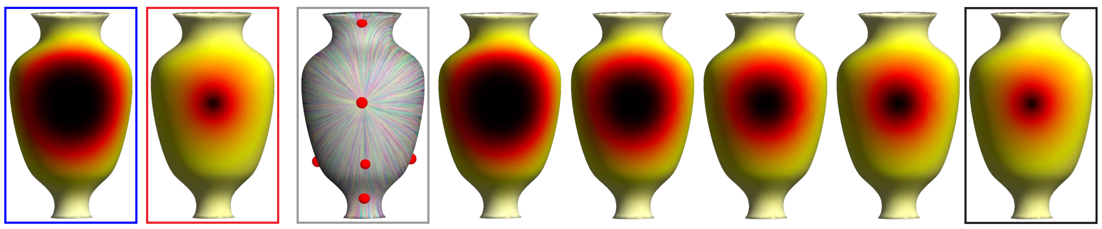


Figure 5.1: Our method takes a source function (blue frame) and a target function (red frame) and finds a single vector field (gray frame) whose associated flow map advects the source function to a function which matches the target function at the end time (black frame). In addition, our method yields a smooth interpolation of functions by advecting the source function for different times (5 frames from the right).

5.1 Introduction

Finding correspondences between geometric objects is a fundamental problem in geometry processing. In many cases, the map between the objects can be represented through correspondences between *scalar functions*. A Gaussian distribution centered at the location of the object [SNB$^+$12], an intensity function representing medical data [BMTY05] or a geometric descriptor [OBCS$^+$12], are all examples utilizing this approach. In fact, matching functions is a *more* general problem, as functions are not restricted to encode shapes, but can represent alternative information, such as distortion information [OBCCG13], appearance properties [BVDPPH11] or texture coordinates.

A natural extension to the function correspondence problem is to additionally compute an *interpolation* between the given functions, namely a time varying function which starts from the source and smoothly interpolates to the target. One possible approach then, is to recast the problem as finding a set of vector fields, whose associated *flow maps* are composed to yield an interpolation between the functions. The flow map of a vector field is computed in any point of the domain by "traveling" from that point and following the trajectory of the vector field (known as the *flow line*), for a specified time. *Advection* of a function is then achieved by composing it with the *inverse* of the flow map (see Figure 5.2), where interpolation is computed by advecting the source function for various times, and the target function is attained at the final time.

This approach to function interpolation has long been considered in medical imaging where anatomical images are deformed from one to another. Several methods designed for solving this problem are currently available [SDP13] of which the Large Deformation Diffeomorphic Metric Mapping (LDDMM) algorithm [BMTY05] is widely adopted. LDDMM tackles the problem by minimizing an objective function, which combines the advection constraint with a regularizing term on the vector fields. In practice, energy minimization is computed by explicitly constructing the flow map and its Jacobian, or *deformation gradient*. Therefore, carrying over this framework to *curved* domains is challenging, as these quantities are difficult to represent and compute on such domains. We suggest to overcome these difficulties by reformulating the energy using the framework of functional vector fields [ABCCO13], leading to a novel advection-based method for spatial interpolation between real-valued functions on curved triangle meshes.

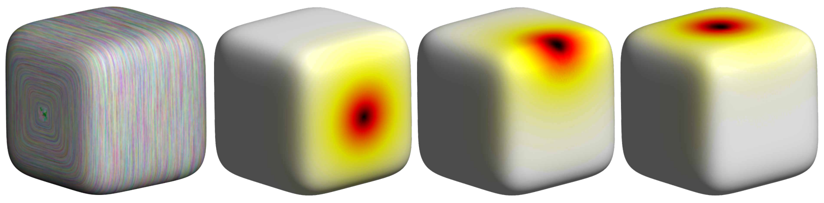


Figure 5.2: The flow map of a vector field (left, shown with the Line Integral Convolution method [PZ11]) is used to advect a function (middle left) for various different times (middle right and right).

The main component required for our method is an advection operator acting on curved domains. Previously, advection has been employed to simulate fluids [SY04], and more recently to compute stable shock filters [PK15]. However, these methods rely on the explicit computation of flow lines which is algorithmically complicated, unstable and error-prone on curved triangle meshes. Alternatively, tangent vector fields can be encoded as directional derivatives of functions, and thus as linear operators acting on the space of functions. Adopting this approach, [ABCCO13] showed that on triangle meshes, discrete tangent vector fields can be encoded as sparse matrices, whose *exponential* represents their associated flow map. Further, the composition of a map with a function is given in this setup as a matrix-vector multiplication, and hence advection can be efficiently approximated by computing the action of the matrix exponential on a vector [AMH11]. Notice that since functions are directly mapped to functions, the explicit computation of flow lines and its inherent difficulties is completely avoided in this setup.

In this paper, we facilitate the functional advection technique for solving the function matching problem. Initially, we propose to optimize for a set of vector fields and use *linear* transport, i.e., taking only the first two terms of the matrix exponential. This approach works well in scenarios as fluid simulation [AWO$^+$14, AVW$^+$15] where the Courant–Friedrichs–Lewy (CFL) condition limits propagation speeds and thus restricts the dynamic time step to be small. However, it is sometimes necessary to find a *single* vector field; a constraint that is rarely attainable with the linearized formulation as the required time step for matching might be too big. For instance, assume that the target function is in fact the advected version of the source function, as is the case in *optical flow* problems [SRB14]. In this context, one hopes to reconstruct the underlying vector field that governs the motion. Thus, we generalize our energy functional to include the *full* matrix exponential, for cases where it is crucial to interpolate using a single velocity field.

Unfortunately, the associated *directional derivative* of the matrix exponential is computationally intractable in most of our problems. Recently, Corman et al. [COC15] noticed that this derivative is in fact a block in the matrix exponential of a bigger operator, and thus used a sum of matrix exponentials of bigger matrices. However, they worked in a *reduced* spectral basis, allowing them to facilitate this observation which requires the computation of a large number of matrix exponentials (as many as the number of basis vectors). We, on the other hand, work with the full basis, thus their approach is less applicable in our case. Instead, we observe that an approximation of the matrix exponential derivative can also be formulated in terms of a *Lie bracket* operator acting on vector fields. Thus, we propose a novel discrete bracket and exploit the relation between vector fields and matrices to arrive at a *tractable* derivative for the matrix exponential. Overall, we emphasize that through the entire computation of the functional's gradient, we never store or explicitly compute the matrix exponential, but only its *action* on vectors.

We demonstrate our machinery in several applications as spatial interpolation between various functions and reconstruction of the governing velocity in an optical flow type scenario. Moreover, we construct a continuous symmetric map based on two descriptors. Finally, we show that our method can be used to extract the point-to-point map that is related to a given functional map.

5.1.1 Related Work

We discuss here various approaches which either solve the same problem on Euclidean domains, solve a related problem on triangle meshes, or target similar applications as ours.

Computational anatomy. The problem of matching consecutive medical images is a classical problem in computational anatomy, and one of the common solutions uses the flow of one or more vector fields, see [SDP13], for a recent review. Among the plethora of such methods, LDDMM [BMTY05] is extremely popular, and has been extended to many settings, though not to curved triangle meshes. On flat domains, discretizations of LDDMM use semi-Lagrangian techniques for vector field integration, and for computing discrete mappings and their differentials, using simple interpolation rules. However, on curved meshes these computations are more challenging, as trajectories should be constrained to remain on the curved surface. Our discretization, on the other hand, is based on the functional approach, thus functions can be advected without explicit computations of mappings and their differentials.

Optical flow. vector field based registration is also popular in computer vision, where it is known as *optical flow*, see [SRB14] for a recent review. In the classical formulation, when two images are given, the goal is to find a smooth displacement vector field which matches the first image to the second. This is in fact the linearized version of advection on Euclidean domains, highly appropriate in optical flow since the change between consecutive images is small. Optical flow has been generalized in many ways, and was recently adapted to advection-based matching of a series of functions on triangle meshes [LB08]. There, however, *multiple* samplings of the interpolated function are given as input, i.e., not only the initial and final functions as in our setup, inherently assuming that the deformation between two consecutive functions is small. Furthermore, they compute multiple vector fields which realize the flow, and their advection approach exhibits far more diffusion than ours.

Optimal transport. Matching between distributions is a prevalent objective in optimal transportation (OT) methods [Vil03, Vil08]. In fact, the Benamou–Brenier [BB00] formulation shares some similarities with our matching approach, with the important difference that their associated vector field is *time-dependent* in general. In the special case when distances are raised to the power of 1 [SRGB14], instead of a general power

p, OT is formulated using a single vector field. However, advecting the function on the resulting vector field leads to a trivial pointwise linear interpolation between the source and target functions, whereas our method yields spatial displacements. Alternatively, using *squared* distances [SDGP$^+$15] (i.e., $p = 2$) yields interpolation results that are closer to ours, yet requires regularization for computational efficiency which leads to blurring, and does not output a single vector field.

Related Applications in Computer Graphics. Our method can be classified as on-surface interpolation of functions, and there exist a small number of works addressing similar problems in Computer Graphics. Perhaps the closest to our approach is the method for continuous matching [COC15]. There, the authors improve a given point-to-point map, by optimizing for a vector field such that the composition of its flow map with a given input map approximates another known map. However, optimization in their setup requires working in a reduced spectral basis in order to be computationally feasible due to their usage of explicit matrix exponentials. In addition, their obtained field is smooth and thus does not account for high frequency deformations whereas our method does. Deformation of functions on surfaces is also addressed in [RTD$^+$10], by computing a map fulfilling some point constraints and composing its inverse with the function to be deformed. This problem is in some sense simpler than the one we address, since the constraints imply that the correspondence between the source and target functions is known, and only interpolation is needed.

5.1.2 Contributions

Our main contribution is a method for solving the inverse problem of computing a vector field whose flow advects one function to another on curved triangle meshes, where the functions are not required to be similar or have overlapping support. To this end we:

- Reformulate the LDDMM energy using functional operators. We explore linear and non-linear advection formulations and provide the associated gradients (Sections 5.3–5.5).
- Present a novel Lie bracket operator on vector fields (Section 5.6) which is instrumental for efficiently computing the derivative of the advection operator (Appendix D.3).
- Present applications of this machinery to optical flow on curved domains, interpolation of scalar functions, extraction of a point-to-point map from a given functional map, and realization of intrinsic symmetry maps. (Section 5.8).

5.2 Vector Fields and Flows

To formally specify our objective we briefly describe the following definitions for vector fields and their flows. In differential geometry, it is well-known [Fra11] that tangent vector fields are fully encoded through their *action* on smooth scalar functions. Given a surface M with its associated metric $\langle \cdot, \cdot \rangle$ and a smooth function $f : M \to \mathbb{R}$, the action of a vector field v on f is given by the *directional derivative* of the function:

$$v(f) = D_v(f) = \langle v, \nabla f \rangle \ , \tag{5.1}$$

where the inner product is computed per point $p \in M$. Vector fields and mappings are tightly linked as any tangent vector field v defines a one-parameter family of self-maps ϕ_v^t, known as the *flow* of v, which satisfies:

$$\frac{\mathrm{d}}{\mathrm{d}t}\phi_v^t = v \circ \phi_v^t, \quad \phi_v^0 = \mathrm{id} \ .$$

Advection of scalar functions is then achieved by composing f with the inverse of the flow map, i.e., $f(t) = f \circ \phi_v^{-t}$. Thus, $f(t)$ is the unique solution of the following partial differential equation,

$$\frac{\mathrm{d}}{\mathrm{d}t} f(t) = -D_v(f(t)), \quad f(0) = f \ . \tag{5.2}$$

The operator of advection plays a key role in our method since it allows to match between functions. We proceed by describing our approach for function matching on surfaces.

5.3 Advection-based Function Matching (ABFM)

A straightforward (and computationally trivial) approach to interpolating functions is to linearly blend them. However, pointwise interpolation is independent of the global structure of the functions and the underlying geometry. Moreover, if the functions are spatial deformations of one another, i.e., an associated field generates the functions (as in optical flow), linear interpolation would not produce satisfying results. These issues motivate a different approach.

Given a surface M and two scalar functions $f, g : M \to \mathbb{R}$, we seek for a time-varying tangent vector field $v(t)$ whose associated flow advects f onto g. In general, this problem is ill-defined, as there can be many such fields, therefore some additional regularization on the vector field is required. To this end, we design an energy functional which includes two terms: a data term that promotes the advection constraint, and a regularization term which enforces smoothness on the vector field. Our approach is inspired by the popular LDDMM framework [BMTY05], which enforces a similar functional. In Figure 5.3, we show matching and interpolation results computed using

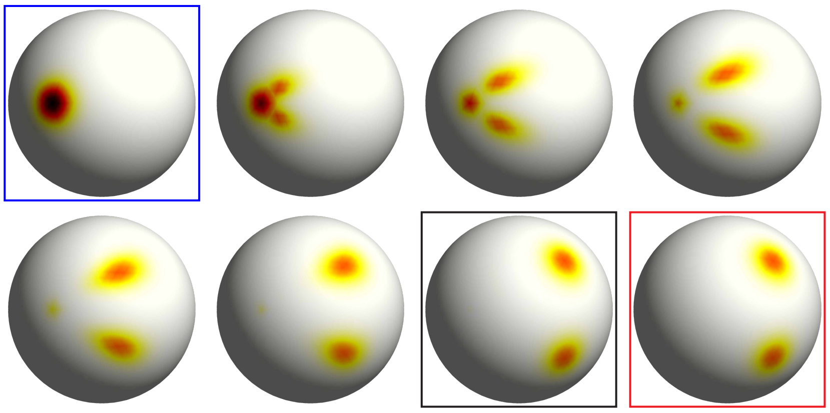


Figure 5.3: Given source and target functions (blue and red frames), our method matches the advected source to the target (black frame). The resulting interpolation is obtained by advecting for different times leading to spatial displacement of values.

our method.

We propose to solve the following optimization problem:

$$\arg\min_{v} \frac{1}{2}\int_0^{\tau} \|v(t)\|_{\alpha}^2 \mathrm{d}t + \frac{1}{2\,\sigma^2}\|f \circ \phi_v^{-\tau} - g\|_{\beta}^2 \ . \tag{5.3}$$

Namely, our data term seeks to minimize the norm of $f(\tau) - g$, where $f(\tau)$ satisfies Eq. (5.2) when advecting f using the optimized velocity. The scaling parameter σ weighs the matching against the penalty due to the regularization of the velocity, i.e., we treat the matching as a *weak* constraint.

We choose a function norm that is more suitable for measuring distances between *disjoint* functions, i.e., functions whose supports (where $f, g \neq 0$) are disjoint. Specifically, increasing the parameter β allows for interpolating functions that are farther apart. The term which regularizes vector fields also uses a modified norm, see e.g., [BMTY05], which promotes a smoother velocity as α grows. Thus, we define the following norms:

$$\|f\|_{\beta}^2 = \int_M f(x)\, C_{\beta}\, f(x)\mathrm{d}x \ , \quad \|v\|_{\alpha}^2 = \int_M \langle v(x), D_{\alpha}\, v(x)\rangle \mathrm{d}x \ ,$$

where C_β is defined as $C_\beta = \mathrm{id} - \beta\,\Delta_{LB}$ with Δ_{LB} the *negative-definite Laplace–Beltrami* operator. Similarly, the operator D_α is given by $D_\alpha = \mathrm{id} + \alpha\,\Delta_H$, where Δ_H is the *Hodge Laplacian*.

5.4 Discretization

The main challenge in the discretization of our objective function on triangle meshes is computing the flow map ϕ_v^t. This requires computing the flow lines of v, which is known to be a non-trivial and error prone problem on curved meshes, requiring combinatoric decisions (e.g., to which triangle should the flow line continue). Furthermore, it is not clear how one could compute the gradient of our objective functional if using such a direct approach for computing the flow lines. In the following, we propose an alternative method, which involves only the advected functions and does not require the computation of the flow lines, based on the functional representation of vector fields [ABCCO13].

Notation. We are given a triangle mesh with face set $\mathcal{F}$ and vertex set $\mathcal{V}$. We define our discretizations in matrix notation, and thus represent functions as vectors of length $|\mathcal{V}|$, and vector fields as vectors of length $3|\mathcal{F}|$. Vertex and face areas are respectively denoted by $A_\mathcal{V} \in \mathbb{R}^{|\mathcal{V}|}$ and $A_\mathcal{F} \in \mathbb{R}^{|\mathcal{F}|}$, where the area of vertex i is computed by one third of the total area of its adjacent triangles. We use diagonal mass matrices given by $G_\mathcal{V} = [A_\mathcal{V}] \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ for vertices and $G_\mathcal{F} = [A_\mathcal{F}] \in \mathbb{R}^{3|\mathcal{F}|\times 3|\mathcal{F}|}$ for faces. The bracket $[\cdot]$ operator converts vectors in $\mathbb{R}^{|\mathcal{V}|}$ and $\mathbb{R}^{|\mathcal{F}|}$ to diagonal matrices in $\mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ and $\mathbb{R}^{3|\mathcal{F}|\times 3|\mathcal{F}|}$ respectively (replicating each entry 3 times for the latter). In addition, we define the interpolation matrix $I_\mathcal{V}^\mathcal{F} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{F}|}$ to average quantities from faces to the vertices, i.e., $I_\mathcal{V}^\mathcal{F}(i,j) = \frac{A_\mathcal{F}(j)}{3A_\mathcal{V}(i)}$, iff vertex i belongs to face j and 0 otherwise. Our differential operators, $\mathrm{grad} \in \mathbb{R}^{3|\mathcal{F}|\times|\mathcal{V}|}$ and $\mathrm{div} \in \mathbb{R}^{|\mathcal{V}|\times 3|\mathcal{F}|}$, are the standard ones as defined in [BKP+10, Chapter 3].

Functional Vector Fields. Following the construction introduced in [ABCCO13] and using our notation, we have that D_v and its *dual* version $\overline{D}_f$, required for derivative computations, can be computed as follows

$$D_v = I_\mathcal{V}^\mathcal{F} [v]_\bullet^T \,\mathrm{grad} \ , \quad \overline{D}_f = I_\mathcal{V}^\mathcal{F} [\mathrm{grad}\, f]_\bullet^T \ ,$$

where $\overline{D}_f$ is a matrix of size $|\mathcal{V}| \times |\mathcal{F}|$ defined as the operator which satisfies $\overline{D}_f(v) = D_v(f)$. Here, $[\cdot]_\bullet \in \mathbb{R}^{3|\mathcal{F}|\times|\mathcal{F}|}$ is a block diagonal matrix which encodes a pointwise multiplication of a vector field by a face-based function, and its transpose evaluates a pointwise inner product.

Functional Advection. A particularly useful property of D_v is that discrete advection of a function f can be computed using the action of the matrix exponential on f, namely,

$$f_t = \exp(-t\, D_v)\, f = \sum_{k=0}^{\infty} \frac{(-t)^k}{k!} D_v^k\, f \ , \tag{5.4}$$

where the action is computed in an efficient manner with methods as [AMH11]. When viewed as an operator that maps functions to their directional derivatives, the discrete D_v with the relation (5.4) is closely related to the functional maps framework [OBCS+12]. In this context, the operator $\exp(-t\, D_v)$ is in fact the functional map associated with the flow map of v. Finally, there are cases (e.g., in fluid simulation) where it is sufficient

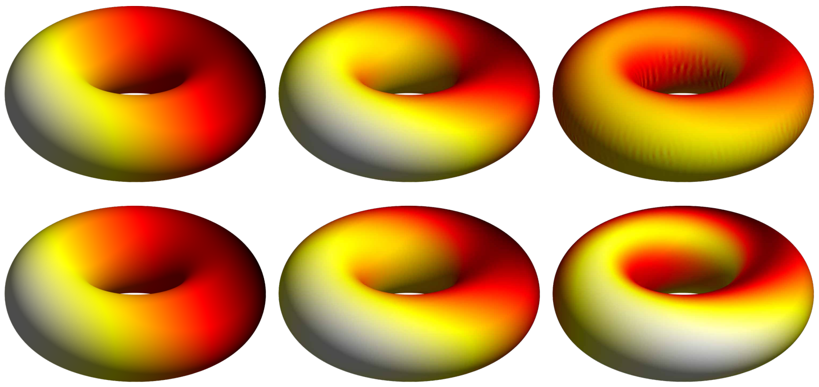


Figure 5.4: Linear advection of an input function (top left) works well for short times (top middle), but discretization errors in the form of oscillations appear for longer times (top right). For comparison, the non-linear transport (bottom) better approximates the flow and it yields a smooth result, even for long times (bottom right).

to use the *linearized* version of advection, i.e.,

$$f_t = (\mathrm{id} - t\, D_v)\; f \;. \tag{5.5}$$

In Figure 5.4, we show a comparison between the linear and non-linear versions of advection. Starting from the same initial function (top and bottom, left), the linear computation (top) exhibits discretization noise, i.e., oscillationst (top right), while the non-linear discretization (bottom) provides a smooth result (bottom right).

Discrete Energy. To fully discretize problem (5.3), we break the time parameter into N segments of equal size $\delta\tau = \tau/N$. Thus, we optimize for a finite set of vector fields $\{v_j\}_{j=1}^N$ that are constant per time segment. Using the above definitions and matrix notations, we arrive at the following discrete optimization problem

$$\underset{\{v_j\}}{\arg\min} \left(\frac{\delta\,\tau}{2} \sum_{j=1}^{N} v_j^T\, G_{\mathcal{F}}\, D_\alpha\, v_j + \frac{1}{2\,\sigma^2} \delta g^T G_{\mathcal{V}}\, C_\beta\, \delta g \right) \;, \tag{5.6}$$

where $\delta g = f \circ \phi_v^{-\tau} - g$. The map $\phi_v^{-\tau}$ is obtained by composing the flow maps of the different velocities, i.e.,

$$f \circ \phi_v^{-\tau} = f \circ \phi_{v_1}^{-\delta\tau} \circ .. \circ \phi_{v_N}^{-\delta\tau} \;,$$

where composition is achieved through matrix multiplication.

The transported function can be computed using the linearized flow (5.5) or the non-linear flow (5.4). In general, the linearized method is preferred in cases where the overall smoothness of advection is of less importance (see applications in Section 5.8 that are related to Figures. 5.11, 5.12), since the composition of discrete mappings may induce some error. Also, the linearized method yields reasonable results when the underlying deformation is small or the flow time is sufficiently short. On the other hand, non-linear advection is crucial when one requires a single vector field that generates a smooth deformation (see e.g., Figures 5.9, 5.10). In the following, we consider two scenarios: linear flows with multiple vector fields and non-linear flows with a single vector field. Hence, we have

$$f \circ \phi_v^{-\tau} = \left(\prod_{j=1}^{N} \left(\mathrm{id} - \delta\tau \, D_{v_j} \right) \right) f \ , \qquad \text{or} \qquad f \circ \varphi_v^{-\tau} = \exp(-\tau \, D_v) f \ ,$$

where we used the notation $\varphi_v^{-\tau}$ in the non-linear case to distinguish it from the linear case. Finally, we use the notation f_t to denote the advected version of f to time t, i.e., $f_t = f \circ \phi_v^{-t}$ or $f_t = f \circ \varphi_v^{-t}$, depending on the associated flow, and $f_0 = f$.

Discrete Gradient. To solve the discrete problem (5.6), the gradient of the energy functional is required. Given that D_α and C_β are *self-adjoint* operators, i.e., in our case these are symmetric matrices with respect to the corresponding inner product, the derivative of the energy functional is given by

$$\frac{\partial}{\partial v_j} E = \delta\tau \, G_{\mathcal{F}} \, D_\alpha \, v_j + \frac{1}{\sigma^2} \left(\frac{\partial}{\partial v_j} f_t \right)^T G_{\mathcal{V}} \, C_\beta \, \delta g \ . \tag{5.7}$$

The full derivation of the gradient appears in Appendix D.1. Note, that the gradient depends on $\partial_{v_j} f_t$, and thus on the choice of advection operator. We provide in Appendices D.2 and D.3 the derivative of the advection for the linearized and non-linear flows, respectively.

5.5 Linear and Non-linear Advection-based Function Matching

The first scenario we consider takes $N > 1$ and employs linearized flows, i.e., uses the advection $f \circ \phi_v^{-\tau}$ as described in Section 5.4. To compute the directional derivative of the energy functional (5.7), we derive the component $\frac{\partial}{\partial v_j} f_t$ and obtain

$$\frac{\partial}{\partial v_j} f_t = -\delta\tau \left(\prod_{i=j+1}^{N} \left(\mathrm{id} - \delta\tau \, D_{v_i} \right) \right) \overline{D}_{f_{(j-1)\delta\tau}} \ , \tag{5.8}$$

where $f_{(j-1)\delta\tau}$ is the (partial) advection of f_0 to time $(j-1)\delta\tau$. We refer to this method as Linearized Advection-based Function Matching (LABFM) and Figures 5.11, and 5.12 were generated using this method.

While the LABFM method is fast and simple to implement, there are certain applications in which $N = 1$ is a design requirement. Thus, we extend the former method to include non-linear flows, i.e., $f \circ \varphi_v^{-\tau}$, and to optimize for a *single* vector field. The modified energy becomes

$$E(v) = \frac{\tau}{2} v^T G_{\mathcal{F}}\, D_\alpha\, v + \frac{1}{2\,\sigma^2} \delta g^T G_{\mathcal{V}}\, C_\beta\, \delta g \ ,$$

where, as before, we need to re-derive the suitable gradient of the component $\partial_v\, f_t$. We emphasize that computing the derivative of this expression is more involved compared to the former case. In particular, $\exp(-\tau\, D_v) \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is a *dense* matrix, thus using finite differencing methods or extracting a block in the exponential of a bigger operator [COC15] is possible only for small problems. Instead, we derive in Appendix D.3 an approximation of the gradient by exploiting the relation between matrices and vector fields. We arrive at the following expression,

$$\frac{\partial}{\partial v} f_t = -\frac{\tau}{k+1} \exp(-\tau\, D_v) \overline{D}_{f_0} \sum_{s=0}^{k} \exp\left(\frac{s\,\tau}{k}\, \mathrm{ad}_v\right) \ , \tag{5.9}$$

where k is a scalar used to approximate a continuous integral, and ad_v is the *Lie bracket* [Fra11], an operator that acts on vector fields, i.e., $\mathrm{ad}_v\, u = [v, u]$, where u is a vector field. Intuitively, the bracket measures the amount of change u exhibits with respect to the flow lines of v. We defer the discussion on the operator ad_v and its exponentiated version to the next section.

5.6 Lie bracket of Vector Fields

The Lie derivative evaluates the change of a vector field over the flow of another vector field. Given two tangent fields v and u, we say that their flows ϕ_v^t and ϕ_u^t *commute* when their bracket is zero. Geometrically, it means that one can apply ϕ_v^t and then ϕ_u^t or the other way around and arrive at the same point. Formally, the bracket is given in operator form by:

$$D_{[v,u]} = D_v D_u - D_u D_v \ , \tag{5.10}$$

where $[v, u]$ denotes the associated vector field. Notice that under the bracket operation, vector fields form a group (since second derivatives cancel), i.e., $D_{[v,u]}$ is a directional derivative operator. We show in Figure 5.5 an example of the smoothest vector field u (right) that commutes with v (left).

Representing and computing the Lie derivative on surfaces is an on-going challenge and several methods try to solve this problem. For instance, [AOCBC15] exploit the functional approach and offer an efficient representation of the bracket in a reduced spectral basis. We choose to follow [ABCCO13] as their bracket discretization is closely related to the directional derivative operators we use. Specifically, the discrete version of Eq. (5.10) is computed in [ABCCO13] by taking the commutator of the respective matrices. However, the resulting matrix acts on scalar functions, whereas $\mathrm{ad}_v \in \mathbb{R}^{3|\mathcal{F}|\times 3|\mathcal{F}|}$ is an operator that takes a vector field and returns a vector field.

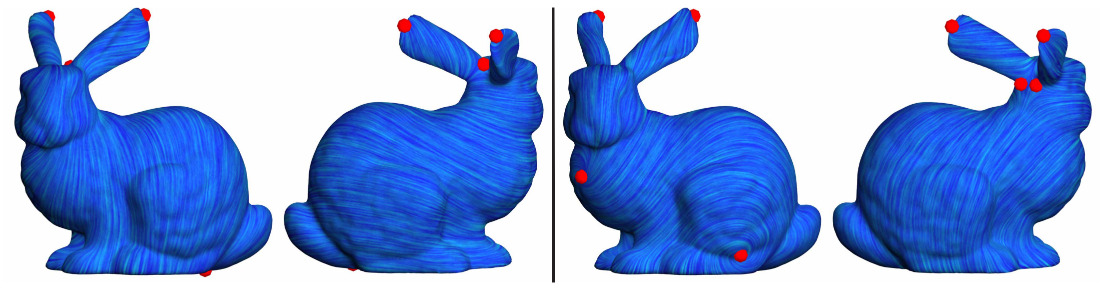


Figure 5.5: Given a vector field v (left), the kernel of its operator ad_v consists many vector fields that commute with v, where we show the smoothest one (right).

Nevertheless, we observe that a vector field $[v, u]$ can be extracted from its directional derivative $D_{[v,u]}$ by applying the operator on the coordinate functions x, y and z. For instance, to reconstruct the x-coordinate we compute $D_{[v,u]}(x) = [v, u]_x$. Repeating this procedure for y and z (the derivation appears in Appendix D.4) yields:

$$\mathrm{ad}_v = \begin{pmatrix} \mathcal{D}_v & 0 & 0 \\ 0 & \mathcal{D}_v & 0 \\ 0 & 0 & \mathcal{D}_v \end{pmatrix} - \begin{pmatrix} \overline{\mathcal{D}}_{v_x} \\ \overline{\mathcal{D}}_{v_y} \\ \overline{\mathcal{D}}_{v_z} \end{pmatrix} \in \mathbb{R}^{3|\mathcal{F}|\times 3|\mathcal{F}|} \ , \tag{5.11}$$

where $\mathcal{D}_v = [v]_\bullet^T \,\mathrm{grad}\, I_\mathcal{V}^\mathcal{F}$ and $\overline{\mathcal{D}}_f = [\mathrm{grad}\, I_\mathcal{V}^\mathcal{F} f]_\bullet^T$.

A novel property of ad_v is its relation to the *differential* of the flow map ϕ_v^t. The differential of a self-map ϕ_v^t generates a self-map $\mathrm{Ad}_{\phi_v^t}$ on the tangent bundle, i.e., $\mathrm{Ad}_{\phi_v^t}$, also known as the *pushforward*, transports vector fields to fields. For finite matrices, our representation is extremely useful since we can discretize $\mathrm{Ad}_{\phi_v^t}$ by taking the exponential of ad_v [Hal15]:

$$\mathrm{Ad}_{\phi_v^t}\, u = \exp\left(t\, \mathrm{ad}_v\right) u \ , \tag{5.12}$$

where the term $\exp(t\,\mathrm{ad}_v)$ appears in our gradient computations (5.9), and we evaluate it using Eq. (5.11) and the matrix exponential. In Figure 5.6, we demonstrate the action of $\mathrm{Ad}_{\phi_v^t}$ associated with v (left) on the field u (middle left) for several times (middle right and right).

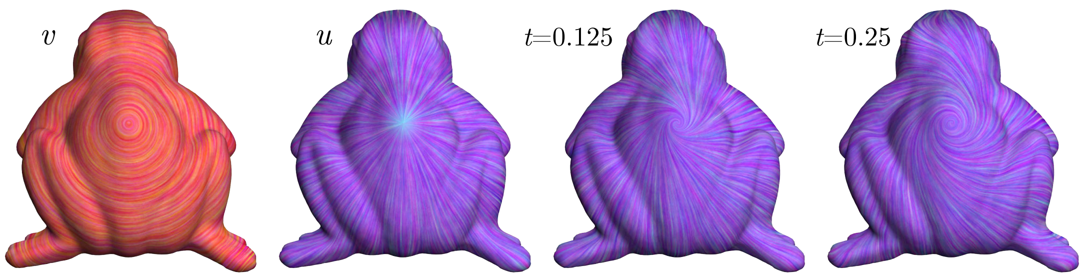


Figure 5.6: Transport (pushforward) of a vector field u (middle left) over the flow lines of v (left) is shown for various times $t = 0.125$ and $t = 0.25$ (middle right and right).

Algorithm 5.1 Energy and gradient calculation for the Quasi-Newton iteration of the non-linear ABFM algorithm. The routines exp and exp_tspan are described in [AMH11].

```
function NLABFM($\mathbf{f}, \mathbf{g}, \mathbf{v}$)
    // Energy computation
    $f_\tau \leftarrow \exp(-\tau,\, D_{\mathbf{v}},\, \mathbf{f})$
    $\delta g \leftarrow f_\tau - \mathbf{g}$
    $\mathbf{E} \leftarrow \frac{\tau}{2}\mathbf{v}^T G_{\mathcal{F}}\, D_\alpha\, \mathbf{v} + \frac{1}{2\,\sigma^2}\delta g^T G_{\mathcal{V}}\, C_\beta\, \delta g$      // Eq. (5.6)
    // Gradient computation
    $d \leftarrow \overline{D}_{\mathbf{f}}^T\, \exp(-\tau,\, D_{\mathbf{v}}^T,\, G_{\mathcal{V}}\, C_\beta\, \delta g)$
    $d_s \leftarrow$ exp_tspan($\mathrm{ad}_{\mathbf{v}}^T,\, d,\, 0,\, \tau,\, k$)
    $\mathbf{D} \leftarrow \tau\, G_{\mathcal{F}}\, D_\alpha\, \mathbf{v} - \frac{\tau}{(k+1)\sigma^2}\sum_s d_s$      // Eq. (5.9)
    return $\mathbf{E}, \mathbf{D}$
```

5.7 Implementation Details and Limitations

We implemented our method in MATLAB using the `minFunc` routine [Sch12] which employs a quasi-Newton algorithm with L-BFGS (limited memory) updating. In Algorithm 5.1, we provide the function handle that computes the energy and derivative of NLABFM. This pseudo-code includes calls to exp and exp_tspan which compute the action of a matrix exponential on a vector for a specific time and for a range, respectively (see [AMH11] for further details). Notice that the gradient computation includes the transpose of the matrices $D_v, \overline{D}_f$ and ad_v. This is due to the use of $(\partial_v f_t)^T$ in the gradient (5.7) and since $\exp(A)^T = \exp(A^T)$ for any matrix A.

Our advection method depends on the end time parameter τ. Unfortunately, the computation of the action of the matrix exponential is not stable for long times. Consequently, the optimization does not find a descent direction and stops immediately. We handle this limitation with a simple modification to the definition of the non-linear flow, i.e.,

$$f \circ \varphi_v^{-\tau} = \exp(-\tau\, D_v) f = \left(\prod_{j=1}^{n} \exp(-\delta\tau D_v) \right) f \ ,$$

where $\delta\tau = \tau/n$. Notice that we use the *same* vector field v in the product and the above relation holds up to machine precision. However, the gradient also changes in a way that is not equivalent in the discrete setting, i.e.,

$$\frac{\partial}{\partial v} f_t = -\frac{\delta\tau}{k+1} \sum_{r=1}^{n} \exp\left(-(n-r)\delta\tau\, D_v\right) \overline{D}_{f_{(r-1)\delta\tau}} \sum_{s=0}^{k} \exp\left(\frac{s\,\delta\tau}{k}\, \mathrm{ad}_v\right) \ .$$

The main difference in the above expression compared to Eq. (5.9) is that we advect for a range of times instead of advecting only for the end time, and, in particular, advection for shorter times contributes to the computation.

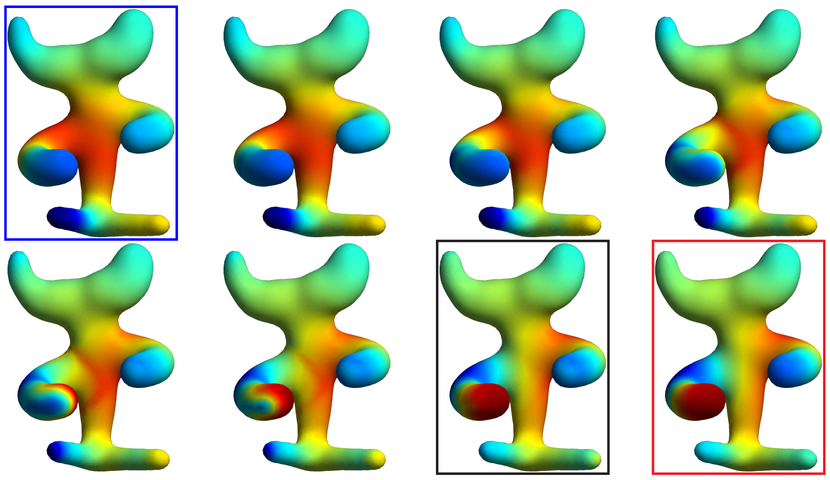


Figure 5.7: Interpolation results between the eigenfunctions 7 and 8 of the Laplace–Beltrami operator (blue and red frames). Notice that our result (black frame) highly matches the target function and that the interpolation path is smooth.

