Topological Aspects of Maps Between Surfaces

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Janis Born, M. Sc. RWTH
aus Duisburg, Deutschland

Berichter: Professor Dr. rer. nat. Leif Kobbelt
Professor Maks Ovsjanikov, PhD

Tag der mündlichen Prüfung: 29. März 2022

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek verfügbar.

Abstract

The generation of high-quality maps between surfaces of 3D shapes is a fundamental task with countless applications in geometry processing. There is a particular demand for maps that offer strict validity properties such as continuity and bijectivity, i. e. *surface homeomorphisms*. Such maps not only define a geometric one-to-one correspondence between surface points, but also a matching of topological features: an identification of handles and tunnels and how the map wraps around them.

Finding a natural, low-distortion surface homeomorphism between a given pair of shapes is a challenging design task that involves both combinatorial (topological) and continuous (geometric) degrees of freedom. However, while powerful methods exist to improve existing homeomorphisms through continuous modifications, these are limited to merely geometric updates, and hence cannot alter map topology.

In this light, it is quite surprising that most existing techniques for the initial construction of homeomorphisms do not systematically deal with questions of map topology and instead relegate these issues to user input or ad-hoc solutions. Unfortunately, this lack of reliable and automatic methods for the critically important topological initialization has so far prevented a further automation of homeomorphic surface map generation.

In this thesis, we aim to close this practical gap by devising new algorithms that specifically address the map-topological issues underlying the construction of surface homeomorphisms. Our theoretical foundation is the study of the *mapping class group*, an algebraic structure which characterizes the entire topological

design space. We approach the task of map topology generation from two different angles, based on different mapping class representations:

We propose a robust method for the construction of maps from sparse landmark correspondences, based on compatible layout embeddings. Our robust embedding strategy systematically searches for short, natural embeddings and therefore reliably avoids a range of sporadic topological initialization errors which can occur with previous heuristic approaches.

Additionally, we introduce a novel algorithm to extract topological map descriptions from approximate, non-homeomorphic input maps. Such a purely abstract description of map topology may then be used to guide the construction of a proper homeomorphism. As our inference method is highly robust to a wide range of map defects and imperfect map representations, this effectively allows to delegate the difficult task of finding a natural map topology to specialized shape matching methods, which have grown increasingly capable.

These advancements promote the further automation of map generation techniques in two regards: They vastly reduce the need for human supervision, and make the results of automatic shape matching methods accessible for topological initialization.

Zusammenfassung

Die Erzeugung von Abbildungen zwischen Oberflächen von 3D-Objekten ist eine wiederkehrende Aufgabe in der Geometrieverarbeitung. Dabei liegt ein besonderes Augenmerk auf hochqualitativen Abbildungen, die besonderen Korrektheitsanforderungen hinsichtlich Stetigkeit oder Bijektivität genügen. Solche *Homöomorphismen* definieren nicht nur eine geometrische Korrespondenz zwischen allen Oberflächenpunkten, sondern auch eine topologische Zuordnung globaler Oberflächenmerkmale wie Löcher, Henkel oder Tunnel.

Die Konstruktion eines natürlichen, verzerrungsarmen Homöomorphismus zwischen zwei gegebenen Oberflächen ist daher eine äußerst filigrane Aufgabe, welche die Berücksichtigung sowohl kombinatorischer (topologischer), als auch kontinuierlicher (geometrischer) Freiheitsgrade erfordert. Es gibt zwar automatische Methoden zur Optimierung von bereits existierenden Homöomorphismen, jedoch beschränken sich diese auf rein kontinuierliche geometrische Verbesserungen und erlauben keine topologischen Korrekturen.

Es mag daher überraschen, dass viele bisherige Algorithmen zur Initialisierung von Homöomorphismen sich wenig darum bemühen, die korrekte Abbildungs-Topologie zu finden: Diese wichtige Aufgabe wird entweder bloß durch (unzuverlässige) Heuristiken gesteuert oder gänzlich der Eingabe der Nutzer:in überlassen. Bislang steht dieser Mangel an topologisch robusten Initialisierungsmethoden einer vollständigen Automatisierung der Erzeugung von Oberflächenabbildungen im Weg.

Ziel dieser Arbeit ist daher die Beschreibung neuer Algorithmen, welche sich explizit der topologischen Schwierigkeiten in der Konstruktion von Homöomor-

phismen annehmen, um diese Automatisierungslücke zu schließen. Theoretische Grundlage dafür ist die Betrachtung der Abbildungsklassengruppe (*mapping class group*) von Oberflächen, welche eine systematische Beschreibung aller möglichen Abbildungs-Topologien erlaubt. Ausgehend von verschiedenen Repräsentationen dieser Gruppe wählen wir zwei unterschiedliche Herangehensweisen:

Wir präsentieren einerseits einen robusten Algorithmus zur Konstruktion von Abbildungen aus vorgegebenen Punkt-Korrespondenzen, basierend auf Einbettungen eines gemeinsamen Zellkomplexes. Unsere Methode sucht systematisch nach natürlichen, möglichst kurzen Einbettungen und vermeidet so die typischen sporadisch auftretenden Initialisierungsfehler vorheriger Methoden, welche bloß heuristisch vorgehen.

Daneben beschreiben wir ein neues Verfahren zur Erstellung rein topologischer Beschreibungen von bereits gegebenen Oberflächenabbildungen. Solche rein abstrakten Beschreibungen können dann zur Festlegung der topologischen Freiheitsgrade neuer Abbildungen herangezogen werden. Unser Analyseverfahren unterstützt eine Reihe gängiger, insbesondere nicht-homöomorpher Abbildungsdarstellungen und verarbeitet selbst fehler- oder lückenhafte Eingabedaten. Dies ermöglicht erstmals die Nutzung spezialisierter Shape-Matching-Verfahren zur vollautomatischen Bestimmung semantischer Korrespondenzen, deren topologische Struktur schließlich durch unser Verfahren zur Herstellung von Homöomorphismen nutzbar gemacht wird.

Diese Neuerungen bieten somit zuverlässige und automatische Alternativen zu bisherigen Erzeugungsmethoden von Oberflächenabbildungen. Sie erfordern ein wesentlich geringeres Maß an menschlicher Interaktion und Beaufsichtigung und machen gleichzeitig die Ergebnisse vollautomatischer Korrespondenzverfahren für topologische Konstruktionen nutzbar.

Acknowledgments

This thesis is the result of my work at the *Visual Computing Institute* and as a fellow of the International Research Training Group *Modern Inverse Problems* of the German Research Foundation (DFG). I gratefully acknowledge the financial support through grant IRTG-2379-5. Additional funding was generously provided through the ERC Advanced Grant *ACROSS* and the Gottfried Wilhelm Leibniz Prize of the DFG.

I sincerely thank my doctoral advisor, Leif Kobbelt, for his distinguished supervision. He allowed me to freely pursue my research interests and provided excellent guidance and advice whenever I needed. I was honored to have Maks Ovsjanikov bring his expertise as co-reviewer and second examiner for my dissertation. I also kindly thank Erika Ábrahám and Klaus Wehrle for serving on my examination committee.

I am grateful to everyone who has contributed directly or indirectly to the success of this work: I wish to thank my former advisors Jan-Robert Menzel, Lars Krecklau, and Hans-Christian Ebke for their mentorship during my undergraduate studies. I likewise valued the competent help from my student assistants Gökay Baytekin, Anton Florey, Hannes Hergeth, Saša Lukić, and Dario Seyb. In many ways, my research was profoundly shaped by the surefooted guidance from Marcel Campen and David Bommes: Countless times, they have generously shared their vast expertise, patiently listened to my often outlandish ideas and encouraged me to pursue the more promising ones. I also thank Markus Baumeister for introducing me to the fascinating subject of algebraic topology and reviewing parts of my

manuscript. Without him I wouldn't know half of the theory which has now become the foundation of this thesis.

Over the last years, I had the great pleasure of working alongside so many brilliant and kind colleagues at the *Visual Computing Institute*: I will fondly remember Arturs Berzins, Yan-Pei Cao, Peter Collienne, Tim Elsner, Johannes Frohn, Lin Gao, Florian Gawrilowicz, Anne Gehre, Alexandra Heuschling, Joe Jakobi, Javor Kalojanov, Gregor Kobsik, Ming Li, Manish Mandad, Christian Mattes, Sven Middelberg, Moritz Ibing, Andreas Longva, Julius Nehring-Wirxel, Florent Poux, Haingoharijao Faniriniaina Ramandiamanana, Patric Schmitz, Kersten Schuster, Zain Selman, Dominik Sibbing, Andreas Tillmann, Robin Tomcin, Philip Trettner, Ole Untzelmann, Cédric Zanni, Tianyu Zhou, and all my fellows from the IRTG. Jan Möbius deserves particular praise for his outstanding technical support, as do Silke van Betteraey, Monika Maszynkiewicz, and Sabrina Fiedler for their competent assistance with all bureaucratic intricacies. I especially cherished the company of Max Lyon, Isaak Lim, and Patrick Schmidt, not only for many fruitful and illuminating scientific discussions, but for their like-minded spirit, their humor, and friendship. Patrick's impact on my work has been immeasurable: I am thoroughly grateful for his dedication as collaborator and co-author on all our mutual projects and his company as a long-time office mate.

My deepest thanks are to my family and friends. Their patient and loving support was a constant source of encouragement which gave me the confidence and energy to successfully complete this work.

Contents

1 Introduction

Geometry processing studies the digital representation and manipulation of three-dimensional shapes. As the visually most defining part of a shape is its *surface*, a wide range of geometry processing algorithms operate on purely surface-based representations (typically discretized as polygon meshes). Common tasks are the digitization of real-world objects, their editing and synthesis, and their preparation for applications such as visualization, animation, simulation, analysis, or manufacturing.

In this setting, many algorithms do not just consider a single shape in isolation but instead operate on pairs or collections of several shapes simultaneously to synthesize or transfer information across surfaces. Such tasks invariably require some notion of *shape correspondence* that identifies matching parts or regions on different surfaces. Mathematically, such a correspondence is established by a *surface map*, i. e. some function that maps between two shapes. With a vast range of applications, surface maps are ubiquitous in geometry processing. Over the last decades, a considerable amount of research has been dedicated specifically to their generation, encoding, and optimization.

There is a particular demand for high-quality continuous map representations such as *surface homeomorphisms* which offer strict validity properties and are therefore suitable for the most sophisticated mapping tasks. However, the construction of such maps is notoriously complicated as it must not only consider geometrical aspects such as smoothness or distortion, but also aspects of *map topology*, which call for global combinatorial decisions.

Meanwhile, many existing map construction methods mostly avoid a direct engagement with topological questions, and instead rely on user input or ad-hoc heuristics to settle these issues. As we will point out, this cursory treatment of map topology has so far been a major hindrance towards a full automation of map generation tasks. In this thesis, we seek to remedy these shortcomings and focus on algorithmic solutions that properly handle topological degrees of freedom in map construction automatically and reliably.

1.1 Maps Between Surfaces

Surface maps are an essential tool for many computer graphics techniques. Their most common areas of application are the simultaneous analysis or processing of different shapes, and inter-surface data transfer.

For the simultaneous processing of shape collections, surface maps establish mutual surface-to-surface correspondences. One application is statistical co-analysis [MDW08], e. g. the computation of average shapes or modes of variation [PSS01; HRM05]. Another task is compatible re-meshing [KS04; ZCZ+18; YFC+18; YZL+20], i. e. the computation of a shared mesh connectivity with different geometric embeddings that approximate instances of a shape collection (e. g. different poses of an animated character).

Similarly, surface maps may be used for morphing [LDSS99; Ale00; ZSH00; CFB16], i. e. the synthesis of continuous shape interpolation sequences. This enables the recovery of animations from keyframe poses [TSSH15], or the reconstruction of anatomical growth processes [ESD+22].

Probably the most fundamental operation associated with surface maps is surface-to-surface data transfer, where pointwise information defined on some source shape (e. g. color, material, temperature, etc.) is carried over to another target shape through the action of a map. This kind of data transfer is essential for any task where information is only available on a single object

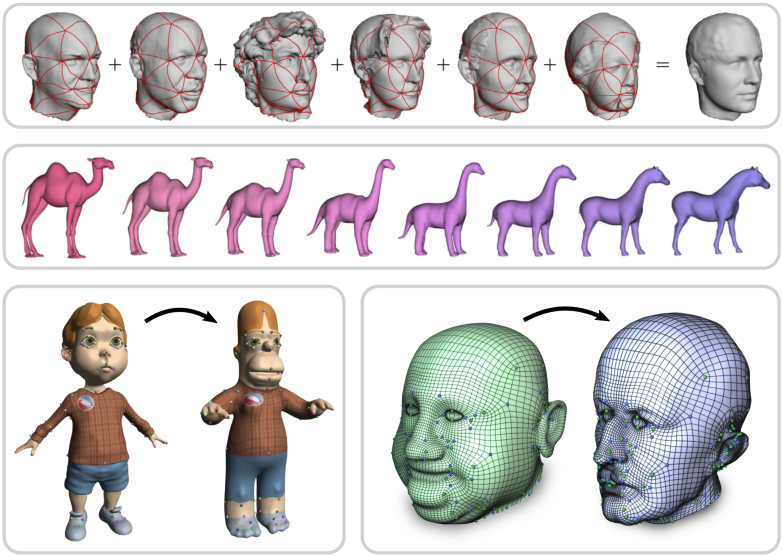


Figure 1.1: *Typical applications for surface maps are shape co-analysis, e. g. the computation of average shapes (top row, by [PSS01]), mesh morphing (center row, by [CFB16]), the transfer of textures (bottom left, by [PBDS13]) or tessellation patterns (bottom right, by [TDN+11]).*

and needs to be propagated onto different instances, which may represent pose variations or level-of-detail versions of the same shape, or entirely different objects. Exemplary applications are the transfer of textures [SSGH01; DYT05], geometric details [BMBZ02], parametrizations [ESCK16], or tessellation patterns for example-based re-meshing [TDN+11; HPG+21].

With this broad range of tasks relying on surface maps as input, it is quite natural to ask how to obtain such maps in the first place. This question of *map generation* has attracted significant research interest and countless methods have been proposed over the past years: In their most basic form, these are map construction tools which heavily rely on human input (e. g. in the form of landmark

annotations). At the other end of the spectrum are fully automatic *shape matching* algorithms that infer a (full or partial) surface-to-surface correspondence without any further user input [KZHC11; Sah20]. These are generally based on empirical correspondence models and aim to find maps that align geometrically similar surface regions or minimize some form of mapping distortion.

Such map generation approaches are of course heavily influenced by the underlying *map representation*, i. e. the discrete encoding of the desired surface map, which by itself has been a subject of extensive study: Motivated by different practical and computational considerations, a variety of representations have been suggested, which differ in their precision, compression, validity guarantees, and their eligibility for editing, composition or optimization. These include e. g. sparse, sample-based correspondences, extrinsic registrations, piecewise-linear continuous mappings, or fuzzy matchings between coarse surface regions. We will review the most relevant approaches for map generation and representation in greater detail in Sections 4.1 and 5.1.

1.2 Surface Homeomorphisms

Despite this range of available practical map representations, many downstream applications are necessarily restricted to a subset of input maps that fulfill certain formal validity requirements. This is generally the case for mapping tasks that must in some way preserve the structural integrity of the data being transmitted from one surface to the other. One prototypical example is the transfer of texture UV coordinates or 2D surface parametrizations.

A basic requirement for this task is map *sharpness*: An input map must unambiguously send any point from the source surface to a single point on the target surface, i. e. it must be a well-defined point-to-point map — a condition that is violated by many fuzzy or distribution-based map representations.