Table 5.1 shows the parameters we used in all of our experiments. For small deformations, taking $\tau = 1$ is sufficient to achieve good results, whereas large deformations require larger τ. Increasing the parameters α and β corresponds to smoother fields and functions, respectively. Finally, decreasing σ weighs the matching constraint higher with respect to the vector field constraint. In practice, we employ a "cooling" procedure for σ (denoted with asterisk in the table) to achieve better matching, i.e., σ is divided by 10 per a fixed amount of iterations.

5.8 Results

Spatial interpolation of functions. Fig. 5.3 shows an intuitive spatial interpolation between a large smooth Gaussian (blue frame) and two small smooth Gaussians (red frame). Our method yields a result (black frame) which matches the target function, where the rest of the frames are obtained by advecting the source over the resulting velocity field for different times. While our result is similar in nature to those obtained with optimal transportation techniques [SDGP$^+$15], we stress that our method is not

Figure	τ	α	β	σ	N	n
Fig. 5.1	1	1	0	$1e-2^*$	–	10
Fig. 5.3	2	$1e+3$	$1e-5$	$1e-4$	–	40
Fig. 5.7	1	$1e+3$	0	$1e-1^*$	20	–
Fig. 5.8	10	$1e+3$	0	$1e-2^*$	–	200
Fig. 5.9	1	1	$1e-4$	$5e-3$	–	20
Fig. 5.10	1	$1e+3$	0	$1e-2^*$	–	20
Fig. 5.11	1	1	0	$1e-2^*$	10	–
Fig. 5.12	1	1	0	$1e-2^*$	10	–

Table 5.1: The parameters used in our experiments. See the text for details about the effect of each parameter on the obtained results.

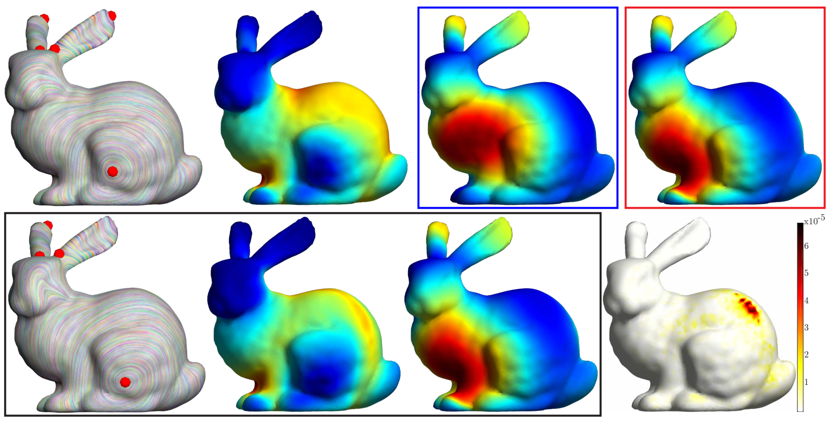


Figure 5.8: Our method takes as input a function (blue frame) and its advected version (red frame) computed using a vector field (top left). The output of our method is the matched function (black frame, right) and the corresponding vector field (black frame, left and middle). Notice that the resulting field highly matches the original field and the error between the function (bottom right) is very small. See the text for additional details.

designed for probability distributions. In particular, our differential operators and their integrated versions do not exhibit a *maximum principle*, i.e., the advected functions are not guaranteed to be probability distributions, even if they originated from a probability distribution. Nevertheless, the drift can be minimized by taking a small σ parameter. Similarly, we show in Fig. 5.7 a smooth interpolation between the eigenfunctions 7 and 8 of the Laplace–Beltrami operator. Matching between eigenfunctions is important for improving maps between surfaces, as was shown in [COC15]. In addition, several mapping techniques rely on associating scalar geometric descriptors [OBCS$^+$12]. Therefore, our method can serve as a building block in such scenarios.

Optical flow on surfaces. In cases when the target function is the advected version of the source function, one common objective is to reconstruct the underlying vector field which generated the motion. For instance, in the context of optical flow, registration between consecutive frames of a movie allows to up-sample the given signal. In Fig. 5.8, we show optical flow on curved surfaces where the source function (blue frame) is advected to time $\tau = 10$ (red frame) over the velocity field (top left, showing its LIC visualization and top middle left, showing its norm). Our method matches the target function (black frame, right) with a vector field (black frame, left and middle) that is very close to the original field. In addition, we show the absolute error between the target function g and the matched function $f \circ \varphi_v^{-10}$, i.e., we compute pointwise $|g - f \circ \varphi_v^{-10}|$ (bottom right). Notice that both f and g are on the scale of 1, thus our matching exhibits significantly small error.

Continuous matching of symmetric surfaces. Given a surface and its symmetry map, our goal is to infer a one-parameter family of maps which continuously matches the surface to its symmetries. In Fig. 5.9, we take a source function (blue frame), map

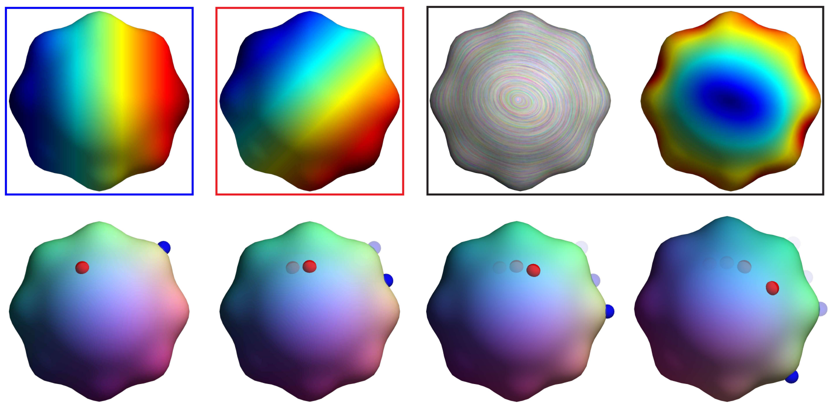


Figure 5.9: Our method handles large deformations between the source function (blue frame) and the target function (red frame), and succeeds in finding a single velocity field whose flow represents the continuous symmetry map (black frame). We show an RGB color coding of the surface mapped under the resulting flow for times $0, \tau/2, \tau$ and $2\,\tau$ (bottom, left to right).

it with the symmetry map (red frame), and we optimize for a single vector field which matches between those functions (black frame, shown with LIC and norm of the field). The bottom row shows an RGB color coding of the coordinate functions mapped with the flow map of the velocity. The initial geometry with two points marked on top of it (left) is advected to time $\tau/2$ (middle left) and then to time τ (middle right). Notice that the points and the coordinate functions are smoothly mapped to the correct values. We additionally show an *extrapolation* of the advection to time $2\,\tau$ (right), where some drift is noticeable, yet it is relatively small. Fig. 5.10 shows a similar experiment on a much more complex data since the geometry contains creases, thus it is not clear that the required vector field even exists. Yet, our method yields a reasonable continuous symmetric mapping which maintains the general behavior.

Function Matching for Mapping. The functional map framework [OBCS$^+$12] provides the basis for many mapping algorithms. In this framework, one can infer pose

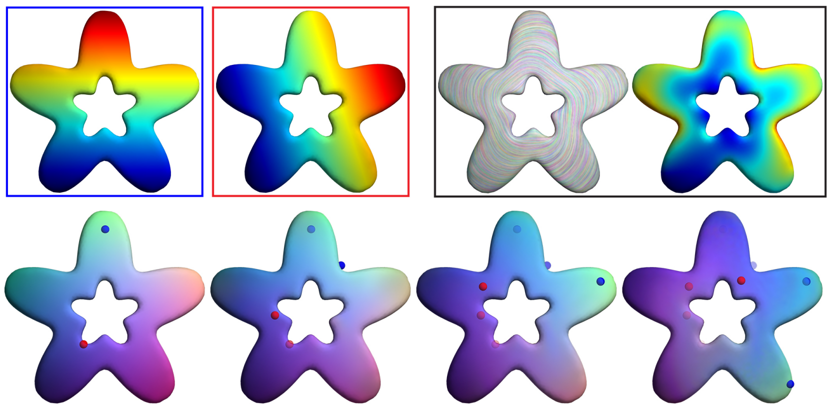


Figure 5.10: An experiment similar to one shown in Fig. 5.9 on a more complex geometry that contains creases. Nevertheless, our method finds a smooth field whose flow generates a continuous symmetric mapping.

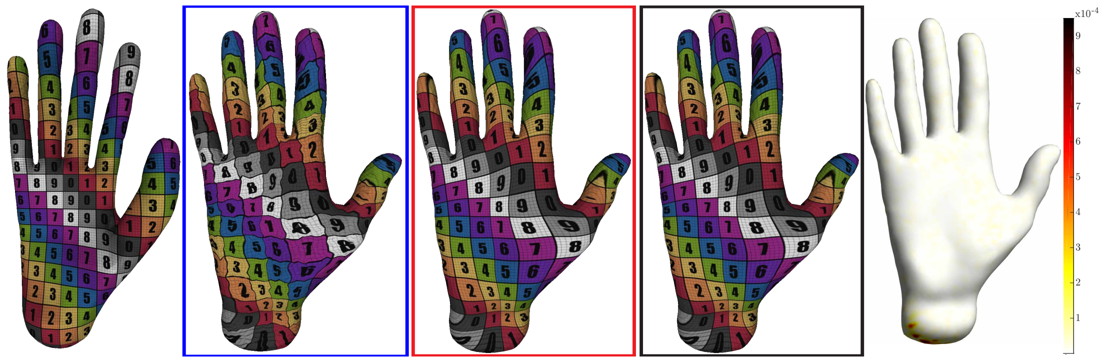


Figure 5.11: Matching between two non-smooth functions (blue frame) to two smooth functions (red frame) shown as texture coordinates. See the text for additional details.

constraints requiring functions to correspond, and compute a map taking functions to functions, which best fulfills these constraints. The effectivity of this framework is somewhat hindered by the fact that it is sometimes difficult to extract a corresponding point-to-point map if it is required. Using our framework, when working with *self-maps*, it is possible to leverage the functional map idea (mapping between corresponding functions), while *simultaneously* maintaining a point-to-point correspondence, defined using the flow map of the computed vector field. This approach was suggested in [COC15], yet as we work in the hat basis both for functions and vector fields, we are not limited to a small subspace of functions and vector fields.

We extend the setting using linearized advection of multiple vector fields to match multiple functions by summing over the errors. Figure 5.11 demonstrates that we can indeed match successfully between two non-smooth functions (blue frame, visualized as transported texture coordinates using a given map from the model on the left) to a smooth version of these functions (red frame, obtained by transportation with the corresponding functional map), and achieve the required functions (black frame). Moreover, we compute the pointwise sum of absolute errors between the target functions $\{g_i\}_{i=1}^2$ and the matched functions $\{f_i \circ \phi_v^\tau\}_{i=1}^2$, i.e., we plot the pointwise difference $\sum_i |g_i - f_i \circ \phi_v^\tau|$ (right). In Figure 5.12 we repeat this experiment on two models from the SCAPE dataset, using the functional map obtained from the ground truth correspondence to transport the functions we use as targets (which are the coordinate functions of the source mesh, two of them visualized as texture coordinates). Computing the flow using one function constraint (gray frame) and 3 function constraints (black frame) - we again match the target functions. Furthermore, we compute the point-to-point map corresponding to our flow and compare the error with respect to the ground truth, with the output of [COC15] on the same inputs. The resulting errors are shown in Figure 5.13, demonstrating that we improve the output point-to-point map.

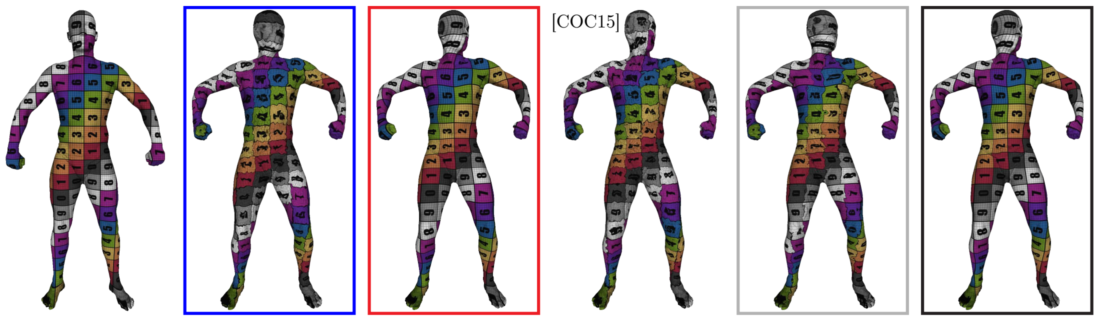


Figure 5.12: A similar experiment to Fig. 5.11 where given a set of non-smooth source functions (blue frame) and smooth target functions (red frame), we optimize for a single match (gray frame) and three matches (black frame). We compare our result to matching obtained with the method of [COC15] (bottom left).

5.9 Conclusion and Future Work

We have presented a novel method for matching scalar functions on curved triangle meshes that is based on the vector field's flow. We designed an energy minimization framework which is inspired by the well-known LDDMM algorithm and facilitates the machinery of functional vector fields. Our unique approach avoids the problematic explicit computation of flow lines and allows to advect in a linear and non-linear fashion. We showed that our matching method is applicable in scenarios of small and large deformations. We also demonstrated its effectiveness in optical flow problems, continuous matching of symmetric surfaces and in the context of the functional maps framework.

Numerous problems which are related to geometry processing can be posed as function matching problems. Thus, we believe that the generality of our framework will make it a valuable tool in many practical scenarios. In particular, we would like to extend our machinery to match and interpolate between tangent vector fields, a problem whose solutions are useful in fluid simulation techniques. Moreover, we would like to generalize our method for computing barycenters or weighted averages of scalar functions. Finally, we believe that our method can be also considered in the context of geometry-aware texture synthesis interpolation.

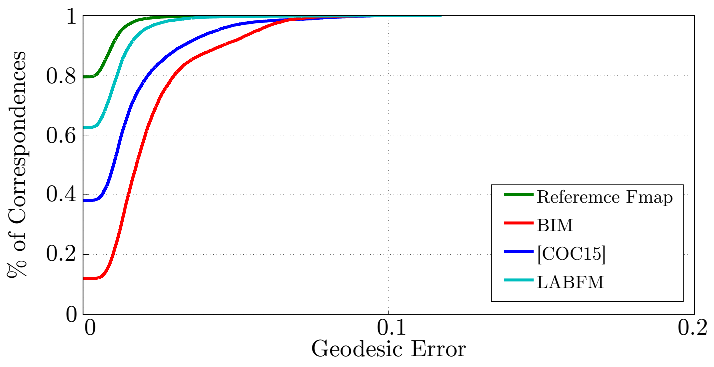


Figure 5.13: Our method matches only three functions on the data of Fig. 5.12, yet it provides a better reconstruction of the reference functional map when compared to the method [COC15].

Chapter 6

Functional Fluids on Surfaces

Fluid simulation plays a key role in various domains of science including computer graphics. While most existing work addresses fluids on bounded Euclidean domains, we consider the problem of simulating the behavior of an incompressible fluid on a curved surface represented as an unstructured triangle mesh. Unlike the commonly used Eulerian description of the fluid using its time-varying velocity field, we propose to model fluids using their vorticity, i.e., by a (time varying) scalar function on the surface. During each time step, we advance scalar vorticity along two consecutive, stationary velocity fields. This approach leads to a *variational integrator* in the space continuous setting. In addition, using this approach, the update rule amounts to manipulating functions on the surface using *linear operators*, which can be discretized efficiently using the recently introduced functional approach to vector fields. Combining these time and space discretizations leads to a conceptually and algorithmically simple approach, which is efficient, time-reversible and conserves vorticity by construction. We further demonstrate that our method exhibits no numerical dissipation and is able to reproduce intricate phenomena such as vortex shedding from boundaries.

6.1 Introduction

Fluids are fascinatingly complex and challenging to simulate, with applications ranging from aerodynamics and meteorology to special effects in computer animation, to name just a few. While fluids in Euclidean domains have been extensively studied in both

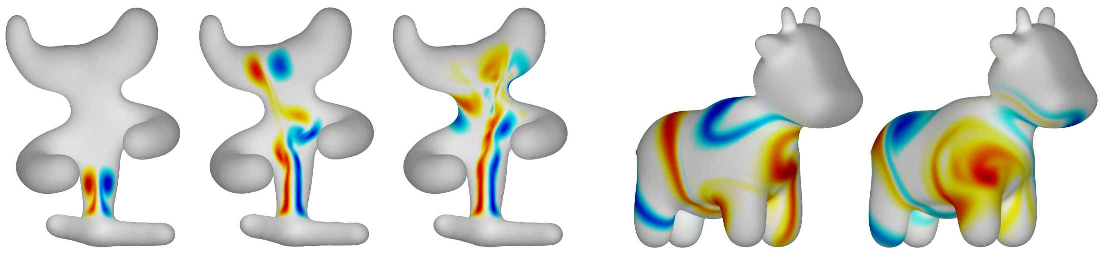


Figure 6.1: Jet flow (left) and shear layer flow (right) on curved surfaces.

computational fluid dynamics and computer graphics [Bri15], fluid simulation on *curved* surfaces has mostly been limited to special cases (e.g., spheres) or particular surface representations, such as subdivision surfaces [Sta03], and practical numerical simulation methods are scarce. This is unfortunate, as simulation of fluids on surfaces has practical value in a variety of domains, including, e.g., atmospheric research [MWC92], the investigation of liquid crystal films [CM05, Bae12], and the entertainment industry.

The main obstacle to adopting successful numerical algorithms from Euclidean domains to surfaces stems from the fact that a fluid is most often represented by its velocity field, and the equations governing the behavior of the physical system require computing derivatives of vector fields, which is challenging on a discrete surface.

Many fluids are naturally *incompressible*, i.e., the flow preserves the volume of the fluid. In this case the flow can be represented by its *vorticity*, given by the *curl* of the velocity field. In general, vorticity is a vector field that describes the local spinning motion of the fluid. On two dimensional domains, such as surfaces in 3D, vorticity can be represented as a (time-varying) *scalar function*. This change of perspective significantly simplifies the analysis and simulation of a fluid, since its behavior can be succinctly described using linear operators that act on real-valued functions on the surface.

Although this fact is well known [Saf92], it has, somewhat surprisingly, received little attention in the context of designing numerical methods for simulating fluids on surfaces. We make use of this formulation in order to construct a time integrator for vorticity on smooth surfaces, which is solely based on first principles of vortex dynamics. Our time symmetric advection scheme is intuitive and easy to implement; yet, it turns out to be *variational*, i.e., belonging to the class of structure preserving Lie group integrators for so-called Lie–Poisson systems [MV91, BS99, MPS99], which can be described in analogy to rigid body dynamics. Thus our method preserves momentum (i.e., vorticity) exactly, despite being of low numerical order. This in turn leads to a method that is qualitatively correct, numerically stable, and largely independent of the chosen time step.

Our resulting integration scheme is based on updating the scalar vorticity function in time. It involves the *push-forward* or *advection* of vorticity along the flow lines of a given vector field. Unlike existing methods which require the explicit computation of the flow lines of a vector field on a surface, we show that this advection can be simply computed as a product of a matrix exponential with a vector in the discrete setting, by leveraging the recently proposed functional framework for vector fields [ABCCO13] and mappings [OBCS$^+$12]. As we show in this paper, this change of viewpoint considerably simplifies the implementation and improves the accuracy of our method.

We demonstrate that our results on Euclidean domains are comparable with existing fluid integrators, while being conceptually simple and straightforward to implement. Furthermore, we describe various experiments where our simulation reproduces the results of analytically derived configurations (for example, spherical solutions), validating its numerical fidelity. Finally, we use our method for simulating flow near the inviscid limit, including effects from vortex shedding at boundaries.

6.1.1 Related Work

The research dedicated to computational fluid dynamics fills numerous books, and a complete survey is beyond the scope of the paper. We thus restrict the discussion of related work to Eulerian methods on compact *two dimensional* manifolds.

Velocity-based approaches. A fluid on a Euclidean domain can be modeled by a time varying vector field by representing the components of the vector field as functions on the domain (see, e.g., [Sta99]). This approach does not immediately generalize to curved surfaces, however, where the representation of vector fields using coordinates is problematic. One option is to use global or local surface parameterizations [LWC05, HAW$^+$09], which may introduce undesired distortion, or to restrict to special cases, such as subdivision surfaces [Sta03]. A pioneering approach for fluid simulation on general triangle meshes was suggested by Shi et al. [SY04], and later extended to deforming surfaces in [NMZ07]; however, these methods require explicit computation of flow lines, which is a challenging task. A related method [FZKH05] used an unstructured Lattice Boltzmann Model to simulate fluid behavior on triangulated surfaces by considering interactions between mesh vertices. This approach, however, also requires an explicit representation for the fluid velocity, unlike our method which relies on manipulating real-valued functions. Auer and colleagues [AMT$^+$12] proposed a different technique based on simulating the flow on a surrounding Euclidean grid and projecting it onto the surface using the Closest Point Method. While simple and efficient, this approach requires a careful construction of the grid, whereas our method works directly on the triangle mesh itself.

Vorticity-based approaches. Discrete Exterior Calculus (DEC) approaches have been developed for simplicial manifolds. In principle, by avoiding parameterization, these approaches provide a natural framework for simulating fluid flow. One of the first methods to adopt this perspective was proposed by Elcott and colleagues [ETK$^+$07], using the vorticity formulation of incompressible fluid flow. Their method preserves circulation, while ours preserves vorticity. We improve on their work by avoiding the computation of flow lines of the velocity vector field, which tends to be both challenging to implement and numerically unstable for triangle meshes. Additionally, our variational approach does not suffer from significant energy dissipation.

Another vorticity based method is proposed by De Witt and coleagues [DWLF12], who use the eigenfunctions of the Laplacian for accelerating the computation, and in order to avoid the somewhat costly Poisson step when computing the velocity from the vorticity. While this method could potentially be extended to curved surfaces by using the eigenfunctions of the Laplace-Beltrami operator, such an approach would require a large number of eigenvectors to correctly represent a detailed flow, which would be prohibitively costly for large models.

Several existing methods exploit the principles of DEC in combination with structure preserving variational time integrators [PMT$^+$11, MCP$^+$09]. We improve on these works by exploiting additional structure that is only available in two dimensions: the real-valued vorticity function. As a consequence, our approach requires only about a fifth of the number of unknowns (vorticity vs. flux and pressure). Further, our formulation avoids the computation of the Lie–Poisson bracket of vector fields, which is used to express the *time continuous* fluid motion on smooth surfaces, but is difficult to discretize. Indeed, different from [PMT$^+$11, MCP$^+$09], by *first* discretizing time and *then* space, we circumvent this difficulty and the attendant need of projecting back onto the subspace of divergence-free vector fields. Then, using the framework of functional vector fields on discrete surfaces [OBCS$^+$12, ABCCO13], the *spatial discretization* is straightforward in our setup, in particular by avoiding the costly computation of the flow lines of a vector field on a surface for advecting the vorticity function.

Thus while being intrinsic, intuitive and easy to implement, our method is variational, and thus preserves many structural properties of the flow even for large time steps.

Our method is time reversible and exhibits no numerical diffusion. Additionally, it conserves vorticity by construction. As a consequence, we *automatically* obtain correct vortex shedding. This contrasts our method with (i) Lagrangian frameworks, where vortex shedding is modeled by adding fractions of the boundary layer vorticity to the flow [PK05, WP10] and (ii) Eulerian approaches where vortex shedding is hampered by numerical diffusion, requiring additional constructions, for instance using precomputed boundary layers [PTSG09] or hybrid approaches [ZLCW13].

6.2 Fluid Flow on Surfaces

The motion of an incompressible and inviscid fluid on a two dimensional Riemannian manifold $(\mathcal{M}, \langle ., . \rangle)$ is described by a time-dependent, divergence-free velocity vector field ν_t, whose time evolution is governed by Euler's equation. Here we discus Euler's equation in its *vorticity* formulation, since this is the formulation that we work with. Before introducing our temporal and spatial discretizations below, we first present the time and space continuous setting. For further details on the vorticity formulation of Euler's equation, we refer to e.g., [Saf92, CMM90].

Fluid velocity. If we restrict our attention to a two dimensional manifold $\mathcal{M}$, then the divergence free velocity vector field ν_t that characterizes the fluid motion has a simple representation in terms of a time varying scalar function σ_t, known as the *stream* *function.* This relationship is given by

$$\nu_t = -\mathcal{J}\nabla\sigma_t + \eta_0 \ , \tag{6.1}$$

where ∇ is the usual gradient operator acting on functions, $\mathcal{J}$ denotes a (pointwise) rotation of a vector field by $\pi/2$ in each tangent plane, and η_0 is a time constant *harmonic* vector field (i.e., both divergence and curl-free). If the manifold has a boundary, then we further require that σ_t vanishes identically on it. Notice that (6.1) corresponds to the Hodge–Helmholtz decomposition of ν_t, using zero boundary conditions for σ_t.

The curl of the velocity vector field,

$$\omega_t = \operatorname{curl} \nu_t \ , \tag{6.2}$$

is known as *vorticity.* While in 3D domains vorticity is itself a vector field, for two dimensional manifolds it can be represented by a *scalar vorticity function.* In this case, the stream function and vorticity are related through

$$\omega_t = -\Delta\sigma_t \ , \tag{6.3}$$

where Δ is the Laplace–Beltrami operator. Note that ν_t is defined by ω_t up to the harmonic component η_0 and, for closed surfaces, σ_t is defined by ω_t up to an additive constant.

Vortex dynamics. The fluid motion is governed by Euler's equation, most compactly expressed in *vorticity form*,

$$\dot{\omega}_t = -\langle \nabla\omega_t, \nu_t \rangle \ , \tag{6.4}$$

where $\dot{\omega}_t = \frac{d}{dt}\omega_t$ is the time derivative of the vorticity function. A direct consequence of this equation is that ω_t is *frozen-in*, i.e., it is transported in the same way as fluid particles (see e.g. [Dav15, pg. 49]). The velocity field can be recovered from vorticity by first computing the stream function σ_t using the *linear* equation (6.3) and then plugging the result into (6.1). Hence, Eq. (6.4) can be viewed as an evolution equation for both ω_t alone and for the whole fluid motion.

Viscosity. So far we have assumed the fluid to be inviscid, i.e., that there is no energy loss due to viscous friction. Nonetheless, the *limit* of zero viscosity yields the dynamical and visual complexity of fluids, such as smoke. The assumption of zero viscosity differs, however, from the limit of zero viscosity: in the absence of viscosity there exists no mechanism for the creation of vorticity, while in the limit vorticity is created through vortex shedding from boundary layers.[1] While the above exposition only treats the inviscid case, viscosity is readily incorporated into the equations of

[1]This insight explained *d'Alembert's paradox*, which predicts vanishing drag on bodies moving with constant velocity [AJ05].

motion using Arnold's observation [AK99] that viscosity can be regarded as an external force acting on the fluid,

$$\dot{\omega}_t = -\langle \nabla \omega_t, \nu_t \rangle - \rho \Delta \omega_t \ . \tag{6.5}$$

Note that this equation represents the vorticity form of the general *Navier–Stokes* equation of fluid motion (in the absence of additional external forces), where ρ is scalar representing the kinematic viscosity. We return to viscosity later in our exposition and for now confine the discussion to the inviscid setting.

Flows of divergence-free vector fields. A key aspect of our method is the evolution of the vorticity function along the flow-lines of the fluid's velocity field. For this, we recall the notion of the flow of a time-varying vector field.

Given a time-dependent velocity field ν_t, its *flow* $\varphi_t(p)$ describes the position after time t of a particle that starts at a point p at time 0. Formally, the flow is defined via

$$\dot{\varphi}_t(p) = \nu_t(\varphi_t(p)), \quad \varphi_0(p) = p, \tag{6.6}$$

for all $p \in \mathcal{M}$. Thus $\varphi_t(p)$ defines a curve on $\mathcal{M}$, and $\dot{\varphi}_t(p)$ is its tangent vector at the point $\varphi_t(p)$.

The flow φ_t is an invertible self-map on $\mathcal{M}$, i.e., $\varphi_t : \mathcal{M} \to \mathcal{M}$, and as such it can also be used to transport real-valued functions on $\mathcal{M}$. In particular, the flow φ_t acts linearly on smooth functions $f \colon \mathcal{M} \to \mathbb{R}$ through the pushforward:

$$\Phi_t(f) = f \circ \varphi_t^{-1} \ . \tag{6.7}$$

Note that Φ_t is a linear operator acting on real-valued functions defined on $\mathcal{M}$. In terms of the flow of the velocity field, vorticity satisfies

$$\omega_t = \Phi_t(\omega_0) \ , \tag{6.8}$$

where $\omega_0 = \operatorname{curl} \nu_0$ and Φ_t is the pushforward of the flow φ_t associated with ν_t. Physically, this resembles the well known fact that vorticity is advected along the fluid flow.

We introduce the flow as a conceptual tool here. However, our implementation does not require an explicit calculation of the flow. Indeed, one of our key observations is that discretizing Eq. (6.8) directly is much simpler than computing the flow φ_t, and can be done efficiently through a simple matrix exponential, by utilizing the recently proposed functional representation for vector fields [ABCCO13], and mappings [OBCS$^+$12]. In this framework, Φ_t is referred to as the *functional map* that corresponds to the flow φ_t.

6.3 Time Discretization

When discretzing time (but not yet space), we seek to determine from an initial vorticity ω_0, a sequence ω_i that *exactly* respects the defining properties of ideal fluid flow (Eqs. (6.1), (6.3), and (6.8)), independently of the time step. We achieve this by introducing a sequence of time-discrete flow updates, resembling the time-continuous setting. Indeed, notice that Eq. (6.8) in the time-continuous case implies that $\omega_{t+s} = \Phi_{t+s}(\omega_0) = \Phi_t(\omega_s)$, suggesting that the time-discrete flow update can be performed incrementally. Notice further that the identity $\Phi_{t+s}(\omega_0) = \Phi_t(\omega_s)$ implies $\Phi_{-t}(\omega_t) = \omega_0$, which is known as *conservation of vorticity.*

Accordingly, we define in the time-discrete case:

Vorticity conservation: Each ω_{i+1} is obtained by pushing forward ω_i, i.e., for two consecutive ω_i, ω_{i+1} there is a functional update map $\Phi_{i\to i+1}$ such that

$$\omega_{i+1} = \Phi_{i\to i+1}(\omega_i) \ . \tag{6.9}$$

It remains to specify the exact structure of the linear operators $\Phi_{i\to i+1}$, which we do as follows:

Self advection: The update $\Phi_{i\to i+1}$ is obtained by the flows of the (time-constant) divergence free velocity fields ν_i and ν_{i+1}, which are (linearly) coupled with vorticity via $\omega_{i+1} = \operatorname{curl}\nu_{i+1}$ and $\omega_i = \operatorname{curl}\nu_i$. Namely, we first flow on ν_i for a fraction $1-s$ of the time step, and then on ν_{i+1} for a fraction s of the time step. Hence, for $s \in [0,1]$ we require

$$\Phi_{i\to i+1} = \Phi_s^{i+1} \circ \Phi_{1-s}^{i} \ , \tag{6.10}$$

where Φ_s^i is the functional representation of the flow of ν_i for time $t = s\,h$ for a given time step h (see Fig. 6.2).

Combining Eqs. (6.9) and (6.10) we obtain a one-parameter family of time integrators,

$$\Phi_{-s}^{i+1}(\omega_{i+1}) = \Phi_{1-s}^{i}(\omega_i) \ . \tag{6.11}$$

Note that Eq. (6.11) is a non-linear implicit equation for ω_{i+1} since Φ^{i+1} depends non-linearly on ω_{i+1}. We describe a way to solve this equation in practice in Sec. 6.4, in particular using Eq. (6.16), which considers the explicit dependence of Φ^{i+1} on ν^{i+1} (and thus ω^{i+1}) in the discrete setting.

Two particular choices of s stand out: For $s = 0$ we obtain the explicit update equation

$$\omega_{i+1} = \Phi_1^i(\omega_i) \ ,$$

which gives rise to a particularly efficient (but less accurate and stable) implementation. Instead, in order to maintain *structure preservation*, we work with $s = 1/2$ to obtain an

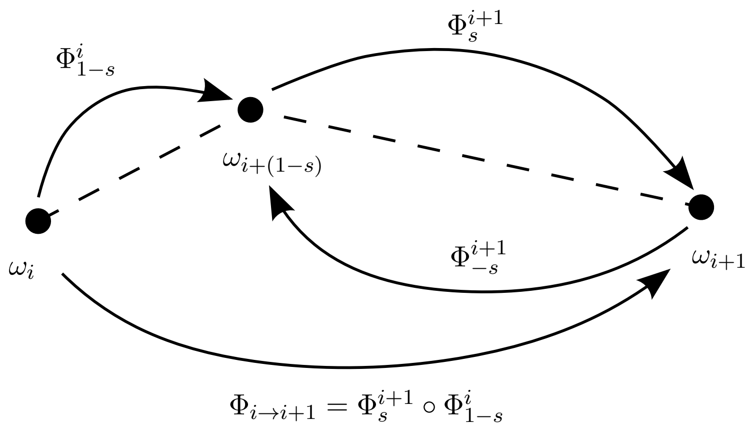


Figure 6.2: The new vorticity ω_{i+1} is obtained by first advecting ω_i along ν_i for a fraction $1-s$ of the time step, and then along ν_{i+1} for the remainder of the time step. This is equivalent to matching the forward advected ω_i (along ν_i) with the backward advected ω_{i+1} (along ν_{i+1}).

implicit, *time-reversible* trapezoidal scheme,

$$\Phi^{i+1}_{-1/2}(\omega_{i+1}) = \Phi^i_{1/2}(\omega_i)\ . \tag{6.12}$$

In summary, the above update method computes ω_{i+1} from ω_i through the flows $\Phi^{i+1}_{-1/2}$ and $\Phi^{i+1}_{1/2}$ of the two stationary vector fields ν_{i+1} and ν_i. We stress again that in our implementation we avoid explicit computation of flows, as explained further below.

Note that the reversible nature of the time-integrator immediately implies that there is no loss of information in between time steps, i.e., our algorithm has no numerical diffusion. As we further conserve vorticity by construction, we achieve plausible and stable fluid simulations even for large time steps over long simulation periods.

Discussion. Our method is a trapezoidal rule and thus second order in time. Apart from the invariants enforced by construction, our time discretization also preserves the Hamiltonian structure of ideal fluid flow in continuous space. In fact, our method can be derived along the lines of [BS99] using a suitable discrete Lagrangian and thus belongs to the class of structure preserving Lie group integrators for Lie–Poisson systems [MV91, BS99, MPS99]. Note, however, that this depends crucially on the fact that the space of smooth functions on $\mathcal{M}$ carries a so-called Poisson structure. In the spatial discretization we are unaware of such a structure, leading to a method which is no longer Poisson but still variational, similar to existing variational fluid integrators on spatial discretizations [PMT$^+$11].

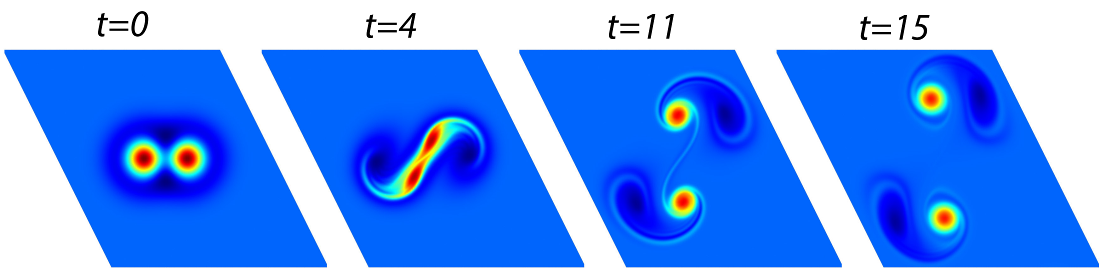


Figure 6.3: Taylor vortices in the plane with periodic boundary conditions. Compare with [MCP$^+$09, Fig. 4].

6.4 Spatial Discretization

In the discrete case, we are given a triangle mesh $\mathscr{M} = (\mathscr{V}, \mathscr{E}, \mathscr{F})$, and need to represent scalar functions, vector fields, and the corresponding operators that map between them. In addition, we require a spatial discretization for the advection operators Φ_s^i.

We represent real-valued functions as scalars on the vertices of the mesh, i.e., $f\colon \mathscr{V} \to \mathbb{R}$, and extend them to the whole surface using the standard piecewise linear hat basis functions. We also consider vector fields as being piecewise constant on the faces of the mesh, i.e, $\nu\colon \mathscr{F} \to \mathbb{R}^3$, s.t. each ν is parallel to the plane spanned by face i.

Differential operators. We use two sets of functions as the representatives of our vector fields: the stream functions σ_t, and the vorticities ω_t. To mimic the continuous case, we require operators ∇, curl, div, and Δ to be such that the following relationships hold:

$$\begin{gathered} \nu_t = -\mathcal{J}\nabla\sigma_t + \eta_0, \quad \omega_t = \operatorname{curl}\nu_t \\ \Downarrow \\ \operatorname{div}\nu_t = 0, \quad \omega_t = -\Delta\sigma_t\ . \end{gathered} \tag{6.13}$$

Remarkably, the standard operators used in the literature fulfill these properties (as is shown in [PP03]), making spatial discretization straightforward in our setting.

Functional operators. In the time-continuous setting, vorticity is pushed forward from the initial vorticity ω_0 using the (functional representation) Φ_t of the flow of ν_t. In the time-discrete setting, where the vector fields ν_i are stationary in between time steps, we can in principle directly compute the flow of ν_i and advect ω_i along this flow. This computation, however, is both costly, difficult to implement and is prone to instabilities. Instead, by considering the recently proposed functional representation of vector fields [ABCCO13] we show that the advection of vorticity can be done directly without computing the flow.

In particular, we follow [ABCCO13] to represent vector fields by their action on scalar functions. Namely, given a vector field ν, the authors propose to consider the

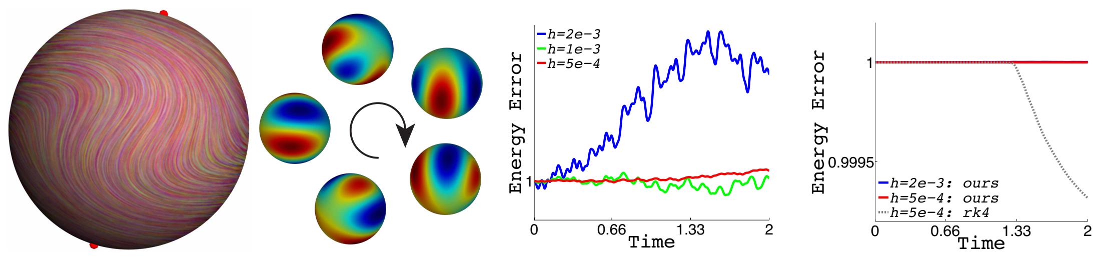


Figure 6.4: When the vector field on the left is used as initial condition, the corresponding vorticity simply rotates on the sphere (middle left). The graph (middle right) shows the relative kinetic energy of the vector field at time t compared to the initial energy, for different time steps h. The maximum change is on the order of 10^{-5}. Compared to the Runge-Kutta time integrator our method is more stable for a longer time with a larger time step (right).

associated derivation operator, given by:

$$\mathcal{V}(f) = \langle \nabla f, \nu, \ \rangle \, . \tag{6.14}$$

Following [ABCCO13], we discretize Eq. 6.14 so that given a scalar function f on the vertices of the mesh, $g = \mathcal{V}(f)$ is another such function, whose value at vertex i is given by:

$$g_i = \frac{1}{\sum_{j \in N(i)} A_j} \sum_{j \in N(i)} \langle \nabla f_j, \nu_j, A \rangle_j \ , \tag{6.15}$$

where the sums run over all faces *adjacent* to vertex i, ∇f_j denotes the value of ∇f in face j, and A_j is the area of the j^{th} face. The resulting linear operator is given by a sparse matrix of size $|\mathscr{V}| \times |\mathscr{V}|$, which is obtained by applying $\mathcal{V}$ to the piecewise linear hat basis functions. In the following we identify $\mathcal{V}$ with this matrix.

The main advantage of representing vector fields as linear operators acting on functions in our setup is that (the functional representation of) the flow of a stationary vector field ν is simply given by the matrix exponential of $\mathcal{V}$ (see [ABCCO13, Lemma 2.5]). Thus, the functional map corresponding to the flow of ν can be computed directly from $\mathcal{V}$ via

$$\Phi_s = \exp\left(s\, h\, \mathcal{V}\right) \ . \tag{6.16}$$

Furthermore, advecting a function f with the flow Φ_s of ν, can be done simply by the matrix vector multiplication $\exp\left(s\, h\, \mathcal{V}\right) f$. Crucially, in the discrete setting, this product can be computed efficiently without evaluating the full matrix exponential, which can be both dense and difficult to approximate [AMH11]. We leverage this insight in the context of fluid simulation by using Equation (6.16) to derive an accurate and stable advection procedure, which circumvents the need to compute the full flow of the velocity field.

This leads to the following space-discrete version of Eq. (6.12):

$$\exp\left(-\frac{h}{2}\mathcal{V}_{i+1}\right) \omega_{i+1} = \exp\left(\frac{h}{2}\mathcal{V}_i\right) \omega_i \ . \tag{6.17}$$

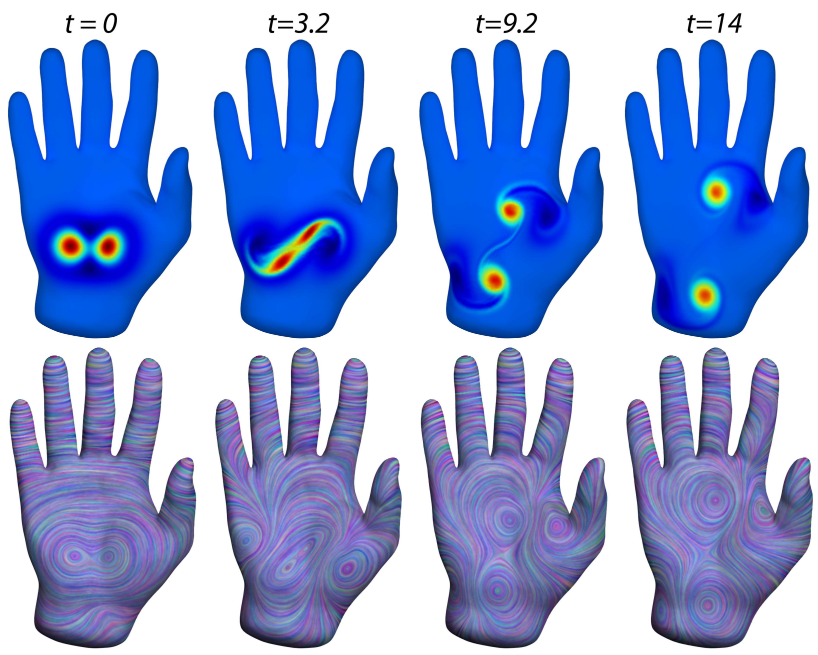


Figure 6.5: Taylor vortices on curved surfaces exhibit the same qualitative behavior as in the plane.

Notice that the above equation is an *exact* integration of ω advected along piecewise constant flows. This is in contrast to approaches advecting velocity [MCP$^+$09, PMT$^+$11], where by construction only low order approximations can be used. We practically observe that for our method a first order approximation of the exponential map is sufficient and does not decrease simulation quality. This amounts to time integration using the trapezoidal rule in the spatial discretization, preserving second order accuracy from the space continuous setting. Since, to our knowledge, a proper spatially discrete Poisson structure is missing, our time integrator may no longer be Lie–Poisson in this case. Nonetheless, it is still variational, time reversible, and vorticity conserving.

6.5 Implementation

Our temporal and spatial discretizations lead to a simple algorithm, which evolves the vorticity in time, by computing ω_{i+1} from a given ω_i, so that the whole fluid motion is obtained from an initial vorticity ω_0. Below we discuss implementation details required for making our method practical and efficient.

Solution of the non-linear equation. The update rule we suggest in Eq. (6.17) is a non-linear equation for ω_{i+1}, since $\mathcal{V}_i$ and $\mathcal{V}_{i+1}$ depend (linearly) on ω_i and ω_{i+1} through $\omega_i = \operatorname{curl}\nu_i$ and $\omega_{i+1} = \operatorname{curl}\nu_{i+1}$, respectively. For an efficient solve, we (i) express ω_{i+1} in terms of the stream function σ_{i+1} and (ii) use the first order approximation

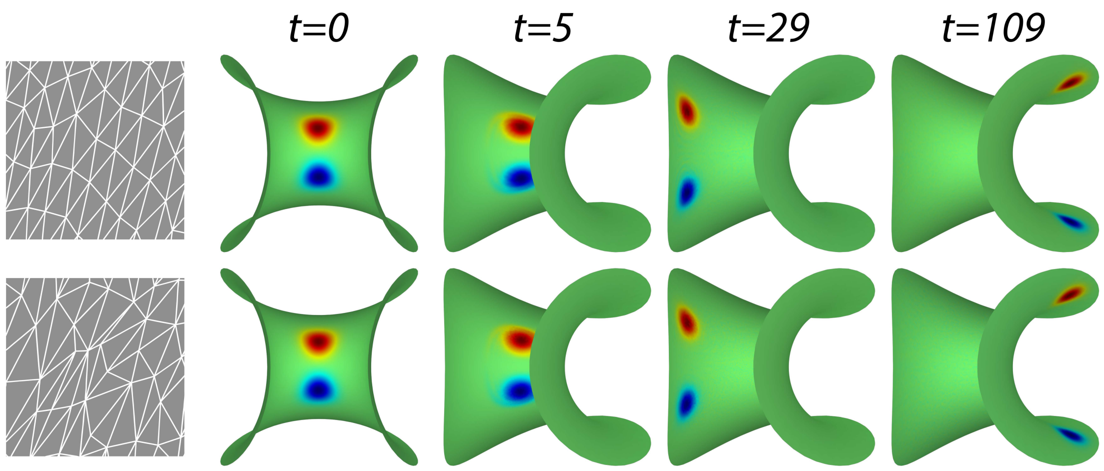


Figure 6.6: A pair of vortices with equal but opposite strength on a hyperbolic surface. The vortices move to the boundary where they separate, travel independently along the boundary, and join again at the opposite side. The bottom row shows the same experiment on a poor triangulation, emphasizing the robustness of our method to the underlying mesh. This experiment is with zero viscosity. Compare with Fig. 6.11 for similar initial conditions, with *non zero* viscosity, which yields vortex shedding from the boundary.

$\exp(\varepsilon A) \approx I + \varepsilon A$ of the matrix exponential. That is, we solve

$$-\left(I + \frac{h}{2}\mathcal{V}_i\right)\omega_i = \left(I - \frac{h}{2}\mathcal{V}_{i+1}\right)\Delta\,\sigma_{i+1}\ , \tag{6.18}$$

which is quadratic in σ_{i+1}, and then recover ω_{i+1} and ν_{i+1} using Equations (6.13): $\omega_{i+1} = -\Delta\sigma_{i+1}$, $\nu_{i+1} = -\mathcal{J}\nabla\sigma_{i+1} + \eta_0$, where η_0 is computed from ν_0. This formulation allows for deriving an analytic expression of the attendant Jacobian as a sparse matrix—an essential feature for efficiency. This, in turn, allows for using a standard Gauss–Newton solver, which typically converges in two to five iterations. As an initial guess for the solver, we use the one-point quadrature $-\Delta\sigma_{i+1} \approx \exp(h\mathcal{V}_i)\,\omega_i$, which can be computed efficiently [AMH11].

Viscosity. So far we have only discussed inviscid fluids. However, as explained above, the treatment of viscosity can be readily integrated into our method. Adding the viscous force to both sides of the update equation (6.18) leads to

$$-\left(I + \rho\frac{h}{2}\Delta\right)\left(I + \frac{h}{2}\mathcal{V}_i\right)\omega_i = \left(I - \rho\frac{h}{2}\Delta\right)\left(I - \frac{h}{2}\mathcal{V}_{i+1}\right)\Delta\,\sigma_{i+1}. \tag{6.19}$$

Adding viscosity not only allows for simulating complex physical phenomena, such as *vortex shedding*, as we explain in the next section, but also has numerical advantages. Indeed, various Eulerian methods suffer inherently from numerical dissipation, to the extent that makes it necessary to re-inject lost vorticity [FSJ01]. Other Eulerian methods with no numerical dissipation (such as [MCP$^+$09, PMT$^+$11] and ours) require explicit addition of viscosity in order to prevent discretization artefacts. Indeed, while in the

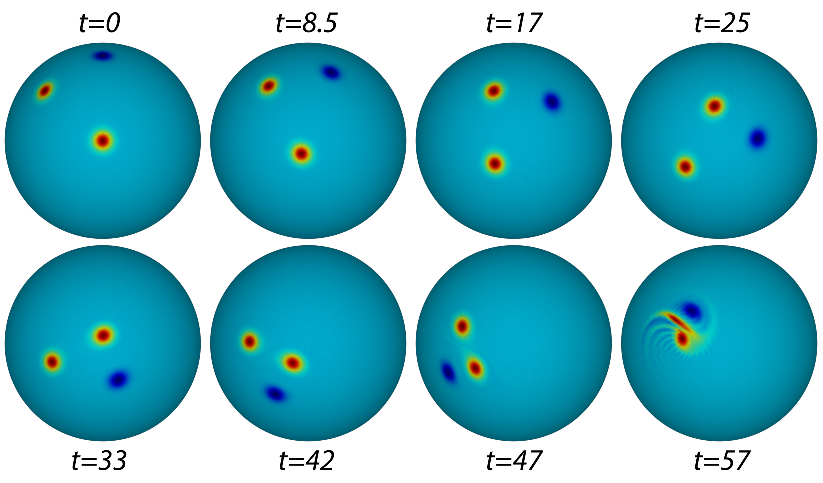


Figure 6.7: The above initial configuration on a sphere is known to collapse for singular point vortices [VL13]. Our method successfully reproduces this result.

spatially continuous case energy cascades to smaller and smaller scales [Cho94], the number of available frequencies in the spatially discrete setting is limited. Without viscosity, this results in accumulation in the highest available frequencies, leading to discretization artefacts.

Boundaries. The treatment of domains with boundary is straightforward in our approach. We solve Eq. (6.19) under the constraint that the stream function σ is zero along every boundary component. In practice we use a sparse matrix that maps functions from all mesh vertices to interior vertices (by simply ignoring boundary vertices). This matrix is applied to Eq. (6.19) as well as to its Jacobian, and we solve for inner vertices only using Gauss–Newton's method.

6.6 Evaluation

We have extensively evaluated our algorithm, with focus on numerical stability, energy behavior, and physical correctness. We have simulated known solutions on planar domains in order to compare our results with existing methods, and used flows with known analytic solutions on curved surfaces. Further, we have simulated a number of interesting flows on curved surfaces, including effects from vortex shedding at boundaries near the inviscid limit, which are inspired by previous work. All the results are also shown in the accompanying video. All figures, unless mentioned otherwise, show vorticity of the flow, color coded on the surface. In figures 6.4 and 6.5 the flow is additionally visualized using the method of [PZ11]. Finally, we investigated how the time varying operator $\mathcal{V}_t$ can be used to uncover global behavior of the flow.

Performance. We used a non-optimized MATLAB implementation, with a standard Gauss–Newton solver for Eq. (6.19), using the analytic, sparse Jacobian. In all our

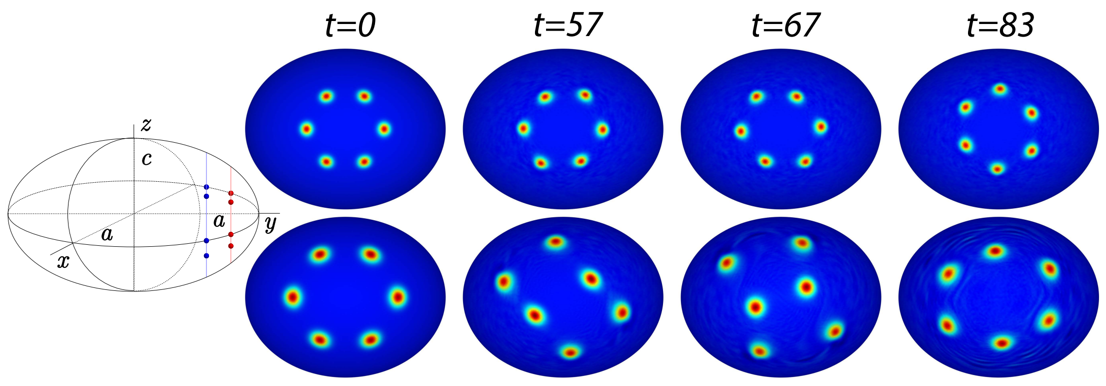


Figure 6.8: A ring of 6 vortices exhibits different behavior when placed on different parts of an oblate spheroid (left). The top row shows vortices placed closer to the tip of the spheroid (red dots in the illustration), and the bottom row shows the behavior for similar vortices placed closer to the $x - z$ plane (blue dots in the illustration).

experiments, the method was very stable and converged in $2-5$ Newton iterations (using a tolerance of 10^{-7}), depending on time step size and flow complexity. The experiments were performed on an Intel i7 processor, with 16 GB RAM. In our experiments the method scaled linearly with mesh size, and a single Newton iteration typically took around 1 second per 10K vertices.

6.6.1 Analytic solutions

Planar Taylor vortices. We first tested our method on the well-known Taylor vortices configuration on a planar Euclidean domain with periodic boundary conditions (see e.g., [MCP+09]). In this experiment, two Taylor vortices either merge or separate depending on their initial distance. Taking this distance to be just above the critical bifurcation threshold (i.e., the vortices should separate) provides an excellent test case for fluid simulation methods (see [McK07, Eq. (1.16)] for initial conditions). Our method reproduces this complex dynamical behavior and produces correct results as shown in Fig. 6.3 (compare with [MCP+09, Fig. 4]).

Rotating sphere flow. On the sphere an analytic solution exists for fluid flow, whose initial condition consist of the combination of a Killing vector field with a rotated gradient of an eigenfunction of the Laplace-Beltrami operator. In particular, it can be shown that the energy of *inviscid* flow with these initial conditions remains in the low frequencies, giving a periodic motion. This makes the configuration a good test case for energy conservation, and for validating qualitative behavior of our solver. Energy plots for different time steps are shown in Fig. 6.4. The relative change in energy is on the order of 10^{-5}. Note that while our method remains stable for a time step of $h = 2 \cdot 10^{-3}$, replacing our time integrator with a Runge-Kutta time integrator leads to significant energy loss for a much smaller time step.

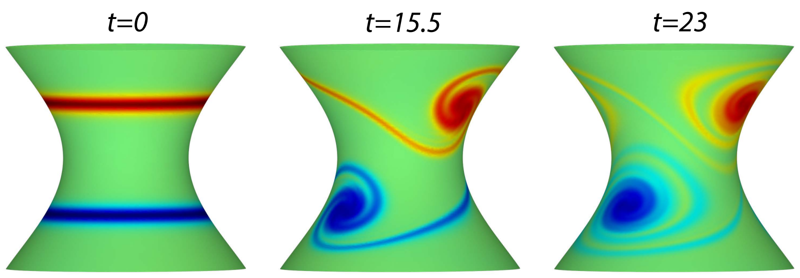


Figure 6.9: A double shear flow on a hyperboloid. Note the thinning of the shear layers, and the creation of vortices. See [SS13] for the reference behavior in the plane.

6.6.2 Vortex configuration on surfaces

Taylor vortices on a curved surface. We generated initial conditions similar to the Taylor vortices in the plane on a curved surface representing a hand. We used the same parameters as in [McK07, Eq. (1.16)], measured as geodesic distances (using [CWW13]) on the mesh. Fig. 6.5 shows frames from the animation, yielding the same qualitative behavior as in the plane.

Vortex pair. In the plane, two point vortices of equal and opposite strength translate along the orthogonal bisector of their connecting line. This can be viewed as a zero dimensional vortex ring, i.e., the 2D equivalent to a circular vortex filament in 3D. In Fig. 6.6 we demonstrate the same qualitative behavior on a hyperbolic surface. In the absence of viscosity the vortices travel towards the boundary, where they separate and move independently along the boundary, until they meet again at the opposite side. This configuration is extremely stable over very long simulation times (performing the periodic motion several times), demonstrating the absence of numerical diffusion, vorticity conservation, and excellent energy behavior of our method.

The qualitative behavior of other point vortex configurations is also reliably reproduced by our method. For instance, Fig. 6.7 shows a configuration on the sphere which is known to collapse [VL13].

Oblate sphere. It is known that N point vortices (with $N < 7$) on a round sphere stay in a stable relative equilibrium, when equally spaced along a latitude circle below a critical colatitude [VL13]. We demonstrate a similar configuration on a stretched (along one axis) sphere, where point vortices are replaced by Gaussian vortices. As shown in Fig. 6.8 in the top row, when placing the vortices around a "flat" pole below the critical angle (for point vortices on a round sphere), the flow preserves the six fold symmetry, as in on the round sphere. In the bottom row, the vortices are positioned *above* the critical angle (i.e. further from the tip), and symmetry breaks. Still, the vortices show periodic behavior, indicating that the corresponding point vortex configuration might be integrable. To our knowledge such configurations have not been studied analytically.

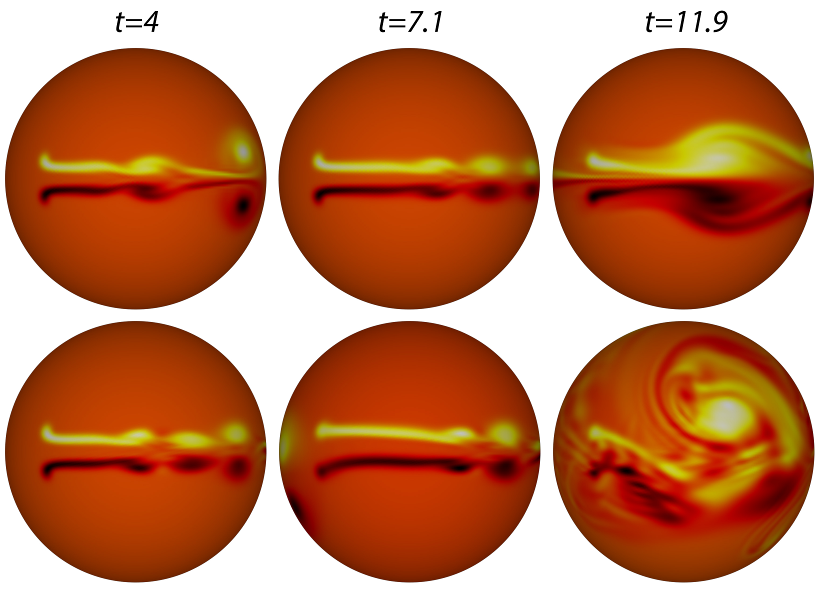


Figure 6.10: Two jet simulations on a sphere with a symmetric triangulation. On the top row the initial vorticity function shares the symmetry of the mesh. Our method preserves this symmetry, yielding a symmetric flow. On the bottom row the initial vorticity is no longer symmetric, yielding a more realistic turbulent flow. For such unstable flows the simulation is sensitive to the discretization of the initial vorticity.

Double shear flow. Two vortex layers of equal but opposite strength with small perturbations generate a vortical flow with a symmetric structure [SS13, Section 4.2]. We use similar initial conditions on the hyperboloid. The resulting flow exhibits the same qualitative behavior, including the intricate symmetries. Fig. 6.9 shows frames from the resulting simulation. Note how vortices curl up, creating thinner and thinner vortex lines, similarly to the reference behavior in the plane.

6.6.3 Turbulent flow and vortex shedding

Jet on a sphere. A jet is modeled by steadily injecting vorticity of equal but opposite strength at both sides of the jet's orifice. Fig. 6.10 shows two jet simulations on a symmetric triangulation of the sphere. In the top row, the initial vorticity function shares the symmetry of the mesh, and due to the stability of our method, this symmetry is exactly preserved by the flow. In the bottom row, the initial vorticity is not exactly symmetric with respect to the triangulation, which introduces instabilities in the flow, leading to a more realistic simulation. In general, unstable flows such as this are sensitive to the discretization of the initial conditions.

Vortex shedding. As explained before, our method naturally handles vortex shedding from boundaries. We used a pair of point vortices on the Enneper's surface, as in Fig. 6.6, but with non-zero viscosity. In contrast to the inviscid case, where the two vortices separate and travel along the whole boundary, with viscosity vortex shedding from the

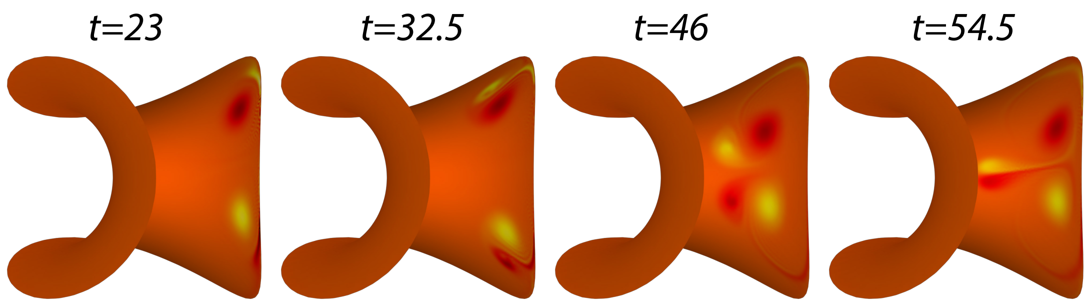


Figure 6.11: Two vortices collide with the boundary on the Enneper's surface. Note the details in shed vorticity generated by the boundary due to viscosity. Compare with Fig. 6.6 for similar initial conditions but zero viscosity.

boundary generates two additional small vortices which "trap" the original vortices and prevent them from circulating, see Fig. 6.11.

In the accompanying video we also show the wake behind a two dimensional cylinder on a round sphere, generated through vortex shedding.

6.6.4 Flow properties

Globally invariant functions. In addition to being instrumental in the computation of vorticity advection, the functional representation of vector fields of Azencot et al. [ABCCO13] also allows us to gain information about different properties of the flow, that would be difficult to obtain otherwise. Here, we briefly outline one such application and leave further exploration as future work.

Given the solution to a flow, we may be interested in finding regions of the surface that are invariant under the flow, i.e., regions from which the fluid does not leave or regions into which the fluid does not enter during the simulation. We relax this problem to consider all functions f such that $\Phi_t(f) = f$ for all t. In order to find such a function, note that if f is mapped to itself under the flow of a constant vector field ν, then $\exp(t\mathcal{V})f = f$ for all t, which means that $\mathcal{V}f = 0$, or equivalently that f is in the kernel of $\mathcal{V}$. Therefore, we are looking for a function f that is *simultaneously* in the kernels of all $\mathcal{V}_t$. Notice that this is the case if and only if f is in the kernel of $\sum_t \mathcal{V}_t^T \mathcal{V}_t$, where we are using that in the time-discrete case there exist only finitely many temporal sample points. Using this observation, we compute the kernel of $\sum_t \mathcal{V}_t^T \mathcal{V}_t$ for the rotating flow from Fig. 6.4 and the stable vortex ring from Fig. 6.8. The resulting functions in the respective kernels are shown in Fig. 6.12.

6.7 Conclusion

Building on the vorticity formulation of fluids, we presented a method for temporally and spatially discretizing the equations of fluid flow. The attendant time integrator for the inviscid case is variational, preserves vorticity exactly, is time reversible, and

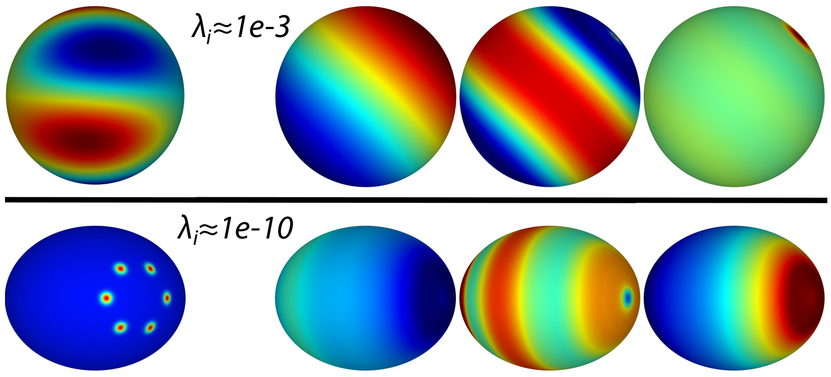

$\lambda_i \approx 1e\text{-}3$
$\lambda_i \approx 1e\text{-}10$

Figure 6.12: A function which is invariant to the whole flow, computed from the flow's kernel. Top row, for the flow from Fig. 6.4, and bottom row for the flow from Fig. 6.8.

does not exhibit numerical diffusion. Additionally, our approach allows for adding a principled treatment of viscosity, enabling the simulation of complex phenomena, such as vortex shedding, without any special or unphysical treatment of boundary layers. The resulting algorithm is efficient, simple to implement, and leads to unprecedented simulation results of fluid flow on curved surfaces, which were previously only possible for flat Euclidean domains.

In our derivation, we have *first* discretized time and *then* space, thereby suggesting a variational integrator in the spatially continuous setting. At the core of our integration lies the observation that each time step can be performed along two consecutive stationary velocity segments. The flow corresponding to these stationary segments can efficiently be computed using the functional point of view in the spatially discrete case, where it simply corresponds to a matrix exponential. We have demonstrated how to achieve efficiency by approximating this exponential up to first order terms, *without* sacrificing stability.

Chapter 7

Functional Thin Films on Surfaces

The motion of a thin viscous film of fluid on a curved surface exhibits many intricate visual phenomena, which are challenging to simulate using existing techniques. A possible alternative is to use a reduced model, involving only the temporal evolution of the mass density of the film on the surface. However, in this model, the motion is governed by a fourth-order nonlinear PDE, which involves geometric quantities such as the curvature of the underlying surface, and is therefore difficult to discretize. Inspired by a recent variational formulation for this problem on smooth surfaces, we present a corresponding model for triangle meshes. We provide a discretization for the curvature and advection operators which leads to an efficient and stable numerical scheme, requires a single sparse linear solve per time step, and exactly preserves the total volume of the fluid. We validate our method by qualitatively comparing to known results from the literature, and demonstrate various intricate effects achievable by our method, such as droplet formation, evaporation, droplets interaction and viscous fingering. Finally, we extend our method to incorporate non-linear van der Waals forcing terms which stabilize the motion of the film and allow additional effects such as pearling.

7.1 Introduction

The intricate motion of a viscous thin film subject to external forces, such as gravity, inspires research in physics, mathematics and computer science, among other scientific disciplines. In many scenarios the domain on which the fluid resides is curved rather than flat. The tear film on the cornea of the eye [BUM$^+$12], the dynamics of lava flows [Gri00] and the formation of ice on the aerofoil of an aircraft [MC04], are all examples related to the evolution of thin films on curved geometries. The goal of this paper is to suggest a method for simulating thin films on surfaces, which is based on *gradient flow* evolution and the *operator view* of the flow induced by tangent vector fields.

Generally, the *Navier–Stokes equations* coupled with appropriate boundary conditions are assumed to give a good approximation of the film's dynamics. However, for the flows we are interested in, these equations are considered difficult to solve numerically, especially on curved domains. Moreover, in the case of thin films we can assume an extremely small height-to-length ratio which leads to a substantial simplification through the *lubrication approximation* [Rey86]. Namely, under the assumptions of the lubrication model, the evolution of the film's mass density is governed by a fourth-order nonlinear partial differential equation (PDE).

A natural approach to simulate thin films within this reduced model would then be to discretize the resulting PDE (e.g., [RRS02]). Choosing such a strategy, however, one will be faced with two main challenges. First, one will need to derive a suitable set of discrete differential operators acting on discrete curved domains (e.g., triangle meshes). Then, the second task will be to construct a proper numerical time integration scheme. While any attempt to discretize general PDEs will encounter these obstacles, in the particular case of thin films, the restriction on the time step size (see e.g., [GBS06]) makes the usage of explicit schemes impractical. Although it is possible to use implicit schemes instead, such schemes do not guarantee in general the preservation of the underlying structure. For example, conserved quantities in the continuous setting (such as the total volume of the thin film) may become non-conserved in a discrete framework. Due to the above obstacles, direct discretization of the PDE is usually considered less attractive.

An alternative point of view is to leverage the *gradient flow* structure which is known to exist for thin film equations (see e.g., [GO03, RV13]). In this model, the motion of the film is determined by the minimizer of a certain cost function, which is defined over the manifold of all possible densities of the film with prescribed volume. Intuitively, the cost function is minimized when the resistance of the fluid to flow due to dissipation induced by friction balances the additional forces (e.g., surface tension and gravity) that act on the film. One of the advantages of this approach is that every gradient flow has a natural time discretization which leads to a variational problem. In practice, it allows for significantly larger time steps compared to explicit numerical schemes. Furthermore, by construction, the associated energy is guaranteed to decrease at each step.

However, we still need to address the issues of modeling the underlying mass transport and the conservation of fluid volume. A reasonable choice within the gradient flow model is to minimize the cost function under an additional constraint given by the transport equation. Intuitively, the transport equation describes how the mass density is affected by the motion of the fluid through the corresponding velocity field. Recently, [ABCCO13] suggested a coordinate-free approach for solving the transport equation on triangulated surfaces by representing tangent vector fields as *linear operators* on scalar functions. Their method is advantageous since it avoids the complicated integration of the fluid's motion, while ensuring the preservation of the integral of the transported quantity.

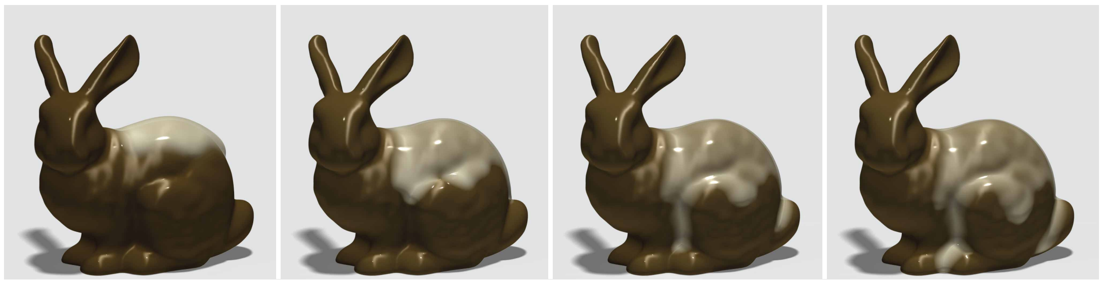


Figure 7.1: Vanilla sauce on a chocolate bunny. The physical parameters are $\mathfrak{b} = 20, \epsilon = 0.1, \beta = 0$.

In this work, we argue that the gradient flow model combined with the operator view of tangent vector fields leads to a robust and highly efficient simulation tool. Specifically, we consider the thin film model of [RV13] in the presence of a *precursor layer* (i.e., the film resides on top of a very thin layer defined over the whole domain) and in the geometric setting of triangulated surfaces. Under the assumption that we are given an approximate normal field, we present formulations of discrete curvature operators which are tailored for our model. In addition, we employ insights from [ABCCO13] to advect the mass function of the thin film in a manner which causes very little numerical dissipation, and is guaranteed to conserve exactly the total volume of the fluid. The resulting method boils down to a *linear* solve of a sparse system per time step. We demonstrate the effects of curvature, gravity (see e.g., Fig. 7.1) and material parameters on the flow, and qualitatively compare our results to previous numerical simulations. Finally, we present various effects (e.g., droplet formation and interaction) which are achievable within our framework.

7.1.1 Related Work

As the behaviour of viscous thin films on surfaces has not, to the best of our knowledge, been previously simulated in the graphics community, we focus our attention on Eulerian methods from the computational fluid dynamics community, and to work on similar phenomena which appeared in the computer graphics literature.

The evolution of thin films over arbitrary domains has been an active area of research in CFD for many decades. We refer the interested reader to the seminal review by [ODB97] and to the more recent review by [CM09]. These reviews present a continuous model for thin films, based on *lubrication theory*, which defines a reduced model for the 3D Navier–Stokes equations given the assumption of a small thickness of the film.

One approach to thin film simulation is to directly discretize the governing PDE as was shown for planar (see e.g., [ZB99, GR00]) and curved (see e.g., [RRS02]) domains. In general, this point of view leads to several challenges, of which the restriction on the time step size for explicit schemes is perhaps the most problematic. Namely, the application of a CFL-type condition leads to the requirement that the time step τ is on the order of $(\delta x)^4$, where δx is the minimal edge length. To overcome this constraint, [GBS06] employed convexity splitting for their time integration scheme (within a level-set framework). Nevertheless, their scheme does not guarantee conservation of the fluid's volume, and has additional restrictions due to the level-set formulation.

An alternative discretization for thin films can be derived from the gradient flow model, for which a natural variational time integrator exists. In general, variational integrators are known to conserve the underlying structure, e.g., the variational scheme in [MCP$^+$09] preserves a notion of discrete momentum. For the case of thin films over curved domains (see e.g., [Van14, RV13]), the gradient flow approach leads to an attractive numerical scheme. In the latter work, which is closest to our approach, the authors used Discrete Exterior Calculus (DEC) [Hir03] for the spatial discretization, representing the flux field with discrete 1-forms. Our approach differs from their work as we use a velocity based formulation, leverage [ABCCO13] for the advection, and suggest discrete curvature operators. These changes allow us to generate stable simulations on meshes with obtuse triangles which are common in graphics. A detailed comparison with [RV13] is given in Sections 7.2 and 7.4.