More importantly, maps for texture transfer must be *dense* and *continuous*: For textures with seams, it does generally not suffice to map UV coordinates at a few sampled positions and interpolate the results in between: A proper transfer of texture seams (or chart boundaries) requires a fully continuous mapping of *all* source points that allows a faithful reconstruction of seams on the target surface. In this setting, continuity additionally prevents *tearing*, i. e. jumps or jarring cuts in the mapped texture image.

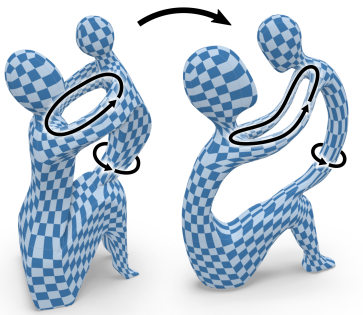
Another essential prerequisite for valid data transfer is map *injectivity*, i. e. that no two different source points map to the same target location. This condition alone rules out many potential mapping artifacts such as local inversions and foldovers, or global double-coverings. If the input map additionally covers the entire target surface (i. e. is *bijective*), it becomes fully *reversible*: Such a map describes a bidirectional one-to-one correspondence between surface points of the two shapes. Information transfer may then be implemented equivalently by *pushing* data from the source surface, or by *pulling* it onto the target surface through the inverse map, whatever is more convenient.

Mathematically, the class of maps that subsumes all of the above favorable properties is that of a *surface homeomorphism*: a point-to-point map between two surfaces that is bijective and continuous in both directions (Fig. 1.2). In

Figure 1.2: *A homeomorphism (a), generated by [SCBK20], defines a continuous one-to-one correspondence between two surfaces. Homeomorphisms provide strict correctness properties, e. g. for guaranteed bijective and inversion-free texture transfer (b).*

practice, only few surface map representations meet these strict requirements. Those that do are typically more involved to construct: Examples are piecewise-linear maps defined over mutual tessellations, or parametrizations onto some common intermediate domain, e. g. the plane or some other constant-curvature space. In return, such discrete homeomorphisms can handle the most intricate mapping tasks with rigorous correctness guarantees: They ensure that no data is lost or invalidated during transfer and are therefore indispensable for applications that need to transmit delicate, high-frequency surface signals.

Unlike other non-continuous map representations, (discrete) homeomorphisms fully and unambiguously describe a structural correspondence between two surfaces. Not only does such a mapping preserve local neighborhood relations, but it also defines a global *topological equivalence*: It uniquely identifies a matching between high-level topological features such as handles or tunnels of the two objects. By observing how paths that travel around topological features of the source surface are mapped to corresponding trajectories around handles or tunnels on the target surface, one obtains a purely abstract description of the *topological type* or the *mapping class* of a homeomorphism. This topological classification coincides with a notion of continuous equivalence between maps, as we will examine in Chapter 3: Two homeomorphisms in the same mapping class are merely geometric variations of each other, in the sense that they can be transformed into one another through a continuous deformation (a *homotopy*). Conversely, it is plainly impossible to alter the topological type of a given map through only continuous modifications in the space of homeomorphisms. The fact that this space cannot be continuously searched is indeed one of the major obstacles that complicates the construction of homeomorphisms.



1.3 Surface Homeomorphism Generation

Despite the practical advantages of discrete homeomorphisms, their automatic generation between arbitrary input shapes remains a challenging problem. The general objective remains the same: finding a map that optimizes some global quality measure (typically: low map distortion) while optionally incorporating constraints such as landmark correspondences. Map generation methods that target homeomorphisms must additionally enforce the mandatory continuity and bijectivity requirements. However, doing so invariably forces them to make clear-cut decisions about map topology: Whether they deal with this aspect explicitly or not, they must ultimately commit to a specific mapping class.

A proper optimization in this setting must therefore consider a solution space with both discrete combinatorial (topological) and continuous (geometric) degrees of freedom, which makes a systematic search extremely difficult: While already the continuous optimization within a single mapping class is generally a challenging non-convex problem, there is currently no viable method for a joint, simultaneous optimization of map topology and geometry. Instead, the predominant strategies in this area are two-phase algorithms that separate topological and geometric concerns into different algorithm stages.

Topology-First Map Generation

The standard process for the generation of discrete homeomorphisms is to first construct a rough but valid initial map which is then continuously optimized to improve geometric quality in a second step.

The initial map construction is typically based on compatible surface decompositions where both source and target surface are partitioned into corresponding collections of disk-like patches or cells [PSS01; KSG03; KS04]. Such decompositions are often defined as *layout embeddings* where the patches and patch boundaries correspond to embedded faces and edges of an abstract cell complex

or *layout*. A globally bijective and continuous map is then established through canonical planar parametrizations of the individual patches.

The result of this first step is — by construction — a valid discrete homeomorphism, but its geometric quality can be extremely poor. Therefore, it is subsequently improved by applying a homeomorphism-preserving map optimization [KS04; APL14; APL15; AL16; AKL17], often driven by a distortion objective that simultaneously acts as a barrier against discontinuities or local inversions [SAPH04; SBCK19; SCBK20]. While these methods guarantee that the map remains a valid homeomorphism throughout the entire process, this necessarily restricts the optimization to homotopic map updates and thus strictly prohibits any topological changes: It remains confined to the mapping class selected by the initial map construction.

This restriction puts an immense responsibility on the primary homeomorphism initialization phase: Without any way to alter the map topology afterwards, it must select the correct mapping class right at the beginning, or the following continuous optimization will never be able to reach the desired result.

In this light, it is somewhat surprising that existing methods for this stage are typically not driven by a rigorous search for the most suitable mapping class, but instead resort to greedy heuristic methods, where layout embeddings are simply constructed by a best-first sequence of shortest-path insertions. Accidental blocking of paths may force embeddings into detours, which can accumulate to cause severely entangled and topologically unintended map initializations. Due to such chaotic and unpredictable failure cases, this initial map construction phase can require a significant degree of human attention and intervention: Often, this involves several trial-and-error iterations of tweaking the landmark constraints until a desired result is achieved. Due to their weak reliability, it is quite risky to use these greedy initialization methods as part of automatic processing pipelines where potential errors cannot be manually detected and corrected.

We will address these issues in Chapter 4 where we propose a robust alternative to purely heuristic-based map initialization schemes. Based on a branch-and-bound optimization strategy, it can reliably find short, natural layout embeddings.

Geometry-First Map Generation

Current fully automatic shape matching methods generally do not directly produce valid homeomorphisms. However, a significant number of methods *can* generate discrete shape correspondences that approximate or resemble a surface homeomorphism reasonably well, albeit in a non-homeomorphic representation.

This suggests an alternative construction strategy: Given some non-homeomorphic input map, one could try to convert it to an actual homeomorphism through local continuous extensions or corrections of non-injectivities.

The promise of this approach lies in its potential to leverage state-of-the-art shape matching algorithms for map generation. These automatic methods have grown increasingly capable at identifying natural, semantically meaningful shape correspondences, e. g. based on learned features [Sah20]. However, these generally operate on non-homeomorphic map representations, and produce merely geometric (but not topologically well-defined) results, which are not suitable for mapping tasks with strict input requirements.

Despite the obvious practical appeal, an automatic recovery of surface homeomorphisms from such low-quality maps is largely unexplored territory: While there exist several methods to disentangle non-injective input maps [FL16; DAZ+20; DKZ+21], these only pertain to surface-to-plane parametrizations. Before similar methods can be applied for surface-to-surface maps, one must deal with the topological issues that naturally arise in this setting: One must choose the correct mapping class for the homeomorphic reconstruction, which requires an identification of the intended or implied topology of the non-homeomorphic input map.

In practice, this identification turns out to be a rather difficult task, as the topological information often cannot be simply "read off" from the input map: Beyond the inherent restrictions of their map representation, such maps are often further afflicted by numerous defects and inconsistencies, e. g. noise, gaps, outliers, or non-injective overlaps. A proper map topology inference must therefore gracefully handle these local imperfections and consolidate ambiguities to form a unique, globally consistent description of the given map topology.

We propose such a topology inference algorithm for ambiguous input maps in Chapter 5. As the first work to provide such a solution, this marks an essential step towards connecting general shape matching methods to strictly homeomorphic map applications.

1.4 Contributions

Our work covers both theoretical and practical aspects of map topology in homeomorphic surface map generation. We make the following major contributions:

- *Map Topology*. We provide a thorough theoretical analysis of the topological aspects of surface map design: We identify the *mapping class group* as the central object of study, a well-known structure from geometric topology which has so far seen surprisingly little attention in geometry processing literature. It offers a complete, canonical description of the topological design space of surface homeomorphisms. We present a concise, accessible exposition of the underlying foundational concepts from algebraic topology, specifically targeted at geometry processing practitioners and applications.

- *Map Topology Representations*. We discuss several mapping class representations and how they translate to practical map topology encodings. In particular, we focus on descriptions in the form of discrete layout embeddings, and the so-called symplectic representation, which set the stage for our main algorithmic contributions:

- *Robust Layout Embedding*. We propose a novel method for the computation of landmark-constrained layout embeddings — an essential component for the initialization of surface homeomorphisms. In contrast to previous works, our algorithm systematically searches for short, natural embeddings across different mapping classes and therefore reliably avoids the sporadic topological initialization errors that can occur with previous heuristic-based approaches.

- *Robust Map Topology Inference*. Towards the goal of homeomorphic map completion, we introduce a new algorithm that infers the intended or implied topological type from a given non-homeomorphic input map. The resulting description may then serve to guide the construction of a topologically unambiguous homeomorphism. In this setting, we leverage the symplectic representation of the mapping class group to devise an extremely robust inference formulation that supports a wide range of typical input map formats and gracefully handles local defects and ambiguities.

Our contributions thus extend the current capabilities of surface map generation techniques in two areas:

For the conventional ab-initio construction of homeomorphisms from landmark correspondences, we eliminate the risk of topological initialization errors due to heuristic mispredictions. This increased reliability significantly reduces the required degree of human supervision and therefore enables applications in large-scale automated settings.

On the other hand, we lay the groundwork for the automatic conversion of non-homeomorphic input maps to true homeomorphisms; bridging the gap between automatic shape matching algorithms that produce imperfect maps, and downstream applications that demand strict map validity. Such a conversion fundamentally requires a reliable estimate of the desired map topology — which is precisely what our inference algorithm provides.

1.5 Overview

We begin our exposition with a review of basic topological properties and algebraic structures related to surfaces and surface-embedded objects such as paths or fields (Chapter 2).

These serve as foundation for our main theoretical focus: the topology of surface homeomorphisms (Chapter 3), which is fully characterized by the mapping class group (Section 3.2), an algebraic structure that encompasses and relates all potential map topologies. We investigate various potential representations of mapping classes and their practical merits (Section 3.4), in particular compatible layout embeddings (Section 3.4.1), and the symplectic representation (Section 3.4.4), which are the respective settings of our following technical contributions to surface map generation methods:

We then present our robust algorithm for the computation of short, landmark-constrained layout embeddings (Chapter 4), and our method to infer purely topological descriptions from only approximately homeomorphic input maps (Chapter 5), which were previously published in [BSK21] and [BSCK21]. We conclude our work by reflecting on some remaining challenges and potential courses of action towards fully automatic, high-quality homeomorphism generation (Chapter 6).

2 Preliminaries

We begin with a review of some basic topological properties of surfaces, embedded paths, and the algebraic structures that arise from them. While algebraic topologists have studied these concepts in great generality, we focus on concrete instances that are specifically relevant for the description and construction of surface maps, with special attention to their algorithmic applications in geometry processing. We will therefore provide most results without proof and refer to the literature [Bre93; Sti93; Hat02; Lee11; LM18; Mun18] for a formal and generalized treatment.

2.1 Surfaces

In the following, we consider surfaces that represent the outer hull or shell of objects. More precisely, by a *surface* (denoted by $\mathcal{S}$) we always mean a compact, connected, orientable two-manifold (with or without boundaries), typically embedded into three-dimensional Euclidean space $\mathcal{S} \subset \mathbb{R}^3$. While we are ultimately interested in results for discrete surfaces (e. g. defined by triangle meshes), we will state most concepts in terms that also apply to more general continuous surfaces.

We consider two surfaces $\mathcal{S}$ and $\mathcal{S}'$ to be topologically equivalent if they are *homeomorphic*, i. e. if there exists a *homeomorphism*, a bijective map $f : \mathcal{S} \to \mathcal{S}'$ such that both f and and its inverse f^{-1} are continuous.

Up to homeomorphism, all closed surfaces can be classified by their *genus* $g \in \mathbb{N}$. Informally, g counts the number of handles of a surface, or the maximal number of cuts one can make before it becomes disconnected. We write $\mathcal{S}_g$ to denote a closed genus g surface. Examples are the sphere $\mathcal{S}_0$, or the torus $\mathcal{S}_1$. Two

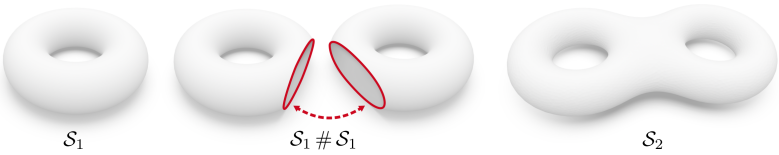


Figure 2.1: *A torus $\mathcal{S}_1$ is a surface with a single handle (left). The connected sum $\mathcal{S}_1 \# \mathcal{S}_1$ of two tori (center) is a double torus $\mathcal{S}_2$ (right).*

closed surfaces $\mathcal{S}_{g_1}$ and $\mathcal{S}_{g_2}$ are homeomorphic if and only if they have identical genus $g_1 = g_2$ [Lee11, Thm. 10.22]. Surfaces of arbitrary genus can be generated by successively attaching handles: Given two surfaces $\mathcal{S}$ and $\mathcal{S}'$, we obtain the *connected sum* $\mathcal{S} \# \mathcal{S}'$ by cutting some disk-shaped region out of $\mathcal{S}$ and $\mathcal{S}'$ and gluing them together along the resulting boundaries (Fig. 2.1). With this operation, a surface of genus g is constructed as the connected sum of a sphere and g tori: $\mathcal{S}_g = \mathcal{S}_0 \# \mathcal{S}_1 \# \cdots \# \mathcal{S}_1$.

A related way to create surfaces of arbitrary genus is by gluing a single planar *fundamental polygon* to itself in a certain way [Lee11, Prop. 6.14]. To generate a surface of genus $g \geq 1$, take a $4g$-sided polygon and label the sides in a counterclockwise sense with oriented arrows according to the pattern $a_1 b_1 \overline{a}_1 \overline{b}_1 \cdots a_g b_g \overline{a}_g \overline{b}_g$, where bars indicate arrows with reversed orientation (Fig. 2.2 left). Then, glue the polygon together by identifying each pair of sides with the same label, such that the arrows have matching orientations. The resulting surface is homeomorphic to $\mathcal{S}_g$ [Bre93, Ch. 9]. All vertices of the fundamental polygon coincide at a common point and each group of sides $a_i b_i \overline{a}_i \overline{b}_i$ is glued to a pair of loops that surround one handle and corresponding tunnel of the surface (Fig. 2.2 right).

2.2 Combinatorial Surfaces

For many computational tasks, we deal more concretely with surfaces based on finite, discrete representations, most importantly: polygon meshes. A polygon

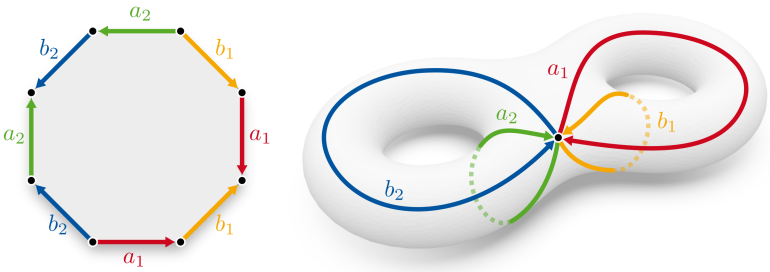


Figure 2.2: *Gluing the sides of a $4g$-sided fundamental polygon (left) according to a specific pattern (oriented arrows) yields a closed surface $\mathcal{S}_g$ (right), here illustrated for $g = 2$.*

mesh separately describes two different aspects of a surface: First, it establishes a purely combinatorial description of the surface's topology in the form of an *abstract cell complex*. Second, it defines the surface's geometry by providing an *embedding* of the cell complex into an embedding space, typically $\mathbb{R}^3$.