We conclude with some representative related work from the graphics community literature. Free surface flows for highly viscous fluids were suggested in [CMVHIT02], where effects such as melting wax are demonstrated. While one could consider adding a surface as a solid boundary and using a similar approach for simulating viscous films, it would be quite difficult to achieve the intricate effects we show without using a very dense grid resolution. More recently, various methods were proposed for modeling thin features in free surface flows by explicitly tracking the free surface mesh [WMFB11, ZWW$^+$12], by using thickened triangle meshes [BUAG12], tetrahedral elements [CWSO13], or simplicial complexes [ZQC$^+$14], to mention just a few. Such approaches, however, require careful manipulation of the connectivity and topology of the free surface geometry, which are avoidable when simulating films on surfaces, as the free surface can be represented as a scalar function.

Finally, some approaches simulate water related phenomena. [WMT05] model the contact angle with the surface, representing the free surface with a level-set based distance field. While various effects are achievable with this approach, the method requires a high-resolution grid which leads to a time-consuming system requiring a few days of computation per simulation. On the other hand, using a height field based method as in [WMT07] considerably reduces computational complexity, however, the instabilities and effects we demonstrate below were not shown there.

7.1.2 Contributions

Our main contributions can be described as follows:

- A discrete model for thin film evolution on *general* triangle meshes.

- An efficient and robust scheme, which exactly preserves the total fluid's volume.

- Simulation of various intricate effects, such as fingering, evaporation and droplet formation, interaction between droplets and pearling.

7.2 Dynamics of thin films

We investigate the evolution of a layer of an incompressible viscous fluid flowing with velocity v on top of a curved surface Γ, under the influence of surface tension and, potentially, gravity. The liquid layer is attached to the surface at the liquid-solid interface, i.e., no-slip boundary condition (we extend this later), whereas the liquid-air surface is evolving freely. A typical scenario is illustrated in Figure 7.2 showing the notation for various related quantities.

Navier–Stokes equations. A common approach for modeling the evolution of thin liquid films is to consider the *Navier–Stokes equations.* These equations describe the fluid's velocity in the liquid phase (the *bulk*), the surface tension on the liquid-air interface (i.e., the *free surface*), and a suitable boundary condition for the velocity in the liquid-solid interface (i.e., on the solid surface). Formally, the fluid velocity v and the pressure p satisfy the equations:

$$\begin{aligned} \partial_t v + (v \cdot \nabla)v - \mu\Delta v + \nabla p &= 0 \text{ in the bulk} \\ \operatorname{div} v &= 0 \text{ in the bulk} \\ v &= 0 \text{ on the surface} \\ \sigma n - \gamma \mathcal{H} n &= 0 \text{ at the free surface} \end{aligned} \tag{7.1}$$

where $\sigma = -p\,\mathrm{id} + \mu(\nabla v + \nabla v^T)$ is the stress tensor, μ and γ are the viscosity and the capillary constants (see Fig. 7.2). Furthermore, the free surface χ itself evolves according to the kinematic condition $\partial_t \chi = v$.

Unfortunately, a straightforward discretization of these equations is challenging. In particular, to achieve the type of effects we show below, the main obstacle is due to the prohibitively small time steps which are imposed by such a method. Moreover, the spatial discretization is also challenging since Eulerian methods will require dense sampling of the domain, whereas Lagrangian techniques will involve complex tracking of the free surface. Therefore, direct discretization of equations (7.1) is not practical for graphics applications for this type of problems.

Lubrication approximation. Since we are interested in *thin* films, a reduction in dimensionality can be achieved by using the *lubrication approximation* model (see e.g., [ODB97]). In this model, the dynamics of the film are governed by the evolution of a function (i.e., a scalar quantity) defined on the surface Γ.

Given a characteristic scaling of height and length, the key assumption to consider is a small height to length ratio, i.e., $\epsilon = \frac{\text{height}}{\text{length}} \ll 1$. Then, one takes into account

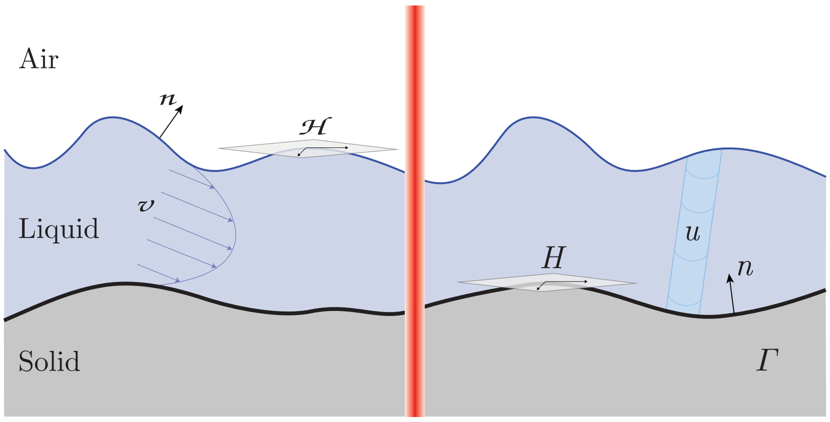


Figure 7.2: A typical scenario is illustrated for the full 3D Navier–Stokes (left) compared to the reduced lubrication model (right). Notice that under the lubrication assumptions the involved quantities are computed directly on Γ, e.g., u is a scalar function.

an asymptotic expansion of the Navier–Stokes equations with respect to ϵ, where the resulting thin film equations are composed of the leading order terms. Taking this path, a derivation of a lubrication model without gravity for the *mass density* u on curved domains yields equations of the form (see [RRS02] and [RV13]):

$$\partial_t u = \operatorname{div}_\Gamma \left(M(u) \nabla_\Gamma p \right) \tag{7.2a}$$

$$M(u) = \frac{1}{3} u^3 \operatorname{id} + \frac{\epsilon}{6} u^4 (H \operatorname{id} - S) \tag{7.2b}$$

$$p = -H - \epsilon T u - \epsilon \Delta_\Gamma u \tag{7.2c}$$

where $M(u)$ is the mobility tensor (to be discussed later) and p can be considered as a pressure-like quantity on the surface, i.e., the fluid moves away from areas of high p. H and K are the mean and Gaussian curvatures, $T = H^2 - 2K$, and S is the shape operator.

Notice that inertia effects are neglected in this model, i.e., the Reynolds number is assumed to be small, Re $\ll 1$, as expected (by simple scaling arguments) for a thin enough film. Moreover, we assume that the mass density u is a proper function. As u is closely related to the fluid's height h, that is $u = h - \frac{\epsilon}{2} H h^2$, the consequence of the former constraint is that the free surface is assumed to be representable as a height function over Γ, and hence, e.g., contact angles higher than $\pi/2$ and wave-like structures cannot be modeled with equations (7.2).

In addition to providing a reduced model for the Navier–Stokes equations, the thin films equations are also instrumental for analyzing the behavior of the flow. As mentioned above, the fluid flows towards low pressure areas thus visualizing p allows to evaluate the underlying dynamics of the film. Moreover, a qualitative study of the expected flow can be done by estimating the different scales of the various components in p. For instance, the dominating term in Eq. (7.2c) is the mean curvature and hence the dynamics on curved domains are expected to be completely different when compared

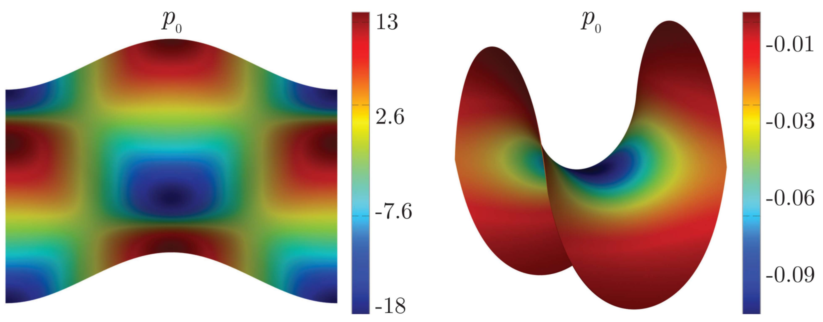


Figure 7.3: By visualizing the pressure we can identify regions where the fluid is likely to accumulate. For example, for an initially uniform layer of fluid, the initial pressure p_0 indicates that fluid is expected to concentrate at the respective centers, where the pressure is lowest. See Fig. 7.4 for the temporal evolution of the flow.

to the flat case (where $H = 0$). Indeed, we demonstrate this and other effects in the following example.

In Figure 7.3 we show the color coding of the pressure computed for an initial uniform deposition of liquid on a bumpy plane (left) and on the Scherk surface (right). These figures suggest that the fluid is most likely to accumulate at the center of the respective surfaces, where the pressure is low. Indeed, we show in Figure 7.4 (top) the color coding of the evolution of the mass density u on the bumpy plane, starting from a uniform layer of fluid. In this case, since the dominating term is H (top, left), the film flows towards the maximal mean curvature, at the center of the basin. Similarly, for a minimal surface, namely when $H = 0$, the terms that govern the dynamics are the Gaussian curvature and the Laplacian of u. In Figure 7.4 (bottom), we show frames of the flow on the Scherk minimal surface, starting again from a uniform layer of fluid. Here, the initial Laplacian of u is 0 thus the minimal Gaussian curvature (bottom, left) drives the fluid towards the center of the surface.

Unfortunately, the simulation of thin film flow based on a PDE of the form (7.2) suffers from serious drawbacks. First, explicit discretization of equation (7.2) requires very strong time step restrictions, and stable (semi-)implicit discretizations allowing for large time steps, are unknown. Second, qualitative properties, such as volume preservation and energy decay, are difficult to ensure. Finally, on general triangulated surfaces it is unclear how to discretize the geometric quantities in a physically consistent way.

These issues motivate a different approach—instead of directly discretizing the PDE, it is possible to model the evolution from the *variational* perspective of gradient flows, as was first suggested in [RV13]. To introduce the concepts to the graphics community, and to keep the paper self contained, we first briefly describe the gradient flow model of thin films, and then discuss our modifications in the next section.

Gradient flow model. The key insight behind the variational approach is that the quantity p can be viewed as the negative (Fréchet) derivative of the *free energy functional*

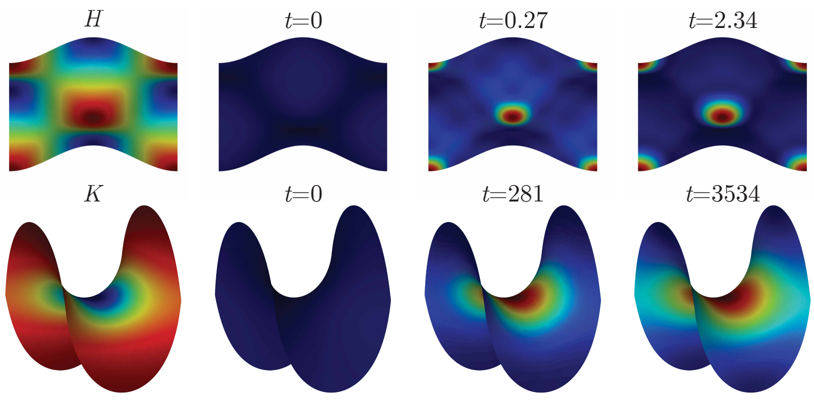


Figure 7.4: (top) The motion of the film primarily depends on the mean curvature thus the fluid concentrates in the center basin, $u_0 = 0.1, \epsilon = 0.1$. (bottom) For minimal surfaces (i.e., when $H = 0$) the film is mostly influenced by the Gaussian curvature as shown for the Scherk's surface, $u_0 = 0.1, \epsilon = 1$.

$\mathcal{E}^\epsilon(u) = \displaystyle\int_\Gamma \left\{-Hu - \frac{\epsilon}{2}Tu^2 + \frac{\epsilon}{2}|\nabla_\Gamma u|^2\right\} \,\mathrm{d}x$ so that the PDE (7.2) is of the *gradient flow* form $\partial_t u = -G(\frac{\delta\mathcal{E}^\epsilon(u)}{\delta u})$. The evolution of u then can be understood as a "steepest" descent for the free energy $\mathcal{E}^\epsilon$, at a rate regulated by the *mobility* $M(u)$ via the function $G(\phi) = \mathrm{div}_\Gamma\,(M(u)\nabla_\Gamma\phi)$. The previous statement can be made precise by introducing the flux $f = -M(u)\nabla_\Gamma p$, so that the PDE can be written in the form of a flow equation as

$$\partial_t u = -\,\mathrm{div}_\Gamma\, f. \tag{7.3}$$

Then the gradient flow is equivalent to the statement that the free energy decays as $\frac{d}{dt}\mathcal{E}^\epsilon(u) = -\mathcal{D}^\epsilon_u(f,f) \le 0$, where the bilinear form $\mathcal{D}^\epsilon_u(f,f) = \displaystyle\int_\Gamma f\cdot M(u)^{-1}f\,\mathrm{d}x$ is known as the (viscous) *dissipation*. This in turn is equivalent to the variational requirement that the density variation $\partial_t u$ and the flux f minimize (at each time t) the so-called Rayleigh functional $\frac{1}{2}\mathcal{D}^\epsilon_u(f,f) + \frac{\delta\mathcal{E}^\epsilon(u)}{\delta u}(\partial_t u)$ under the transport constraint (7.3).

Intuitively, the energy is an approximation of the area of the free surface, which should be minimized due to surface tension, and the dissipation is the "price to pay" for the total shear stress due to the flow inside the film. Hence, among all the possible flows which preserve the mass of the fluid, we look for the one which optimally minimizes the area of the free surface and the stress inside the film.

Finally, following the idea of *natural time discretization* of gradient flows [Ott01] and *minimizing movements* [GA06], we integrate in time to arrive at a variational approximation of $u^{k+1} = u(t^k + \tau)$ given $u^k = u(t^k)$:

$$u^{k+1} = \operatorname*{arg\,min}_{u=\mathcal{F}_\tau(u^k,f)} \left\{\frac{1}{2\tau}\mathcal{D}^\epsilon_u(f,f) + \mathcal{E}^\epsilon(u)\right\} \tag{7.4}$$

where $\mathcal{F}_\tau(u^k, f)$ denotes a suitable (approximate) solution at $t^k + \tau$ of the initial value transport problem (7.3) with $u(t^k) = u^k$. The constrained minimization problem (7.4) is equivalent to discretizing the original PDE (7.2) in time; instead of the PDE then, one can describe (and discretize) the thin film flow through the three components of the gradient flow: the free energy $\mathcal{E}^\epsilon$, the dissipation $\mathcal{D}$ and the flow equation (7.3) (or in the time-discrete setting the flow operator $\mathcal{F}_\tau$).

In [RV13], suitable energy and dissipation functionals are derived for gravity- and surface tension-driven thin film flow on a *smooth* curved surface. The variational time discretization (7.4) is coupled then with a spatial discretization based on Discrete Exterior Calculus, resulting in a fully discrete scheme on triangulated surfaces that addresses some of the shortcomings of PDE-based solvers pointed out previously. Specifically, discrete qualitative properties are straightforward to preserve: the energy decay is built into the time discretization (7.4), as will be shown later, and it is also easier to set up discrete mass conservation for the flow equation than for the full PDE (7.2). In addition, because of the explicit control on the energy decay, the variational scheme is very stable, allowing for large time steps.

Unfortunately, directly applying that scheme for graphics purposes on *general* triangle meshes is challenging since curvature quantities and mass preserving transport are more difficult to discretize in this setting. In [RV13] mass preservation was achieved by working with a flux-based formulation, that lends itself naturally to a finite-volume approach such as Discrete Exterior Calculus. However, in the presence of obtuse triangles, i.e., triangles with angles larger than $\pi/2$, negative entries can arise in the diagonal matrices that the scheme uses to define inner products between discrete k-forms. This can lead to non-convexity and eventually to instability and/or non-convergence of the variational scheme. Notice that for general meshes, eliminating these obtuse triangles is highly non-trivial.

In the next section we present our approach for discretizing the thin film gradient flow model on general triangulated surfaces. We first develop the discrete energy and dissipation terms by modeling the fluid as a prismatic layer formed by an offset surface to the triangle mesh, which naturally introduces discrete curvature quantities. In addition, we switch to a velocity-based formulation of the transport equation $\partial_t u + \mathrm{div}_\Gamma(uv) = 0$, which allows us to use the new discretization suggested in [ABCCO13], that does not suffer from the aforementioned problem.

7.3 Thin films on triangulated surfaces

As we have previously seen in Figure 7.4, the film dynamics are heavily dependent on the curvature operators, H, K and S. In their work [RV13] presented one dimensional applications and simulations on two dimensional surfaces where the curvatures are easy to compute analytically (such as surfaces of revolution and graphs). One could, of course, extend their method to triangulated surfaces by choosing a set of discrete curvature operators from the many available in the literature (see e.g., [GG06]). We chose instead to go back to fluid mechanics and look for a definition of the energy and dissipation functionals that could be applied on continuous but non-smooth surfaces, such as a triangulated mesh. We present the resulting model in this section, but have reserved a more technical derivation for the supplemental material.

Our main observation is that if Γ is equipped with a continuous vector field n that is *approximately* normal, one can follow similar derivations as in [RV13], and arrive at energy and dissipation functionals given by (up to an $O(\epsilon^2)$ error):

$$\mathcal{E}^\epsilon(u) = \int_\Gamma (\mathfrak{b}z - H)u + \frac{\epsilon}{2}(\mathfrak{b}\cos\theta - T)u^2 + \frac{\epsilon}{2}|\nabla_\Gamma u|^2 \,\mathrm{d}a \tag{7.5}$$

and

$$\mathcal{D}^\epsilon_u(v,v) = \int_\Gamma v \cdot M(u)^{-1} v \,\mathrm{d}a \tag{7.6}$$

$$M(u) = \left(\beta + \frac{u}{3}\right)\mathrm{id} + \epsilon\frac{u^2}{12}\left(7H\,\mathrm{id} - 3S - 5\bar{S}\right) \tag{7.7}$$

respectively, where (unlike in [RV13]) the curvature quantities in these equations are now given in terms of the approximate normal field n. In (7.5) we included the gravity terms that involve the Bond number $\mathfrak{b}$, which measures the relative strength of gravity vs. surface tension, the altitude z, and the angle θ of the surface normal with the vertical direction. The discrete total curvature T and shape operator S are given in section 7.3.1 and the rotated shape operator $\bar{S}$ is given in section 7.3.3. Moreover, we incorporated in (7.7) a constant β which allows for various slip conditions.

7.3.1 Geometry of thin films on triangulated surfaces

For a smooth surface, the geometry of a liquid layer is modeled by a scalar height function h, which describes the extension of the liquid along a *surface normal direction* at each surface point. In the limit of thin films, this height field is scaled by a global scaling parameter ϵ. Then, the liquid layer is bounded by the surface on one side and by an offset along the surface normal by ϵh on the other. The laws of physical motion of the liquid are expressed by expanding the 3D motion up to second powers in ϵ.

Adopting this perspective for the case of a triangulated surface Γ, we take the approach of associating surface normals n as well as the offset function h with vertices and extending the resulting offset field linearly across triangles, leading to a prismatic liquid layer per triangle, see Figure 7.5 (left). This approach ensures continuity of the offset field across edges, which we harness to ensure mass conservation when the liquid evolves.

There is, however, a caveat with this approach: it is widely accepted that there exist no “best” vertex normals in the discrete case. Consequently, we only require *consistent* normals in the following sense. If the average edge length of the mesh is δx, it suffices that we are provided with a set of (unit length) vertex normals n such that the difference

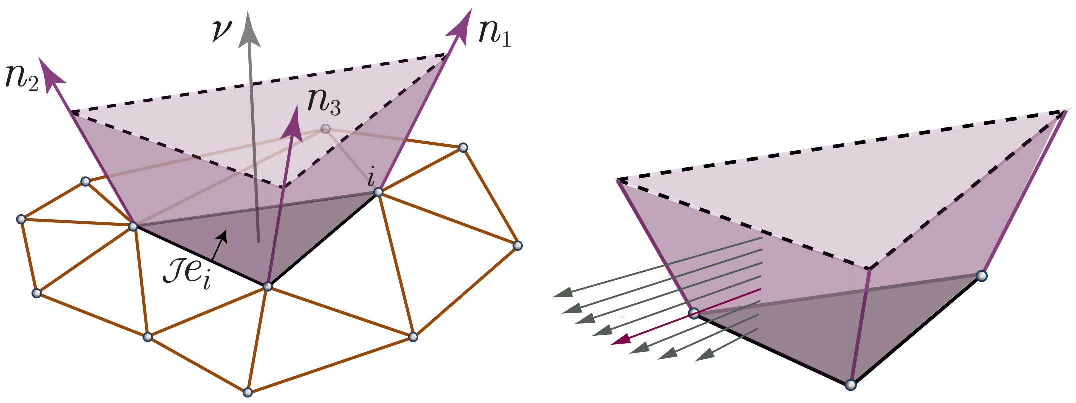


Figure 7.5: (left) Prismatic layer of viscous fluid, depicted as a piecewise linear field over a triangle. (right) Prismatic volume with tangential vector field v (red) and attached Hagen-Poiseuille type velocity profile $\Pi_s(v)$.

$|\nu - n|$, between the normal ν of any (flat) triangle of the mesh and the vertex normal n of its vertices, is of order δx^2. [1]

As in the smooth case, the lubrication approximation requires an additional scaling variable ϵ in which the relevant physical terms are developed up to second order. With the lateral extension of the film being measured in direction of the discrete normal n, we obtain the free surface

$$\Gamma_{\epsilon h} = \{x + \epsilon h(x) n(x) \,|\, x \in \Gamma\}$$

of the thin film at the liquid-gas interface and the fluid volume $V_{\epsilon h} = \{x + s\epsilon h(x) n(x) \,|\, x \in \Gamma,\, s \in (0,1]\}$.

In order to derive the variational time discretization of the evolution of the thin film we make use of three different expansion formulas, namely the expansion of volume, area, and length with respect to the thickness parameter ϵ. Returning to the smooth case for a moment, such an expansion leads to expressions in terms of curvatures of the underlying surface, containing the shape operator S, its trace, and its determinant, known as mean and Gauss curvature, respectively [Car94].

We exactly recover this geometric description in our discrete model. Indeed, first recall that in the smooth setting the shape operator is defined as the tangential gradient of the (smooth) unit normal field. Accordingly, we define in the discrete case a *generalized* shape operator (in the sense of considering arbitrary "normals" n) by

$$S := -\frac{1}{2} P(\nabla_\Gamma n + (\nabla_\Gamma n)^T) P \,, \tag{7.8}$$

where $\nabla_\Gamma = P\nabla_{\mathbb{R}^3}$ is the (triangle-based) tangential gradient on Γ and $P = \mathrm{id} - \nu \otimes \nu$ is the projection onto the (triangle-based) tangent space. From this shape operator we deduce a discrete mean curvature $H = \mathrm{Tr}(S)$ and a discrete Gaussian curvature

[1]Notice that this condition implies that $\nabla_\Gamma n$ is both tangential and symmetric up to order δx.

$K = \frac{1}{2}\left(\mathrm{Tr}(S)^2 - \mathrm{Tr}(S^2)\right)$. Notice that in this setup S, and therefore also H and K, are constant per face.

Second recall that in the smooth case, mean and Gaussian curvatures *alternatively* arise by considering first and second variations of offset volume and surface area. The same holds true in the discrete case, i.e., for our prismatic layer. For example, for *the expansion of offset volume* we obtain (up to an $\mathrm{O}(\epsilon^3 + \delta x)$ error)

$$\int_{V_{\epsilon h}} \mathrm{d}x = \int_\Gamma \left(\epsilon h - \frac{\epsilon^2}{2} H h^2\right) \mathrm{d}a \ .$$

Here H equals the trace of our generalized shape operator S defined above. Hence, the two alternative *discrete* definitions of mean curvature (as the trace of the tangential gradient of the normal, and through the second order expansion of the of the offset volume) are consistent. Intuitively, the correction term $\frac{\epsilon^2}{2} H h^2$, and in particular the appearance of mean curvature, accounts for change of surface area in the lateral direction.

Notice that the integrand can be written as ϵu, with $u = h - \frac{\epsilon}{2} H h^2$. Thus u describes (up to a factor of ϵ) the *fluid volume per surface area* and can be considered as the local mass density. This quantity is an alternative and, from the viewpoint of the underlying conservation law, preferable variable.

Likewise, for *the expansion of the surface area* we obtain (up to an $\mathrm{O}(\epsilon^3 + \delta x)$ error) that

$$\int_{\Gamma_{\epsilon h}} \mathrm{d}a = \int_\Gamma \left(1 - \epsilon h H + \frac{\epsilon^2}{2}\left(2h^2 K + |\nabla_\Gamma h|^2\right)\right) \mathrm{d}a \ .$$

Notice that when $h = 1$, i.e., when one considers constant offsets, then this expression is equal to the famous Steiner formula, known from differential geometry [Fed69]. As before, H and K that arise from the expansion of the surface area are exactly the mean and Gaussian curvatures, respectively, defined using our generalized shape operator S.

7.3.2 Energy

The first ingredient of our variational time discretization is the energy of the thin film, given by the sum of surface energy (the total area of the free surface $\Gamma_{\epsilon h}$, which tends to be minimized due to surface tension) and gravitational energy (weighted by the Bond number $\mathfrak{b}$):

$$\mathcal{E}(h) = \int_{\Gamma_{\epsilon h}} \mathrm{d}a + \mathfrak{b} \int_{V_{\epsilon h}} z \, \mathrm{d}x \, .$$

Here the Cartesian coordinate z denotes the altitude, i.e., we assume that gravity is acting along the z-direction.

The surface energy was spelled out above. Analogously to the expansion of the offset volume, we obtain for *the expansion of gravitational energy* (up to an $\mathrm{O}(\epsilon^3 + \delta x)$ error) that

$$\int_{V_{\epsilon h}} z\,\mathrm{d}V = \int_\Gamma \left(\epsilon z h + \frac{\epsilon^2}{2}\left(-h^2 H z + h^2\cos\theta\right)\right)\mathrm{d}a\,.$$

Here, per triangle, θ is the angle of the direction of gravity with the triangle normal. Exchanging the height h against the mass density u and restricting to the (non constant) leading order terms we finally end up with the energy functional

$$\mathcal{E}^\epsilon(u) = \int_\Gamma (\mathfrak{b} z - H)u + \frac{\epsilon}{2}\left(\mathfrak{b}\cos\theta - T\right)u^2 + \frac{\epsilon}{2}|\nabla_\Gamma u|^2 \mathrm{d}a \tag{7.9}$$

with $T = H^2 - 2K$.

7.3.3 Conservation law for the flow

Mass conservation during the temporal evolution of the fluid is one of the central physical principles of viscous flow [CMM90]. Violations of this principle in numerical simulations lead to undesirable artefacts. For our approach, we outline how mass conservation can be *exactly* maintained by working with a conservation law in *divergence* form. Mass conservation is a balance principle: the change of volume must equal the flux of material across the volume boundary. On an arbitrary (triangular) patch T this translates into the balance equation

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{V_{\epsilon h}(T)} \mathrm{d}x = \int_{F_{\epsilon h}(T)} \mathfrak{v}\cdot\mu\,\mathrm{d}a\ ,$$

where $\mathfrak{v}$ is the fluid's velocity vector and μ is the (inward pointing) normal of the faces $F_{\epsilon h}(T)$ of the prism $V_{\epsilon h}(T)$ above T (cf. Fig. 7.5 (right)). Using the divergence theorem of Gauss and Taylor expansions in the height, which corresponds to an expansion of the length functional on the edges of the patch, we obtain the conservation law

$$\partial_t u = -\operatorname{div}_\Gamma\left(u\int_0^\epsilon Q_s \mathfrak{v}_{\Gamma,s}\,\mathrm{d}s\right)\ ,$$

where $\mathfrak{v}_{\Gamma,s}(x)$ is the tangential component of the velocity in the liquid layer and the tensor $Q_s = \mathrm{id} - su\left(\bar{S} - H\,\mathrm{id}\right)$ accounts for the geometry of the prism $V_{\epsilon h}(T)$. The rotated shape operator $\bar{S} = -[\nu]_\times S[\nu]_\times$ is defined via the skew-symmetric matrix $[\nu]_\times$, which in turn is given by requiring that $[\nu]_\times \cdot x = \nu \times x$ for any vector x. We define the (weighted) average velocity $v = \int_0^\epsilon Q_s \mathfrak{v}_{\Gamma,s}\,\mathrm{d}s$, independent of s, so that the conservation law is restricted to the triangulated surface Γ and takes the simple form

$$\partial_t u + \operatorname{div}_\Gamma(u\,v) = 0\,. \tag{7.10}$$

The weighting reflects the inclination and torsion of the faces of the prisms. The advantage of working with an averaged velocity is that it resides directly on the surface Γ. In the discrete case, this velocity field can be modeled using piecewise constant (per triangle) vector fields, and mass balance can be expressed using commonly used discrete differential operators.

7.3.4 Dissipation and mobility

In the previous section, we used averaging in order to reduce the velocity field in the bulk to a velocity field on the surface. For treating dissipation, we require the opposite direction, i.e., to reconstruct a velocity field in the bulk from the velocity filed on the surface. Since the inverse of averaging allows for many solutions, this reconstruction step is not unique a priori. In order to single out a unique velocity field in the bulk, we invoke a physical principle by considering the field that causes least energy dissipation.

Concretely, we require a (tensor) profile function Π_s such that $\mathfrak{v}_{\Gamma,s} = \Pi_s v$ and $\int_0^\epsilon Q_s \Pi_s \,\mathrm{d}s = \mathrm{id}$ (see Fig. 7.5 (right)). Note that there are many possible velocity profiles $\Pi : s \mapsto \Pi_s$ that satisfy this integral constraint. From the theory of viscous flows [Poz11] we know that the physically observed profile minimizes the viscous dissipation rate $\int_{V_{\epsilon h}} |\nabla \mathfrak{v} + \nabla \mathfrak{v}^T|^2 \,\mathrm{d}x$. This is dominated by the vertical shear stress, i.e., the normal derivative of the tangential velocity, which can be expressed as a quadratic form in $\mathfrak{v}$. Approximating this quadratic form to leading order in ϵ, substituting $\mathfrak{v}$ by $\Pi(v)$, and optimizing the transportation cost for given boundary conditions $\Pi_0 = 0$ (no-slip at substrate) and zero shear stress at free surface under the integral constraint $\int_0^\epsilon Q_s \Pi_s \,\mathrm{d}s = \mathrm{id}$, yields an optimal profile Π^*, which to leading order matches the well-known Hagen-Poiseuille profile. We thus obtain the dissipation as a function of the averaged velocity v as

$$\mathcal{D}_u^\epsilon(v,v) = \int_\Gamma v \cdot M(u)^{-1} v \,\mathrm{d}a \;, \tag{7.11}$$

where the mobility tensor is defined as

$$M(u) = \frac{u}{3} + \epsilon \frac{u^2}{12} \left(7H \,\mathrm{id} - 3S - 5\bar{S}\right) .$$

For a more detailed derivation of the optimal profile see the supplemental material.

7.3.5 Minimizing movement approach

Combining the three building blocks we have derived, and using the minimizing movement approach, we arrive at an effective variational time discretization for the evolution of a thin film on a triangulated surface. The energy $\mathcal{E}^\epsilon$ (7.9) depends on the mass density u, whereas the dissipation $\mathcal{D}_u^\epsilon$ (7.11) is a quadratic form on motion fields v. For given u^k at time step k any mass density u at time step $k+1$ results from the transport of u^k via an underlying motion field. Hence, the time discrete conservation law (7.10) has to be handled as a constraint representing the coupling of u and v. Altogether, we iteratively define u^{k+1} as the minimizer u of the following constrained optimization problem:

$$\min_{u,v} \left\{ \frac{1}{2\tau} \mathcal{D}^{\epsilon}_{u^k}(v,v) + \mathcal{E}^{\epsilon}(u) \right\}$$
$$\text{subject to } u = T_\tau(v)(u^k),$$

where $T_\tau(v)$ denotes the operation of transporting u^k with constant velocity v for a time interval of length τ. The factor $\frac{1}{2\tau}$ reflects the proper rescaling in time to obtain the dissipation to be spent to transform u^k into u.

We consider a number of extensions to this model, which are known for the flat case [ODB97]. The first one replaces on Γ the no-slip $v = 0$ by the *Navier slip condition* $v = \beta \partial_n v$ with β denoting the slip length (in case of large variation of the velocity in the normal direction, the fluid undergoes slipping on the surface Γ). To reflect this one has to add β to the mobility M. This slip boundary conditions accelerates the motion of the fluid. Furthermore, we consider *evaporation.* It takes the form of a sink term in the right hand side of the conservation law and is modeled in the time discrete setup by the constraint $u - T_\tau(v)(u^k) = -\frac{u}{(u^k+c_e)^2}$, for a small constant c_e. Intuitively, the evaporation rate is faster for thinner films, which reflects a faster heating of thinner films.

7.4 Spatial discretization

The main challenge here is to define a stable discretization of the transport equation (7.10) such that various properties (e.g., energy decay and mass preservation) will hold on general triangle meshes. While many of the operators we use are standard in geometry processing, we highlight the properties these operators should possess such that the resulting optimization scheme would indeed be stable.

Notation. We consider a triangle mesh and denote by $\mathcal{V}$ its vertex set and by $\mathcal{F}$ its face set. We use bold faced symbols to denote the spatial discrete analogues of continuous quantities (e.g., $\mathbf{u}$ is the discrete mass density). When required, we use the subscripts $\mathcal{V}$ and $\mathcal{F}$ to denote quantities on the vertices and the faces, respectively. The bracket $[\cdot]$ operator is used to convert vectors in $\mathbb{R}^{|\mathcal{V}|}$ and $\mathbb{R}^{|\mathcal{F}|}$ to block diagonal matrices in $\mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ and $\mathbb{R}^{3|\mathcal{F}|\times 3|\mathcal{F}|}$ respectively (replicating each entry 3 times for the latter).

Functions, vector fields and inner products. We use a typical setup, i.e., piecewise-linear functions and piecewise-constant vector fields, with corresponding inner products. Specifically, we represent real-valued functions as scalars on the vertices of the mesh, i.e.,

$\mathbf{u} \in \mathbb{R}^{|\mathcal{V}|}$, and extend them to the whole mesh using piecewise linear hat basis functions. Similarly, vector fields are treated as piecewise-constant on the faces of the mesh, i.e., $\mathbf{v} \in \mathbb{R}^{3|\mathcal{F}|}$.

For defining discrete inner products we require vertex and face areas, denoted by $\mathbf{A}_{\mathcal{V}} \in \mathbb{R}^{|\mathcal{V}|}$ and $\mathbf{A}_{\mathcal{F}} \in \mathbb{R}^{|\mathcal{F}|}$, respectively. For the vertex area we use 1/3 of the total area of its adjacent triangles, and we define an interpolating matrix $\mathbf{I}_{\mathcal{V}}^{\mathcal{F}} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{F}|}$ which interpolates quantities from faces to the vertices, i.e., $\mathbf{I}_{\mathcal{V}}^{\mathcal{F}}(i,j) = \frac{\mathbf{A}_{\mathcal{F}}(j)}{3\mathbf{A}_{\mathcal{V}}(i)}$, iff vertex i belongs to face j and 0 otherwise. This choice implies that $\mathbf{A}_{\mathcal{F}} = (\mathbf{I}_{\mathcal{V}}^{\mathcal{F}})^T\mathbf{A}_{\mathcal{V}}$, which will be important for consistency later. Now, discrete inner products are defined by:

$$\int_\Gamma \mathbf{u_1}\mathbf{u_2} da = \mathbf{u_1}^T G_{\mathcal{V}} \mathbf{u_2}, \qquad \int_\Gamma \langle \mathbf{v_1}, \mathbf{v_2}\rangle da = \mathbf{v_1}^T G_{\mathcal{F}} \mathbf{v_2},$$

where $G_{\mathcal{V}} = [\mathbf{A}_{\mathcal{V}}] \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ and $G_{\mathcal{F}} = [\mathbf{A}_{\mathcal{F}}] \in \mathbb{R}^{3|\mathcal{F}|\times 3|\mathcal{F}|}$ denote the diagonal mass matrix of the vertices and the faces.

Differential Operators. Equations (7.5) and (7.10) require discrete gradient and divergence operators. In the smooth case, these operators fulfill integration by parts, namely on a surface without boundary we have: $\int_\Gamma \langle v, \nabla_\Gamma u\rangle\, da + \int_\Gamma u \cdot \mathrm{div}_\Gamma\, v\, da = 0$. In order to maintain discrete preservation of mass (see appendix E.1), we need the operators grad $\in \mathbb{R}^{3|\mathcal{F}|\times|\mathcal{V}|}$ and $div \in \mathbb{R}^{|\mathcal{V}|\times 3|\mathcal{F}|}$ to fulfill this discretely, namely:

$$\mathbf{v}^T G_{\mathcal{F}}(\mathrm{grad}\,\mathbf{u}) + (div\mathbf{v})^T G_{\mathcal{V}}\mathbf{u} = 0,$$

for arbitrary $\mathbf{v}$ and $\mathbf{u}$. Interestingly, the standard operators (e.g., as defined in [BKP+10, Chapter 3]) fulfill this property.

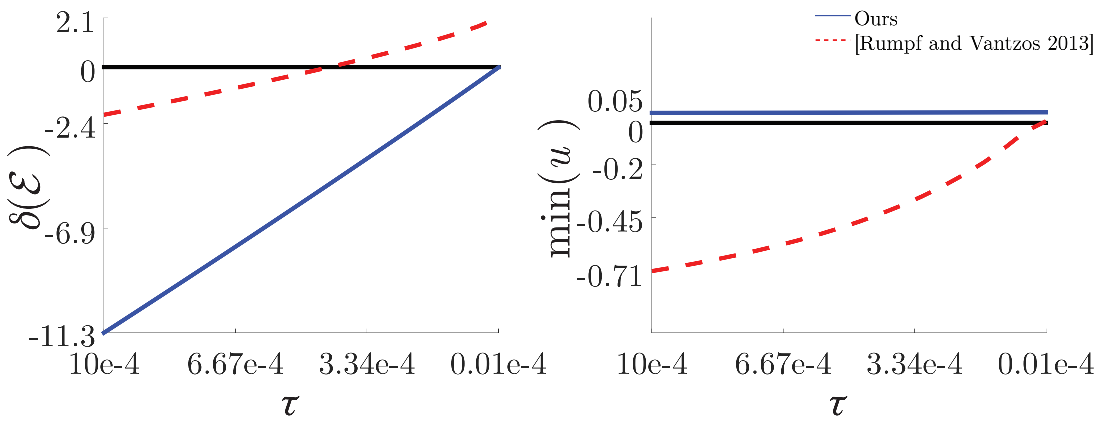


Figure 7.6: Comparison with [RV13]. (left) Plot of the observed energy reduction $\delta(\mathcal{E}) = \mathcal{E}(t+\tau) - \mathcal{E}(t)$ as a function of the time step τ, on a mesh with obtuse triangles. The present scheme consistently decreases the energy ($\delta(\mathcal{E}) \leq 0$), whereas the other method has trouble with small time steps. (right) Regarding the positivity of the solution, again on a mesh with obtuse triangles, the present method preserves the initial minimum u, whereas the other method exhibits negative values of u.

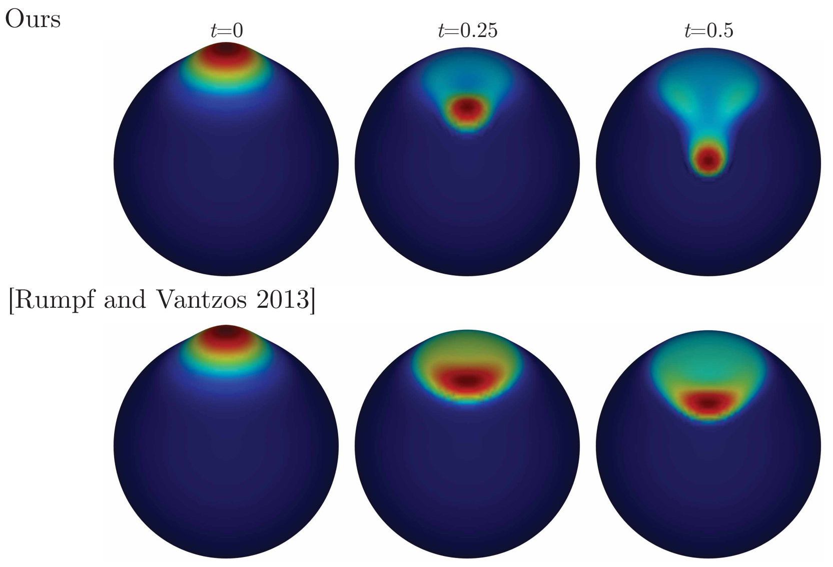


Figure 7.7: Starting from the same initial conditions and physical parameters, our transport scheme (top) achieves a better resolved finger compared to the result (bottom) generated with the more diffusive scheme suggested in [RV13].

Approximate normal field, curvature and gravity. As described in the previous section, all of the required curvature quantities can be computed once a suitable approximate normal field is given. In practice, we use the area-weighted averages of triangle normals [BKP+10, pg. 42] as vertex normals. By applying the discrete gradient operator defined previously, the tangential gradient of the discrete normal field per face j is:

$$(\nabla_\Gamma \mathbf{n})_j = \frac{1}{2\mathbf{A}_{\mathcal{F}}(j)} \left(\sum_{i=1}^{3} \mathbf{n}_{j_i} (\mathcal{J}\mathbf{e}_{j_i})^T \right)$$

where the sum runs over the three vertex normals $\mathbf{n}_{j_i}$ of the face and $\mathcal{J}\mathbf{e}_{j_i}$ is the rotated (by $\pi/2$) edge opposite to vertex i in the triangle j (see Figure 7.5). The gravity quantities can be computed as follows: $\mathbf{z}$ is the vertical coordinate function and $\cos\theta$ is the vertical component function of $\mathbf{n}$.

Mobility. The discrete mobility $\mathbf{M}(\mathbf{u})$ is a $3|\mathcal{F}| \times 3|\mathcal{F}|$ *diagonal* matrix, where for each face the associated quantities can be computed using Eq. (7.7), the curvature operators, and the interpolated mass density $\mathbf{u}_{\mathcal{F}}$ on the faces ($\mathbf{u}$ is defined on vertices).

Transport operator. In the continuous case, equation (7.10) guarantees that the integral of $\partial_t u$ vanishes on a closed surface (since the divergence of any vector field integrates to 0). However, once we discretize u and v then $\mathrm{d}iv(\mathbf{uv})$ is no longer well defined using our discrete operators, since $\mathbf{uv}$ is not a piecewise constant vector field. To avoid this issue, we first apply the product rule to (7.10) and reformulate the constraint as $\partial_t u = -(v \cdot \nabla_\Gamma u + u \operatorname{div}_\Gamma v)$. We then follow [ABCCO13] and define a directional derivative $\mathbf{D}(\mathbf{v})$ such that $\mathbf{1}_{\mathcal{V}}^T G_{\mathcal{V}} (\mathbf{D}(\mathbf{v}) + [\mathrm{d}iv\mathbf{v}])\, \mathbf{u} = 0$ for any $\mathbf{u}$ and $\mathbf{v}$ (see appendix E.1 for the proof). Specifically, the directional derivative is given as $\mathbf{D}(\mathbf{v}) \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$

by $\mathbf{D}(\mathbf{v}) = \mathbf{I}_{\mathcal{V}}^{\mathcal{F}}[\mathbf{v}]_{\bullet}^{T}\,\mathrm{grad}$, where $[\cdot]_{\bullet} \in \mathbb{R}^{3|\mathcal{F}|\times|\mathcal{F}|}$ converts vector fields to block diagonal matrices.

The main advantage of this point of view is that in the discrete case the transport equation turns into a system of ODEs of the form $\partial_t \mathbf{u} + \mathbf{A}\mathbf{u} = 0$, for a constant matrix $\mathbf{A}$, which can be solved using a matrix exponential [HO10]. Thus, for a velocity $\mathbf{v}$ constant in time, the discrete transport equation can be solved in the time interval $[t^k, t^k + \tau]$ to yield the solution

$$\mathbf{u} = \exp\left(-\tau\mathbf{D}(\mathbf{v}) - \tau[\mathrm{d}iv\mathbf{v}]\right)\mathbf{u}^k \tag{7.12}$$

at $t = t^k + \tau$, where τ is the time step. In the case of evaporation, we have an additional term $-\tau[\mathbf{u}^k + c_e]^{-2}$ in the exponential.

We compared our transport scheme to the method of [RV13] on the bunny model which has obtuse triangles. Specifically, we computed the difference in energy and the minimal u in the first iteration for different time step sizes. In Figure 7.6 (left) we show that our method is consistently decreasing the energy, whereas the method of [RV13] actually increases the energy for small time steps. In addition, we show in Figure 7.6 (right) that their method yields negative values for u even for very small time steps, whereas ours preserves the initial value of the precursor layer.

Furthermore, the suggested transport mechanism is more appropriate to the flows we are interested in than the one suggested by [RV13]. In particular, droplet formation and fingering instabilities are transport-dominated effects. Thus, a natural requirement from a transport mechanism is to exhibit minimum diffusion, allowing to capture better resolved fingers on relatively coarse meshes as we demonstrate. We show in Figure 7.7 that starting from the same initial conditions, our scheme is qualitatively less diffusive compared to the method of [RV13].

7.5 Fully discrete model

Given the above discrete operators and quantities, we can write the fully-discrete optimization problem for computing $\mathbf{u}, \mathbf{v}$ given $\mathbf{u^k}$:

$$\begin{gathered}\min_{\mathbf{u},\mathbf{v}} \left\{ \frac{1}{2\tau}\mathcal{D}^{\epsilon}_{\mathbf{u}^k}(\mathbf{v},\mathbf{v}) + \mathcal{E}^{\epsilon}(\mathbf{u}) \right\}, \\ \text{subject to } \mathbf{u} = \exp\left(-\tau\mathbf{D}(\mathbf{v}) - \tau[\mathrm{d}iv\mathbf{v}]\right)\mathbf{u}^k.\end{gathered} \tag{7.13}$$

Then, the fully-discrete energy and dissipation are given by:

$$\begin{aligned}\mathcal{E}^{\epsilon}(\mathbf{u}) &= \mathbf{a}^T G_{\mathcal{V}}\mathbf{u} + \frac{\epsilon}{2}\mathbf{u}^T(G_{\mathcal{V}}\mathbf{B} + \mathbf{L})\mathbf{u}, \\ \mathcal{D}^{\epsilon}_{\mathbf{u}^k}(\mathbf{v},\mathbf{v}) &= \mathbf{v}^T G_{\mathcal{F}}\mathbf{M}(\mathbf{u}^k)^{-1}\mathbf{v},\end{aligned}$$

where $\mathbf{a} = \mathfrak{b}\mathbf{z} - \mathbf{H}$, $\mathbf{B} = \mathfrak{b}\cos\theta - \mathbf{H^2} + 2\mathbf{K}$, and the stiffness matrix $\mathbf{L} = -G_{\mathcal{V}}\mathrm{d}iv\,\mathrm{grad}$.

7.5.1 Properties

Discrete energy. *The discrete energy $\mathcal{E}^\epsilon(\mathbf{u}^k)$ is non increasing.*

Proof: Noticing that $\mathbf{u} = \mathbf{u}^k$ and $\mathbf{v} = 0$ is an admissible pair for the minimization problem (7.13) since they satisfy the constraint, we have immediately that:

$$\frac{1}{2\tau}\mathcal{D}^\epsilon_{\mathbf{u}^k}(\mathbf{v}^{k+1}, \mathbf{v}^{k+1}) + \mathcal{E}^\epsilon(\mathbf{u}^{k+1}) \le \frac{1}{2\tau}\mathcal{D}^\epsilon_{\mathbf{u}^k}(0,0) + \mathcal{E}^\epsilon(\mathbf{u}^k) \rightarrow \mathcal{E}^\epsilon(\mathbf{u}^{k+1}) \le \mathcal{E}^\epsilon(\mathbf{u}^k)$$

since $\mathcal{D}^\epsilon_{\mathbf{u}^k}(\mathbf{v}^{k+1}, \mathbf{v}^{k+1}) \ge 0$ and $\mathcal{D}^\epsilon_{\mathbf{u}^k}(0,0) = 0$.

Intuitively, since $\mathcal{D}$ is non-negative, if the fluid moved and "paid" with dissipation, then it found a smaller energy solution (otherwise it will have remained at the previous state, with the same energy).

Discrete mass. *The total discrete mass $m(\mathbf{u}) = \int_\Gamma \mathbf{u}\, \mathrm{d}a = \mathbf{1}_\mathcal{V}^T G_\mathcal{V}\mathbf{u}$ is exactly preserved.*

Proof: The transport equation (7.12) can be written as $\mathbf{u} = \exp(-\tau\mathbf{A})\mathbf{u}^k$, where $\mathbf{A} = \mathbf{D}(\mathbf{v}) + [\mathrm{d}iv\mathbf{v}]$. In appendix E.1 we show $\mathbf{1}_\mathcal{V}^T G_\mathcal{V}\mathbf{A} = 0$ for any velocity $\mathbf{v}$. Hence, we have $m(\mathbf{u}^k) - m(\mathbf{u}) = \mathbf{1}_\mathcal{V}^T G_\mathcal{V}\{\mathrm{id} - \exp(-\tau\mathbf{A})\}\mathbf{u}^k = \mathbf{1}_\mathcal{V}^T G_\mathcal{V}\left\{\tau\mathbf{A} - \frac{\tau^2}{2}\mathbf{A}^2 + \ldots\right\}\mathbf{u}^k = 0$.

7.5.2 Optimization

To solve the discrete variational model (7.13) we use the first order approximation $\exp(-\tau\mathbf{A}) \approx \mathrm{id} - \tau\mathbf{A}$ of the matrix exponential, so that the linear equation:

$$\mathbf{u} = \mathbf{u}^k - \tau(\mathbf{D}(\mathbf{v}) + [\mathrm{d}iv\mathbf{v}])\mathbf{u}^k \tag{7.14}$$

replaces the non-linear constraint (7.12). Hence, at every time step we solve *a quadratic problem with a linear constraint*, which is convex for a small enough τ (see §5.3 "Dynamic Time-stepping"). As we will show next, this can be done very efficiently, by solving a single linear system for $\mathbf{u}$. Note that it is straightforward to check that the results of §5.1 hold for the linearized constraint as well, hence we gain efficiency yet do not lose stability.

The linear system. Using the method of Lagrange multipliers we obtain the *first order necessary conditions*:

$$\begin{aligned} G_\mathcal{F}\mathbf{M}(\mathbf{u}^k)^{-1}\mathbf{v} - \left(\overline{\mathbf{D}}(\mathbf{u}^k) + [\mathbf{u}^k]\mathrm{d}iv\right)^T G_\mathcal{V}\mathbf{p} = 0 \\ G_\mathcal{V}\left(\mathbf{a} + \epsilon\mathbf{B}\mathbf{u}\right) + \epsilon\mathbf{L}\mathbf{u} - G_\mathcal{V}\mathbf{p} = 0 \\ G_\mathcal{V}\left(\mathbf{u} - \mathbf{u}^k + \tau(\mathbf{D}(\mathbf{v}) + [\mathrm{d}iv\mathbf{v}])\mathbf{u}^k\right) = 0, \end{aligned} \tag{7.15}$$

where $\mathbf{p}$ is the dual variable.

Figure	$\|\mathcal{V}\|$	Avg. per step	#steps	Total time
Fig. 7.1, Bunny*	38306	0.484	1999	967.8
Fig. 7.4, Bumpy plane	40401	0.683	4996	3410.4
Fig. 7.4, Scherk surface	40401	0.627	1997	1252.4
Fig. 7.9, Rounded cube*	19728	0.142	4991	709.5
Fig. 7.10, Sphere	40962	1.645	300	493.5
Fig. 7.12, Moomoo*	16710	0.080	1981	158.4
Fig. 7.13, Torus	40000	1.079	456	491.8
Fig. 7.14, Moai	89126	3.106	314	975.3
Fig. 7.15, Rain	10242	0.198	18001	3570.1
Fig. 7.17, Pensatore	27732	0.818	991	810.3
Fig. 7.16, Wine glass*	38976	0.708	496	351.1

Table 7.1: Timing statistics (in seconds). Asterisk denotes simulations where an iterative solver was used, whereas for the rest, we used a direct non-iterative solver.

A key ingredient to deriving (7.15) is the *dual operator* $\overline{\mathbf{D}}(\mathbf{u})$, defined such that $\mathbf{D}(\mathbf{v})\mathbf{u} = \overline{\mathbf{D}}(\mathbf{u})\mathbf{v}$, as it allows us to take derivatives with respect to $\mathbf{v}$. This operator is: $\overline{\mathbf{D}}(\mathbf{u}) = \mathbf{I}_{\mathcal{V}}^{\mathcal{F}}[\operatorname{grad}\mathbf{u}]_{\bullet}^{T}$. Similarly, it holds that $([\mathbf{u}]\mathrm{d}iv)\,\mathbf{v} = [\mathrm{d}iv\mathbf{v}]\mathbf{u}$.

Finally, eliminating $\mathbf{v}$ and $\mathbf{p}$, we arrive at the following *reduced linear system* for $\mathbf{u}$:

$$\Big(\mathrm{id} + \tau\epsilon\,\mathbf{R}(\mathbf{u}^k, \mathbf{u}_e^k)(G_{\mathcal{V}}\mathbf{B} + \mathbf{L})\Big)\mathbf{u} = \mathbf{u}_e^k - \tau\mathbf{R}(\mathbf{u}^k, \mathbf{u}_e^k)G_{\mathcal{V}}\mathbf{a} \tag{7.16}$$

where $\mathbf{R}(\mathbf{u}^k, \mathbf{u}_e^k) = \mathbf{F}(\mathbf{u}_e^k)\mathbf{M}(\mathbf{u}^k)G_{\mathcal{F}}^{-1}\mathbf{F}(\mathbf{u}_e^k)^T$ and $\mathbf{F}(\mathbf{u}_e^k) = \overline{\mathbf{D}}(\mathbf{u}_e^k) + [\mathbf{u}_e^k]\mathrm{d}iv$ and $\mathbf{u}_e^k = \exp(-\tau[\mathbf{u}^k + c_e]^{-2})\mathbf{u}^k$ if evaporation is included and $\mathbf{u}_e^k = \mathbf{u}^k$ otherwise.

Thus, we obtain a fully discrete scheme where given an initial mass density $\mathbf{u}_0$, we evolve it in time using the above update rule.

We implemented our method in MATLAB using standard linear solvers for Eq. ((7.16)). In all our experiments, the method was very stable allowing for large time steps (on the scale of $\mathrm{O}(\epsilon + \delta x)$, which is excellent for 4th order problems) depending on the initial conditions and the underlying mesh. The experiments were performed on an Intel(R) Xeon(R) processor with 32 GB RAM, and we show in Table 7.1 the statistics for the different simulations.

7.5.3 Limitations

Dynamic Time-Stepping. Given that the stiffness matrix $\mathbf{L}$ is positive semi-definite, the system (7.16) is invertible as long as $\tau_1\,\epsilon\|\mathbf{R}(\mathbf{u}^k)\|_2\|G_{\mathcal{F}}\|_2\underline{\mathbf{B}} \leq 1$, where $\underline{\mathbf{B}}$ is the absolute taken on the minimum value of $\mathbf{B}$ and it is a measure of how strongly negative the quantity $\mathfrak{b}\cos\theta - T$ is on the surface. Moreover, we employ a CFL-type condition depending on the maximum velocity of the film $\underline{\mathbf{v}}$, and grid size, i.e., we require that $\tau_2\underline{\mathbf{v}} \leq \delta x$. Finally, we take the time step to be $\tau = \min\{\tau_1, \tau_2\}$.

Positivity Preservation. Unfortunately, even if we start from a strictly positive $\mathbf{u}_0$, the evolution of the film $\mathbf{u}^k$ is not guaranteed to stay positive [RV13]. Aside from being non-physical, in the case of negative values, droplets might rupture. In practice, all of our simulations remain positive, excluding the evaporation example. Nevertheless, the evaporation term has a stabilizing effect, indeed, negative mass concentrations are also

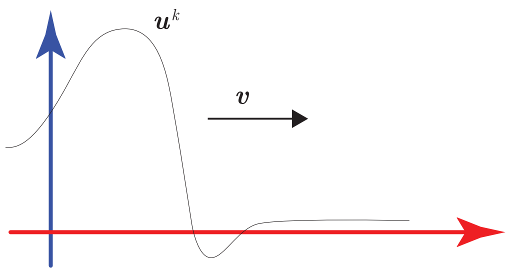


Figure 7.8: Capillarity ridge with high velocity and undershooting.

evaporated. Intuitively, positivity is difficult to maintain due to the jump in pressure along the triple line (the interface where air, solid and liquid meet). Moreover, the so-called capillary ridge is formed, due to the competition between surface tension and other forcing effects, e.g., gravity, see Figure 7.8 and 7.9. Thus, right where the film is at its thinnest, the resulting velocity is high, implying instability along the direction of motion. We leave further investigation of the issue of positivity preservation for future work.

Meshes with creases. In general, the model we developed in Section 7.2 has a strong dependency on the consistency of the vertex normals. In practice, general meshes might have creases, or small dihedral angles, which will cause H to be arbitrarily negatively large and non-smooth. This can have a detrimental effect on the simulation, as the fluid will be drawn towards these singular locations. There are two possible remedies for this situation: we can either refine the mesh (possibly non-uniformly), however that would require additional pre-processing before one can apply our scheme to an arbitrary model. Alternatively, we can add a regularizer to the energy so that it is easier to control the simulation. We opted for the second option, as it makes our method easier to use, and can allow the artist some freedom to control the simulation in a non-physical way. Hence, for meshes with creases (see e.g., Fig. 7.17), we multiply the stiffness matrix $\mathbf{L}$ defined in Section 7.5 by a constant $1 \leq \mathbf{r} \leq 100$. This effectively adds some numerical diffusion, allowing for more smooth solutions. Note that discrete conservation of mass is not affected by this modification.

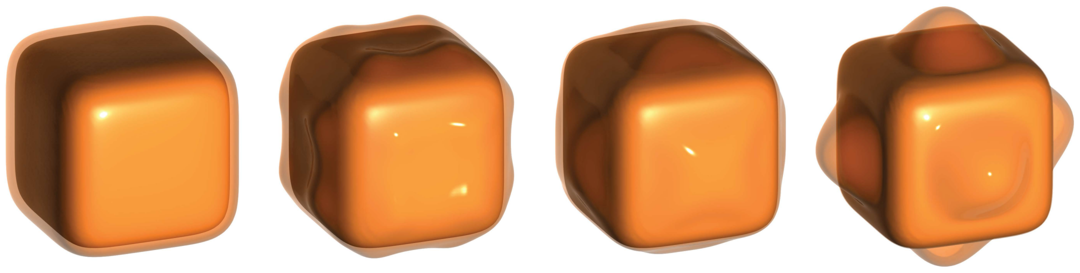


Figure 7.9: In the absence of gravity, the fluid departs areas where the mean curvature is strongly negative and capillary ridges form. Later, surface tension balances the fluid on top of every face, cf. [RRS02] ($u_0 = 0.1, \mathfrak{b} = 0, \epsilon = 0.1, \beta = 0$).

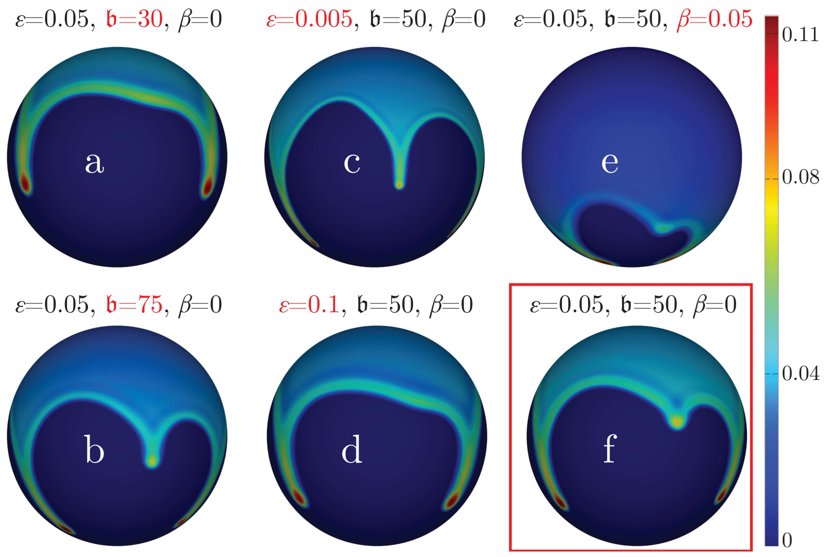


Figure 7.10: Fingering behavior for varying parameters, at $t = 10$. In every column, one parameter is modified from the reference configuration (f). See the text for details.

Detachment of fluid. As the fluid is "tied" to the surface, droplets cannot detach when they become too large. In these cases, the droplets grow narrower and taller until equilibrium is reached and the approximate lubrication solution is stable, although the full 3D flow is not. Note, that in this case one could potentially switch to a full 3D simulation, which will allow the droplet to separate from the surface. This is an interesting direction for future research.

7.6 Experimental results

Parameter exploration. We begin by exploring the effect of various parameter choices on the simulation of the thin film. For this example, we choose a sphere as a simple geometric model with limited curvature effects on the flow. The basic experiment includes placing a concentration of fluid at the top of the sphere, with slightly perturbed initial conditions to avoid perfect symmetry. Due to gravity the fluid flows downward, and the initial perturbations give rise to fingering instabilities, (see [TH10] for an experimental demonstration of fingering on a sphere). The result for the parameters $\epsilon = 0.05, \mathfrak{b} = 50, \beta = 0$ is shown in Figure 7.10 (f), demonstrating the emergence of a secondary finger in the center (see also Fig. 7.16, showing multiple fingers in a wine glass).

We refer to this setup as the *reference configuration*, and now modify in every column of the figure a single parameter to isolate its effect on the simulation, for which we show a snapshot at time $t = 10$. Left: varying $\mathfrak{b}$ changes the speed with which the film flows downward, without strongly affecting the shape of the fingers. Specifically, for a lower $\mathfrak{b}$ value (a), the secondary finger does not emerge yet, whereas for a higher $\mathfrak{b}$ value (b) it is more pronounced than in the reference configuration. Middle: changing ϵ

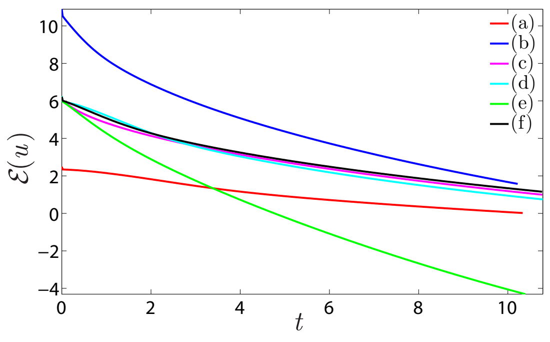


Figure 7.11: The energy $\mathcal{E}(\mathbf{u})$ for the simulations in Figure 7.10.

affects the surface tension component, and therefore the shape of the fingers. Reducing ϵ yields thinner fingers (c), whereas increasing it (d) makes more viscous thick fingers, and eliminates the secondary finger. Right: increasing β considerably speeds up the fluid (e), allowing it to flow more freely in all directions (as opposed to increasing $\mathfrak{b}$ which causes faster flow in the direction of gravity).

Energy reduction. The numerical scheme we use is guaranteed by construction to reduce the energy $\mathcal{E}(\mathbf{u})$ at every time step. Figure 7.11 shows the energy decay in time, for the different simulations in Figure 7.10. We observe that the slip parameter β affects the speed with which the energy is reduced, the gravity parameter $\mathfrak{b}$ also affects the initial value of the energy, and the parameter ϵ has a minor impact on the energy, as it is dominated by the leading order term.

Thin films interaction. Figure 7.12 demonstrates the flow and interaction of thin films on the moomoo model. The higher bulk of fluid accumulates beneath the horns of the model, followed by a faster motion when it comes in contact with the lower bulk of fluid (see also Figure 7.14). Then, the motion is mostly determined by the two main fingers flowing on the sides of the model. In Fig. 7.15 we show the interaction of many droplets viewed from four sides of the unit sphere. We repeatedly pour new droplets at the top of the sphere at a fixed rate and drain the liquid from the bottom.

Droplet formation. A thin film concentrating *beneath* a flat surface develops an instability called *droplet formation* (cf. [SK98]). In Figure 7.13, we start with a uniform layer of fluid on the torus with small perturbations, and allow it to drop beneath the torus due to gravity. As the fluid accumulates around the circular set of lowest points, droplets form.

Evaporation. Figure 7.14 shows how evaporation ($c_e = 0.01$) and the precursor layer affect the motion of the film. We deposit precursor layers of different heights on the two halves of the Moai model and place a similar bulk of fluid near the eyes. Due to the initially thicker precursor layer, even though it evaporates quickly, the film on the

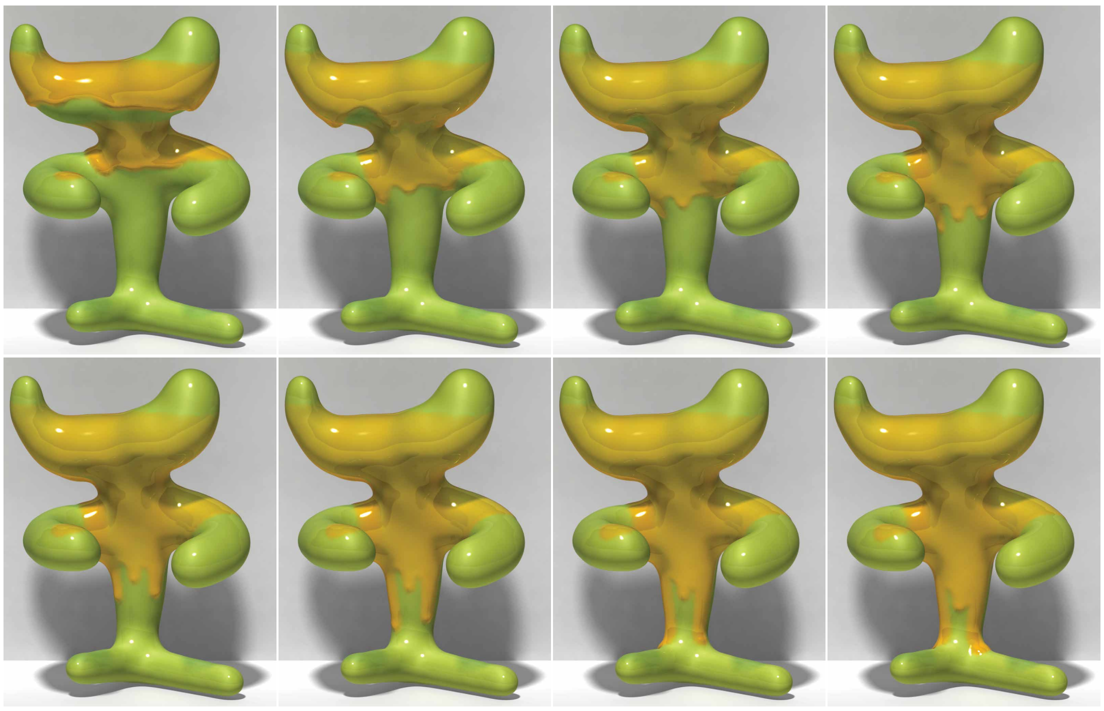


Figure 7.12: Flow on the moomoo model ($\mathfrak{b} = 20, \epsilon = 0.1, \beta = 0$). Note how the upper and lower films interact: the larger mass density of the upper film causes it to catch up with the lower front leading to the formation of quickly propagating fingers.

left part of the model flows to a greater distance compared to the film on the right. Eventually, all the film evaporates.

7.7 Van der Waals potential term.

As mentioned in the limitations section, a major drawback of our method is that the positivity of the mass density $\mathbf{u}$ is *not* guaranteed. One approach towards solving this issue is to add the integrated non-linear potential term $\int_\Gamma W(u)\mathrm{d}a \approx \mathbf{1}_\mathcal{V}^T \mathbf{G}_\mathcal{V} \mathbf{W}(\mathbf{u})$ to the discrete energy $\mathcal{E}^\epsilon(\mathbf{u})$. The purpose of this term is to penalize values of $\mathbf{u}$ that are under a certain threshold $\mathbf{u}_\mathbf{p}$. A computationally simple choice, commonly used to model intermolecular forces, is the well-known *Lennard-Jones (LJ) potential* [Jon24] given by

$$\mathbf{W}(\mathbf{u}) = \frac{1}{2}\left(\frac{\mathbf{u}_\mathbf{p}}{\mathbf{u}}\right)^4 - \left(\frac{\mathbf{u}_\mathbf{p}}{\mathbf{u}}\right)^2 .$$

In the context of thin films, the LJ potential was used in [GR01], and in addition to maintaining the height of the precursor layer, it also leads to the spontaneous formation of droplets (*pearling*) due to the potential well (see inset figure).

Namely, the modified energy favors large densities of fluid (where the potential is zero) or densities of the precursor layer size (where the potential has a minimum). Overall, using the LJ potential stabilizes the simulation by promoting the continued positivity of the solution. Although it is not entirely accurate physically, it is similar

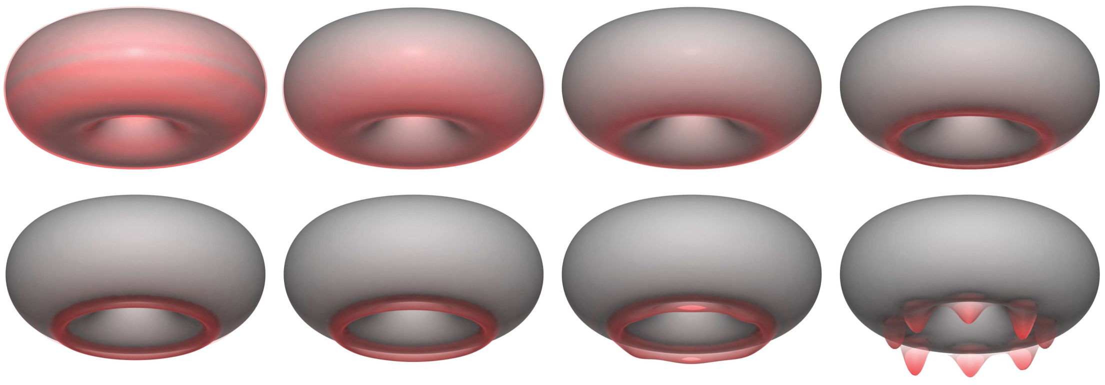
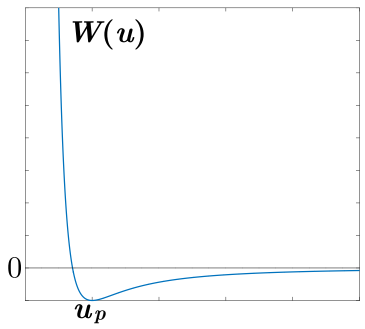


Figure 7.13: Starting from a perturbed uniform layer of fluid, the fluid flows downwards, accumulates and finally forms droplets.

enough to the real intermolecular interactions, that occur between substrate, liquid film and the air and determine the hydrophobic/hydrophilic properties of the surface, to achieve visually appealing results.



The modifications needed to incorporate the LJ potential can be summarized as follows. The new energy is given by $\mathcal{E}^{\epsilon}_{\mathbf{W}}(\mathbf{u}) = \mathcal{E}^{\epsilon}(\mathbf{u}) + \mathbf{1}^T_{\mathcal{V}}\mathbf{G}_{\mathcal{V}}\mathbf{W}(\mathbf{u})$. Thus, the new Euler–Lagrange equations (7.15) can be reduced to the following primal-dual system

$$\begin{aligned} \mathbf{p} &= \mathbf{a} + \epsilon\left(\mathbf{B} + G_{\mathcal{V}}^{-1}\mathbf{L}\right)\mathbf{u} + \mathbf{W}'(\mathbf{u})\ , \\ \mathbf{u} &= \mathbf{u}^k - \tau\mathbf{R}(\mathbf{u}^k)G_{\mathcal{V}}\mathbf{p}\ . \end{aligned} \tag{7.17}$$

Therefore, the resulting Newton system can be written as

$$\begin{pmatrix} \mathrm{id} & -\epsilon(\mathbf{B} + G_{\mathcal{V}}^{-1}\mathbf{L}) - [\mathbf{W}''(\mathbf{u})] \\ \tau\mathbf{R}(\mathbf{u}^k)G_{\mathcal{V}} & \mathrm{id} \end{pmatrix} \begin{pmatrix} \delta\mathbf{p} \\ \delta\mathbf{u} \end{pmatrix} = \begin{pmatrix} \mathbf{r_p} \\ \mathbf{r_u} \end{pmatrix}\ , \tag{7.18}$$

where

$$\begin{aligned} -\mathbf{r_p} &= \mathbf{p} - \mathbf{a} - \epsilon(\mathbf{B} + G_{\mathcal{V}}^{-1}\mathbf{L})\mathbf{u} - \mathbf{W}'(\mathbf{u})\ , \\ -\mathbf{r_u} &= \mathbf{u} - \mathbf{u}^k + \tau\mathbf{R}(\mathbf{u}^k)G_{\mathcal{V}}\mathbf{p}\ . \end{aligned}$$

The correction for $\delta\mathbf{p}$ can be eliminated, resulting in a single equation for the update of

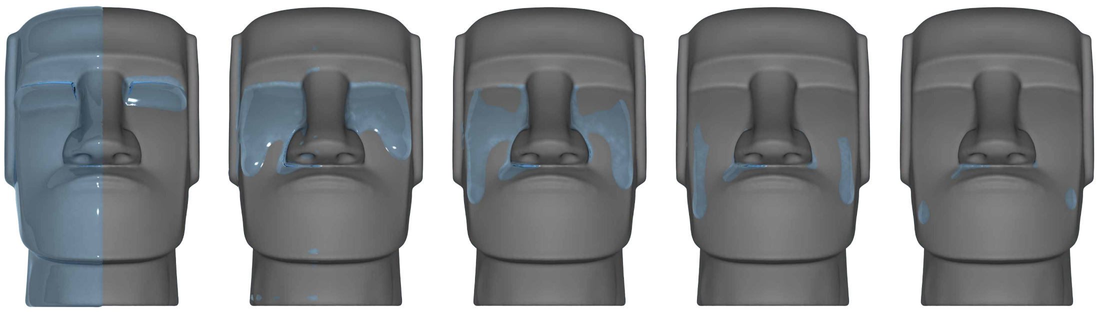


Figure 7.14: Evaporation effect on the evolution of the film.