An abstract cell complex describes the combinatorial structure of a manifold polygon mesh: a collection of cells (vertices V, edges E, faces F) and their connectivity. This entire structure can be constructed from a finite set of halfedges H, related by a *rotation system* (ρ, ν) describing two operations [LM18, Ch. 3.2]:

- *Opposite halfedge*. $\rho : H \to H$ maps each halfedge h to its counterpart with reversed orientation. It is an involution (i. e. $\rho(\rho(h)) = h$) without fixed points ($\rho(h) \neq h$ for all h).

- *Next halfedge*. $\nu : H \to H$ maps each halfedge to the next halfedge within the same face, in a counterclockwise sense. ν is a permutation.

These operations allow to navigate a set of halfedges H, as illustrated in Fig. 2.3, and are found in many software implementations of halfedge data structures, such as [BSBK02]. More fundamentally, the combinatorial structure of the rotation system fully defines the cells of the complex, which can be discovered through repeated applications of the navigation operations. For a given halfedge $h \in H$,

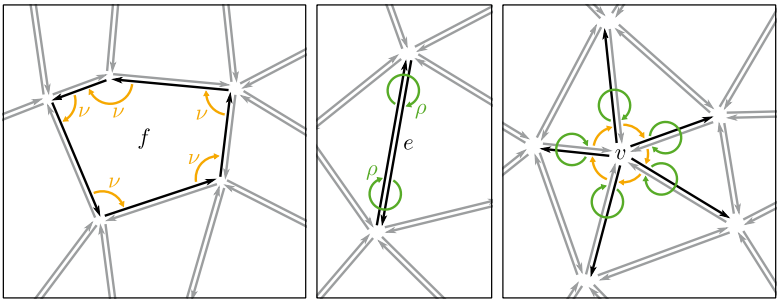


Figure 2.3: *An abstract cell complex is defined by a rotation system (ρ, ν) acting on halfedges (arrows). Faces (left), edges (center), and vertices (right) arise from different orbits of these rotation operations.*

we define the *orbit* $g^*(h)$ of an operation g as the set of all halfedges reachable from h by repeated applications of g. We obtain the cells as follows:

- *Faces* F. Repeated applications of ν circulate a halfedge $h \in H$ counterclockwise along the boundary of a single face (Fig. 2.3 left). We thus identify a face with an orbit $f = \nu^*(h)$. We define the set of all faces F as the distinct orbits of ν.

- *Edges* E. For any $h \in H$, the orbit $e = \rho^*(h)$ consists of exactly two elements: two opposite halfedges (Fig. 2.3 center). We identify e as an edge and define the set of all edges E as the distinct orbits of ρ.

- *Vertices* V. The combined operation $\nu \circ \rho$ rotates some halfedge $h \in H$ clockwise around its source vertex (Fig. 2.3 right). As before, we therefore identify the orbit $v = (\nu \circ \rho)^*(h)$ as a vertex, and define the set of all vertices V as the distinct orbits of $\nu \circ \rho$.

The halfedges H and the rotation system (ρ, ν) thus fully define the cells of an *abstract cell complex* $K = V \cup E \cup F$ and their connectivity.

Some additional terminology: We associate every cell $\sigma \in K$ with a *dimension* $\dim \sigma$, and refer to a cell with $\dim \sigma = k$ as a *k-cell*: 0 for vertices, 1 for edges, and 2 for faces. A cell σ *contains* a halfedge $h \in H$ if $h \in \sigma$. The *valence* of

a cell σ is the number of its contained halfedges $|\sigma|$. We say that two cells σ_1, σ_2 are *incident* if they contain some common halfedge, i. e. if $\sigma_1 \cap \sigma_2 \neq \emptyset$. If additionally $\dim \sigma_1 < \dim \sigma_2$, we say that σ_1 is *on the boundary* of σ_2 and write $\sigma_1 \prec \sigma_2$. The *source vertex* of a halfedge h is the unique vertex $v \in V$ that contains h, denoted by $\operatorname{src} h$. Its *target vertex* is the source vertex of the opposite halfedge: $\operatorname{tar} h = \operatorname{src} \rho(h)$.

The purely combinatorial structure defined by an abstract cell complex K can be realized geometrically by an *embedding*, a function that maps each k-cell of K to some k-dimensional subregion of an embedding space X. This embedding space can be e. g. the Euclidean space $\mathbb{R}^3$ or some surface $\mathcal{S}$. The following definition ensures that the individual cells are properly glued together to form a continuous embedding.

- For every k-cell σ of K, we choose some k-dimensional parameter domain P_σ (e. g. $[0, 1]^k$) and define a *cell embedding* $M_\sigma : P_\sigma \to X$, a continuous, injective map. We call its image $\operatorname{Im} M_\sigma$ the *embedded cell*.

- Embedded cells can only mutually intersect on their boundaries, never on their interior: For any pair of cells $\sigma_1 \neq \sigma_2$, we require that

$$\operatorname{Im} M_{\sigma_1} \cap \operatorname{Im} M_{\sigma_2} = \partial \operatorname{Im} M_{\sigma_1} \cap \partial \operatorname{Im} M_{\sigma_2}.$$

- The boundary of each embedded cell σ is the union of all embedded cells on its boundary:

$$\operatorname{Im} M_\sigma|_{\partial P_\sigma} = \bigcup_{\sigma' \prec \sigma} \operatorname{Im} M_{\sigma'}.$$

By combining all cell embeddings M_σ to a single, globally injective map, we obtain an embedding of the entire cell complex, denoted by $M : K \to X$.

A common special case of a combinatorial surface is a *triangle mesh*, an abstract cell complex where all faces have valence 3. Triangle meshes typically come with an embedding into $\mathbb{R}^3$ where all cell maps are affine functions. In this case,

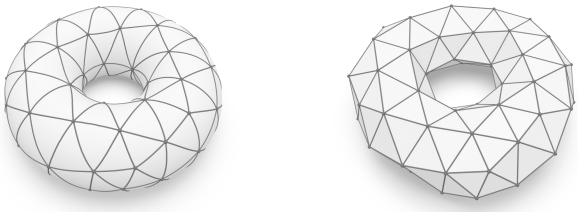


Figure 2.4: *Two geometric realizations of the same abstract cell complex, a triangle mesh. The left embedding into $\mathbb{R}^3$ uses curved elements while the right one is piecewise linear.*

the embeddings of edges and faces are fully defined via linear and barycentric interpolation of the vertex positions (Fig. 2.4 right).

Given some abstract cell complex K with an embedding M into $\mathbb{R}^3$, its image $\operatorname{Im} M$ defines a *geometric realization* of K, which is a surface (in the sense of Section 2.1). Conversely, all surfaces can be represented by a cell complex via triangulation: For any surface $\mathcal{S}$, there exists some cell complex K with an embedding such that its geometric realization is homeomorphic to $\mathcal{S}$ [Lee11, Thm. 5.36].

2.3 Homotopy

Homotopy formalizes a different notion of topological equivalence from a homeomorphism. While a homeomorphism describes a static one-to-one correspondence, homotopy formalizes the intuitive notion that two objects can be transformed into the other through a continuous deformation process without cutting or tearing, which may be sensitive to the topology of the surrounding space. For now, we will focus on the homotopy of curves embedded in a surface, specifically: loops that are anchored at a common base point. Homotopy classes of such loops form the homotopy group, an important topological invariant of a surface.

Let $p \in \mathcal{S}$ denote a fixed *base point* on a surface $\mathcal{S}$. Then, a *loop* is any continuous function $\gamma : [0, 1] \to \mathcal{S}$ describing a parametric curve on $\mathcal{S}$ that starts

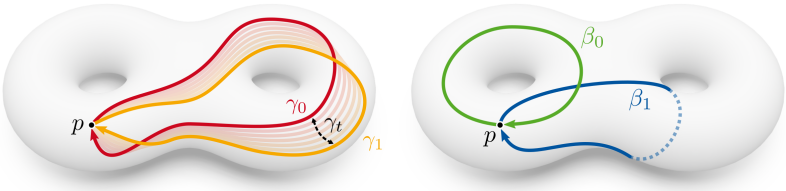


Figure 2.5: *Left: Two loops γ_0 and γ_1 with base point p are connected by a homotopy γ_t, a continuous deformation. Right: No such homotopy exists between the loops β_0 and β_1 as they are separated by topological obstacles. They are non-homotopic.*

and ends in in the base point, i. e. $\gamma(0) = \gamma(1) = p$. A loop γ is called *simple* if it does not self-intersect, i. e. if γ is injective.

We say that two loops γ_0, γ_1 are *homotopic* (and write $\gamma_0 \simeq \gamma_1$) iff there exists a *homotopy*, i. e. a continuous family of loops $\gamma_t : [0, 1] \to \mathcal{S}$ for the range $t \in [0, 1]$ that interpolate between γ_0 and γ_1, again with the requirement that $\gamma_t(0) = \gamma_t(1) = p$ for any $t \in [0, 1]$. This homotopy γ_t can be understood as a deformation of γ_0 into γ_1, which must be continuous in the following sense:

- For any fixed interpolation parameter $t \in [0, 1]$, the curve γ_t must itself be continuous (i. e., γ_t never "breaks apart" at any time during the deformation).
- For any fixed curve parameter $u \in [0, 1]$, the point $\gamma_t(u)$ moves continuously over $\mathcal{S}$ as t varies over $[0, 1]$ (i. e. the function γ_t makes no "sudden jumps" in its deformation process).

This strong notion of continuity makes the homotopy relation sensitive to topological obstacles in $\mathcal{S}$: Homotopies cannot push a loop over handles or holes in $\mathcal{S}$ (or avoid them by temporarily breaking and reconnecting it). Figure 2.5 shows examples of homotopic and non-homotopic loops on a surface.

Homotopy is an equivalence relation on loops [Hat02, Prop. 1.2] and hence defines equivalence classes: Given some loop γ, we write $[\gamma]$ to denote its *homotopy class*, i. e. the set of all loops that are homotopic to γ (examples shown

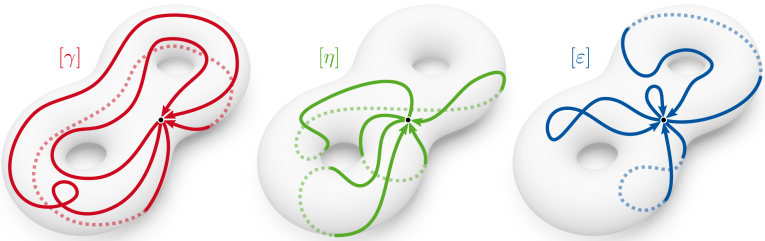


Figure 2.6: *Examples of mutually homotopic loops in different homotopy classes* $[\gamma]$, $[\eta]$, *and* $[\varepsilon]$. *The loops in* $[\varepsilon]$ *are contractible: They can shrink together to the base point.*

in Fig. 2.6). Intuitively, different homotopy classes of loops represent topologically distinct ways in which loops can wind around handles or tunnels of a surface $\mathcal{S}$.

The *constant loop* $\varepsilon : [0, 1] \to \mathcal{S}$ just remains at the base point, $\varepsilon(u) = p$ for all $u \in [0, 1]$. Its homotopy class $[\varepsilon]$ is the set of all *contractible loops*, i. e. those that do not enclose any topological features and hence can be pulled together at the base point (Fig. 2.6 right).

2.3.1 Homotopy Group

The set of loops with a common base point p on a surface $\mathcal{S}$ is closed under the following operations:

- *Concatenation* (Fig. 2.7 left). Given two loops γ, η, we define their concatenation (written as $\gamma \cdot \eta$) as the loop that travels first along γ and then along η, i. e.

$$(\gamma \cdot \eta)(u) = \begin{cases} \gamma(2u) & u \in [0, \frac{1}{2}], \\ \eta(2u - 1) & u \in [\frac{1}{2}, 1]. \end{cases}$$

- *Reversal* (Fig. 2.7 right). For any loop γ, there is a corresponding reverse loop $\overline{\gamma}$ with $\overline{\gamma}(u) = \gamma(1 - u)$.

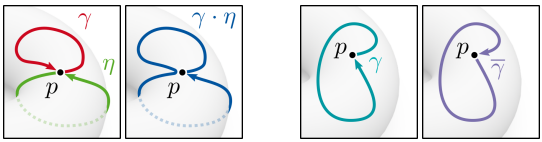


Figure 2.7: *Left: Concatenation of two loops γ and η. Right: Loop reversal.*

These operations are compatible with homotopies between loops in the following sense: Given two pairs of homotopic loops $\gamma_0 \simeq \gamma_1$ and $\eta_0 \simeq \eta_1$, their concatenations are homotopic as well, i. e. $\gamma_0 \cdot \eta_0 \simeq \gamma_1 \cdot \eta_1$. In other words, the homotopy class of a concatenated loop $\gamma \cdot \eta$ only depends on the homotopy classes of γ and η. Analogously, if two loops $\gamma_0 \simeq \gamma_1$ are homotopic, then so are the reversed loops $\overline{\gamma_0} \simeq \overline{\gamma_1}$.

This independence from homotopic variations allows us to abstract away from the particular shapes of loops and instead consider loop concatenation and reversal directly as operations on homotopy classes: We define $[\gamma] \cdot [\eta] = [\gamma \cdot \eta]$ and $\overline{[\gamma]} = [\overline{\gamma}]$. We can now observe that the set of all homotopy classes of loops with base point p on $\mathcal{S}$ forms a group under this concatenation operation [Hat02, Prop. 1.3]:

- Concatenation is *associative*, i. e. $[\alpha] \cdot ([\beta] \cdot [\gamma]) = ([\alpha] \cdot [\beta]) \cdot [\gamma]$. This follows from the observation that $\alpha \cdot (\beta \cdot \gamma) \simeq (\alpha \cdot \beta) \cdot \gamma$, where the homotopy is just a continuous re-parametrization of the same loop.
- The class of contractible loops $[\varepsilon]$ is the *neutral element* w. r. t. concatenation. Concatenating some loop γ with the constant loop ε just extends it by a constant segment which is again homotopic via re-parametrization. Therefore, $[\varepsilon] \cdot [\gamma] = [\gamma] \cdot [\varepsilon] = [\gamma]$.
- For every homotopy class $[\gamma]$, there exists an *inverse element* $\overline{[\gamma]} = [\overline{\gamma}]$, corresponding to the homotopy class with reversed loop orientation. Indeed, concatenating any loop γ with its reversal $\overline{\gamma}$ yields a loop $\gamma \cdot \overline{\gamma}$ that travels back and forth along the same path and is hence contractible: $\gamma \cdot \overline{\gamma} \simeq \varepsilon$ (and analogously, $\overline{\gamma} \cdot \gamma \simeq \varepsilon$). Therefore, $[\gamma] \cdot \overline{[\gamma]} = \overline{[\gamma]} \cdot [\gamma] = [\varepsilon]$.