$\mathbf{u}$. The modified update rule (7.16) for the correction is given by

$$\begin{aligned}\Big(\mathrm{id} + \tau\,\mathbf{R}(\mathbf{u}^k)G_{\mathcal{V}}\left(\epsilon(\mathbf{B}+G_{\mathcal{V}}^{-1}\mathbf{L}) + [\![\mathbf{W}''(\mathbf{u})]\!]\right)\Big)\delta\mathbf{u} = \\ \mathbf{u} - \mathbf{u}^k + \tau\mathbf{R}(\mathbf{u}^k)G_{\mathcal{V}}\left(\mathbf{a} + \epsilon\left(\mathbf{B}+G_{\mathcal{V}}^{-1}\mathbf{L}\right)\mathbf{u} + \mathbf{W}'(\mathbf{u})\right)\ .\end{aligned} \tag{7.19}$$

A single Newton step takes the form of $\mathbf{u} \leftarrow \mathbf{u} - \gamma\delta\mathbf{u}$, with $0 < \gamma \leq 1$ such that the energy is reduced. In practice we took $\gamma = 1$ in all of the examples that we show. As for the initial guess for the Newton iterations, we took $\mathbf{u} = \mathbf{u}^k$. Unfortunately, the concavity of $\mathbf{W}(\mathbf{u})$ poses a too strict requirement on τ for the system (7.19) to be invertible. In practice, we split the potential to its convex $\mathbf{W}_+(\mathbf{u})$ and concave $-\mathbf{W}_-(\mathbf{u})$ parts, so that the usual bound discussed in subsection 7.5.3 can be used. Specifically, we define

$$\tilde{\mathbf{W}}(\mathbf{u}^k, \mathbf{u}) = \mathbf{W}_+(\mathbf{u}) - \Big(\mathbf{W}_-(\mathbf{u}^k) + \mathbf{W}'_-(\mathbf{u}^k)(\mathbf{u} - \mathbf{u}^k)\Big)\ ,$$

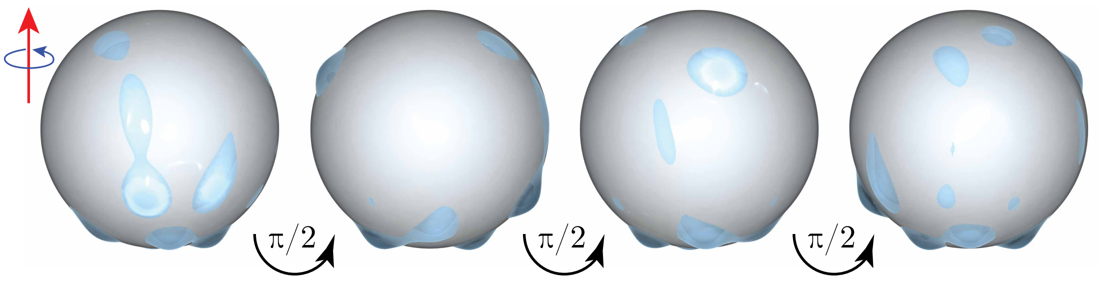


Figure 7.15: Rain of droplets lead to their interesting interaction over the sphere (see the video for the full simulation). The sphere is shown from its four sides, where the axis of rotation is shown above.

where for the LJ potential we have $\mathbf{W}_+(\mathbf{u}) = \frac{1}{2}\left(\frac{\mathbf{u_p}}{\mathbf{u}}\right)^4$ and $\mathbf{W}_-(\mathbf{u}) = \left(\frac{\mathbf{u_p}}{\mathbf{u}}\right)^2$. Finally, we modify the system (7.19) such that $\mathbf{W}''(\mathbf{u}) \to \mathbf{W}''_+(\mathbf{u})$ and $\mathbf{W}'(\mathbf{u}) \to \mathbf{W}'_+(\mathbf{u}) - \mathbf{W}'_-(\mathbf{u}^k)$. Note that a small value for the threshold $\mathbf{u_p}$ can approximate an effectively de-wetted surface (i.e. a very thin precursor layer) with droplets of apparently compact support. As small values of $\mathbf{u_p}$ exacerbate the non-convexity of the LJ potential, this necessitates smaller time steps.

In Figure 7.18 we show the effect of *pearling* that occurs whenever a trail of thin layer of liquid appears during the motion. In Figure 7.19 we show that due to the high attractive and repulsive forces, droplets emerge spontaneously. Both of these effects are achievable due to the Van der Waals non-linear potential term.

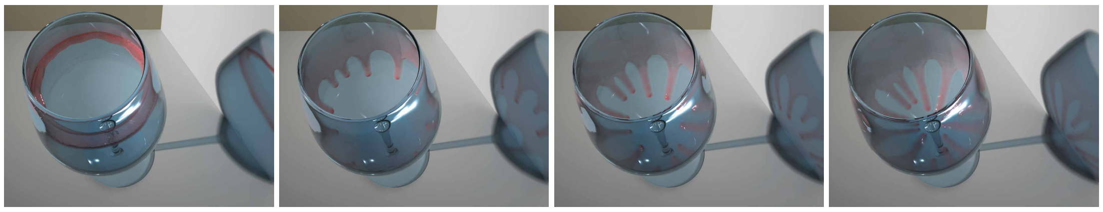


Figure 7.16: Multiple fingers on the inside of a glass of wine ($\mathfrak{b} = 500, \epsilon = 0.0001, \beta = 0$).

7.8 Conclusion

We presented a novel method for simulating viscous thin film flow on triangulated meshes. Our approach is based on a variational time discretization and is therefore stable and allows for large time steps. Furthermore, we guarantee by construction that the discrete total mass is preserved and that the discrete energy is non-increasing. The algorithm is based on a single sparse linear solve per iteration, and is therefore very efficient. We demonstrated various intricate film motions, such as viscous fingering and droplet interaction.

There are many potential extensions to our model. For instance, it might be possible to extend the model to handle effects due to surface tension gradient. Also, our discretization of the mass transport constraint might be potentially useful in additional applications. Finally, we mentioned various extensions throughout the paper such as positivity preservation and fluid detachment which might be interesting to achieve.

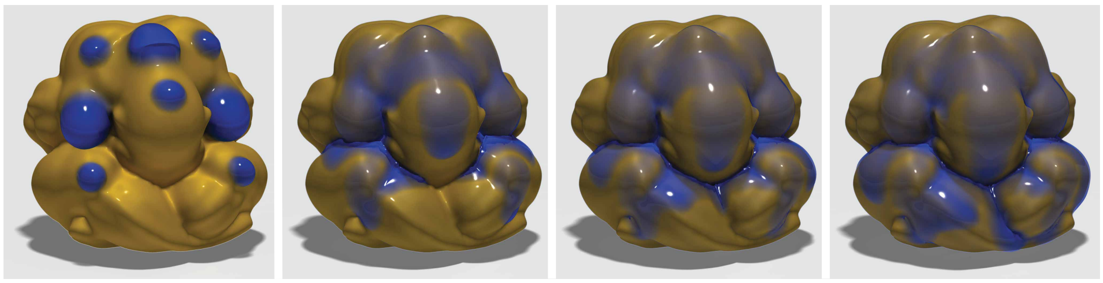


Figure 7.17: Thin film flow on a geometrically complicated model. Note how starting from a few blobs of fluid, the film naturally follows the creases of the object, merges and splits accordingly.

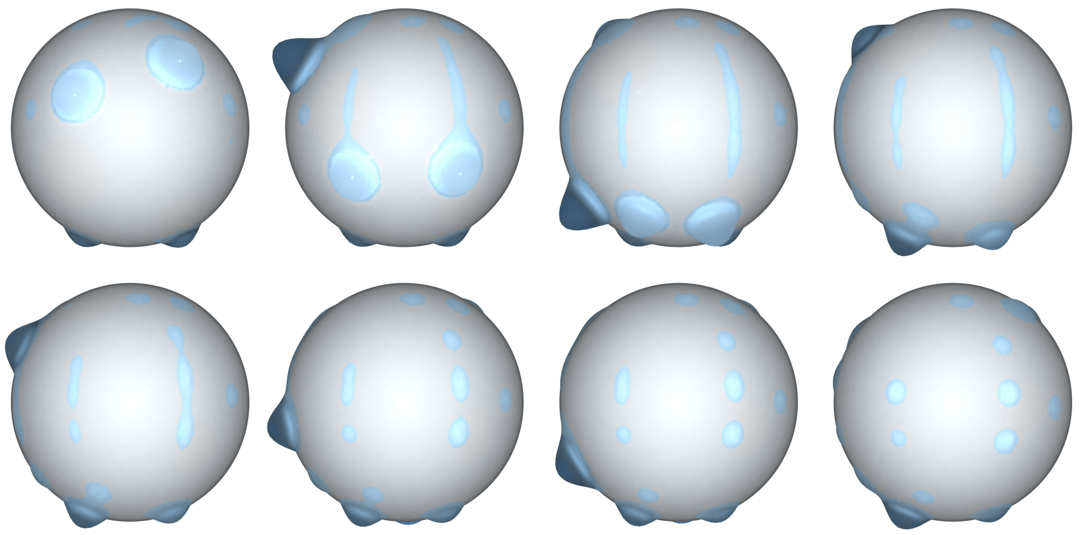


Figure 7.18: The two big droplets (top, left) travel fast enough to leave a trail of fluid behind (top, middle and right), which later separates into several smaller droplets (bottom). This phenomenon is known as *pearling*.

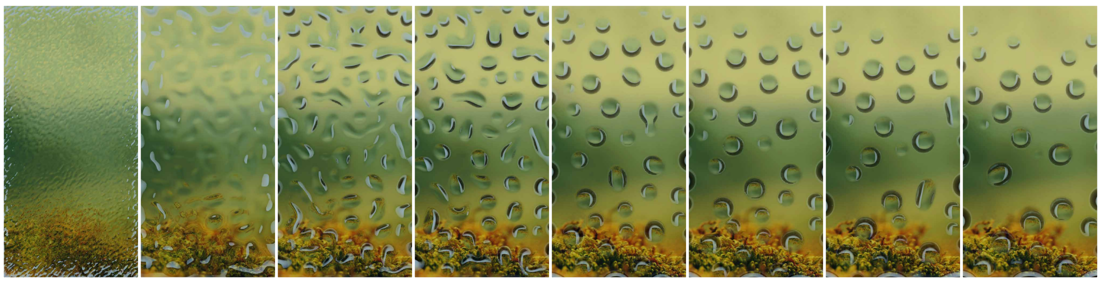


Figure 7.19: Starting from a slightly perturbed uniform layer of fluid (top, left), droplets quickly form due to the attractive/repulsive forces resulting from the van der Waals non-linear potential (top, middle and right). Later, the droplets travel downwards due to gravity and several merges between droplets occur (bottom).

Appendix A

Appendix of Chapter 2

A.1 Proof of lemma 2.2.1

Lemma 2.2.1. Let T_F^t, $t \in \mathbb{R}$ be the functional operator representations of the flow diffeomorphisms $\Phi_V^t : M \to M$ of V, defined by $T_F^t(f) = f \circ \Phi_V^t$ for any function $f \in C^\infty(M)$. If D is a linear partial differential operator then $D_V \circ D = D \circ D_V$ if and only if for any $t \in \mathbb{R}$, $T_F^t \circ D = D \circ T_F^t$.

Proof. Let $p \in M$ and $f \in C^\infty(M)$ be a smooth function. If $V(p) = 0$, then $\Phi_V^t(p) = p$ and $D_V(f)(p) = 0$. It immediately follows that $D_V \circ D(f)(p) = D \circ D_V(f)(p)$ if and only if $T_F^t \circ D(f)(p) = D \circ T_F^t(f)(p)$ because the right hand side of both equation is equal to 0.

Now assume that $V(p) \neq 0$. There exists (see, e.g. [Spi99] Theorem 7, p.148) a local coordinate system in an open neighborhood of p such that $V = \frac{\partial}{\partial x}$ and D can be written as

$$D = \sum_{0<|\alpha|\leq n} a_\alpha(x,y)\partial^\alpha$$

where $\alpha = (i,j)$ is a multi-index , $|\alpha| = i + j$ and $\partial^\alpha = \frac{\partial^{|\alpha|}}{\partial x^i \partial x^j}$.

First assume that $T_F^t \circ D = D \circ T_F^t$. Since the derivative (with respect to t) of $f \circ \Phi_V^t(p)$ at $t = 0$ is equal to $D_V(f)(p)$, the differentiation with respect to t of the equality $D(f)(\Phi_V^t(p)) = D(f \circ \Phi_V^t(p))$ gives at $t = 0$: $D_V(D(f))(p) = D(D_V(f))(p)$. As this holds for any f and p, we deduce that $D_V \circ D = D \circ D_V$.

Assume now that $D_V \circ D = D \circ D_V$. As in the proof of Lemma 2.2.4, since the flow of V is a one parameter group we just need to prove that $T_F^t \circ D = D \circ T_F^t$ for t contained in an arbitrarily small interval containing 0 but not reduced to 0. Using the product rule we have

$$0 = D_V \circ D(f) - D(D_V(f)) \quad = \sum_{0<|\alpha=(i,j)|\leq n} \frac{\partial a_\alpha}{\partial x} \frac{\partial^\alpha f}{\partial x^i \partial x^j}.$$

Since this equality holds for any f we deduce that for any α, $\frac{\partial a_\alpha}{\partial x} = 0$. As a consequence, the coefficients a_α of D are constant along the trajectories of V in the local coordinate system and thus for $|t|$ small enough we obtain $T_F^t \circ D(f)(p) = D \circ T_F^t(f)(p)$. □

A.2 Proof of lemma 2.2.2

Lemma 2.2.2. A vector field V is a Killing vector field if and only if $D_V \circ L = L \circ D_V$.

Proof. As L is a differential operator, it follows from Lemma 2.2.1 that $D_V \circ L = L \circ D_V$ if and only if $T_F^t \circ L = L \circ T_F^t$. Recalling that the Laplace-Beltrami operator is invariant under the action of isometries of M, we immediately deduce that if V is a Killing vector field then $D_V \circ L = L \circ D_V$. Now, if $T_F^t \circ L = L \circ T_F^t$, then the Laplace-Beltrami operator L is preserved by the action of the diffeomorphisms Φ_V^t. Since L determines the metric on M, Φ_V^t have to be isometries. □

Lemma A.2.1. *Given two vector fields D_{V_1} and D_{V_2} that both commute with some operator D, the Lie derivative $\mathcal{L}_{V_1}(V_2)$ will also commute with D.*

Proof. Using that $DD_{V_1} = D_{V_1}D$ and $DD_{V_2} = D_{V_2}D$ we immediately obtain

$$\begin{aligned} D(D_{V_1}D_{V_2} - D_{V_2}D_{V_1}) &= DD_{V_1}D_{V_2} - DD_{V_2}D_{V_1} \\ &= D_{V_1}D_{V_2}D - D_{V_2}D_{V_1}D \\ &= (D_{V_1}D_{V_2} - D_{V_2}D_{V_1})D. \end{aligned}$$ □

A.3 Proof of lemma 2.2.4

Lemma 2.2.4. $D_{V_2} = (T_F)^{-1} \circ D_{V_1} \circ T_F$.

Proof. Given $p \in M$, by definition of the push forward we have $V_2(T(p)) = dT(V_1(p))$ where dT denotes the differential of the diffeomorphism T. Now if $f \in C^\infty(N)$ is a smooth function, then using the chain rule we get

$$\begin{aligned} D_{V_1} \circ T_F(f)(p) = D_{V_1}(f \circ T)(p) &= d(f \circ T)(V_1(p)) \\ &= df(dT(V_1(p))) \\ &= df(V_2(T(p))) \\ &= D_{V_2}(f)(T(p)) \\ &= T_F \circ D_{V_2}(f)(p) \end{aligned}$$

As T is a diffeomorphism, T_F is an isomorphism and we obtain $D_{V_2} = (T_F)^{-1} \circ D_{V_1} \circ T_F$. □

A.4 Proof of lemma 2.2.5

Lemma 2.2.5. Let $T^t = \Phi_V^t$ be the self-map associated with the flow of V at time t. Then if T_F^t is the functional representation of T^t, for any real analytic function f (see [DFN92], p. 210):

$$T_F^t f = \exp(t\ D_V) f = \sum_{k=0}^{\infty} \frac{(tD_V)^k f}{k!}.$$

Proof. The set of diffeomorphisms associated to the flow of V is a one parameter group: for $t, s \in \mathbb{R}$, $\Phi_V^{t+s} = \Phi_V^t \circ \Phi_V^s$ (see [Spi99, pg. 147, thm, 6]). The right hand side of the equality of the Lemma also having the same property, it sufficies to show it for t contained in any arbitrarily small interval containing 0 but not reduced to 0. Given $p \in M$, if $V(p) = 0$, then for any k, $(D_V)^k(f)(p) = 0$ and both hand sides of the equality are equal to $f(p)$. Now assume that $V(p) \neq 0$. There exists (see, e.g. [Spi99] Theorem 7, p.148) an analytic local coordinate system in an open neighborhod of p in which V is equal to $\frac{\partial}{\partial x}$. As a consequence without loss of generality we can assume that $V = \frac{\partial}{\partial x}$ and $p = 0$, and prove the equality in this coordinate system. As the flow of $\frac{\partial}{\partial x}$ is just a translation, the left hand side of the equality becomes $T_F f(0) = f(t)$. As $D_{\frac{\partial}{\partial x}}(f) = \frac{\partial f}{\partial x}$, the right hand side is just the Taylor expansion of f at 0 in the direction of x:

$$\sum_{k=0}^{\infty} \frac{t^k}{k!} \frac{\partial^k f}{\partial x^k}(0).$$

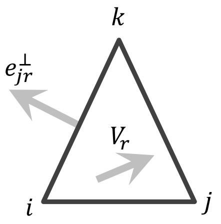
Since f is an analytic function, for $|t|$ small enough, this Taylor expansion is equal to $f(t)$. $\square$

A.5 Proof of lemma A.5.1

To compute the entries in the matrix S, we need to compute integrals of the form $d_{ij}^r = \int_{t_r} \gamma_i \langle \nabla\gamma_j, V_r \rangle \, d\mu$, where t_r is a triangle, γ_i is the hat basis function of the vertex i, and V_r is a constant vector in t_r. These integrals are non zero only if both i and j are vertices of t_r, and their value is given by the following Lemma.

Lemma A.5.1. Let $M = (X, F, N)$ and let V be a piecewise constant vector field on M. In addition, let $t_r = (i, j, k) \in F$ be a triangle and V_r be the value of V on t_r. Then:

$$d_{ij}^r = \int_{t_r} \gamma_i \langle \nabla\gamma_j, V_r \rangle \, d\mu = \frac{1}{6} \left\langle e_{jr}^{\perp}, V_r \right\rangle,$$



where $e_{jr}^{\perp}$ is the edge of t_r opposite to vertex j rotated by $\pi/2$, such that it points outside the triangle (see the inset figure for the notations).

Proof. The gradient of a basis hat function is given by (see e.g. [BKP+10]): $\nabla\gamma_j = e_{jr}^{\perp}/(2\mathcal{A}_r)$, where $\mathcal{A}_r$ is the area of the triangle t_r. This value is constant in t_r, as is V_r, and therefore we have:

$$d_{ij}^r = \int_{t_r} \gamma_i \left\langle \nabla\gamma_j, V_r \right\rangle d\mu = \frac{1}{2\mathcal{A}_r} \left\langle e_{jr}^{\perp}, V_r \right\rangle \int_{t_r} \gamma_i d\mu.$$

The integral of a basis hat function on the whole triangle is exactly the volume of a pyramid with basis t_r and height 1. Hence, $\int_{t_r} \gamma_i d\mu = \mathcal{A}_r/3$. Plugging this in d_{ij}^r we get:

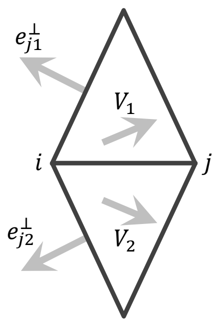
$$d_{ij}^r = \frac{1}{6} \left\langle e_{jr}^{\perp}, V_r \right\rangle.$$

Note, that this expression holds also when $j = i$. □

Now, computing the values of S_{ij} and S_{ii} is simply a matter of identifying on which set of triangles d_{ij}^r is not zero.

For S_{ij}, these are only the two triangles t_1, t_2 neighboring the edge (i, j). Hence we have:

$$S_{ij} = \frac{1}{6} \left(\left\langle e_{j1}^{\perp}, V_1 \right\rangle + \left\langle e_{j2}^{\perp}, V_2 \right\rangle \right),$$



where the notations are given in the inset figure.

For S_{ii}, the relevant triangles are the faces t_r which are near the vertex i (denoted by $N_F(i)$), hence we have:

$$S_{ii} = \frac{1}{6} \sum_{t_r \in N_F(i)} \left\langle e_{ir}^{\perp}, V_r \right\rangle.$$

Finally, we would like to show that $S_{ii} = -\sum_j S_{ij}$. From the definition of S_{ij} we have that:

$$\sum_j S_{ij} = \frac{1}{6} \sum_{j \in N(i)} \left(\left\langle e_{j1}^{\perp}, V_1 \right\rangle + \left\langle e_{j2}^{\perp}, V_2 \right\rangle \right).$$

By re-arranging the sum as a sum on the neighboring faces, we get:

$$\sum_j S_{ij} = \frac{1}{6} \sum_{r=(i,j,k) \in N_F(i)} \left(\left\langle e_{jr}^{\perp}, V_r \right\rangle + \left\langle e_{kr}^{\perp}, V_r \right\rangle \right).$$

It is easy to check that for a triangle $r = (i, j, k)$ we have:

$$e_{jr} + e_{kr} = (p_i - p_k) + (p_j - p_i) = p_j - p_k = -e_{ir},$$

and hence:

$$\sum_j S_{ij} = \frac{1}{6} \sum_{r=(i,j,k)\in N_F(i)} \left(\left\langle -e_{ir}^{\perp}, V_r \right\rangle\right) = -S_{ii}.$$

A.6 Proof of lemma 2.4.1

Lemma 2.4.1. Let $M = (X, F, N)$ and let V_1, V_2 be two piecewise constant vector fields on M. Then: $\hat{D}^F_{V_1} = \hat{D}^F_{V_2}$ if and only if $V_1 = V_2$.

Proof. We will show that given a tangent vector field V, and a corresponding operator $\hat{D}^F_V$, we can reconstruct V uniquely from $\hat{D}^F_V$. Since $\hat{D}^F_V$ is defined locally per face, where V is smooth, the uniqueness is in fact implied by the uniqueness property in the smooth case. However, for completeness we will validate this explicitly, by providing a reconstruction method that extracts V given $\hat{D}^F_V$.

Given a face $r = (i, j, k)$ we compute $c_i = (\hat{D}^F_V(\gamma_i))_r$ and similarly for c_j, c_k, where γ_i is the hat basis function of vertex i. Now, we consider the set of constraints we have on V_r. First, by definition we have that $(\hat{D}^F_V(\gamma_i))_r = \langle \nabla\gamma_i, V_r\rangle = c_i$. In addition, V_r should be tangent to the triangle, hence $\langle V_r, N_r\rangle = 0$, where N_r is the normal. This yields the following linear system for V_r:

$$\begin{pmatrix} (\nabla\gamma_i)_r^T \\ (\nabla\gamma_j)_r^T \\ (\nabla\gamma_k)_r^T \\ N_r^T \end{pmatrix} V_r = \begin{pmatrix} c_i \\ c_j \\ c_k \\ 0 \end{pmatrix}$$

However, since $s = \gamma_i + \gamma_j + \gamma_k = 1$, we have that $\hat{D}^F_V(s) = c_i + c_j + c_k = 0$, and similarly $\nabla\gamma_i + \nabla\gamma_j + \nabla\gamma_k = 0$. Therefore, one of the equations is redundant. Furthermore, $\nabla\gamma_i$ is in the direction of the edge (j, k) rotated by $\pi/2$, and similarly for $\nabla\gamma_j$ and they are both orthogonal to N_r. Therefore, if the triangle is not degenerate, $\nabla\gamma_i, \nabla\gamma_j, N_r$ are linearly independent, and the system is full rank. Since we know that $\hat{D}^F_V$ was constructed from V, the system has a unique solution given by V_r. □

A.7 Proof of lemma 2.4.2

Lemma 2.4.2. Let $M_1 = (X_1, F, N_1)$ and $M_2 = (X_2, F, N_2)$ be two triangle meshes with the same connectivity but different metric (i.e. different embedding). Additionally, let V_1 be a piecewise constant vector field on M_1, then $\hat{D}^F_{V_1} = \hat{D}^F_{V_2}$.

Here $(V_2)_r = A(V_1)_r$, where A is the linear transformation that takes the triangle r in M_1 to the corresponding triangle in M_2. Note that $\hat{D}_{V_i}$ is computed using the embedding X_i.

Proof. By definition we have that

$$(\hat{D}^F_{V_1})_{ri} = \langle (\nabla \gamma_i)_1, (V_1)_r \rangle = \left\langle \frac{R^{90}(p^1_k - p^1_j)}{2\mathcal{A}_1}, (V_1)_r \right\rangle,$$

where the face $r = (i, j, k)$, p^1_i are the coordinates in X_1 of vertex i and R^{90} is counter-clockwise rotation by $\pi/2$ in the plane of the triangle r. On the other hand we have

$$\begin{aligned}(\hat{D}^F_{V_2})_{ri} = \langle (\nabla \gamma_i)_2, (V_2)_r \rangle &= \left\langle \frac{R^{90}(p^2_k - p^2_j)}{2\mathcal{A}_2}, (V_2)_r \right\rangle \\ &= \left\langle \frac{R^{90}A(p^1_k - p^1_j)}{2|A|\mathcal{A}_1}, A(V_1)_r \right\rangle,\end{aligned}$$

where $|A|$ is the determinant of A. It is easy to check directly, that for any A we have that: $A^T(R^{90})^T A = |A|(R^{90})^T$, which implies $\hat{D}^F_{V_1} = \hat{D}^F_{V_2}$, as required. □

A.8 Proof of lemma 2.4.3

Lemma 2.4.3. Let $M = (X, F, N)$, V a piecewise constant vector field on M, $f = \sum_i f_i \gamma_i$ a PL function on M, and w_i the Voronoi area weights, then:

$$\sum_{i=1}^{|X|} w_i (\hat{D}_V f)_i = \sum_{i=1}^{|X|} w_i f_i (\nabla \cdot V)_i.$$

Proof. From the definition of $\hat{D}_V$, we have that

$$\sum_{i=1}^{|X|} w_i (\hat{D}_V f)_i = \sum_{i=1}^{|X|} (W \hat{D}_V f)_i = \sum_{i=1}^{|X|} (Sf)_i = \sum_{i=1}^{|X|} \sum_{j=1}^{|X|} S_{ij} f_j.$$

Switching the roles of the indices i, j, we get:

$$\sum_{i=1}^{|X|} \sum_{j=1}^{|X|} S_{ji} f_i = \sum_{i=1}^{|X|} g_i f_i, \quad g_i = \sum_{j=1}^{|X|} S_{ji}.$$

The only non-zero entries in the i-th column of S are on the diagonal and entries S_{ji} such that j is a neighbor of i. Thus we have:

$$g_i = S_{ii} + \sum_{j \in N(i)} S_{ji}.$$

Plugging in the definition of S_{ji} and S_{ii} we get:

$$g_i = \frac{1}{6} \sum_{t_r \in N_F(i)} \left\langle e^{\perp}_{ir}, V_r \right\rangle + \frac{1}{6} \sum_{j \in N(i)} \left(\left\langle e^{\perp}_{i1}, V_1 \right\rangle + \left\langle e^{\perp}_{i2}, V_2 \right\rangle \right).$$

Again, we can re-arrange the second sum as a sum on neighboring faces and get:

$$\begin{aligned} g_i &= \frac{1}{6} \sum_{t_r \in N_F(i)} \left\langle e_{ir}^{\perp}, V_r \right\rangle + \frac{1}{6} \sum_{t_r \in N_F(i)} \left(\left\langle e_{ir}^{\perp}, V_r \right\rangle + \left\langle e_{ir}^{\perp}, V_r \right\rangle \right) \\ &= \frac{1}{2} \sum_{t_r \in N_F(i)} \left\langle e_{ir}^{\perp}, V_r \right\rangle = w_i (\div V)_i . \end{aligned}$$

Finally, we get:

$$\sum_{i=1}^{|X|} w_i (\hat{D}_V f)_i = \sum_{i=1}^{|X|} g_i f_i = \sum_{i=1}^{|X|} w_i (\div V)_i f_i,$$

as required. □

Appendix B

Appendix of Chapter 3

B.1 Challenges in the discrete setting

We would like to show that the metric compatibility is impossible to achieve in the discrete setting even when functions and vector fields do not live on the same domain. Below we will assume that vector fields are discretized on the faces of a triangle mesh and functions are discretized on some other domain (vertices, edges, faces, etc.). However, the proof is quite general, and can potentially be extended to the case where even the vector fields are also discretized on some other domain (e.g. on edges), depending on the choice of inner product.

We will use the following formulation of the metric compatibility:

$$\tilde{D}_X \mathcal{A}(\langle U, V\rangle) = \mathcal{A}(\left\langle \tilde{\nabla}_X U, V \right\rangle + \left\langle \tilde{\nabla}_X V, U \right\rangle). \tag{B.1}$$

Here $\tilde{D}_X$ is a covariant derivative for functions with respect to the vector field X. I.e., $\tilde{D}_X$ takes a function defined on some domain (e.g., vertices or edges) and produces a function defined on the same domain. $\tilde{\nabla}_X U$ is the covariant derivative for vector fields, and the inner product is the standard inner product of vector fields in $\mathbb{R}^3$. Since the inner product $\langle U, V\rangle$ produces a function on the faces of the triangle mesh, we need an operator $\mathcal{A}$ that takes functions on faces and produces functions on vertices or edges.

We will assume that $\mathcal{A}$ has the following properties:

1. It is linear: $\mathcal{A}(f+g) = \mathcal{A}(f) + \mathcal{A}(g)$.
2. It maps constant functions to constant functions. I.e., if we are given a function f, such that the value of f on face i is equal to its value on face j for every j, then $\mathcal{A}(f)$ is also a constant function on the target domain (e.g, vertices or edges).
3. It is non-negative.

Under these conditions, we have the following result:

Lemma B.1.1. *If $\tilde{D}_X$ is a linear operator such that $\tilde{D}_X f = 0$ if f is a constant function, and the covariant derivative for vector fields is linear: $\tilde{\nabla}_X(U_1 + U_2) = \tilde{\nabla}_X U_1 + \tilde{\nabla}_X U_2$, then the metric compatibility condition (Eq. B.1) implies that $\tilde{D}_X f = 0$ for all f in the range of $\mathcal{A}$. I.e., $\tilde{D}_X \mathcal{A} h = 0$ for any h.*

Proof. We will use V_i to denote a vector field which is non-zero on face i and has unit norm, and use $e_i = \langle V_i, V_i \rangle$, as the indicator function of face i.

1. The metric compatibility condition implies that:

$$\tilde{D}_X \mathcal{A}(e_i) = \tilde{D}_X \mathcal{A}(\langle V_i, V_i \rangle) = 2\mathcal{A}(\langle \nabla_X V_i, V_i \rangle)$$

Since $V_i = 0$ on any face other than i, we have $\nabla_{\mathcal{X}} \mathcal{V}_\rangle V_i = a_i e_i$ for some scalar a_i. Thus,

$$\tilde{D}_X \mathcal{A}(e_i) = 2\mathcal{A}(a_i e_i) = 2a_i \mathcal{A}(e_i).$$

In other words, $\mathcal{A}(e_i)$ is an eigenvector of $\tilde{D}_X$ with eigenvalue $2a_i$. Our goal will be to show that $a_i = 0$ for all i, since in that case $\tilde{D}_X \mathcal{A}(h) = 0$ for any h.

2. For any $i \neq j$, we have $\mathcal{V}_\rangle V_j = 0$. Thus:

$$0 = \tilde{D}_X \mathcal{A}(\langle V_i, V_j \rangle) = \mathcal{A}(\langle \nabla_X V_i, V_j \rangle) + \mathcal{A}(\langle \nabla_X V_j, V_i \rangle).$$

3. Let $V = \sum_i V_i$. Note that $\langle V, V \rangle = \sum e_i = c$ a constant function on the faces. Thus $\tilde{D}_X \mathcal{A}(\langle V, V \rangle) = 0$. But

$$\begin{aligned} \tilde{D}_X \mathcal{A}(\langle V, V \rangle) &= 2\mathcal{A}(\langle \nabla_X V, V \rangle) = 2\mathcal{A}(\left\langle \nabla_X \sum_i V_i, \sum_i V_i \right\rangle) \\ &= 2\mathcal{A}(\left\langle \sum_i \nabla_X V_i, \sum_i V_i \right\rangle) = 2\mathcal{A}(\sum_{i,j} \langle \nabla_X V_i, V_j \rangle) \\ &= 2\sum_{i,j} \mathcal{A}(\langle \nabla_X V_i, V_j \rangle) = 0 \end{aligned}$$

Using parts 1. and 2. above (which states that that the cross terms cancel out), this further simplifies to:

$$\tilde{D}_X \mathcal{A}(\langle V, V \rangle) = 2\sum_i \mathcal{A}(\langle \nabla_X V_i, V_i \rangle) = 2\sum_i a_i \mathcal{A}(e_i) = 0.$$

4. Since $\mathcal{A}(e_i)$ is an eigenvector of $\tilde{D}_X$ with eigenvalue $2a_i$, the previous part can be rewritten as (by summing eigenvectors with the same eigenvalue): $\sum_j \lambda_j \phi_j = 0$, where λ_j are all distinct and non-zero, and $\phi_j = \sum_i \mathcal{A}(e_i)$ such that $2a_i = \lambda_j$.

We claim that ϕ_j are all linearly independent. To see this suppose that $\phi_k = \sum_{j\neq k} b_j\phi_j$, for some k, such that $\{\phi_j\}$ are linearly independent and $b_j \neq 0$. Then since $\tilde{D}_X\phi_i = \lambda_i\phi_i$ we have:

$$\sum_{j\neq k} b_j\lambda_j\phi_j = \sum_{j\neq k} b_j\lambda_k\phi_j,$$

which implies (since b_j is non-zero) that $\lambda_k = \lambda_j$ for all j, which is a contradiction.

5. Since ϕ_j are all linearly independent, $\sum_j \lambda_j\phi_j = 0$, implies that $\lambda_j\phi_j = 0$ for all j. Thus, either $\lambda_j = 0$ or $\phi_j = 0$. But $\phi_j = \sum_i \mathcal{A}(e_i)$ for some index i, and A is assumed to be non-negative, $\sum_i \mathcal{A}(e_i) = 0$ only if $\mathcal{A}(e_i) = 0$ for every i. But this means that $\lambda_j = a_i = 0$. Therefore, $a_i = 0$ for all i, which implies that $\tilde{D}_X\mathcal{A}(h) = 0$ for all h. □

B.2 Properties of the continuous operators associated with the Levi-Civita covariant derivative

The following lemmas all deal with smooth manifolds. We will assume that each manifold is compact and without boundary. Moreover we will assume that all vector fields are not only smooth but *analytic*. Note that this requirement is necessary for Lemma B.3.1, and not e.g. for Lemma B.2.3 but we will assume it throughout for simplicity.