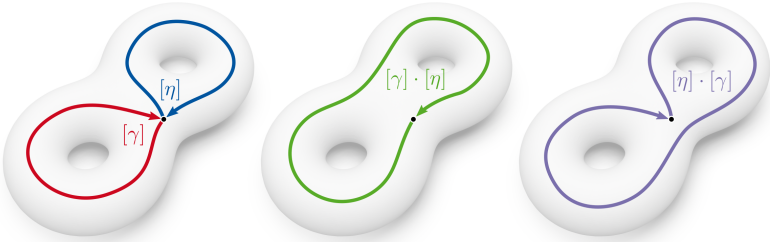


Figure 2.8: *Composition of homotopy classes is non-commutative: The homotopy classes* $[\gamma] \cdot [\eta]$ *(center) and* $[\eta] \cdot [\gamma]$ *(right) are distinct.*

We call this group $\pi_1(\mathcal{S}, p)$, the *homotopy group* of $\mathcal{S}$ at p (other names in the literature are *fundamental group* or *first homotopy group*). Its elements are the different homotopy classes of loops on $\mathcal{S}$ with base point p. The group operation describes the effect of loop composition up to homotopy.

Strictly speaking, different base points $p_0 \neq p_1$ on the same surface $\mathcal{S}$ define different homotopy groups $\pi_1(\mathcal{S}, p_0) \neq \pi_1(\mathcal{S}, p_1)$. This is due to the fact that the homotopy classes in $\pi_1(\mathcal{S}, p_0)$ and $\pi_1(\mathcal{S}, p_1)$ are sets of loops with different base points and therefore necessarily distinct. However, if $\mathcal{S}$ is path-connected, the choice of base point does not affect the purely algebraic structure of the homotopy group: Imagine some curve μ connecting p_0 with p_1 on $\mathcal{S}$, then any loop γ at p_0 corresponds to a loop $\overline{\mu} \cdot \gamma \cdot \mu$ at p_1. As it turns out, this correspondence yields an isomorphism between $\pi_1(\mathcal{S}, p_0)$ and $\pi_1(\mathcal{S}, p_1)$ for any p_0, p_1 [Hat02, Prop. 1.5]. Therefore, we often omit the specification of a particular base point and simply speak of *the* homotopy group of a surface $\pi_1(\mathcal{S})$.

In general, the homotopy group is non-abelian, i. e. its group operation is non-commutative: Consider a double torus $\mathcal{S}_2$ and two homotopy classes $[\gamma]$, $[\eta]$ that surround different handles (Fig. 2.8 left). The concatenations $[\gamma] \cdot [\eta] \neq [\eta] \cdot [\gamma]$ describe different homotopy classes (Fig. 2.8 center and right): They contain loops that wind around the two handles in a different order.

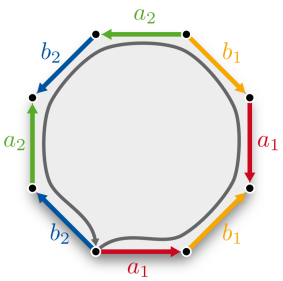
Certain surfaces have a trivial homotopy group, for example the sphere $\mathcal{S}_0$ and the disk $\mathcal{S}_{0,1}$. On those so-called *simply-connected* surfaces, any loop is

contractible, hence there exists only a single homotopy class $[\varepsilon]$ (the contractible loops). Otherwise, all surfaces that we consider have homotopy groups of infinite order: As soon as there is some non-trivial homotopy class $[\gamma] \neq [\varepsilon]$ in $\pi_1(\mathcal{S})$, then immediately all repetitions $[\gamma]^n$ (for any $n \in \mathbb{N}$) are distinct and part of $\pi_1(\mathcal{S})$ as well.

2.3.2 Homotopy Basis

While the homotopy group $\pi_1(\mathcal{S})$ of a surface generally consists of an infinite number of homotopy classes, it is still *finitely generated*. That means, one can find some finite set of loops $S = \{\gamma_1, \ldots, \gamma_n\}$ such that any element of $\pi_1(\mathcal{S})$ corresponds to a loop generated by concatenating items from S (potentially with repetitions or reversals). A minimal generating set S is called a *homotopy basis* of $\mathcal{S}$. For a given basis S, any homotopy class of $\pi_1(\mathcal{S})$ can be encoded by a word (or string) of symbols over the alphabet of basis elements $\gamma_i \in S$ and their reversals $\overline{\gamma_i}$. An exemplary homotopy basis of loops $S = \{\gamma_1, \gamma_2, \gamma_3, \gamma_4\}$ on a double torus $\mathcal{S}_2$ is shown in Fig. 2.9 (top left).

For a general genus g surface, the homotopy basis has exactly $2g$ elements, which may be constructed as follows: Each of the g handles can be associated with two loops: one that circles around it, and one that goes through it. Recall that any closed surface $\mathcal{S}_g$ of genus $g \geq 1$ can be constructed by gluing together a $4g$-sided fundamental polygon (see Section 2.1). The seams of this gluing process then form a *system of loops* $S = \{a_1, b_1, \ldots, a_g, b_g\}$, a set of $2g$ non-intersecting loops on $\mathcal{S}_g$ with a common base point (where all corners of the fundamental polygon are joined together). Such a system of loops S then constitutes a homotopy basis of $\mathcal{S}_g$ [EW05]. Furthermore, this construction reveals an important relation between homotopy classes of $\pi_1(\mathcal{S}_g)$: The sequence



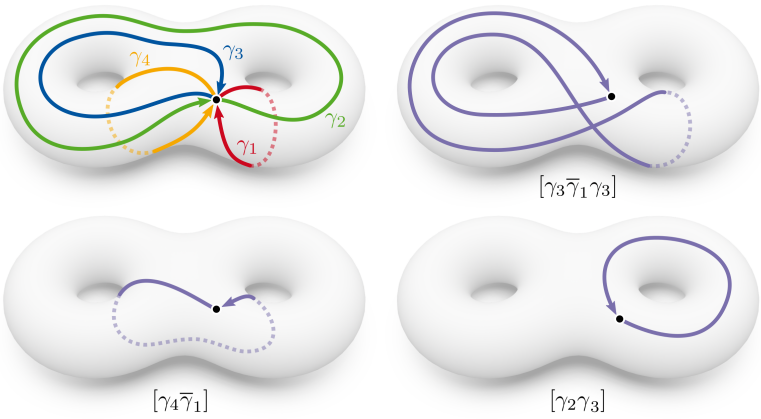


Figure 2.9: *The set of loops $\{\gamma_1, \gamma_2, \gamma_3, \gamma_4\}$ forms a homotopy basis (top left) for the surface $\mathcal{S}_2$. Any homotopy class is uniquely represented by a sequence of these basis elements or their reversals, as exemplified by representative cycles (purple).*

$a_1 b_1 \overline{a}_1 \overline{b}_1 \cdots a_g b_g \overline{a}_g \overline{b}_g$ describes a loop that travels once along the boundary of the fundamental polygon (cf. Fig. 2.2 right). Since the fundamental polygon is simply-connected, this loop is contractible and hence

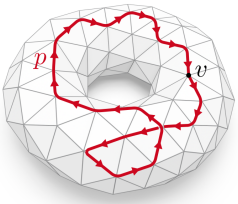
$$[a_1 b_1 \overline{a}_1 \overline{b}_1 \cdots a_g b_g \overline{a}_g \overline{b}_g] = [\varepsilon], \tag{2.1}$$

which is the defining relation for the presentation of the homotopy group $\pi_1(\mathcal{S}_g)$. Note that the edges of a fundamental polygon are only one possible choice for a homotopy basis: Generally, one can find many other sets of loops which generate the homotopy group but do not adhere to the standard gluing pattern (such as Fig. 2.9 top left), or even contain mutual intersections.

2.3.3 Discrete Representation

The previously introduced notions of the homotopy group are defined for general continuous surfaces and loops. We now describe analogous concepts for combinatorial surfaces (Section 2.2) where paths are represented by sequences of edges

in a cell complex and continuous homotopies are replaced by equivalent discrete operations. Similar representations have been described e. g. in [Rot73] or [LM18, Ch. 3.2].



Given an abstract cell complex $K = V \cup E \cup F$ with halfedges H, we choose some fixed *base vertex* $v \in V$. We define an *edge path* p as a connected sequence of halfedges starting at v. It is represented by a word $p = h_0 \cdots h_n$ over the alphabet H with the condition that the first halfedge starts at v ($\operatorname{src} h_0 = v$), and every pair of successive halfedges h_i, h_{i+1} is connected through a common vertex $\operatorname{tar} h_i = \operatorname{src} h_{i+1}$. We define the *target* of a nonempty path p as the target of the last halfedge $\operatorname{tar} p = \operatorname{tar} h_n$. An *edge loop* is an edge path p which returns to the base vertex, i. e. with $\operatorname{tar} p = v$. The empty word ε represents an empty loop with $\operatorname{tar} \varepsilon = v$.

Homotopies between continuous loops are defined via continuous deformations. Their discrete analogon are sequences of *elementary moves*: modifications of an edge loop which are consistent with homotopies on a geometric embedding of the surface. There are two operations:

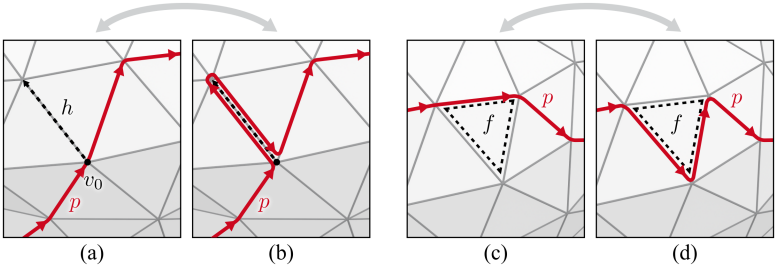


Figure 2.10: *Discrete path homotopies are generated by elementary moves: An edge move (a–b) pulls the path p over an incident halfedge h. A face move (c–d) pushes the path over an incident face f.*

- *Edge move*. This operation extends or shrinks an edge loop p by pulling a piece of it over a single edge (Fig. 2.10 (a–b)). For any decomposition of p into two parts $p = p_0 \cdot p_1$, let $v_0 = \operatorname{tar} p_0$ denote the vertex at which the split occurs. We can now extend p at the split vertex by inserting or removing a back-and-forth movement over any edge incident to v_0: For any halfedge $h \in H$ with $\operatorname{src} h = v_0$, we define an *edge move* as a replacement of $p_0 \cdot p_1$ by $p_0 \cdot h \cdot \rho(h) \cdot p_1$, or vice-versa.

- *Face move*. This operation corresponds to a deformation that pulls a halfedge of a loop p over one of its incident faces (Fig. 2.10 (c–d)). For any halfedge h_0 contained in a face $f \in F$, let $h_1 \cdots h_n$ be the sequence of the other halfedges along the boundary of f. Whenever the element h_0 appears in an edge loop p, we may apply a *face move*, a replacement of h_0 by $\rho(h_n) \cdots \rho(h_1)$, or vice-versa.

Any two edge loops p_1, p_2 are *homotopic* $p_1 \simeq p_2$ if one can be transformed into the other through a finite sequence of elementary moves. This gives rise to the familiar notion of *homotopy classes* of edge loops.

Just as in the continuous setting (Section 2.3.1), the set of edge loops with a fixed base vertex v is closed under the following operations:

- *Concatenation*. Given two edge loops p_1, p_2, the concatenation $p_1 \cdot p_2$ is again an edge loop that first traverses p_1 and then p_2.

- *Reversal*. An edge loop $p = h_0 \cdots h_n$ can be reversed by inverting the order and orientation of the halfedges, i. e. $\overline{p} = \rho(h_n) \cdots \rho(h_0)$.

Again, we can take these operations to act directly on homotopy classes. Under concatenation, the set of homotopy classes of edge loops based at v forms a discrete version of the *homotopy group* $\pi_1(K, v)$, or $\pi_1(K)$ if we neglect the choice of base vertex. Given some embedding M of K defining a geometric realization $\mathcal{S}$, the discrete homotopy group $\pi_1(K)$ of the abstract cell complex is indeed isomorphic to its continuous counterpart $\pi_1(\mathcal{S})$ [LM18, Thm. 3.4.7].

2.3.4 Computing Homotopy Bases

The combinatorial representations from Section 2.2 and Section 2.3.3 enable the algorithmic analysis of loops on surfaces and their homotopy.

One task with significant practical utility is the computation of a homotopy basis (in the sense of Section 2.3.2) for a given combinatorial surface. In this context, a homotopy basis of a cell complex K is a set B of edge loops with common base vertex v that generate the homotopy group $\pi_1(K)$. A general approach is to find B as a system of loops that cuts K to a single polygonal region, i. e. a fundamental polygon bounded by $2g$ edges corresponding to the homotopy generators. One simple way to achieve this is to decimate K down to a minimal cell complex with a single vertex v and a single face by successively contracting edges and merging adjacent faces [LM18, Ch. 3.2]. By marking the remaining $2g$ edges and reversing the decimation process, one obtains a homotopy basis B represented by edge loops in the original complex K.

Other approaches avoid decimation and instead compute loops directly on K. Given some notion of edge lengths (e. g. through a geometric embedding of K), it is especially desirable to find the overall *shortest* system of loops at some base vertex v. While similar optimal surface decomposition problems are known to be computationally difficult, e. g. for shortest cut graphs [EH04], it can be shown that a simple greedy strategy suffices for this task: By successively cutting K along $2g$ locally shortest non-separating edge loops, one indeed obtains a globally shortest system of loops [EW05]. Moreover, these $2g$ cuts can be computed simultaneously using a *tree-cotree decomposition* algorithm [Epp03], as follows: Starting from the base vertex $v \in V$, compute a *primal spanning tree* $T \subseteq E$ of shortest edge paths to all vertices V of K using Dijkstra's algorithm (Fig. 2.11 (a)). Then, every edge $e \in E$ not contained in T closes a cycle in T, corresponding to a shortest edge loop from v that passes through e. It remains to select a set of suitable *bridge edges* whose associated loops in T cut the surface into a single disk. This is achieved by constructing a second *dual spanning tree* T^* over the faces F

without crossing any primal edges of T (Fig. 2.11 (b)). As a result, there will be exactly $2g$ bridge edges which are neither contained in T nor in T^*, corresponding to homotopically different ways to close loops in T (Fig. 2.11 (c)). If T^* is additionally computed as a maximal spanning tree (e. g. using Kruskal's algorithm) with weights chosen according to the length of the associated loops, the result is a globally shortest system of loops at v. It is simultaneously a shortest homotopy basis of the combinatorial surface [EW05]. A similar method for the construction of loop systems is described in [DS95].

While any system of loops cuts K into a single polygon, the sequence and orientation of the $2g$ loops along its boundary will not necessarily correspond to the standard pattern $a_1 b_1 \overline{a}_1 \overline{b}_1 \cdots a_g b_g \overline{a}_g \overline{b}_g$ (as in Eq. (2.1)). Such *canonical* systems of loops, required for certain mapping or homotopy detection tasks, may be generated by more specialized algorithms [VY90; LPVV01].

Note that different edge loops of a shortest homotopy basis never cross, but they may merge and run along the same edges in parallel, especially near the base vertex. If a fully disjoint set of edge loops is desired, it is possible to apply a

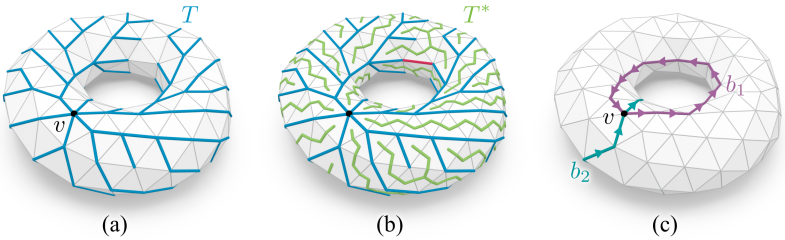


Figure 2.11: *Computing a homotopy basis of a cell complex via tree-cotree decomposition. (a): Primal spanning tree T (blue) rooted at a base vertex v. (b): Dual spanning tree T^* (green) which crosses no edges of T. Edges that are neither on T nor crossed by T^* are classified as bridge edges (red; a second one is occluded behind the torus). (c): Each bridge edge closes a loop b_i in T, which together form a homotopy basis.*

post-process that incrementally pulls loops apart via local refinement of the cell complex [LBG+08; Liv20].