Lemma B.2.1. *For a closed oriented surface M without boundary, $\nabla_U V = 0\ \forall\ U$ if and only if $V = 0$ or M is a flat torus.*

Proof. The proof of this lemma relies on the result of Lemma B.3.1 below. First, note that if M is not a genus 1 surface, then according to Hopf index theorem [Mor01, pg. 256], there must be some point p s.t. $V(p) = 0$. But then pick another point p' and construct a vector field Z such that the flow-lines of Z connect p and p'. I.e. $\Phi_Z(p,t) = p'$ for some t. Since $\nabla_Z V = 0$ this implies, using Lemma B.3.1 below, that $V(p') = V(p) = 0$, and thus $V = 0$ everywhere, since p' was arbitrary. Let assume now that M is a torus. Since $\nabla_U V = 0\ \forall\ U$, parallel transport around any paths must commute, so there is no curvature and thus M must be a flat torus. □

Lemma B.2.2. *Two vector fields U and V are equal if and only if $\nabla_U W = \nabla_V W$ for all vector fields W.*

Proof. Recall from the definition of parallel transport that if $\nabla_X V = 0$, then V is preserved by parallel transport along the trajectories of X. Suppose $X \neq 0$, so that there is some point p, s.t. $\Phi_t(p) \neq p$ for some t. Then for any vector field V, $V(\Phi_t(p))$ is the parallel transport of $V(p)$ along the trajectory of X from p to $\Phi_t(p)$. As the parallel transported image of $V(p)$ is uniquely defined, it is easy to build two vector fields V_1 and V_2 such that $V_1(p) = V_2(p)$ but $V_1(\Phi_t(p)) \neq V_2(\Phi_t(p))$, a contradiction. □

Lemma B.2.3. *A vector field U is divergence-free if and only if ∇_U is anti-symmetric with respect to the inner product on the surface. I.e., if and only if $\int_M \langle \nabla_U X, Y \rangle \, dx = -\int_M \langle \nabla_U Y, X \rangle \, dx$ for all vector fields X and Y.*

Proof. Suppose U is divergence-free. Then, using the metric compatibility of the covariant derivative:

$$\begin{aligned}&\int_M \left(\langle \nabla_U X, Y \rangle + \langle \nabla_U Y, X \rangle \right) dx \\ &= \int_M \nabla_U \langle X, Y \rangle \, dx = \int_M \mathrm{div}(U) \langle X, Y \rangle \, dx = 0\end{aligned}$$

where the second to last equality uses Stokes' theorem and integration by parts. Now, suppose that ∇_U is anti-symmetric. Then by the same argument we get: $\int_M \mathrm{div}(U) \langle X, Y \rangle \, dx = 0$ for any X and Y. Suppose $f = div(U)$ is not zero. Then there exists a point p such that $f(p) = \epsilon > 0$. Let Ω be a small neighborhood of p such that $f(p)$ does not change sign and is strictly greater than 0. By constructing a vector field X that vanishes outside of Ω, and considering $\int_M \mathrm{div}(U) \langle X, X \rangle \, dx$ it is easy to see that this integral must be positive. But this contradicts the assumption of anti-symmetry.□

B.3 Properties of the discrete operators associated with the Levi-Civita covariant derivative

Lemma B.3.1. *Let $\tilde{\Gamma}_{U,t} = \exp(t\tilde{\nabla}_U)$, where $\tilde{\nabla}_U$ is the matrix representation of the discrete covariant derivative operator defined in the main paper, and* exp *is matrix exponentiation. Then:*

$$\tilde{\nabla}_U(V)(p) = \frac{d}{dt}\Big(\tilde{\Gamma}_{U,t}(V)(p)\Big)\Big|_{t=0}. \tag{B.2}$$

Proof. We have: $\frac{d}{dt}\tilde{\Gamma}_{U,t} = \frac{d}{dt}\exp(t\tilde{\nabla}_U) = \tilde{\nabla}_U \exp(t\tilde{\nabla}_U)$, where we can use standard matrix derivative rules, as $\tilde{\nabla}_U$ does not depend on t. Hence, for $t = 0$ we get: $\frac{d}{dt}\tilde{\Gamma}_{U,t}\big|_{t=0} = \tilde{\nabla}_U$, as required. □

Lemma B.3.2. *If $\tilde{D}_U$ uses a full basis, then the operator $\tilde{\nabla}$ is invariant to rigid transformations. Namely, let $M = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ be a mesh embedded with coordinates $X \in \mathbb{R}^3$, and let U, V be two tangent vector fields on M. In addition, let $\mathcal{T}$ be a global rigid transformation $\mathcal{T} : \mathbb{R}^3 \to \mathbb{R}^3$. Then:*

$$(\tilde{\nabla}^{\mathcal{T}(X)})_{\mathcal{T}(U)} \mathcal{T}(V) = \mathcal{T}((\tilde{\nabla}^X)_U V), \tag{B.3}$$

where $\tilde{\nabla}^X$ is the discrete covariant derivative operator on a mesh embedded with coordinates X.

Proof. Since we only deal with vector quantities (e.g. the input vectors, and the edge vectors of the mesh), and intrinsic scalar quantities (e.g. triangle areas) it is clear that the definition is invariant to global translation.

Let $\mathcal{T}$ be given by a global rotation matrix R^T. Consider the gradient of the three components of V in the face $i \in \mathcal{F}$, namely the 3×3 matrix $G^X_{V,i}$ whose columns are $\big[(\nabla^X \mathcal{A} v_x)(i), (\nabla^X \mathcal{A} v_y)(i), (\nabla^X \mathcal{A} v_z)(i)\big]$, where $\mathcal{A}$ is an intrinsic averaging operator, and ∇^X is the gradient of non-conforming elements on a mesh embedded with coordinates X.

From the definition of the gradient, it is easy to check that

$$G^X_{V,i} = -(E^X_i)^T \mathcal{C}_i \mathcal{A} V / \triangle_i, \tag{B.4}$$

where E^X_i is a 3×3 matrix whose rows are the rotated vector edges of the face i, $\mathcal{C}_i$ is a $3 \times |\mathcal{E}|$ matrix which chooses the edges in the face i, V is a $|\mathcal{F}| \times 3$ matrix where the i-th row represents the vector in face i, and $\triangle_i$ is the area of face i. Similarly, for the rotated mesh we have:

$$G^{XR}_{VR,i} = -(E^{XR}_i)^T \mathcal{C}_i \mathcal{A} V R / \triangle_i, \tag{B.5}$$

since $\mathcal{C}_i$ is combinatorial, $\mathcal{A}$ and $\triangle_i$ are intrinsic, and rotating the vector field can be expressed as post multiplying by R. Similarly, rotating the coordinates (and thus the edge vectors) of X can also be expressed as post multiplying by R, hence we have: $E^{XR}_i = E^X_i R$. Combined with (B.4) and (B.5) we get:

$$G^{XR}_{VR,i} = R^T G^X_{V,i} R. \tag{B.6}$$

By definition, we have that $(\tilde{D}^X_U V)(i) = \big(\tilde{D}^X_U v_x, \tilde{D}^X_U v_y, \tilde{D}^X_U v_z\big)(i) = U(i) G^X_{V,i}$, where $U(i)$ is the vector U in the face i. Hence, plugging in (B.6) we get:

$$\big(\tilde{D}^{XR}_{UR}(VR)\big)(i) = (UR)(i) G^{XR}_{VR,i} = U(i) R R^T G^X_{V,i} R = U(i) G^X_{V,i} R, \tag{B.7}$$

hence: $\big(\tilde{D}^{XR}_{UR}(VR)\big)(i) = \big(\tilde{D}^X_U V\big)(i) R$.

It is straightforward to check that by projecting out the normal component we get $\big(\tilde{\nabla}^{XR}_{UR}(VR)\big)(i) = \big(\tilde{\nabla}^X_U(V)\big)(i) R$ as required. □

B.4 Periodic solution to Euler's Equation

In this section we consider the evolution of an incompressible inviscous fluid on a 2-dimensional sphere. Our goal is to show that if the velocity field at time 0 equals: $V(0) = U_0 + J\nabla\phi_j$ where U_0 is a Killing vector field, J is an operator that rotates a given vector field by $\pi/2$ in each tangent plane and ϕ_j is an eigenfunction of the Laplace-Beltrami operator corresponding to the j^{th} eigenvalue, then the solution to

Euler equation at time t will have the form

$$V(t) = U_0 + \sum_i a_i(t) J\nabla\phi_i.$$

Here, $a_i(t)$ are scalar-valued functions and ϕ_i are eigenfunctions of the Laplace-Beltrami operator corresponding to *the same* j^{th} eigenvalue. Thus, $V(t)$ is a linear combination of a KVF and a rotated gradient of an eigenfunction corresponding to the j^{th} eigenvalue for all times t. Moreover, we would also like to show that the vorticity $\omega(t) = \text{curl}(V(t))$ is advected isometrically by the flow.

To show that $V(t) = U_0 + \sum_i a_i(t) J\nabla\phi_i$, for all t, recall the vorticity formulation of Euler equation:

1. $V(t) = J\nabla\psi(t)$

2. $\omega(t) = -L\psi(t)$

3. $\frac{d}{dt}\omega(t) = -D_{V(t)}\omega(t)$

where $V(t)$ is the velocity field, ω is the vorticity, ψ is called the *stream function*, L is the Laplace-Beltrami operator and $D_{V(t)}$ the covariant derivative (of functions) in the direction of $V(t)$ (see e.g. [Tay96, pg. 536, Eq. (1.27)]).

Suppose $\psi(0) = \phi_1 + \phi_j$ where ϕ_1 corresponds to the first non-zero eigenfunction of the Laplace-Beltrami (note that $J\nabla\phi_1$ is a Killing vector field). Thus, we have: $w(0) = -L\psi(0) = -(\lambda_1\phi_1 + \lambda_j\phi_j)$ and $V(0) = -J\nabla\phi_1 - J\nabla\phi_j$. Now:

$$\begin{aligned} L\frac{d}{dt}\psi(0) &= D_{V(0)}L\psi(0) = \langle V(0), \nabla L\psi(0)\rangle \\ &= \langle J\nabla\phi_1 + J\nabla\phi_j, \lambda_1\nabla\phi_1 + \lambda_j\nabla\phi_j\rangle \\ &= \langle J\nabla\phi_1, \lambda_j\nabla\phi_j\rangle + \langle J\nabla\phi_j, \lambda_1\nabla\phi_1\rangle \\ &= (\lambda_j - \lambda_1)\langle J\nabla\phi_1, \nabla\phi_j\rangle \end{aligned}$$

Now since $U_0 = J\nabla\phi_1$ is a Killing vector field, $L\langle U_0, \nabla f\rangle = \langle U_0, \nabla Lf\rangle$ for any f, which implies in particular that $L\langle U_0, \nabla\phi_j\rangle = \lambda_j\langle U_0, \nabla\phi_j\rangle$, and therefore $\langle U_0, \nabla\phi_j\rangle$ is an eigenfunction of L corresponding to the j^{th} eigenvalue. Note that this implies that $\frac{d}{dt}\psi(0)$ is contained in the span of the eigenfunctions corresponding to the j^{th} eigenvalue. Moreover, using the same argument as above, the same is true for any t. Thus we have: $\psi(t) = \phi_1 + \sum_i a_i(t)\phi_i$ and $V(t) = U_0 + \sum_i a_i(t) J\nabla\phi_i$, for all t, where $a_i(t)$ are scalar valued functions of time and ϕ_i are eigenfunctions corresponding to the j^{th} eigenvalue of L.

Note that $V(t)$ is not a Killing Vector field for any time t. However, as we will show the vorticity function ω *is* advected isometrically by $V(t)$.

For this note that $w(t) = -L\psi(t) = -(\lambda_1\phi_1 + \lambda_j \sum_i a_i(t)\phi_i)$ for all t, and

$$\frac{d}{dt}\omega(t) = -D_{V(t)}\omega(t) = (\lambda_j - \lambda_1)\left\langle J\nabla\phi_1, \sum_i a_i(t)\nabla\phi_i\right\rangle.$$

Now consider another PDE for the evolution of ω (which would a-priori give a different flow).

$$\frac{d}{dt}\omega(t) = -D_{J\nabla\phi_1}\omega(t) = \lambda_j\left\langle J\nabla\phi_1, \sum_i a_i(t)\nabla\phi_i\right\rangle.$$

Note that when $\omega(t)$ has the form as above, these two equations only differ by a scalar, i.e. the speed of evolution. Moreover note that when $w(0) = -(\lambda_1\phi_1 + \lambda_j\phi_j)$ then $w(t)$ will have this form for all t for both PDEs. Thus, whether w is advected by $V(t)$ or by a constant $U_0 = J\nabla\phi_1$ the trajectory will be the same. Since we know that $J\nabla\phi_1$ is a Killing Vector field, this means that $\omega(t)$ is advected isometrically by $V(t)$.

Appendix C

Appendix of Chapter 4

C.1 Proof of lemma 4.3.1

Lemma 4.3.1. Given a cross field x and an arbitrary point $q \in M$, we compute the associated power vectors y_1 and y_2 at q using two different basis vectors b_1 and b_2, respectively. Then, for any real-valued function f, the following relation holds

$$\langle y_1, (\operatorname{grad} f)_{1,p} \rangle = \langle y_2, (\operatorname{grad} f)_{2,p} \rangle \ ,$$

where $(\operatorname{grad} f)_{i,p}$ is the power vector of $(\operatorname{grad} f)$ at q in the basis b_i.

Proof. Let x be one of the 4 vectors of the cross field at q. We represent it using b_i as $x = s_x R^{\theta_i} b_i$, where $s_x = \|x\|$, $\theta_i \in [0, 2\pi)$ and R^{θ} is counter-clockwise rotation by angle θ in the tangent plane of q. Similarly, let $(\operatorname{grad} f)(q) = s_f R^{\alpha_i} b_i$, where $s_f = \|(\operatorname{grad} f)(q)\|$, and we assumed that both s_x and s_f are not zero. Now, the power vector of x w.r.t the basis b_i is given by $y_i = R^{4\theta_i} b_i$, and similarly $(\operatorname{grad} f)(q)_{i,p} = R^{4\alpha_i} b_i$. The inner product is therefore:

$$\langle y_i, (\operatorname{grad} f)(q)_{i,p} \rangle = b_i^T R^{-4\theta_i} R^{4\alpha_i} b_i = \cos(4(\alpha_i - \theta_i)),$$

since the bases b_i are unit-length. Now simply note that the difference of angles is independent of the basis which gives us the result. □

C.2 A Practical Optimization Approach

In what follows, we re-formulate our minimization problem (4.7) as a standard quadratic programming optimization problem. Our analysis shows that the involved Hessian is composed of a sparse term and a dense component, which is the product of a matrix and its transpose. Thus, we can facilitate MATLAB's `quadprog` with the `trust-region-reflective` method, allowing to solve a large and dense problem as long as its Hessian is *structured.* As the derivation for the case of a pair of shapes closely follows the single shape scenario, we omit the discussion of re-formulating the problem given in Eq. (4.11).

Recalling the smoothness and alignment terms, Eqs. (4.1) and (4.2), respectively, we observe that their associated Hessian matrices, H_s and H_l are extremely sparse. We denote

$$\begin{aligned} H_s &= \mathrm{grad}_p^T \, G_{\mathcal{E}} \, \mathrm{grad}_p \ , \\ H_l &= S^T \, G_{\mathcal{F}} \, S \ , \end{aligned}$$

where H_s has a sparsity structure of a Laplacian matrix (one-ring of faces) and H_l is a diagonal matrix. In addition, the alignment component includes a linear term which we denote by $f = -H_l \cdot w$ and a quadratic part in w which does not affect the optimization.

For the consistency component given in Eq. (4.6), we distinguish between two cases. In the first case, we use a reduced functional basis, i.e., $k < 300$, and we denote $G_{c,i} = C\, D(f_i) - D\,(C \cdot f_i)$, with $G_{c,i}$ being a *constant* matrix of size $k \times 2|\mathcal{F}|$, since C and f_i are fixed throughout the optimization. The consistency condition has the following Hessian:

$$H_c = \sum_{i=1}^{m} G_{c,i}^T \, G_{c,i} \ .$$

Unfortunately, direct computation of H_c results in a large and dense matrix and thus, in practice, we only perform manipulations of the form $G_{c,i}^T \cdot (G_{c,i} \cdot y)$. The second case, when $k = |\mathcal{V}|$, is much simpler as H_c is sparse and problem (4.7) can be solved directly in this scenario. Overall, we achieve the following Hessian,

$$H = (1 - \alpha_l)[(1 - \alpha_c)\, H_s + \alpha_c \, H_c] + \alpha_l \, H_l \ .$$

Finally, using the above notation, our problem (4.7) can be written as

$$\arg\min_y \frac{1}{2} y^T H y + f^T \, y \ . \tag{C.1}$$

Appendix D

Appendix of Chapter 5

D.1 Directional derivatives of the discrete energy (5.6).

We take variations of E with respect to the velocities v_j, and idenitfy the coefficient of v_j with the partial derivative $\frac{\partial}{\partial v_j}E$:

$$\begin{aligned}
\frac{\partial}{\partial t}E(v_j + t\delta v_j)\Big|_{t=0} &= \frac{\delta\tau}{2}\,\delta v_j^T G_{\mathcal{F}}\, D_\alpha\, v_j + \frac{\delta\tau}{2}\, v_j^T G_{\mathcal{F}}\, D_\alpha\, \delta v_j \\
&+ \frac{1}{2\,\sigma^2}\left(\frac{\partial f_t}{\partial v_j}\delta v_j\right)^T G_{\mathcal{V}}\, C_\beta\, \delta g \\
&+ \frac{1}{2\,\sigma^2}\,\delta g G_{\mathcal{V}}\, C_\beta \left(\frac{\partial f_t}{\partial v_j}\delta v_j\right) \\
&= \delta\tau\,\delta v_j^T G_{\mathcal{F}}\, D_\alpha\, v_j + \frac{1}{\sigma^2}\,\delta v_j^T \left(\frac{\partial}{\partial v_j} f_t\right)^T G_{\mathcal{V}}\, C_\beta\, \delta g \\
&\equiv \delta v_j^T \left(\frac{\partial}{\partial v_j}E\right)\ .
\end{aligned}$$

Notice that the above holds in our setup since D_α and C_β are self-adjoint operators with respect to the inner products defined by $G_{\mathcal{F}}$ and $G_{\mathcal{V}}$ respectively.

D.2 Directional derivatives of $f \circ \phi_v^{-\tau}$ (linear advection).

The key insight for deriving the gradient for $f \circ \phi_v^{-\tau}$ is to employ the dual operator $\overline{D}_f$ in order to extract the particular v_j. As in Appendix D.1, we have:

$$
\begin{aligned}
\frac{\partial}{\partial t} f \circ \phi^{-\tau}_{v+t\delta v}\Big|_{t=0} &= \frac{\partial}{\partial t} \prod_{j=1}^{N} \left(\mathrm{id} - \delta\tau\, D_{v_j + t\delta v_j}\right) f \Big|_{t=0} \\
&= \frac{\partial}{\partial t} \prod_{j=1}^{N} \left(\mathrm{id} - \delta\tau\, D_{v_j} - t\,\delta\tau\, D_{\delta v_j}\right) f \Big|_{t=0} \\
&= \sum_{j=1}^{N} \left(\prod_{i=j+1}^{N} (\mathrm{id} - \delta\tau\, D_{v_i}) \right) (-\delta\tau\, D_{\delta v_j}) \left(\prod_{i=1}^{j-1} (\mathrm{id} - \delta\tau\, D_{v_i}) \right) f \\
&= -\delta\tau \sum_{j=1}^{N} \left(\prod_{i=j+1}^{N} (\mathrm{id} - \delta\tau\, D_{v_i}) \right) D_{\delta v_j}\, f_{(j-1)\delta\tau} \\
&= -\delta\tau \sum_{j=1}^{N} \left(\prod_{i=j+1}^{N} (\mathrm{id} - \delta\tau\, D_{v_i}) \right) \overline{D}_{f_{(j-1)\delta\tau}} \delta v_j \\
&\equiv \sum_{j=1}^{N} \frac{\partial f \circ \phi^{-\tau}_{v}}{\partial v_j} \delta v_j
\end{aligned}
$$

D.3 Directional derivative of $f \circ \varphi^{-\tau}_{v}$ (non-linear advection).

In what follows, we describe our approximation to the derivative of $f \circ \varphi^{-\tau}_{v}$. Notice that for finite matrix groups, a direct differentiation is available (see e.g., [Hal15]). However, the resulting expression is not computationally tractable as it contains an exponential of a $9|\mathcal{F}|^2 \times 9|\mathcal{F}|^2$ matrix. We, on the other hand, facilitate the discrete relation between vector fields and matrices and thus obtain an efficient yet approximate expression for the derivative.

$$
\begin{aligned}
&\frac{\partial}{\partial t}\left(\frac{1}{2\,\sigma^2}\left\|\exp(-\tau\, D_{v+t\,\delta v})f-g\right\|_\beta^2\right)\Big|_{t=0}= \\
&\frac{1}{\sigma^2}\left\langle \exp(-\tau\, D_{v+t\,\delta v})f-g,\left[\frac{\partial}{\partial t}\left(\exp(-\tau\, D_{v+t\,\delta v})f-g\right)\right]\Big|_{t=0}\right\rangle_\beta = \\
&\frac{1}{\sigma^2}\left\langle \exp(-\tau\, D_{v})f-g,\left[\frac{\partial}{\partial t}\exp(-\tau\, D_{v+t\,\delta v})\right]\Big|_{t=0} f\right\rangle_\beta =^{(1)} \\
&\frac{1}{\sigma^2}\left\langle \delta g,\exp(-\tau\, D_v)\left[\int_0^1 \exp(-\operatorname{ad}_{-s\,\tau\, D_v})D_{-\tau\,\delta v}\mathrm{d}s\right] f\right\rangle_\beta =^{(2)} \\
&\frac{-\tau}{\sigma^2}\left\langle \delta g,\exp(-\tau\, D_v)\left[\int_0^1 D_{\exp(s\,\tau\ \operatorname{ad}_v)\delta v}\mathrm{d}s\right] f\right\rangle_\beta = \\
&\frac{-\tau}{\sigma^2}\left\langle \delta g,\exp(-\tau\, D_v)\int_0^1 \overline{D}_f\exp(s\,\tau\ \operatorname{ad}_v)\delta v\mathrm{d}s\right\rangle_\beta = \\
&\frac{-\tau}{\sigma^2}\left\langle \delta g,\exp(-\tau\, D_v)\overline{D}_f\int_0^1 \exp(s\,\tau\ \operatorname{ad}_v)\mathrm{d}s\delta v\right\rangle_\beta =^{(3)} \\
&\frac{-\tau}{(k+1)\sigma^2}\left\langle \delta g,\exp(-\tau\, D_v)\overline{D}_f\sum_{s=0}^{k}\exp\left(\frac{s\,\tau}{k}\operatorname{ad}_v\right)\delta v\right\rangle_\beta \quad .
\end{aligned}
$$

The proof for (1) is given in [Hal15] and (3) is a simple averaging rule for approximating the continuous integral with a finite sum. The pass in (2) states $\exp(\operatorname{ad}_{D_v})D_u = D_{\exp(\operatorname{ad}_v)u}$, i.e., applying this operation to the matrices D_v and D_u is the same as acting the on vector fields v and u. In the discrete setting this relation does not hold, thus pass (2) can be considered as an approximation of the required computation.

D.4 Construction of the operator ad_v.

To derive Eq. (5.11), we employ the following observations. As the current D_v operates on vertex-based functions, yet ad_v is expected to act on vector fields, we define $\mathcal{D}_v = [v]_\bullet^T \operatorname{grad} I_{\mathcal{V}}^{\mathcal{F}}$, an operator on face-based functions. Moreover, a vector field $v = (v_x, v_y, v_z)$ can be reconstructed by applying its directional derivative operator on the coordinate functions of the surface. Namely,

$$
\mathcal{D}_v(x) = v_x \ , \ \mathcal{D}_v(y) = v_y \ , \ \mathcal{D}_v(z) = v_z \ ,
$$

where any point $p \in M$ is given by $(x_p, y_p, z_p) \in \mathbb{R}^3$. Thus, using the above observations we obtain,

$$
\begin{aligned}
\mathcal{D}_{[v,u]}(x) &= \mathcal{D}_v\mathcal{D}_u(x) - \mathcal{D}_u\mathcal{D}_v(x) \\
&= \mathcal{D}_v(u_x) - \mathcal{D}_u(v_x) \\
&= \mathcal{D}_v(u_x) - \overline{\mathcal{D}}_{v_x}(u) \ ,
\end{aligned}
$$

where $\overline{\mathcal{D}}_f = [\operatorname{grad} I_{\mathcal{V}}^{\mathcal{F}} f]_{\bullet}^{T}$. Finally, applying the above argument to the coordinate functions y and z yields,

$$\begin{aligned}
\operatorname{ad}_v(u) &= [v, u] \\
&= \big(\mathcal{D}_{[v,u]}(x), \mathcal{D}_{[v,u]}(y), \mathcal{D}_{[v,u]}(z)\big) \\
&= \big(\mathcal{D}_v(u_x) - \overline{\mathcal{D}}_{v_x}(u), \mathcal{D}_v(u_y) - \overline{\mathcal{D}}_{v_y}(u), \mathcal{D}_v(u_z) - \overline{\mathcal{D}}_{v_z}(u)\big) \ ,
\end{aligned}$$

where Eq. (5.11) is the matrix form of the above computation.

Appendix E

Appendix of Chapter 7

E.1 Mass preservation

To prove that mass is strictly preserved we recall the first order necessary conditions in the context of the Lagrangian.

$$\begin{gathered}
\tau \mathbf{G}_{\mathcal{F}} \mathbf{M}(\mathbf{u}^k)^{-1} \mathbf{v} - \tau \left(\overline{\mathbf{D}}(\mathbf{u}^k) + [\mathbf{u}^k] \operatorname{div} \right)^T \mathbf{G}_{\mathcal{V}} \mathbf{p} = 0 \\
\mathbf{G}_{\mathcal{V}} \left(\mathbf{a} + \epsilon \mathbf{B} \mathbf{u} \right) + \epsilon \mathbf{L} \mathbf{u} - \mathbf{G}_{\mathcal{V}} \mathbf{p} = 0 \\
\mathbf{G}_{\mathcal{V}} \left(\mathbf{u} - \mathbf{u}^k + \tau (\mathbf{D}(\mathbf{v}) + [\operatorname{div} \mathbf{v}]) \mathbf{u}^k \right) = 0
\end{gathered}$$

It is interesting to note that, using the definition of $\mathbf{L}$ and a discrete integration by parts, the second equation is equivalent to $\mathbf{p} = \mathbf{a} + \epsilon \mathbf{B} \mathbf{u} - \epsilon \operatorname{div} \operatorname{grad} \mathbf{u}$ in correspondence to the corresponding continuous equation $p = a + \epsilon b u - \epsilon \Delta_\Gamma u$.

At first, we rewrite the first equation and get

$$\begin{aligned}
\mathbf{v} =& \mathbf{M}(\mathbf{u}^k) \mathbf{G}_{\mathcal{F}}^{-1} \left(\overline{\mathbf{D}}(\mathbf{u}^k) + [\mathbf{u}^k] \operatorname{div} \right)^T \mathbf{G}_{\mathcal{V}} \mathbf{p} \\
=& \mathbf{M}(\mathbf{u}^k) \mathbf{G}_{\mathcal{F}}^{-1} \left([\operatorname{grad} \mathbf{u}^k]_\bullet (\mathbf{I}_{\mathcal{V}}^{\mathcal{F}})^T \mathbf{G}_{\mathcal{V}} \mathbf{p} + \operatorname{div}^T [\mathbf{u}^k] \mathbf{G}_{\mathcal{V}} \mathbf{p} \right) \\
=& \mathbf{M}(\mathbf{u}^k) \mathbf{G}_{\mathcal{F}}^{-1} \left([(\mathbf{I}_{\mathcal{V}}^{\mathcal{F}})^T \mathbf{G}_{\mathcal{V}} \mathbf{p}] \operatorname{grad} \mathbf{u}^k + \operatorname{div}^T \mathbf{G}_{\mathcal{V}} [\mathbf{u}^k] \mathbf{p} \right) \\
=& \mathbf{M}(\mathbf{u}^k) \mathbf{G}_{\mathcal{F}}^{-1} \left([\mathbf{G}_{\mathcal{F}} \mathbf{I}_{\mathcal{F}}^{\mathcal{V}} \mathbf{p}] \operatorname{grad} \mathbf{u}^k - \mathbf{G}_{\mathcal{F}} \operatorname{grad} [\mathbf{u}^k] \mathbf{p} \right) \\
=& \mathbf{M}(\mathbf{u}^k) \left([\mathbf{p}_{\mathcal{F}}] \operatorname{grad} \mathbf{u}^k - \operatorname{grad} [\mathbf{u}^k] \mathbf{p} \right)
\end{aligned}$$

using the facts that $\mathbf{G}_{\mathcal{V}}$ and $[\mathbf{u}^k]$ commute as diagonal matrices, that $[\mathbf{v}]_\bullet \mathbf{u}_{\mathcal{F}} = [\mathbf{u}_{\mathcal{F}}] \mathbf{v}$ for any $\mathbf{u}_{\mathcal{F}}$ (discrete scalar on faces) and $\mathbf{v}$ (discrete vector), and that the interpolation matrices are defined so that $\mathbf{I}_{\mathcal{F}}^{\mathcal{V}} = \mathbf{G}_{\mathcal{F}}^{-1} (\mathbf{I}_{\mathcal{V}}^{\mathcal{F}})^T \mathbf{G}_{\mathcal{V}}$. Again, it is interesting to note that the equation above is an approximation of the corresponding continuous one:

$$v = M(u^k) \left(p \nabla_\Gamma u^k - \nabla_\Gamma (u^k p) \right) = -u^k M(u^k) \nabla_\Gamma p$$

Now, we consider the discrete $m(\mathbf{u}) = \mathbf{1}_{\mathcal{V}}^T \mathbf{G}_{\mathcal{V}} \mathbf{u}$ with $\mathbf{1}_{\mathcal{V}}$ a vector of ones of length $|\mathcal{V}|$. Indeed, multiplying the third equation with $\mathbf{1}_{\mathcal{V}}^T$, using the duality of $\mathbf{D}$ and $\overline{\mathbf{D}}$, and taking into account that the interpolation matrix $\mathbf{I}_{\mathcal{F}}^{\mathcal{V}}$ is defined so that $\mathbf{I}_{\mathcal{F}}^{\mathcal{V}} \mathbf{1}_{\mathcal{V}} = \mathbf{1}_{\mathcal{F}}$ we obtain

$$\begin{aligned}
m(\mathbf{u}^{k+1}) - m(\mathbf{u}^k) &= -\tau\, \mathbf{1}_{\mathcal{V}}^T \mathbf{G}_{\mathcal{V}} \left(\mathbf{D}(\mathbf{v}) + [\operatorname{div} \mathbf{v}]\right) \mathbf{u}^k \\
&= -\tau\, \mathbf{1}_{\mathcal{V}}^T \mathbf{G}_{\mathcal{V}} \left(\overline{\mathbf{D}}(\mathbf{u}^k) + [\mathbf{u}^k] \operatorname{div}\right) \mathbf{v} \\
&= -\tau\, \mathbf{v}^T \left(\overline{\mathbf{D}}(\mathbf{u}^k) + [\mathbf{u}^k] \operatorname{div}\right)^T \mathbf{G}_{\mathcal{V}} \mathbf{1}_{\mathcal{V}} \\
&= -\tau\, \mathbf{v}^T \mathbf{G}_{\mathcal{F}} \left([\mathbf{I}_{\mathcal{F}}^{\mathcal{V}} \mathbf{1}_{\mathcal{V}}] \operatorname{grad} \mathbf{u}^k - \operatorname{grad}[\mathbf{u}^k] \mathbf{1}_{\mathcal{V}}\right) \\
&= -\tau\, \mathbf{v}^T \mathbf{G}_{\mathcal{F}} \left(\operatorname{grad} \mathbf{u}^k - \operatorname{grad} \mathbf{u}^k\right) = 0\,.
\end{aligned}$$

The key step is applying the previous calculation with $\mathbf{p} = \mathbf{1}_{\mathcal{V}}$ and so *the discrete conservation of mass is equivalent to the fact that a constant discrete pressure gives zero discrete velocity.*

Bibliography

[ABCCO13] Omri Azencot, Mirela Ben-Chen, Frédéric Chazal, and Maks Ovsjanikov. An operator approach to tangent vector field processing. In *Computer Graphics Forum*, volume 32, pages 73–82. Wiley Online Library, 2013.

[AFW06] Douglas N Arnold, Richard S Falk, and Ragnar Winther. Finite element exterior calculus, homological techniques, and applications. *Acta numerica*, 15(1):1–155, 2006.

[AJ05] John D Anderson Jr. Ludwig prandtl's boundary layer. *Physics Today*, 58(12):42–48, 2005.

[AK99] Vladimir I Arnold and Boris A Khesin. *Topological methods in hydrodynamics*, volume 125. Springer Science & Business Media, 1999.

[AMH11] Awad H Al-Mohy and Nicholas J Higham. Computing the action of the matrix exponential, with an application to exponential integrators. *SIAM journal on scientific computing*, 33(2):488–511, 2011.

[AMT$^+$12] Stefan Auer, Colin B Macdonald, Marc Treib, Jens Schneider, and Rüdiger Westermann. Real-time fluid effects on surfaces using the closest point method. In *Computer Graphics Forum*, volume 31, pages 1909–1923. Wiley Online Library, 2012.

[AOCBC15] Omri Azencot, Maks Ovsjanikov, Frédéric Chazal, and Mirela Ben-Chen. Discrete derivatives of vector fields on surfaces–an operator approach. *ACM Transactions on Graphics (TOG)*, 34(3):29, 2015.

[APL15] Noam Aigerman, Roi Poranne, and Yaron Lipman. Seamless surface mappings. *ACM Transactions on Graphics (TOG)*, 34(4):72, 2015.

[ASK$^+$05] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. Scape: shape completion and animation of people. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 408–416. ACM, 2005.

[AVW$^+$15] Omri Azencot, Orestis Vantzos, Max Wardetzky, Martin Rumpf, and Mirela Ben-Chen. Functional thin films on surfaces. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 137–146. ACM, 2015.

[AWO$^+$14] Omri Azencot, Steffen Weißmann, Maks Ovsjanikov, Max Wardetzky, and Mirela Ben-Chen. Functional fluids on surfaces. In *Computer Graphics Forum*, volume 33, pages 237–246. Wiley Online Library, 2014.

[Bae12] Seung Ki Baek. Vortex interaction on curved surfaces. *Physical Review E*, 86(5):056603, 2012.

[Bat00] George Keith Batchelor. *An introduction to fluid dynamics*. Cambridge university press, 2000.

[BB00] Jean-David Benamou and Yann Brenier. A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem. *Numerische Mathematik*, 84(3):375–393, 2000.

[BCBSG10] Mirela Ben-Chen, Adrian Butscher, Justin Solomon, and Leonidas Guibas. On discrete killing vector fields and patterns on surfaces. In *CGF*, volume 29, pages 1701–1711, 2010.

[BCE$^+$13] David Bommes, Marcel Campen, Hans-Christian Ebke, Pierre Alliez, and Leif Kobbelt. Integer-grid maps for reliable quad meshing. *ACM Transactions on Graphics (TOG)*, 32(4):98, 2013.

[BKP$^+$10] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. CRC press, 2010.

[BLP$^+$13] David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. Quad-mesh generation and processing: A survey. In *Computer Graphics Forum*, volume 32, pages 51–76. Wiley Online Library, 2013.

[BMTY05] M Faisal Beg, Michael I Miller, Alain Trouvé, and Laurent Younes. Computing large deformation metric mappings via geodesic flows of diffeomorphisms. *International journal of computer vision*, 61(2):139–157, 2005.

[Bri15] Robert Bridson. *Fluid simulation for computer graphics*. CRC Press, 2015.

[BS99] Aleksandr I Bobenko and Yu B Suris. Discrete time lagrangian mechanics on lie groups, with an application to the lagrange top. *Communications in mathematical physics*, 204(1):147–188, 1999.

[BUAG12] Christopher Batty, Andres Uribe, Basile Audoly, and Eitan Grinspun. Discrete viscous sheets. *ACM Trans. Graph.*, 31(4):113, 2012.

[BUM$^+$12] RJ Braun, R Usha, GB McFadden, TA Driscoll, LP Cook, and PE King-Smith. Thin film dynamics on a prolate spheroid with application to the cornea. *J. of Eng. Math.*, 73(1):121–138, 2012.

[BVDPPH11] Nicolas Bonneel, Michiel Van De Panne, Sylvain Paris, and Wolfgang Heidrich. Displacement interpolation using lagrangian mass transport. In *ACM Transactions on Graphics (TOG)*, volume 30, page 158. ACM, 2011.

[BZK09] David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. In *ACM Transactions On Graphics (TOG)*, volume 28, page 77. ACM, 2009.

[Car94] Manfredo P Carmo. Differential geometry of surfaces. *Differential Forms and Applications*, pages 77–98, 1994.