2.4 Homology

As introduced in the previous section, homotopy relates loops on a surface and considers them equivalent if they can be deformed into one another. In the following, we describe a similar relation, *homology*, which is not based on continuous deformation. Instead, it derives from the concept of cycle addition, i. e. the operation of adding or superimposing oriented loops, which is commutative. As we will see, homology defines a coarser relation than homotopy: Certain loops can be distinguished by homotopy but not by homology. The benefit of this somewhat coarser classification is a much simpler algebraic representation: Due to the commutativity of cycle addition, structures from homology are generally isomorphic to vector spaces and can hence be represented and manipulated using standard tools from linear algebra.

In the previous section, it was convenient to first define homotopy on general continuous surfaces (Section 2.3) and then transfer it to the discrete setting (Section 2.3.3). For the following introduction to homology, we will work the other way around and rely on a discrete, combinatorial surface representation right from the start.

2.4.1 Homology Group

Consider a combinatorial surface without boundary, represented by an abstract cell complex K (see Section 2.2). We assume that each cell of K is (arbitrarily) assigned a fixed orientation (i. e. a direction for each edge, and a winding direction for each face).

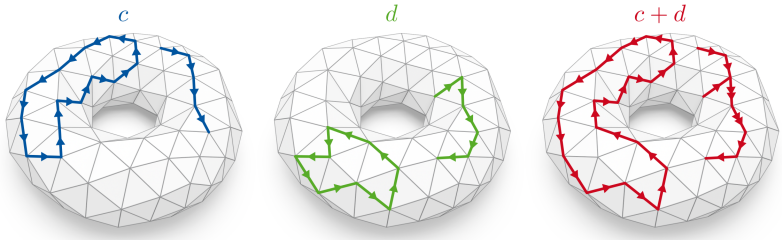


Figure 2.12: *Two* 1*-chains* c *and* d *on a triangular cell complex* K*. Coefficients are indicated by oriented arrows on edges. The sum* $c+d$ *superimposes the two chains by adding their coefficients.*

The basic elements of the following constructions are *chains*: collections of oriented cells of a certain dimension. Formally, a k*-chain* c is a linear combination of k-cells

$$c = \sum_{\sigma_i \in K_k} c_i \sigma_i$$

with coefficients $c_i \in \mathbb{Z}$. Each coefficient c_i indicates how often the cell σ_i occurs in c, and with which orientation: For positive c_i, the orientation of σ_i in c agrees with the predefined orientation of σ_i. For negative c_i, it is the opposite. The magnitude of c_i counts the multiplicity, i. e. the number of occurrences. Cells with coefficient $c_i = 0$ are not part of the chain c.

Chains can be added by simply adding their coefficients: Given two k-chains c and d, their sum $c+d$ is again a k-chain where each cell σ_i has coefficient $c_i + d_i$. Cells with opposite coefficients in c and d cancel (as in Fig. 2.12). A chain c can be reversed as $-c$ by negating all coefficients. We denote the set of all k-chains by C_k. It forms a free abelian group under addition, called the k-th *chain group*.

For a single k-cell σ_i, we define its oriented boundary $\partial_k \sigma_i$ as a chain of $(k-1)$-cells along the boundary of σ_i, with coefficients $+1$ or -1, depending on whether

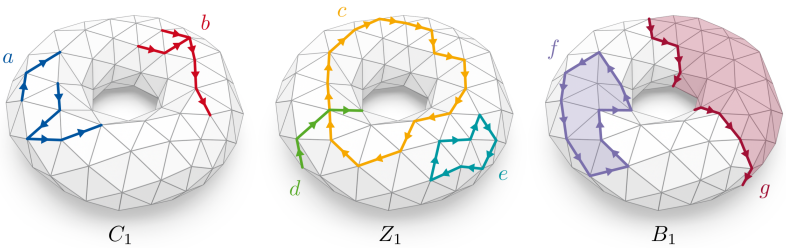
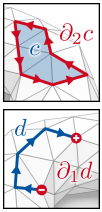


Figure 2.13: *Left:* 1*-chains* ($a, b \in C_1$) *on a cell complex. Center:* 1*-cycles* ($c, d, e \in Z_1$) *are* 1*-chains without boundary i. e. closed chains with no endpoints. Right:* 1*-boundaries* ($f, g \in B_1$) *are* 1*-chains which are themselves boundaries of* 2*-chains (shaded regions).*

their predefined orientations coincide with that of σ_i or not. Based on this, we define the *boundary map* $\partial_k : C_k \to C_{k-1}$ on k-chains as

$$\partial_k(c) = \sum_{\sigma_i \in K_k} c_i \partial_k \sigma_i.$$



We only consider combinatorial surfaces with cells of dimension $k \in \{0, 1, 2\}$. Hence, there are only two relevant boundary maps: For a 2-chain $c \in C_2$ representing a region of the surface as a collection of faces, the boundary map $\partial_2 : C_2 \to C_1$ yields the 1-chain $\partial_2 c$ consisting of edges that run along the contour of the region. For a 1-chain $d \in C_1$, representing a path (or set of paths) as a collection of edges, the boundary map $\partial_1 : C_1 \to C_0$ yields the 0-chain $\partial_1 d$ which is a collection of signed vertices at the endpoints, with negative / positive sign at the source / target points of paths, respectively.

A k-chain c is called a *k-cycle* if its boundary is empty, i. e. $\partial_k c = 0$. We denote the set of all k-cycles by Z_k. It is closed under addition (if $\partial_k c = 0$ and $\partial_k d = 0$, then $\partial_k(c + d) = 0$). We are primarily interested in 1-cycles: Similar to edge loops (as introduced in Section 2.3.3), 1-cycles represent closed chains of edges (e. g. Fig. 2.13 center). However, 1-cycles neither encode a base vertex nor a sequence of traversal, nor are they necessarily connected.

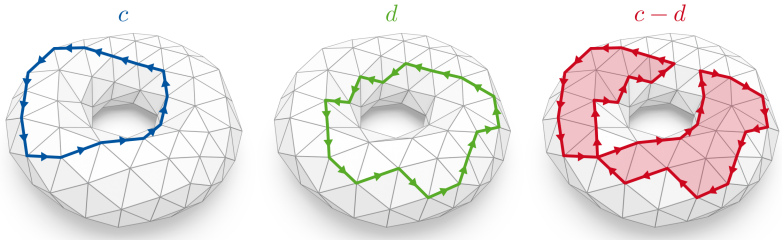


Figure 2.14: *The two* 1*-cycles* c *and* d *are homologous: Their difference* $c - d$ *is a* 1*-boundary (the boundary of the shaded region).*

A k-chain b is called a *k-boundary* if it is the boundary of some $(k+1)$-chain, i. e. if there exists some $c \in C_{k+1}$ such that $\partial_{k+1} c = b$. By this definition, any 1-cycle of edges surrounding a face or a collection of faces of K is a 1-boundary (e. g. f and g in Fig. 2.13). The set of all k-boundaries is denoted by B_k, which is again closed under addition. Every k-boundary is a k-cycle, i. e. $B_k \subseteq Z_k$, but the reverse is generally not true: For example, a 1-cycle that goes around a handle of a torus (e. g. c or d in Fig. 2.13) is not the boundary of any 2-cycle. In fact, homology studies precisely this difference and centers on the question: Which are the cycles that are not boundaries, and how can we classify them?

Homology is thus defined as a relation on cycles that ignores boundary variations. Formally, two k-cycles $c, d \in Z_k$ are *homologous*, $c \sim d$, if they only differ by a k-boundary, i. e. if there exists some $b \in B_k$ such that $c - d = b$. Figure 2.14 shows an example of two homologous 1-cycles. Their difference is a 1-boundary.

Homology is an equivalence relation and hence partitions the set of cycles Z_k into equivalence classes: Each cycle $c \in Z_k$ belongs to a unique homology class $[c] = \{c + b \mid b \in B_k\}$. The homology class $[0]$ of an empty cycle is the class of all boundary cycles B_k.

Cycle addition is compatible with homology, i. e. if $c_0 \sim c_1$ and $d_0 \sim d_1$, then $(c_0 + d_0) \sim (c_1 + d_1)$. Therefore, the operation $[c] + [d] = [c + d]$ is also a well-defined operation on homology classes, with $[0]$ acting as a neutral element. Together with this addition operation, the set of all homology classes of k-

cycles forms a group $H_k(K)$, called the *k-th homology group* of the combinatorial surface K. In the following, we will focus on the *first homology group* $H_1(K)$, which describes homology classes of edge cycles.

2.4.2 Homology vs. Homotopy

Note the conceptual similarities between the homology group $H_1(K)$ and the homotopy group $\pi_1(K)$ (as defined in Section 2.3.3): Both describe equivalence classes of edge collections on a cell complex; cycles in the case of $H_1(K)$ and loops in the case of $\pi_1(K)$. Elements of both groups correspond to topological classifications of loops or cycles according to how they wind around handles and tunnels of the surface. The important distinction lies in the fact that $\pi_1(K)$ considers paths as ordered sequences while $H_1(K)$ ignores order, which is also reflected in the commutativity of the composition operators: Cycle addition commutes ($c + d = d + c$) while loop concatenation does not ($p_1 \cdot p_2 \neq p_2 \cdot p_1$ in general). The commutativity of cycle composition extends to the addition of homology classes. Thus, an element of $H_1(K)$ merely describes *how often* a cycle surrounds certain handles, while an element of $\pi_1(K)$ additionally encodes in what sequence they are traversed.

We can state the relationship between these two groups more explicitly: $H_1(K)$ can be derived from $\pi_1(K)$ through a deliberate linearization of its elements and operations. Any edge loop $p = h_1 \cdots h_n$ may be considered as a cycle by forgetting the order of elements and just taking them as a linear combination of oriented edges $p = h_1 + \cdots + h_n$. In this light, cycle addition can be seen as a linearized version of loop concatenation: Due to the representation of its operands as unordered linear combinations, it has no sense of sequence and is thus commutative. Algebraically, this process is equivalent to an enforcement of commutativity on the operation level: The group $H_1(K)$ is precisely the *abelianization* of $\pi_1(K)$, which is obtained by forcing the group operation to commute [Lee11, Thm. 13.14].

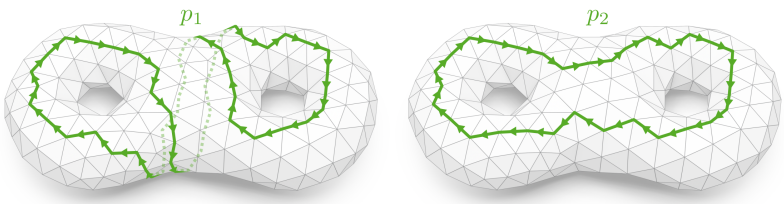


Figure 2.15: *The two edge loops p_1 and p_2 are non-homotopic ($p_1 \not\simeq p_2$): The twist around the waist of the double torus cannot be undone by elementary moves. However, when considered as cycles, p_1 and p_2 are homologous ($p_1 \sim p_2$): The different central segment can be simply removed and replaced through the addition of boundary cycles.*

By considering loops as cycles, we obtain the following relation between homology and homotopy: Any pair of homotopic edge loops $p_1 \simeq p_2$ is also homologous: $p_1 \sim p_2$ [Bre93, Ch. 5, Lem. 3.3]. However, the reverse implication generally does not hold: There are homologous paths that are not homotopic; see Fig. 2.15 for an example. Homology is hence a coarser relation than homotopy.

2.4.3 Homology Basis

The fact that $H_1(K)$ is the abelianization of $\pi_1(K)$ implies that generators of $\pi_1(K)$ are also generators of $H_1(K)$. In other words, any homotopy basis may also serve as a *homology basis*, when considered as a set of cycles. Therefore, the tree-cotree-algorithm described in Section 2.3.4 also suffices for the computation of a homology basis. More generally, one can also choose homology bases that are not homotopy bases, i. e. not necessarily rooted in a single common base point (as e. g. in Fig. 2.17 left).

For a closed combinatorial surface K of genus g, the homology group $H_1(K)$ is a finitely-generated free abelian group [Sti93, Ch. 5.3.3] and thus isomorphic to $\mathbb{Z}^{2g}$. This enables a particularly simple way to represent and manipulate its elements: Let $B_K = \{b_1, \ldots, b_{2g}\}$ be a given homology basis for K. Then, any

homology class $[c] \in H_1(K)$ is uniquely represented by a finite linear combination of those basis cycles, i. e.

$$[c] = \sum_{i=1}^{2g} h_i [b_i]$$

where $h \in \mathbb{Z}^{2g}$ is a vector of integer coefficients. We also use the shorthand $[c] = [B_K h]$ to denote this linear combination. In this encoding, any vector $h \in \mathbb{Z}^{2g}$ represents a homology class. Addition and reversal of homology classes directly correspond to vector addition and negation.

2.4.4 Algebraic Intersection Numbers

Consider two 1-cycles c, d on a cell complex. We define their *algebraic intersection number* $\omega(c, d)$ by counting their oriented transversal intersections: Following c, we count $+1$ whenever c crosses d left-to-right (Fig. 2.16 (a)) and -1 if c crosses d right-to-left (Fig. 2.16 (b)). Where c and d only merge or diverge without crossing, we instead count $+\frac{1}{2}$ or $-\frac{1}{2}$ (Fig. 2.16 (c–d)). The algebraic intersection number is always an integer $\omega(c, d) \in \mathbb{Z}$. It is linear in both arguments (w. r. t. cycle addition) and anti-symmetric, i. e. $\omega(c, d) = -\omega(d, c)$. Moreover, it is invariant w. r. t. homologous arguments [FM11, Ch. 1.2.3]: For pairs of homologous cycles $c_1 \sim c_2$ and $d_1 \sim d_2$, it is $\omega(c_1, d_1) = \omega(c_2, d_2)$. Therefore

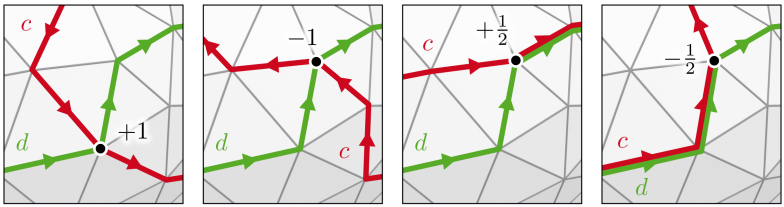
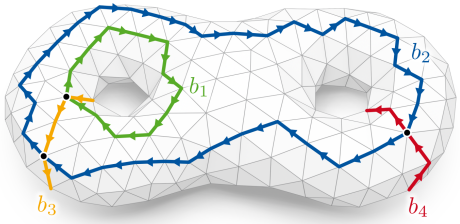


Figure 2.16: *The algebraic intersection number $\omega(c, d)$ counts the number of oriented intersections between a pair of cycles c and d, distinguishing between left-handed (a) and right-handed crossings (b). Merging (c) and diverging (d) paths only count as half.*



$$\begin{array}{c} \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{array} \begin{array}{c} \begin{array}{cccc} b_1 & b_2 & b_3 & b_4 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 \\ -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array} = \Omega_K$$

Figure 2.17: *Left: Homology basis of cycles $B_K = \{b_1, b_2, b_3, b_4\}$. Right: The matrix Ω_K with entries $(\Omega_K)_{ij} = \omega(b_i, b_j)$ encodes the algebraic intersection form of homology classes (represented as linear combinations of B_K).*

$\omega([c],[d]) = \omega(c,d)$ also defines a bilinear form $H_1(K) \times H_1(K) \to \mathbb{Z}$ directly on homology classes.

Choosing a fixed homology basis $B_K = \{b_1, \dots, b_{2g}\}$ for $H_1(K)$ as in Section 2.4.3 allows an explicit representation of the algebraic intersection form ω as an inner product: For two homology classes given by coefficient vectors $[c] = [B_K h_c]$ and $[d] = [B_K h_d]$, one can compute the intersection number as

$$\omega([c],[d]) = \omega([B_K h_c],[B_K h_d]) = \langle h_c, h_d \rangle_{\Omega_K} = h_c^\mathsf{T} \Omega_K h_d \tag{2.2}$$

where $\Omega_K \in \mathbb{Z}^{2g \times 2g}$ with entries $(\Omega_K)_{ij} = \omega(b_i, b_j)$ is the Gram matrix of the intersection form ω (see Fig. 2.17 for an example). The matrix Ω_K is non-singular and anti-symmetric.

2.4.5 Continuous Representation

The previous definition of homology in terms of combinatorial surfaces is also known as *cellular homology*. It only represents discrete chains on a given abstract cell complex K. With an embedding $M : K \to \mathcal{S}$, those chains also obtain geometric realizations: Formally, each chain $c = \sum_i c_i \sigma_i$ is realized as an *embedded chain*, defined as the linear combination of the associated cell embeddings $\sum_i c_i M_{\sigma_i}$ (see Section 2.2). For a 1-chain, this corresponds to a collection of curves $[0,1] \to \mathcal{S}$.

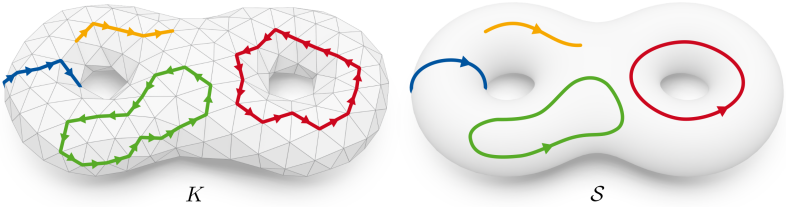


Figure 2.18: *Cellular homology considers chains and cycles defined as linear combinations over elements of a given cell complex K (left). In singular homology, the domain is instead a continuous surface $\mathcal{S}$, where chains and cycles may take arbitrary (in particular smooth) shapes (right).*

A given geometric embedding M of K prescribes a discrete set of possible embedded chains on $\mathcal{S}$. This restriction is lifted by *singular homology* [Hat02, Ch. 2.1] which considers embedded chains as linear combinations over *all* possible cell embeddings. Thus, cycles may take arbitrary continuous shapes on $\mathcal{S}$. In this context, one can define a homology group $H_1(\mathcal{S})$ for general surfaces which is independent of any discrete representation: For any cellular decomposition K of $\mathcal{S}$, the group $H_1(\mathcal{S})$ is isomorphic to $H_1(K)$ and thus inherits all algebraic properties discussed in this section. For the most part, we will therefore directly refer to $H_1(\mathcal{S})$ as the homology group of $\mathcal{S}$, as it does not presume or depend on an underlying cell complex. While singular homology has therefore broader theoretical applications, cellular homology still remains relevant from an implementation perspective as it naturally applies to polygon mesh data structures.

2.5 Cohomology

The theory of cohomology is the dual counterpart to homology. While the objects of homology are cycles, the objects of cohomology are cocycles, i. e. functions that can be *measured* along cycles. In the following, we describe a cohomology group (specifically: the *first de Rham cohomology group*, see e. g. [Lee13, Ch. 17]

for more details), where cocycles are similar to tangent vector fields that can be integrated along curves on the surface.

Homology arises from the notion of a boundary operator on cycles. The corresponding operator in cohomology is the exterior derivative which acts on cocycles. Apart from that, the remaining definitions closely mirror those from homology and most concepts have directly corresponding counterparts.

2.5.1 Cohomology Group

Consider a surface $\mathcal{S}$. A *cochain* is a continuous 1-form on $\mathcal{S}$, given by a function $x : T\mathcal{S} \to \mathbb{R}$, where $T\mathcal{S}$ denotes the tangent bundle of $\mathcal{S}$. For every point $p \in \mathcal{S}$ of the surface, it locally defines a linear map $x_p : T_p\mathcal{S} \to \mathbb{R}$ that measures vectors from the tangent space around p. Cochains can be visualized as tangent vector fields where each vector indicates the local gradient of the 1-form (Fig. 2.19 (a)). We illustrate them as black dashes in our figures. Cochains can be composed via addition and negation, which act pointwise on the local linear maps: Given cochains x and y, their sum $x + y$ at any point $p \in \mathcal{S}$ is just $(x + y)_p = x_p + y_p$, and likewise $(-x)_p = -x_p$.

(a) (b) (c)

Figure 2.19: *A cochain (a) can be visualized as a tangent vector field on a surface. A cocycle (b) is a cochain with zero exterior derivative, corresponding to a curl-free vector field. A coboundary (c) is a cochain that is itself the exterior derivative of some scalar potential (blue).*

A *cocycle* is a cochain x with zero exterior derivative, i. e. $\mathrm{d}x = 0$. Cocycles can be pictured as curl-free vector fields on $\mathcal{S}$ (Fig. 2.19 (b)). Just like cochains, they may be composed and are closed under addition and negation.

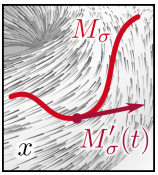
A *coboundary* is any cochain x that is itself the exterior derivative $x = \mathrm{d}\phi$ of some scalar field $\phi : \mathcal{S} \to \mathbb{R}$. Coboundaries correspond to gradient vector fields of scalar potentials (Fig. 2.19 (c)). Any coboundary x is also a cocycle because $\mathrm{d}x = \mathrm{d}\mathrm{d}\phi = 0$, but not vice-versa: Some cocycles are not derivatives of scalar fields and instead flow around handles or tunnels of the surface $\mathcal{S}$.

Cohomology defines the following equivalence relation: Two cocycles x and y on a surface $\mathcal{S}$ are *cohomologous*, $x \sim y$, if their difference $x - y$ is a coboundary, as in Fig. 2.20. This relation partitions the set of all cocycles of a surface $\mathcal{S}$ into equivalence classes: Every cocycle x belongs to a *cohomology class* $[x]$, i. e. a set of mutually cohomologous cocycles. The elements of some cohomology class $[x]$ are all cocycles with the same global flow behavior around handles and tunnels of the surface. Cocycle addition induces a well-defined operation on cohomology classes: $[x]+[y] = [x+y]$. The neutral element $[0]$ is the class of all coboundaries. Under this operation, the set of all cohomology classes of a surface $\mathcal{S}$ forms the *cohomology group* $H^1(\mathcal{S})$, which is a vector space [Bre93, Ch. 5, Def. 2.5].

Figure 2.20: *Two cocycles x, y are cohomologous ($x \sim y$) if and only if their difference $x - y$ is a coboundary, i. e. the exterior derivative of some scalar potential (blue). Despite their local geometric variations, the global flow of x and y around the hole of the torus is the same.*

2.5.2 Connection to Homology

Homology and cohomology can be connected through an operation that pairs cycles with cocycles by taking integrals. Consider a surface $\mathcal{S}$ defined by a cell complex K with a geometric embedding $M : K \to \mathcal{S}$. Cochains on $\mathcal{S}$ can then



be measured along embedded edges of K via integration: Let σ be an edge of K with a smooth embedding $M_\sigma : [0, 1] \to \mathcal{S}$ which defines a tangent vector $M'_\sigma(t)$ for every point $M_\sigma(t)$. We can now measure a cochain x along σ by taking the line integral of x along the curve M_σ, which is

$$\int_\sigma x = \int_0^1 x(M_\sigma(t))(M'_\sigma(t))\,\mathrm{d}t. \tag{2.3}$$

Based on this, we can define integration of cochains along chains, i. e. linear combinations of edges: Given a chain $c = \sum_i c_i\sigma_i$ with an embedding on $\mathcal{S}$, we define

$$\int_c x = \sum_i c_i \int_{\sigma_i} x. \tag{2.4}$$

Note that the above integral is linear both in its integration domain (the chain c) and its integrand (the cochain x), i. e.

$$\int_{c+d} x = \int_c x + \int_d x, \qquad \text{and} \qquad \int_c x + y = \int_c x + \int_c y. \tag{2.5}$$

In the following, we mostly deal with integrals of cocycles along cycles, which have further useful properties:

- *Homology invariance*. Integrals of a cocycle x along a *boundary cycle* b (i. e. with $b = \partial d$ for some chain d) are always zero:

  $$\int_c x = \int_{\partial d} x = \int_d \mathrm{d}x = 0,$$

  which uses Stokes' theorem and the fact that x is a cocycle, i. e. $\mathrm{d}x = 0$. This implies that any two homologous cycles $c_1 \sim c_2$ (which, by definition, only differ by a boundary cycle) produce the same integral $\int_{c_1} x = \int_{c_2} x$

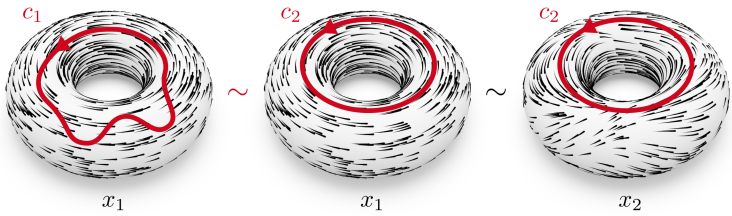


Figure 2.21: *Integration of cocycles along cycles is invariant under homology and cohomology: Homologous cycles $c_1 \sim c_2$ (left and center) have the same integral over the cocycle x_1, i. e. $\int_{c_1} x_1 = \int_{c_2} x_1$. Similarly, the cycle c_2 has the same integral along cohomologous cocycles $x_1 \sim x_2$ (center and right), i. e. $\int_{c_2} x_1 = \int_{c_2} x_2$.*

(cf. Fig. 2.21 left and center). In other words: The value of an integral $\int_c x$ is invariant w. r. t. homologous variations of c. It is identical for any cycle in the same homology class $[c]$.

- *Cohomology invariance*. Similarly, a *coboundary* x (which is $x = \mathrm{d}\phi$ for some cochain ϕ) integrates to zero over any cycle c:

$$\int_c x = \int_c \mathrm{d}\phi = \int_{\partial c} \phi = 0,$$

which again relies on Stokes' theorem and the fact that $\partial c = 0$. Therefore, any two cohomologous cocycles $x_1 \sim x_2$ (which only differ by a coboundary) will have the same integral $\int_c x_1 = \int_c x_2$ (cf. Fig. 2.21 center and right). Thus, the value of an integral $\int_c x$ is identical for all cocycles in the same cohomology class $[x]$.

In light of the above two points, it makes sense to define integration directly as an operation on homology and cohomology classes as $\int_{[c]}[x] = \int_c x$, see also [Lee13, Thm. 18.12]. It enjoys the same bi-linearity properties as in Eq. (2.5).

We say that a cohomology class $[x]$ is *integral* if the value of $\int_{[c]}[x] = n$ is an integer $n \in \mathbb{Z}$ for any homology class $[c]$. $H^1(\mathcal{S})$ remains a group if we restrict to only integral cohomology classes. Furthermore, under this integrality

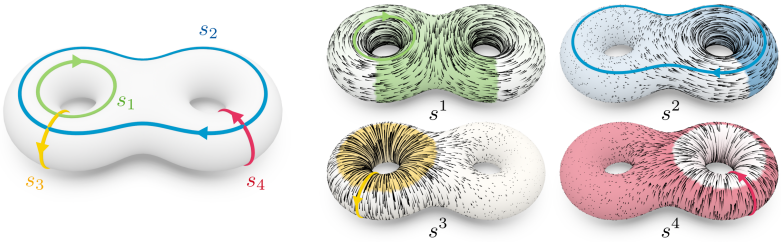


Figure 2.22: *Left: a homology basis $B_\mathcal{S} = \{s_1, s_2, s_3, s_4\}$. Right: a corresponding dual cohomology basis $B^\mathcal{S} = \{s^1, s^2, s^3, s^4\}$ for the same surface, fulfilling $\int_{s_i} s^j = \delta_{ij}$ for any basis cycle s_i and basis cocycle s^j.*

constraint, the cohomology group $H^1(\mathcal{S})$ becomes isomorphic to the homology group $H_1(\mathcal{S})$, which is a consequence of *Poincaré duality* [Hat02, Ch. 3.3].

2.5.3 Cohomology Basis

Just like the homology group $H_1(\mathcal{S})$, the cohomology group $H^1(\mathcal{S})$ thus admits a very simple linear representation of its elements: For a closed genus g surface $\mathcal{S}$, one can again choose a *cohomology basis*, a set $B^\mathcal{S} = \{s^1, \dots s^{2g}\}$ of $2g$ cocycles such that any cohomology class $[x] \in H^1(\mathcal{S})$ is uniquely represented by some linear combination

$$[x] = \sum_{i=1}^{2g} h_i [s^i]$$

where h is a vector of coefficients, with $h \in \mathbb{Z}^{2g}$ if we consider integral cohomology classes. Again, we use $[x] = [B^\mathcal{S} h]$ as a shorthand notation for this linear combination.

We can make the duality of $H_1(\mathcal{S})$ and $H^1(\mathcal{S})$ explicit by choosing their respective homology and cohomology bases $B_\mathcal{S}$ and $B^\mathcal{S}$ in a specific compatible way: Suppose $B_\mathcal{S} = \{s_1, \dots, s_{2g}\}$ is a given homology basis for a surface $\mathcal{S}$.

We can then find a *dual cohomology basis*, a set of cocycles $B^{\mathcal{S}} = \{s^1, \ldots, s^{2g}\}$ such that

$$\int_{s_i} s^j = \delta_{ij} \tag{2.6}$$

for any $s_i \in B_{\mathcal{S}}$ and $s^j \in B^{\mathcal{S}}$, where δ_{ij} is the Kronecker delta. In the resulting pair of bases, any basis cycle s_i integrates to 1 along the corresponding basis cocycle s^i and to 0 along any other cocycle s^j with $i \neq j$ (see Fig. 2.22).

This dual choice of bases greatly simplifies the computation of integrals: Let $B_{\mathcal{S}}$ and $B^{\mathcal{S}}$ be a pair of dual homology and cohomology bases for the same surface $\mathcal{S}$. Then, any homology class $[c]$ and cohomology class $[x]$ may be represented as linear combinations $[c] = [B_{\mathcal{S}} h_c]$ and $[x] = [B^{\mathcal{S}} h_x]$ with integer coefficients $h_c \in \mathbb{Z}^{2g}$ and $h_x \in \mathbb{Z}^{2g}$. Then, due to basis duality (Eq. (2.6)) and the bi-linearity of cycle-cocycle integration (Eq. (2.5)), one can compute the integral of $[x]$ along $[c]$ simply as a standard dot product of the coefficient representations, i. e.

$$\int_{[c]} [x] = \int_{[B_{\mathcal{S}} h_c]} [B^{\mathcal{S}} h_x] = \langle h_c, h_x \rangle. \tag{2.7}$$

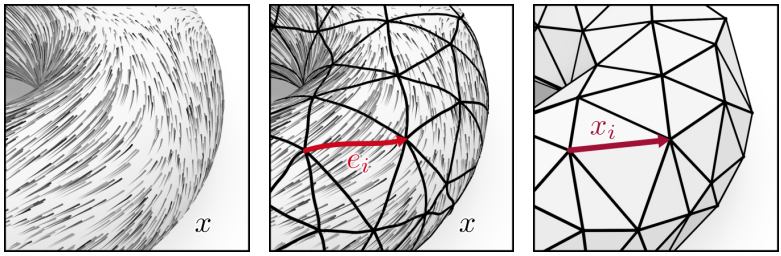


Figure 2.23: *A continuous cochain x on a surface $\mathcal{S}$ (left) is discretized by triangulating the surface and taking integrals $x_i = \int_{e_i} x$ along every edge e_i (center). The discrete cochain is encoded by storing the resulting values $x_i \in \mathbb{R}$ for all edges of the abstract cell complex K of the triangulation (right).*

2.5.4 Discrete Representation

For computations on polygon meshes, we require a discretized representation of cochains and cocycles. A common way to encode cochains on a discrete surface is to take measurements along all mesh edges and only store the resulting integral values, which is the standard representation in discrete exterior calculus [Hir03; CGDS13].