[CBK12] Marcel Campen, David Bommes, and Leif Kobbelt. Dual loops meshing: quality quad layouts on manifolds. *ACM Transactions on Graphics (TOG)*, 31(4):110, 2012.

[CDS10] Keenan Crane, Mathieu Desbrun, and Peter Schröder. Trivial connections on discrete surfaces. In *CGF*, volume 29, pages 1525–1533, 2010.

[Cho94] Alexandre J Chorin. *Vorticity and turbulence*, volume 103. Springer Science & Business Media, 1994.

[CM05] Darren Crowdy and Jonathan Marshall. Analytical solutions for rotating vortex arrays involving multiple vortex patches. *Journal of Fluid Mechanics*, 523:307–337, 2005.

[CM09] RV Craster and OK Matar. Dynamics and stability of thin liquid films. *Reviews of modern physics*, 81(3):1131, 2009.

[CMM90] Alexandre Joel Chorin, Jerrold E Marsden, and Jerrold E Marsden. *A mathematical introduction to fluid mechanics*, volume 3. Springer, 1990.

[CMVHIT02] Mark Carlson, Peter J Mucha, R Brooks Van Horn III, and Greg Turk. Melting and flowing. In *Proc. of SCA 2002*, pages 167–174, 2002.

[COC15] Etienne Corman, Maks Ovsjanikov, and Antonin Chambolle. Continuous matching via vector field flow. In *Computer Graphics Forum*, volume 34, pages 129–139. Wiley Online Library, 2015.

[CWSO13] Pascal Clausen, Martin Wicke, Jonathan R Shewchuk, and James F O'brien. Simulating liquids and solid-liquid interactions with lagrangian meshes. *ACM Trans. Graph.*, 32(2):17, 2013.

[CWW13] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics (TOG)*, 32(5):152, 2013.

[Dav15] Peter Davidson. *Turbulence: an introduction for scientists and engineers*. Oxford University Press, 2015.

[dCV92] Manfredo Perdigao do Carmo Valero. *Riemannian geometry*. 1992.

[DE13] Gerhard Dziuk and Charles M Elliott. Finite element methods for surface pdes. *Acta Numerica*, 22:289–396, 2013.

[DFN92] BA Dubrovin, A Fomenko, and S Novikov. Modern geometry-methods and applications, part i: The geometry of surfaces, transformation groups, and fields, vol. 93. *Graduate texts in mathematics*, 1992.

[DP80] John R Dormand and Peter J Prince. A family of embedded Runge-Kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

[DVPSH14] Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. Designing n-polyvector fields with complex polynomials. In *Computer Graphics Forum*, volume 33, pages 1–11. Wiley Online Library, 2014.

[DWLF12] Tyler De Witt, Christian Lessig, and Eugene Fiume. Fluid simulation using Laplacian eigenfunctions. *ACM Transactions on Graphics (TOG)*, 31(1):10, 2012.

[EBCK13] Hans-Christian Ebke, David Bommes, Marcel Campen, and Leif Kobbelt. QEx: robust quad mesh extraction. *ACM Transactions on Graphics (TOG)*, 32(6):168, 2013.

[ETK$^+$07] Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. Stable, circulation-preserving, simplicial fluids. *ACM Transactions on Graphics (TOG)*, 26(1):4, 2007.

[Fed69] Herbert Federer. *Geometric measure theory*, volume 1996. Springer New York, 1969.

[Fra11] Theodore Frankel. *The geometry of physics: an introduction.* Cambridge University Press, 2011.

[FSDH07] Matthew Fisher, Peter Schröder, Mathieu Desbrun, and Hugues Hoppe. Design of tangent vector fields. *ACM Transactions on Graphics (TOG)*, 26(3):56, 2007.

[FSJ01] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 2001.

[FZKH05] Zhe Fan, Ye Zhao, Arie Kaufman, and Ying He. Adapted unstructured lbm for flow simulation on curved surfaces. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 245–254. ACM, 2005.

[GA06] Ennio Giorgi and Luigi Ambrosio. New problems on minimizing movements. In *Ennio De Giorgi selected papers*, pages 699–714. Springer, 2006.

[GBS06] John B Greer, Andrea L Bertozzi, and Guillermo Sapiro. Fourth order partial differential equations on general geometries. *J. of Comp. Physics*, 216(1):216–246, 2006.

[GG06] Timothy D Gatzke and Cindy M Grimm. Estimating curvature on triangular meshes. *Int. J. of shape modeling*, 12(01):1–28, 2006.

[GO03] Lorenzo Giacomelli and Felix Otto. Rigorous lubrication approximation. *Interfaces and Free boundaries*, 5(4):483–530, 2003.

[GR00] Günther Grün and Martin Rumpf. Nonnegativity preserving convergent schemes for the thin film equation. *Numerische Mathematik*, 87(1):113–152, 2000.

[GR01] Günther Grün and Martin Rumpf. Simulation of singularities and instabilities arising in thin film flow. *European Journal of Applied Mathematics*, 12(03):293–320, 2001.

[Gri00] RW Griffiths. The dynamics of lava flows. *Ann. Rev. of Fluid Mechanics*, 32(1):477–518, 2000.

[Hal15] Brian C Hall. *Lie groups, Lie algebras, and representations: an elementary introduction*, volume 222. Springer, 2015.

[HAW$^+$09] Kyle Hegeman, Michael Ashikhmin, Hongyu Wang, Hong Qin, Xianfeng Gu, et al. Gpu-based conformal flow on surfaces. *Communications in Information & Systems*, 9(2):197–212, 2009.

[Hir03] Anil N Hirani. *Discrete exterior calculus*. PhD thesis, California Institute of Technology, 2003.

[HO10] Marlis Hochbruck and Alexander Ostermann. Exponential integrators. *Acta Numerica*, 19:209–286, 2010.

[Jon24] John Edward Jones. On the determination of molecular fields. ii. from the equation of state of a gas. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 106, pages 463–477. The Royal Society, 1924.

[JPS$^+$13] A Jacobson, D Panozzo, C Schüller, O Diamanti, Q Zhou, N Pietroni, et al. libigl: A simple c++ geometry processing library, 2013.

[KCPS13] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Globally optimal direction fields. *ACM Transactions on Graphics (TOG)*, 32(4):59, 2013.

[KLF11] Vladimir G Kim, Yaron Lipman, and Thomas Funkhouser. Blended intrinsic maps. In *ACM Transactions on Graphics (TOG)*, volume 30, page 79. ACM, 2011.

[KNP07] Felix Kälberer, Matthias Nieser, and Konrad Polthier. Quadcover-surface parameterization using branched coverings. In *Computer Graphics Forum*, volume 26, pages 375–384. Wiley Online Library, 2007.

[Kol93] Ivan Kolar. *Natural operations in differential geometry*. Springer-Verlag, Berlin New York, 1993.

[LB08] Julien Lefèvre and Sylvain Baillet. Optical flow and advection on 2-Riemannian manifolds: a common framework. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(6):1081–1092, 2008.

[LJX$^+$10] Yu-Kun Lai, Miao Jin, Xuexiang Xie, Ying He, Jonathan Palacios, Eugene Zhang, Shi-Min Hu, and Xianfeng Gu. Metric-driven rosy field design and remeshing. *IEEE Transactions on Visualization and Computer Graphics*, 16(1):95–108, 2010.

[LRB$^+$16] O. Litany, E. Rodolà, A. M. Bronstein, M. M. Bronstein, and D. Cremers. Non-rigid puzzles. *Computer Graphics Forum (Proc. SGP)*, 35(5):135–143, 2016.

[LWC05] Lok Ming Lui, Yalin Wang, and Tony F Chan. Solving pdes on manifolds with global conformal parametriazation. In *Variational, Geometric, and Level Set Methods in Computer Vision*, pages 307–319. Springer, 2005.

[MB01] Andrew J Majda and Andrea L Bertozzi. *Vorticity and incompressible flow*, volume 27. Cambridge University Press, 2001.

[MC04] Tim G Myers and Jean PF Charpin. A mathematical model for atmospheric ice accretion and water flow on a cold surface. *Int. J. of Heat and Mass Transfer*, 47(25):5483–5500, 2004.

[McK07] Alexander McKenzie. *HOLA: a High-Order Lie Advection of discrete differential forms, with applications in fluid dynamics.* PhD thesis, California Institute of Technology, 2007.

[MCP$^+$09] Patrick Mullen, Keenan Crane, Dmitry Pavlov, Yiying Tong, and Mathieu Desbrun. Energy-preserving integrators for fluid animation. In *ACM Transactions on Graphics (TOG)*, volume 28, page 38. ACM, 2009.

[MDS$^+$02] Mark Meyer, Mathieu Desbrun, Peter Schröder, Alan H Barr, et al. Discrete differential-geometry operators for triangulated 2-manifolds. *Visualization and mathematics*, 3(2):52–58, 2002.

[MH16] Min Meng and Ying He. Consistent quadrangulation for shape collections via feature line co-extraction. *Computer-Aided Design*, 70:78 – 88, 2016. {SPM} 2015.

[MMP$^+$11] P Mullen, A McKenzie, D Pavlov, L Durant, Y Tong, E Kanso, JE Marsden, and M Desbrun. Discrete lie advection of differential forms. *Foundations of Computational Mathematics*, 11(2):131–149, 2011.

[Mor01] S Morita. *Geometry of differential forms.* American Mathematical Society, Providence, R.I, 2001.

[MPP$^+$13] Giorgio Marcias, Nico Pietroni, Daniele Panozzo, Enrico Puppo, and Olga Sorkine-Hornung. Animation-aware quadrangulation. In *Computer Graphics Forum*, volume 32, pages 167–175. Wiley Online Library, 2013.

[MPS99] Jerrold E Marsden, Sergey Pekarsky, and Steve Shkoller. Discrete euler-poincaré and lie-poisson equations. *Nonlinearity*, 12(6):1647, 1999.

[MPZ14] Ashish Myles, Nico Pietroni, and Denis Zorin. Robust field-aligned global parametrization. *ACM Transactions on Graphics (TOG)*, 33(4):135, 2014.

[MTP$^+$15] Giorgio Marcias, Kenshi Takayama, Nico Pietroni, Daniele Panozzo, Olga Sorkine-Hornung, Enrico Puppo, and Paolo Cignoni. Data-driven interactive quadrangulation. *ACM Transactions on Graphics (TOG)*, 34(4):65, 2015.

[MV91] Jürgen Moser and Alexander P Veselov. Discrete versions of some classical integrable systems and factorization of matrix polynomials. *Communications in Mathematical Physics*, 139(2):217–243, 1991.

[MVL03] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review*, 45(1):3–49, 2003.

[MWC92] Jonathan Miller, Peter B. Weichman, and M. C. Cross. Statistical mechanics, Euler's equation, and Jupiter's Red Spot. *Phys. Rev. A*, 45:2328–2359, Feb 1992.

[New01] Paul K Newton. *The N-vortex problem: analytical techniques*, volume 145. Springer, 2001.

[NMZ07] Patrick Neill, Ron Metoyer, and Eugene Zhang. Fluid flow on interacting deformable surfaces. In *ACM SIGGRAPH 2007 Posters*, SIGGRAPH '07. ACM, 2007.

[NVW12] I Nitschke, A Voigt, and J Wensch. A finite element approach to incompressible two-phase flow on manifolds. *Journal of Fluid Mechanics*, 708:418–438, 2012.

[OBCCG13] Maks Ovsjanikov, Mirela Ben-Chen, Frederic Chazal, and Leonidas Guibas. Analysis and visualization of maps between shapes. In *Computer Graphics Forum*, volume 32, pages 135–145. Wiley Online Library, 2013.

[OBCS$^+$12] Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. Functional maps: a flexible representation of maps between shapes. *ACM Transactions on Graphics (TOG)*, 31(4):30, 2012.

[OCB$^+$16] Maks Ovsjanikov, Etienne Corman, Michael Bronstein, Emanuele Rodolà, Mirela Ben-Chen, Leonidas Guibas, Frederic Chazal, and Alex Bronstein. Computing and processing correspondences with functional maps. In *SIGGRAPH ASIA 2016 Courses*, pages 9:1–9:60, 2016.

[ODB97] Alexander Oron, Stephen H Davis, and S George Bankoff. Long-scale evolution of thin liquid films. *Rev. of modern physics*, 69(3):931, 1997.

[Ott01] Felix Otto. The geometry of dissipative evolution equations: the porous medium equation. *Comm. in Partial Differential Equations*, 26(1-2):101–174, 2001.

[PBB$^+$13] J. Pokrass, A. M. Bronstein, M. M. Bronstein, P. Sprechmann, and G. Sapiro. Sparse modeling of intrinsic correspondences. *Computer Graphics Forum*, 32(2pt4):459–468, 2013.

[Pet06] Peter Petersen. *Riemannian geometry.* Springer, New York, 2006.

[PHD$^+$10] Helmut Pottmann, Qixing Huang, Bailin Deng, Alexander Schiftner, Martin Kilian, Leonidas Guibas, and Johannes Wallner. Geodesic patterns. In *ACM Transactions on Graphics (TOG)*, volume 29, page 43. ACM, 2010.

[PK05] Sang Il Park and Myoung Jun Kim. Vortex fluid for gaseous phenomena. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 261–270. ACM, 2005.

[PK15] F Prada and M Kazhdan. Unconditionally stable shock filters for image and geometry processing. In *Computer Graphics Forum*, volume 34, pages 201–210. Wiley Online Library, 2015.

[PLPZ12] Daniele Panozzo, Yaron Lipman, Enrico Puppo, and Denis Zorin. Fields on symmetric surfaces. *ACM Transactions on Graphics (TOG)*, 31(4):111, 2012.

[PMT$^+$11] Dmitry Pavlov, Patrick Mullen, Yiying Tong, Eva Kanso, Jerrold E Marsden, and Mathieu Desbrun. Structure-preserving discretization of incompressible fluids. *Physica D: Nonlinear Phenomena*, 240(6):443–458, 2011.

[Pol05] Konrad Polthier. Computational aspects of discrete minimal surfaces. *Global theory of minimal surfaces*, 2:65–111, 2005.

[Poz11] Constantine Pozrikidis. *Introduction to theoretical and computational fluid dynamics*. Oxford University Press, 2011.

[PP03] Konrad Polthier and Eike Preuss. Identifying vector field singularities using a discrete hodge decomposition. *Visualization and Mathematics*, 3:113–134, 2003.

[PPTSH14] Daniele Panozzo, Enrico Puppo, Marco Tarini, and Olga Sorkine-Hornung. Frame fields: Anisotropic and non-orthogonal cross fields. *ACM Transactions on Graphics (TOG)*, 33(4):134, 2014.

[PS06] Konrad Polthier and Markus Schmies. *Straightest geodesics on polyhedral surfaces*. ACM, 2006.

[PTSG09] Tobias Pfaff, Nils Thuerey, Andrew Selle, and Markus Gross. Synthetic turbulence using artificial boundary layers. In *ACM Transactions on Graphics (TOG)*, volume 28, page 121. ACM, 2009.

[PZ07] Jonathan Palacios and Eugene Zhang. Rotational symmetry field design on surfaces. In *ACM Transactions on Graphics (TOG)*, volume 26, page 55. ACM, 2007.

[PZ11] Jonathan Palacios and Eugene Zhang. Interactive visualization of rotational symmetry fields on surfaces. *Visualization and Computer Graphics, IEEE Transactions on*, 17(7):947–955, 2011.

[Rey86] Osborne Reynolds. On the theory of lubrication and its application to Mr. Beauchamp tower's experiments, including an experimental determination of the viscosity of olive oil. *Proc. of the Royal Society of London*, 177:157–234, 1886.

[RLL$^+$06] Nicolas Ray, Wan Chiu Li, Bruno Lévy, Alla Sheffer, and Pierre Alliez. Periodic global parameterization. *ACM Transactions on Graphics (TOG)*, 25(4):1460–1485, 2006.

[RMC15] E. Rodolà, M. Moeller, and D. Cremers. Point-wise map recovery and refinement from functional correspondence. In *Proc. Vision, Modeling and Visualization (VMV)*, 2015.

[RRS02] R Valery Roy, Anthony John Roberts, and ME Simpson. A lubrication model of coating flows over a curved substrate in space. *J. of Fluid Mechanics*, 454:235–261, 2002.

[RS14] Nicolas Ray and Dmitry Sokolov. Robust polylines tracing for n-symmetry direction field on triangulated surfaces. *ACM Transactions on Graphics (TOG)*, 33(3):30, 2014.

[RTD$^+$10] Tobias Ritschel, Thorsten Thormählen, Carsten Dachsbacher, Jan Kautz, and Hans-Peter Seidel. Interactive on-surface signal deformation. In *ACM Transactions on Graphics (TOG)*, volume 29, page 36. ACM, 2010.

[Rus04] Szymon Rusinkiewicz. Estimating curvatures and their derivatives on triangle meshes. In *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on*, pages 486–493. IEEE, 2004.

[RV13] Martin Rumpf and Orestis Vantzos. Numerical gradient flow discretization of viscous thin films on curved geometries. *Math. Models and Methods in Applied Sciences*, 23(05):917–947, 2013.

[RVAL09] Nicolas Ray, Bruno Vallet, Laurent Alonso, and Bruno Levy. Geometry-aware direction field processing. *ACM Transactions on Graphics (TOG)*, 29(1):1, 2009.

[RVLL08] Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lévy. N-symmetry direction field design. *ACM Transactions on Graphics (TOG)*, 27(2):10, 2008.

[Saf92] Philip G Saffman. *Vortex dynamics*. Cambridge university press, 1992.

[Sch12] Mark Schmidt. minfunc: unconstrained differentiable multivariate optimization in matlab. *URL http://www. di. ens. fr/mschmidt/Software/minFunc. html*, 2012.

[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 Transactions on Graphics (TOG)*, 34(4):66, 2015.

[SDP13] Aristeidis Sotiras, Christos Davatzikos, and Nikos Paragios. Deformable medical image registration: A survey. *IEEE Transactions on Medical Imaging*, 32(7):1153–1190, 2013.

[SK98] Ashutosh Sharma and Rajesh Khanna. Pattern formation in unstable thin liquid films. *Phys. Rev. Lett.*, 81:3463–3466, Oct 1998.

[SNB$^+$12] Justin Solomon, Andy Nguyen, Adrian Butscher, Mirela Ben-Chen, and Leonidas Guibas. Soft maps between surfaces. In *Computer Graphics Forum*, volume 31, pages 1617–1626. Wiley Online Library, 2012.

[Spi99] Michael Spivak. *A comprehensive introduction to differential geometry. Vol. I.* Publish or Perish Inc., third edition, 1999.

[SRB14] Deqing Sun, Stefan Roth, and Michael J Black. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106(2):115–137, 2014.

[SRGB14] Justin Solomon, Raif Rustamov, Leonidas Guibas, and Adrian Butscher. Earth mover's distances on discrete surfaces. *ACM Transactions on Graphics (TOG)*, 33(4):67, 2014.

[SS13] Omer San and Anne E Staples. A coarse-grid projection method for accelerating incompressible flow computations. *Journal of Computational Physics*, 233:480–508, 2013.

[Sta99] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999.

[Sta03] Jos Stam. Flows on surfaces of arbitrary topology. In *ACM Transactions On Graphics (TOG)*, volume 22, pages 724–731. ACM, 2003.

[SY04] Lin Shi and Yizhou Yu. Inviscid and incompressible fluid simulation on triangle meshes. *Computer Animation and Virtual Worlds*, 15(3-4):173–181, 2004.

[SZ12] Andrzej Szymczak and Eugene Zhang. Robust morse decompositions of piecewise constant vector fields. *Visualization and Computer Graphics, IEEE Transactions on*, 18(6):938–951, 2012.

[Tay96] Michael Taylor. Partial differential equations, Vol. III, 1996.

[TH10] Daisuke Takagi and Herbert E Huppert. Flow and instability of thin films on a cylinder and sphere. *J. of Fluid Mech.*, 647:221, 2010.

[TIN$^+$11] Julien Tierny, Joel Daniels II, Luis G. Nonato, Valerio Pascucci, and Claudio T. Silva. Inspired quadrangulation. *Computer-Aided Design*, 43(11):1516 – 1526, 2011. Solid and Physical Modeling 2011.

[TLHD03] Yiying Tong, Santiago Lombeyda, Anil N Hirani, and Mathieu Desbrun. Discrete multiscale vector field decomposition. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 445–452, 2003.

[Van14] Orestis Vantzos. *Thin viscous films on curved geometries.* PhD thesis, Universitäts-und Landesbibliothek Bonn, 2014.

[VCD$^+$16] Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. Directional Field Synthesis, Design, and Processing. *Computer Graphics Forum*, 2016.

[Vil03] Cédric Villani. *Topics in optimal transportation.* Number 58. American Mathematical Soc., 2003.

[Vil08] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

[VL13] Joris Vankerschaver and Melvin Leok. A novel formulation of point vortex dynamics on the sphere: geometrical and numerical aspects. *Journal of Nonlinear Science*, pages 1–37, 2013.

[War07] Max Wardetzky. *Discrete Differential Operators on Polyhedral Surfaces-Convergence and Approximation.* PhD thesis, Freie Universität Berlin, 2007.

[WMFB11] C. Wojtan, M. Müller-Fischer, and T. Brochu. Liquid simulation with mesh-based surface tracking. In *ACM SIGGRAPH 2011 Courses*. ACM, 2011.

[WMT05] Huamin Wang, Peter J Mucha, and Greg Turk. Water drops on surfaces. *ACM Trans. Graph.*, 24(3):921–929, 2005.

[WMT07] Huamin Wang, Gavin Miller, and Greg Turk. Solving general shallow wave equations on surfaces. In *Proc. of SCA 2007*, pages 229–238, 2007.

[WP10] Steffen Weißmann and Ulrich Pinkall. Filament-based smoke with vortex shedding and variational reconnection. In *ACM Transactions on Graphics (TOG)*, volume 29, page 115. ACM, 2010.

[YCJL09] Chih-Yuan Yao, Hung-Kuo Chu, Tao Ju, and Tong-Yee Lee. Compatible quadrangulation by sketching. *Computer Animation and Virtual Worlds*, 20(2-3):101–109, 2009.

[ZB99] Liya Zhornitskaya and Andrea L Bertozzi. Positivity-preserving numerical schemes for lubrication-type equations. *SIAM J. on Numerical Analysis*, 37(2):523–555, 1999.

[ZLCW13] Jian Zhu, YouQuan Liu, YuanZhang Chang, and EnHua Wu. Animating turbulent water by vortex shedding in pic/flip. *Science China Information Sciences*, 56(3):1–11, 2013.

[ZMT06] Eugene Zhang, Konstantin Mischaikow, and Greg Turk. Vector field design on surfaces. *ACM Transactions on Graphics (ToG)*, 25(4):1294–1326, 2006.

[ZQC$^+$14] Bo Zhu, Ed Quigley, Matthew Cong, Justin Solomon, and Ronald Fedkiw. Codimensional surface tension flow on simplicial complexes. *ACM Trans. Graph.*, 33(4):111, 2014.

[ZWW$^+$12] Yizhong Zhang, Huamin Wang, Shuai Wang, Yiying Tong, and Kun Zhou. A deformable surface model for real-time water drop animation. *Vis. and Comp. Graph., IEEE Trans. on*, 18(8), 2012.

פי כן, הנגזרת הקו-וריאנטית חשובה בהרבה מקרים אחרים---עיצוב שדות וקטורים הוא אחת מהדוגמאות העיקריות שנדון בהן בפרק 3.

לבסוף, יתרון משמעותי נוסף לעבודה עם אופרטורים, הוא שבאמצעותם אנו מובלים באופן טבעי לייצוג חלש של הבעיה. לעומת הייצוג החזק שבו ערכים מחושבים בכל נקודה, הייצוג החלש בד"כ מכיל גרסא סכומה של הערך, באיזור קטן. הייצוג החלש עוזר כאשר נתון לנו מידע שהוא מקורב או אפילו רועש. למשל, בפרק 4, אנו נעסוק בבעיה של הפיכת זוג משטחים משולשיים נתונים, $\mathcal{M}_1$ ו- $\mathcal{M}_2$, למשטחים עם מרובעים תואמים. המשטחים יכולים להיות בעלי מספר שונה של קודקודים ולכן אנו מניחים שנתון בידינו מיפוי $\varphi\colon \mathcal{M}_1 \to \mathcal{M}_2$. על מנת לייצר משטחים עם מרובעים תואמים, אנו נחשב שני שדות וקטורים שהם סימטריים סיבובית, x_1 ו- x_2, על $\mathcal{M}_1$ ו- $\mathcal{M}_2$, בהתאמה, וכדי להחיל תאימות אנו זקוקים לדיפרנציאל המיפוי, $\mathrm{d}\varphi$. כלומר, תנאי התאימות הבסיסי שלנו הוא מהצורה $\mathrm{d}\varphi(x_1) = x_2$ עבור כל נקודה ב- $\mathcal{M}_2$. בפועל, המיפוי φ יכול להיות רועש, מה שיוביל לדיפרנציאל $\mathrm{d}\varphi$ רועש ולאלגוריתם שאינו יציב. במקום להשתמש בתנאי לעיל, אנו נתכנן אלגוריתם יעיל ויציב בפרק 4, ע"י שימוש בגרסא הפונקציונלית של תנאי התאימות הקודם:

$$C[\varphi] \cdot D(x_1) = D(x_2) \cdot C[\varphi]\,,$$

כאשר $C[\varphi]$ הוא האופרטור המשויך למיפוי φ. אנו מפנים את הקורא לדיון המקיף ולמידע נוסף המופיע הפרק הרלבנטי.

לסיכום, גישה מבוססת אופרטורים היא שימושית במספר מקרים, כפי המוצג בתזה הזו, ובאמצעות גישה זו ניתן לתכנן אלגוריתמים שהינם אפקטיביים ויציבים, בעודם יעילים לחישוב. תקוות המחבר היא שמתודולוגית האופרטורים תמשיך לצמוח בעתיד ויתווספו ביטויים ושימושים חדשים לספרות העיבוד ספרתי של גיאומטריה ובתחומים מדעיים אחרים.

הקבוצה הבאה והאחרונה של מד"ח אותן נחקור בעבודה זו הינן משוואות של נוזלים דקים, הקשורות לבעיה שהצגנו בתחילת הפרק. על משטחים עקומים, כאשר u מייצג את צפיפות המסה, המשוואות הללו נתונות ע"י:

$$\partial_t u + \operatorname{div}\big(-M(u)\cdot \operatorname{grad} p(u)\big) = 0\,,$$

$$M(u) = \frac{1}{3}u^3\mathrm{id} + \frac{\epsilon}{6}u^4(H\ \mathrm{id} - S)\,,$$

$$p = -H - \epsilon T u - \epsilon \Delta u\,,$$

כאשר H, T ו- S הם גדלים הקשורים לעקמומיות, $\epsilon \ll 1$ הוא היחס בין גובה לאורך של הנוזל, div הוא אופרטור הדיברגנץ שמודד את כמות השטף הקשורה ל- v, ו- Δ הוא אופרטור הלפלס—בלטרמי. במקרה זה, משוואת ההסעה היא אינה לינארית ומסדר רביעי, ולכן מקרה זה מאתגר במיוחד לפיתרון, כפי שנתאר בפרק 7. למרות זאת, מאחר ונוזלים דקים הם בעלי מבנה מתמטי של gradient flow, אנו מסוגלים לפתור משוואות אלו באופן יעיל ולסמלץ מספר תופעות מורכבות של נוזלים דקים על משטחים עקומים כלליים.

המשוואות שהוצגו בפרק זה, מלבד משוואת שימור התנע, חולקות מבנה משותף בו השינוי בזמן נתון ע"י השינוי הדיפרנציאלי במרחב. בנוסף, נזכיר שמתקיים

$$\operatorname{div}(uv) = \langle v, \operatorname{grad} u\rangle + u\cdot \operatorname{div}(v)\,,$$

ולכן, אנו מבחינים בין אופרטורים דיפרנציאליים כגון grad, div, curl ו- Δ, אותם ניתן לבנות ע"פ הכתוב בספרות לבין הנגזרת הכיוונית, $D(v) = \langle v, \operatorname{grad}\cdot\rangle$, שהיא עיקר העיסוק בפרק 2. נקודת המבט בה $D(v)$, הוא אופרטור המעביר פונקציות סקלריות לנגזרות שלהן, טומנת בחובה הרבה יתרונות. לדוגמא, בהינתן בסיס סופי, $D(v)$ היא למעשה מטריצה ריבועית שתכונותיה ניתנות לחקירה או אילוץ, כפי שמוצג בפרק 2.

בנוסף, כאשר פותרים משוואות דיפרנציאליות נשלטות הסעה, השימוש ב- $D(v)$ מאפשר לנו להימנע מהחישוב הרגיש נומרית של מסלולי הזרימה. כלומר, במקום לחשב ראשית את מסלולי הזרימה ולהסיע את $D(v)$ על גביהם, נוכל להשתמש ישירות ב- $D(v)$ בפותר זמן מפורש או סתום. לאור השימושיות של $D(v)$, הרחבנו רעיון זה למקרה של נגזרות וקטוריות, $\nabla_v v$, בפרק 3 לצורך פיתרון משוואות שימור התנע. למרבה הצער, בעוד $D(v)$ הוא חישוב שהינו פנימי למשטח, הדיסקרטיזציה שלנו כוללת נגזרת חיצונית עם הטלה, ולכן הפותר נוזלים שקיבלנו פחות מוצלח מהפותר שתוכנן בפרק 6. אף על

$$\partial_t u + \langle v, \mathrm{grad}\, u\rangle = 0\ ,$$

כאשר ∂_t היא הנגזרת בזמן, grad זהו אופרטור הגרדיאנט ו- $\langle\cdot,\cdot\rangle$ זוהי המכפלה הסקלרית הסטנדרטית. אכן, משוואת ההסעה מקשרת בין השינוי ב- u במהלך הזמן יחד עם (מינוס) השינוי במרחב, מכיוון שהביטוי $\langle v, \mathrm{grad}\, u\rangle$ הוא הנגזרת הכיוונית של u ביחס ל- v.

בפרק 5, נעסוק במשוואת ההסעה במקרה בו נתונים תנאי שפה לזמן ההתחלה והסוף, כלומר $u(0) = u_0$ ו- $u(1) = u_1$, והמטרה העיקרית היא לחשב את $u(t)$ ו- $v(t)$ עבור $t \in (0,1)$. למעשה, זו בעיה שאינה מוגדרת היטב, ולכן ניאלץ להוסיף רגולריזציה על מנת להגביר את הסיכויים למציאת פיתרון. המבנה הנ"ל יותאם למשימת שיפור מיפוי התחלתי φ בין שני משטחים $\mathcal{M}_1$ ו- $\mathcal{M}_2$. כפי שנראה בפרק 5, אם מרחב הפונקציות של $\mathcal{M}_1$ מושם בהתאמה למרחב הפונקציות של $\mathcal{M}_2$ ניתן לשפר באופן משמעותי את המיפוי הנתון φ. למרבה המזל, בעיית השפה לעיל מתאימה למשימה זו, כאשר ניתן לקחת $u_0 = f \circ \varphi$ ו- $u_1 = g$, עם $f \in L_2(\mathcal{M}_1)$ ו- $g \in L_2(\mathcal{M}_2)$, ולהשתמש ב- $v(t)$ המחושב כדי לייצר את המיפוי החדש φ. אנו מפנים את הקורא לפרק הרלבנטי לקבלת פרטים נוספים בנושא.

המד"ח הבאה שנבחן קשורה לזרימה של נוזלים, היא מבטיחה את שימורו של התנע, והיא חלק ממשוואות נאוויה--סטוקס הידועות:

$$\partial_t v + \nabla_v v - \mu \nabla^2 v + \mathrm{grad}\, p = 0\ ,$$

כאשר v היא המהירות, p זהו הלחץ, ∇ היא נגזרת קו-ווריאנטית, ו- ∇^2 זהו לפלסיאן וקטורי עם μ מקדם הצמיגות. למרבה הצער, פתירת המשוואה הזו על משטחים עקומים זו משימה מאתגרת כפי שנתאר בקצרה בהמשך ובפירוט בפרק 3. אף על פי כן, המד"ח הוקטורית הנ"ל ניתנת לפישוט באופן משמעותי למד"ח סקלרית ע"י הגדרת ערבוליות הזרימה, $\omega = \mathrm{curl}\, v$, המצביעה תמיד בכיוון הנורמל למשטח, במקרה הדו-ממדי. לכן, במקום לפתור את משוואת שימור התנע, אנו נמדל זרימה שאינה צמיגה בפרק 6 ע"י בחינת משוואות הערבוליות, המתקבלת מלקיחת הרוטור של המשוואה לעיל:

$$\partial_t \omega + \langle v, \mathrm{grad}\, \omega\rangle - \mu \Delta \omega = 0\ .$$

אנו שוב עם מד"ח מסוג הזזה, אך הפעם היא לא לינארית בגלל האילוץ המקשר בין ω ל- v ובנוסף היא מכילה רכיב צמיגות.

תקציר המחקר

התזה הבאה מציגה שיטות פרקטיות להתמודד עם כמה בעיות מאתגרות המוגדרות על משטחים עקומים שאינם טרביאליים. לדוגמא, אנו נרצה להעריך באופן נומרי כיצד שכבה דקה של יין זורמת על גבי כוס יין. כיצד ניגשים כדי לפתור בעיה קשה שכזו?
השיקולים הבאים צריכים להילקח בחשבון. ראשית, התנועה וסיבותיה, כלומר הדינמיקה של הבעיה מנותחת ומיוצגת באופן מתמטי ע"י משוואה דיפרנציאלית. בעבודה זו, אנו מניחים שהמשוואות השולטות בבעיה, כבר פותחו בעבודות קודמות, למשל, בדוגמא המנחה שלנו, היין יכול להיות ממודל בקירוב ע"י משוואות נוזלים דקים. שנית, המשטח הגיאומטרי העקום מיוצג בצורה כזו שתאפשר לבצע חישובים פשוטים, תוך כדי שמירת היכולת לתאר גיאומטריות סבוכות. לצורך כך, התחום מקורב באמצעות משטח משולשי, כלומר, קבוצה של קודקודים, צלעות ופיאות אשר נתונות ע"י משולשים מישוריים המחוברים בינהם בצלעות. השיקול השלישי ומוקד העניין העיקרי בעבודה זו, הוא פתירת המשוואות על גבי המשטח הבדיד במונחי שיטה נומרית שניתנת לקידוד בשפת מחשב כלשהי.

משוואות דיפרנציאליות מתארות את השינוי בזמן או מרחב (או שניהם) של משתנה הכרחי לבעיה. נוזלים דקים, למשל, מאופיינים באמצעות גובה הנוזל, h, המחושב בכל נקודה בתחום. האבולוציה בזמן ובמרחב של h נחשבת כפיתרון של המשוואות הנתונות. פיתרון נומרי של משוואות דיפרנציאליות היא בעיה קשה, המצריכה תכנון של כללי אינטגרציה בזמן יחד עם בניה של אופרטורים דיפרנציאליים במרחב על מנת לייצר שערוך אמין של דינמיקת הבעיה. בהינתן הסיווג הנ"ל, החצי הראשון של העבודה הבאה דן בשיקולים המרחביים, והחצי השני בעיקר עוסק בכללי האינטגרציה בזמן.

אנו ממשיכים בתיאור מעלה-מטה של העבודה. כלומר, ראשית נתאר את המשוואות הדיפרנציאליות החלקיות (מד"ח) אותן נרצה לפתור, ולאחר מכן, נתמקד ברכיבים המרחביים הדרושים להרכבת פותרים נומריים אפקטיביים. כפי שהדוגמא לעיל מרמזת, אנו נתעניין בבעיות הסעה בעיקרן, ולכן, אנו מתחילים ממד"ח שמתאר באופן מתמטי כיצד משתנה מסוים, u, נע בהשפעת שדה מהירות, v. למעשה, זוהי אחת מהמד"ח הנפוצות ביותר והיא משמשת לתיאור של מגוון תופעות טבעיות והיא נקראת משוואת ההסעה:

המחקר נעשה בהנחיית פרופמ' מירלה בן-חן, בפקולטה למדעי המחשב.

אני מודה לטכניון, לפקולטה למדעי המחשב, למלגת ג'ייקובס—קואלקום, למלגת אדמס של האקדמיה הישראלית למדעים ואומניות, ולמענק נסיעה ע"ש פרופ' רחמימוף עבור חוקרים צעירים של הקרן הדו-לאומית ישראל-ארה"ב למדע על התמיכה הנדיבה בהשתלמותי.

ייצוג מבוסס אופרטורים
בעיבוד ספרתי של גיאומטריה

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
דוקטור לפילוסופיה

עומרי אזנקוט

הוגש לסנט הטכניון – מכון טכנולוגי לישראל
אייר, התשע"ז　　חיפה　　מאי, 2017

ייצוג מבוסס אופרטורים

בעיבוד ספרתי של גיאומטריה

עומרי אזנקוט