Let x be a cochain on a surface $\mathcal{S}$ given by an abstract cell complex K with an embedding $M : K \to \mathcal{S}$. For each edge $e_i \in E$, we can compute the integral

$$x_i = \int_{e_i} x$$

according to Eq. (2.3). This associates a real value $x_i \in \mathbb{R}$ with every edge e_i, a discretization also known as the *de Rham map* [Hir03, Def. 3.3.1], see Fig. 2.23.

Conversely, any assignment of coefficients $x_i \in \mathbb{R}$ to edges e_i defines a discrete cochain x on K. In this encoding, we can still exactly compute integrals between chains and cochains: Given some chain x over edges of K, the integral of a discretized cochain x, as defined by Eq. (2.4) becomes

$$\int_c x = \sum_i c_i \int_{e_i} x = \sum_i c_i x_i.$$

i. e. just a dot product of the coefficients of c and x.

3 Topology of Surface Maps

Mathematically, a map between two surfaces $\mathcal{A}$ and $\mathcal{B}$ is a function $f : \mathcal{A} \to \mathcal{B}$. Generally, we require that f is a *homeomorphism*, i. e. that f is bijective and both f and its inverse f^{-1} are continuous. In combination, these conditions ensure that homeomorphisms never tear, fold over, or double cover parts of the surface and put all points of $\mathcal{A}$ and $\mathcal{B}$ into a one-to-one correspondence.

As an additional validity criterion, we may require that a homeomorphism $f : \mathcal{A} \to \mathcal{B}$ is *orientation-preserving*, i. e. that the image of $\mathcal{A}$ on $\mathcal{B}$ agrees with the natural orientation of $\mathcal{B}$ (or, more intuitively: preserves the orientation of surface normals). We denote the set of all orientation-preserving homeomorphisms between two surfaces $\mathcal{A}$ and $\mathcal{B}$ by $\mathrm{Homeo}(\mathcal{A}, \mathcal{B})$.

3.1 Surface Map Representations

For geometry processing, where the shapes $\mathcal{A}$ and $\mathcal{B}$ are typically given as polygon meshes (see Section 2.2), various ways to represent surface maps have been proposed. In the following, we briefly review the most important options.

3.1.1 Discrete Non-Homeomorphisms

Most shape correspondence methods employ map representations that rely on some form of sampling or quantization which merely approximate a continuous map.

Sample-based representations encode a map between two surfaces $\mathcal{A}$ and $\mathcal{B}$ as a finite set of corresponding points on the two shapes. It is common to declare

one surface $\mathcal{A}$ as the "source shape" and describe the map in terms of the image of its vertices $V_\mathcal{A}$ on $\mathcal{B}$. Vertex-to-vertex maps [RMC15; VLR+17; LDCK18] assume that those images are also vertices of $\mathcal{B}$, leading to a purely combinatorial map representation in terms of a discrete assignment $V_\mathcal{A} \to V_\mathcal{B}$. Vertex-to-surface maps [EHA+19; ESB19] encode samples on $\mathcal{B}$ more generally as points within faces, e. g. via barycentric coordinates. Other types of sampled maps do not limit their domain to vertices of $\mathcal{A}$ and instead specify an arbitrary number of corresponding point pairs $\{(a_i, b_i)\}_i \subset \mathcal{A} \times \mathcal{B}$ on both surfaces [MCK+17]. This includes maps that only describe a partial matching in certain regions [MGP06; BB08] or sparse landmarks [LF09; EEB20]. By construction, the above representations only encode reliable correspondences at the sample points. Mapping arbitrary points requires some form of extension or interpolation [PBDS13], which may or may not be continuous, depending on the density and quality of the sampling.

Another way to express a correspondence is by describing how to deform mesh $\mathcal{A}$ such that it resembles the surface of $\mathcal{B}$, which is the task of non-rigid registration methods [HAWG08; LSP08; YLSL10]. Such registrations or ambient maps encode either the resulting deformed surface or the deformation itself [ELC19]. Unfortunately, this only brings one surface into proximity of the other. The extraction of a proper mapping $\mathcal{A} \to \mathcal{B}$ typically involves some form of closest-point projection which, depending on the projection operator, may be neither continuous nor injective.

Instead of specifying individual point samples, other approaches represent maps as correspondences between smooth distributions on $\mathcal{A}$ and $\mathcal{B}$. Functional maps [OBS+12] describe surface maps through their action on scalar fields on surfaces. By representing these scalar fields via coefficients in a reduced basis, one obtains a linear encoding of a surface map as a small real-valued matrix. This concept has seen countless applications in different shape correspondence tasks [OCB+17], e. g. the automatic inference, processing and refinement of maps.

A similar representation are soft maps [SNB+12], which explicitly encode map uncertainties and ambiguities as soft correspondences between surface regions.

All of the above options describe maps up to a certain degree of ambiguity: Due to sampling, projection, or fuzzyness, they are partial, non-continuous, or non-injective and therefore not proper homeomorphisms in the mathematical sense. We will revisit such maps in Chapter 5, where we propose a technique that can still robustly extract topological information from non-homeomorphic representations.

3.1.2 Discrete Homeomorphisms

There are other representations that *can* unambiguously describe full homeomorphisms between the surfaces of two triangle meshes $\mathcal{A}$ and $\mathcal{B}$: They are generally defined in terms of a common abstract cell complex K with embeddings $M_\mathcal{A} : K \to \mathcal{A}$ and $M_\mathcal{B} : K \to \mathcal{B}$ into both surfaces (Fig. 3.1). Then, their composition $f = M_\mathcal{B} \circ M_\mathcal{A}^{-1}$ is a homeomorphism $f : \mathcal{A} \to \mathcal{B}$. The cell complex

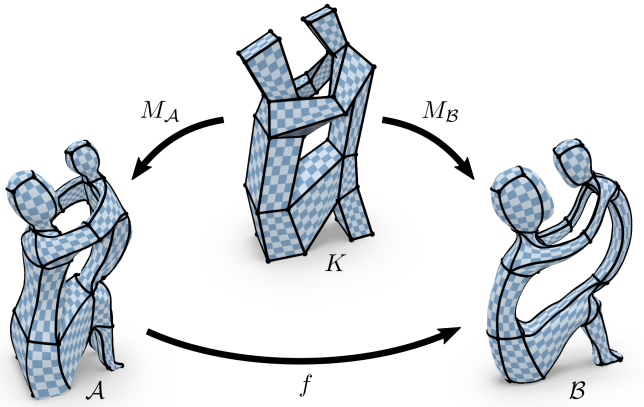


Figure 3.1: *An abstract cell complex K (here: a collection of quadrangular cells) with piecewise linear embeddings $M_\mathcal{A}$ and $M_\mathcal{B}$ into two different triangle meshes $\mathcal{A}$ and $\mathcal{B}$. The composed mapping $f = M_\mathcal{B} \circ M_\mathcal{A}^{-1}$ is a homeomorphism between the two surfaces.*

K may be constructed as a coarse layout [PSS01; KS04] or a single polygonal domain obtained from a cut graph [CJGQ05; APL14; APL15]. Alternatively, one can choose K identical to the connectivity of mesh $\mathcal{A}$ and describe the map only in terms of its embedding on $\mathcal{B}$ [SAPH04; SCBK20]. Individual cell embeddings are then typically defined via piecewise parametrizations onto canonical domains, e. g. (parts of) the plane [PSS01; KSG03; KS04; APL14; APL15] or more general spaces of constant curvature [Ale00; PH03; AL16; AKL17; SCBK20]. Ultimately, a surface map between two polyhedral meshes may be represented in the form of a *meta mesh* where K takes the form of a mutual tessellation or *overlay* of $\mathcal{A}$ and $\mathcal{B}$ with piecewise linear embeddings [Tur92; LDSS99; SAPH04; SBCK19].

The above works propose various approaches to construct the cell complex K and its embeddings, which we review in more detail in Section 4.1. As we will see, this initial construction step is of vital importance for many further processing tasks: It fully defines the topology of the map, which typically can no longer be modified during a subsequent continuous map optimization.

3.1.3 Continuous Homeomorphism Optimization

The generation of high-quality discrete homeomorphisms commonly follows a two-step procedure: First, an initial map f is generated by creating embeddings of a cell complex K into both $\mathcal{A}$ and $\mathcal{B}$, based on a set user-provided landmark constraints. The resulting map is a homeomorphism by construction but may be of poor geometric quality, e. g. introduce excessive stretch. In a second phase, f is thus optimized to reduce distortion while maintaining its status as a homeomorphism. Popular objectives combine a preference fow low mapping distortion (e. g. isometry) with a built-in injectivity-preserving barrier character: Measures such as the symmetric Dirichlet energy [SAPH04] diverge towards infinity whenever f approaches non-injectivity. Under an optimization that continuously stays within the feasible region of this objective [SS15], the map f goes through a continuous deformation towards a low-distortion configuration

while remaining a homeomorphism at any time [JSP17; SBCK19; SCBK20]. Many other approaches, based on distortion objectives without such a barrier character, restrict to continuous homeomorphism-preserving map updates by other means [KS04; APL14; APL15; AL16; AKL17].

While this form of optimization does preserve the initial validity of the map, it comes with a serious limitation: Generally, the space of all homeomorphisms $\mathrm{Homeo}(\mathcal{A}, \mathcal{B})$ consists of different components (corresponding to different map topologies) that are explicitly *not* mutually reachable via continuous homotopic deformations. If the optimal (i. e. minimum-distortion) solution lies in a different component from the initialization, it is therefore completely unreachable, regardless of the optimization trajectory. These components, known as *mapping classes*, are the subject of the following section.

3.2 The Mapping Class Group

Section 2.3 introduced homotopy as a relation between loops, expressing a topological equivalence: Homotopic loops are connected through continuous deformations, which cannot overcome topological obstacles (holes, handles, tunnels) in the surrounding space (i. e. the surface $\mathcal{S}$). This relation can be analogously defined for surface maps.

Consider two surfaces $\mathcal{A}$ and $\mathcal{B}$ and a pair of continuous maps $f_0, f_1 : \mathcal{A} \to \mathcal{B}$. Then, a *homotopy* between f_0 and f_1 is a family of maps $f_t : \mathcal{A} \to \mathcal{B}$ for $t \in [0, 1]$ such that

- for every fixed $t \in [0, 1]$, the map f_t is continuous;
- for every fixed $a \in \mathcal{A}$, the image $f_t(a) \in \mathcal{B}$ varies continuously as t ranges over $[0, 1]$.

Figure 3.2 shows an example. Two maps f_0 and f_1 are called *homotopic*, $f_0 \simeq f_1$, if some homotopy f_t exists. Under this relation, every surface map f belongs to

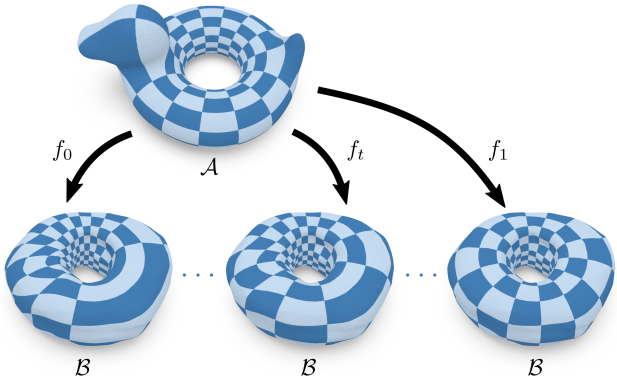


Figure 3.2: *Two maps f_0 (left) and f_1 (right) are homotopic if there exists a continuously varying family of maps f_t that interpolate between them.*

an equivalence class $[f]$ of mutually homotopic maps, known as a *(map) homotopy class*.

A stronger notion of homotopy is that of an *isotopy*. It additionally requires that the homotopy f_t remains bijective throughout the entire deformation. Isotopies therefore characterize precisely the trajectories of continuous homeomorphism-preserving map optimization methods, as described in Section 3.1.3. By definition, any pair of isotopic surface maps is also homotopic. For homeomorphisms, the reverse implication also holds: Whenever two surface homeomorphisms are connected by a homotopy, one can also find an isotopy [Sti93, Ch. 6.2.5]. In this context, the notions of *homotopy class* and *isotopy class* are therefore synonymous.

The different homotopy classes of orientation-preserving homeomorphisms $\mathrm{Homeo}(\mathcal{A}, \mathcal{B})$ are called *mapping classes*. Mapping classes organize the space of all valid surface maps in a way that separates topology from geometry: The space of maps $\mathrm{Homeo}(\mathcal{A}, \mathcal{B})$ decomposes into discrete topological types, each corresponding to one distinct mapping class. In turn, each mapping class contains a continuum of geometrically different maps of the same topology.

Instead of considering two surfaces $\mathcal{A}$ and $\mathcal{B}$, one can study maps from a single surface $\mathcal{S}$ to itself. This perspective gives rise to an algebraic structure that relates all possible mapping classes: We denote by $\mathrm{Homeo}(\mathcal{S})$ the set of all self-homeomorphisms $\mathcal{S} \to \mathcal{S}$ that preserve the orientation of the surface. Elements of $\mathrm{Homeo}(\mathcal{S})$ may be combined via function composition: For any $f, g \in \mathrm{Homeo}(\mathcal{S})$, it is again $f \circ g \in \mathrm{Homeo}(\mathcal{S})$. This composition operation remains well-defined if we consider its action on elements of $\mathrm{Homeo}(\mathcal{S})$ up to homotopy, i. e. on mapping classes: $[f] \circ [g] = [f \circ g]$. Under this operation, the set of all mapping classes of $\mathrm{Homeo}(\mathcal{S})$ forms the *mapping class group* $\mathrm{Mod}(\mathcal{S})$ of the surface $\mathcal{S}$ [FM11, Ch. 2.1].

The mapping class $[id]$ contains all maps homeomorphic to the identity map. It is the neutral element of $\mathrm{Mod}(\mathcal{S})$, as $[id \circ f] = [f \circ id] = [f]$ for any mapping class $[f] \in \mathrm{Mod}(\mathcal{S})$. Simple examples for non-identity elements are mapping classes induced by rotational symmetries of certain surfaces: Figure 3.3 (left) shows different mapping classes of order 2, i. e. with $[r_i]^2 = [id]$. An example of a mapping class of infinite order is a so-called *Dehn twist*, as seen in Fig. 3.3 (right). We will revisit Dehn twists in Section 3.4.2 as they serve as elementary generators of the mapping class group of closed surfaces.

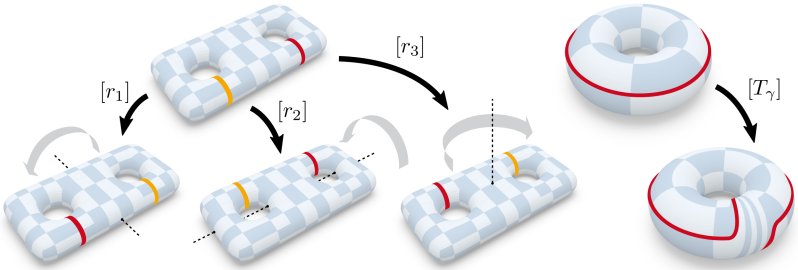


Figure 3.3: *Examples of non-trivial mapping classes. Left: Rotations of a symmetric double-torus are representatives of order-two mapping classes. Right: A Dehn twist around the handle of a torus. Red and yellow loops serve to emphasize the mapping action.*

While mapping class groups are defined in terms of self-maps on a single surface, they can also describe the topology of maps between *two* surfaces $\mathcal{A}$ and $\mathcal{B}$, by a slight extension: By picking some homeomorphism $b : \mathcal{A} \to \mathcal{B}$ as a "base map" between the two shapes, any class $[f] \in \mathrm{Mod}(\mathcal{B})$ of self-maps $\mathcal{B} \to \mathcal{B}$ can be pre-composed with b to define a corresponding class $[f \circ b]$ of maps $\mathcal{A} \to \mathcal{B}$. The mapping class group $\mathrm{Mod}(\mathcal{B})$ thus characterizes the entire topological space of valid homeomorphisms $\mathcal{A} \to \mathcal{B}$, up to the (arbitrary) choice of base map b.

3.3 The Pure Mapping Class Group

For a closed surface $\mathcal{S}_g$ of genus $g \geq 1$, the mapping class group $\mathrm{Mod}(\mathcal{S}_g)$ captures topologically different possibilities of how a self-homeomorphism on $\mathcal{S}_g$ can wrap around surface handles and tunnels. Additional topological degrees of freedom arise when we consider punctured surfaces: Given a closed surface $\mathcal{S}_g$ and a set of n distinct points $P_n = \{p_1, \ldots, p_n\} \subset \mathcal{S}_g$, one obtains the *punctured surface* $\mathcal{S}_{g,n} = \mathcal{S}_g \setminus P_n$ by removing those points from the surface. Note

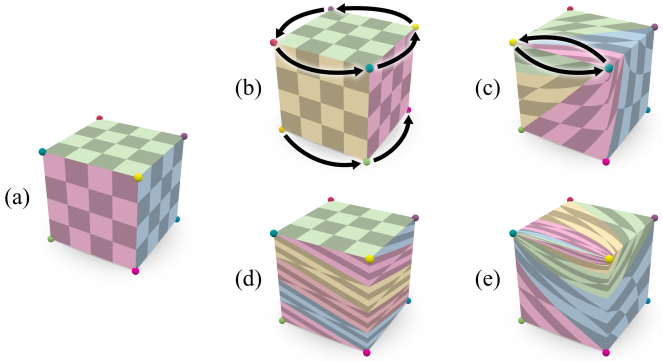


Figure 3.4: *(a): A surface $\mathcal{S}_{0,8}$, with punctures indicated by colored dots. Its mapping class group $\mathrm{Mod}(\mathcal{S}_{0,8})$ also contains maps that permute those punctures (b, c). In contrast, the* pure *mapping class group $\mathrm{PMod}(\mathcal{S}_{0,8})$ only contains maps that leave all punctures fixed and can merely twist around or between them (d, e).*

that self-homeomorphisms on $\mathcal{S}_{g,n}$ may permute these punctures. For example, consider a cube with punctured corners (Fig. 3.4 (a)). Its rotational symmetries are self-homeomorphisms that induce permutations on the punctures (Fig. 3.4 (b)). Since those homeomorphisms are pairwise non-homotopic, they correspond to distinct mapping classes. Generally, the mapping class group $\mathrm{Mod}(\mathcal{S}_{g,n})$ of a punctured surface contains all mapping classes that arbitrarily permute punctures (e. g. Fig. 3.4 (c)).

In some cases, it makes sense to deliberately restrict to self-homeomorphisms on $\mathcal{S}_{g,n}$ that disallow such permutations of punctures. Even under this restriction, there are many different non-isotopic maps which preserve all punctures but still bend and twist around them in various ways (cf. Fig. 3.4 (d, e)). The isotopy classes of this set of maps then form a subgroup of the mapping class group $\mathrm{Mod}(\mathcal{S}_{g,n})$, known as the *pure mapping class group* $\mathrm{PMod}(\mathcal{S}_{g,n})$, which precisely contains those mapping classes of $\mathcal{S}_{g,n}$ that leave all punctures fixed.

The pure mapping class group is of practical interest as it describes the possible topologies of maps under a given set of fixed point constraints (represented by punctures). This becomes relevant for the design of maps between surfaces with landmark correspondences (which we discuss in Chapter 4).

3.4 Mapping Class Representations

Each mapping class describes one possible "topological type" of maps between surfaces. Automatic methods that generate or optimize map topology thus (implicitly or explicitly) operate on some practical representation of mapping classes. In the most basic sense, any encoding of a discrete homeomorphism (Section 3.1.2) already acts as a representative of some mapping class. However, this representation is highly redundant and impractical if one is interested in a purely topological description: Due to its explicit encoding of map geometry, it is difficult to modify

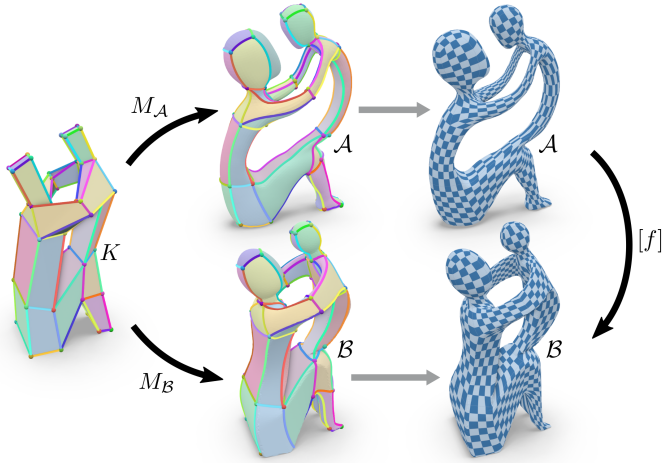


Figure 3.5: *Cellular graph embeddings ($M_{\mathcal{A}}$, $M_{\mathcal{B}}$) map vertices and edges of an abstract cell complex (K, left) into target surfaces ($\mathcal{A}$, $\mathcal{B}$, center), thereby partitioning them into disk patches. These decompositions may be extended to full embeddings via continuous patch-wise parametrizations (right), which uniquely defines a mapping class $[f]$.*

or check for equivalence. In the following, we discuss more concise options for representing mapping classes.

3.4.1 Layout Embeddings

Instead of specifying a full homeomorphism to represent a mapping class, one can reduce the redundant geometric degrees of freedom by omitting certain parts of the map that can be uniquely reconstructed up to homotopy.

Consider an abstract cell complex $K = (V, E, F)$ with an embedding into a surface, defined by $M_{\mathcal{S}} : K \to \mathcal{S}$. By restricting this embedding to the vertices and edges of K, we obtain a *cellular graph embedding*. Its image on $\mathcal{S}$ is a collection of points and curves that cut $\mathcal{S}$ into a set of topological disks. While a cellular graph embedding no longer specifies the geometric embeddings of the

face interiors, these are topologically unambiguous: All homeomorphisms onto a disk with fixed boundary are homotopic [FM11, Lem. 2.1]. Therefore, any full embedding reconstructed by a homeomorphic extension of a given cellular graph embedding has the same map topology.

As described in Section 3.1.2, a homeomorphism $f : \mathcal{A} \to \mathcal{B}$ between two surfaces is often defined as the composition $f = M_{\mathcal{B}} \circ M_{\mathcal{A}}^{-1}$ of two cellular embeddings $M_{\mathcal{A}} : K \to \mathcal{A}$ and $M_{\mathcal{B}} : K \to \mathcal{B}$ from the same abstract cell complex K. By the above argument, restricting $M_{\mathcal{A}}$ and $M_{\mathcal{B}}$ to cellular graph embeddings fully identifies the mapping class $[f]$ without having to specify the entire map geometry (Fig. 3.5). Where $\mathcal{A}$ and $\mathcal{B}$ are given as triangle meshes, the images of the cellular graph embeddings are typically encoded as a collections of triangle edge paths.

Representations of this type are common in practice: Many algorithms initialize a map between two shapes by first computing compatible cellular graph embeddings and then extending these to faces of the cell complex (e. g. via harmonic parametrization). Existing methods differ in the way in which the cell complex K and its graph embeddings into $\mathcal{A}$ and $\mathcal{B}$ are constructed. Commonly, K is chosen to be a very coarse complex consisting of only few cells. In that case, one often refers to such a cell complex as a *layout*, and its maps into $\mathcal{A}$ and $\mathcal{B}$ as *layout embeddings*.

When constructing embeddings of a given layout, a common strategy is to favor embeddings consisting of shortest paths, as they tend to correspond to natural mapping classes. However, many existing methods that follow this approach are based on greedy heuristics and may sporadically produce topologically unintended results, as we will point out in Chapter 5. As a remedy, we propose an embedding algorithm that *systematically* searches for shortest-path layout embeddings, effectively optimizing across different mapping classes.

3.4.2 Dehn Twists

A *Dehn twist* is a self-homeomorphism which twists a part of a surface along a band-shaped region. It can be constructed as follows [FM11, Ch. 3.1.1]: Consider a loop γ without self-intersections on a surface $\mathcal{S}$ (Fig. 3.6 left). By slightly thickening the loop γ, one obtains a *ribbon* $\Gamma \subset \mathcal{S}$, a narrow cyclic strip around γ. This region Γ is homeomorphic to a cylinder C by some map $c : \Gamma \to C$. Now consider a self-homeomorphism $T_C : C \to C$ that twists the cylinder by $360°$ around its central axis (Fig. 3.6 bottom). T_C leaves the two boundaries of C fixed and only affects its interior. By composition with c, one obtains a self-homeomorphism $T_\Gamma : \Gamma \to \Gamma$ that applies a similar $360°$ twist to the ribbon, defined by $T_\Gamma = c^{-1} \circ T_C \circ c$. This map can be extended to act on the entire surface $\mathcal{S}$ by just prescribing the identity for all points beyond Γ. Then, this self-homeomorphism on $\mathcal{S}$ is called a *Dehn twist*. With regard to map topology, the width or precise shape of the ribbon Γ does not matter: In fact, the mapping class of a Dehn twist only depends on the homotopy class of the loop γ. Therefore, such a map is called a *Dehn twist along* γ, denoted by $T_\gamma : \mathcal{S} \to \mathcal{S}$.

A contractible simple loop ε encloses a simply-connected region. Hence, a Dehn twist T_ε just creates a vortex around the enclosed region, which can be undone

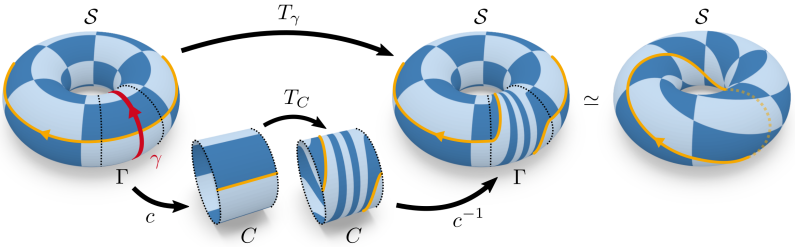


Figure 3.6: *A ribbon-shaped region Γ around a simple loop γ is excised from the surface $\mathcal{S}$. It is homeomorphic to a cylinder C. Twisting C by $360°$ and re-embedding the result into $\mathcal{S}$ yields a map $T_\gamma : \mathcal{S} \to \mathcal{S}$, a so-called Dehn twist, in a non-trivial mapping class (indicated by its action on the yellow loop).*

by a continuous deformation and is thus homotopic to the identity map. Dehn twists along contractible loops therefore correspond to the identity element of the mapping class group $\mathrm{Mod}(\mathcal{S})$. Every Dehn twist T_γ along a non-contractible loop γ has infinite order in $\mathrm{Mod}(\mathcal{S})$: Repeated applications T_γ^n produce distinct mapping classes for any $n \in \mathbb{N}$.

Dehn Twist Relations

If two loops α and β are non-intersecting, a Dehn twist along one loop does not affect the other one. They can thus be applied in any order, i. e. they commute: $T_\alpha \circ T_\beta = T_\beta \circ T_\alpha$. However, this only holds if α and β do not intersect. In general, Dehn twists are non-commutative and their interactions can be quite intricate. Some relations have been stated for very specific configurations of twist loops, e. g. the braid relation for pairs of loops with a single intersection [FM11, Prop. 3.11], or the lantern relation for a particular arrangement of seven loops [FM11, Prop. 5.1].

Generating the Mapping Class Group from Dehn Twists

Dehn Twists are of central importance to the study of mapping class groups as they serve as elementary generators: It is possible to express any mapping class in $\mathrm{Mod}(\mathcal{S})$ as a composition of a finite number of Dehn twists along non-contractible simple loops on $\mathcal{S}$ [FM11, Thm. 4.1]. In other words: Given two mapping classes $[f], [g] \in \mathrm{Mod}(\mathcal{S})$, one can always find some sequence of Dehn twists that transforms $[f]$ into $[g]$. While the previous statement allows Dehn twists along arbitrary non-contractible simple loops, one can find much smaller generating sets of carefully chosen twists that still generate the entire mapping class group. One example are the so-called *Lickorish generators* (Fig. 3.7 (a)), which generate the mapping class group $\mathrm{Mod}(\mathcal{S}_g)$ of a closed genus $g \geq 1$ surface from a set of Dehn twists along $3g - 1$ different loops, arranged around the handles of the surface [Lic64]. For genus $g \geq 2$, this set can be further reduced to twists along

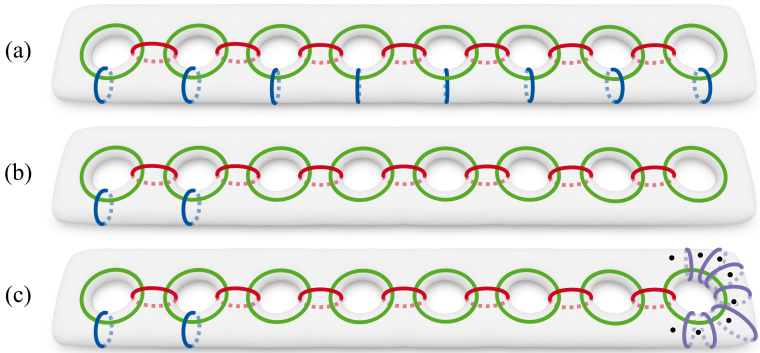


Figure 3.7: *Dehn twists along a finite set of carefully chosen loops, the Lickorish generators (a), suffice to generate the mapping class group. All but two of the blue loops are actually redundant; removing them yields the minimal set of Humphreys generators (b). Punctures (black dots) induce additional generators (purple), which can be lined up around one handle (c).*

$2g + 1$ loops, known as the *Humphries generators* (Fig. 3.7 (b)), which form a minimal Dehn twist basis for $\mathrm{Mod}(\mathcal{S}_g)$ [Hum79].

Dehn Twists and Punctured Surfaces

For closed surfaces, Dehn twists generate the entire mapping class group $\mathrm{Mod}(\mathcal{S}_g)$. The same does not hold for *punctured surfaces*: For a surface $\mathcal{S}_{g,n}$ with $n > 1$ punctures, the group $\mathrm{Mod}(\mathcal{S}_{g,n})$ contains mapping classes that act as permutations on the punctures. However, such permutations are not generated by Dehn twists: Since twist loops cannot pass directly through punctures, Dehn twists may only modify the way a map twists around or between punctures, but they can never exchange the punctures themselves. By keeping all punctures fixed, they only generate a subgroup of $\mathrm{Mod}(\mathcal{S}_{g,n})$, namely the *pure mapping class group* $\mathrm{PMod}(\mathcal{S}_{g,n})$, as introduced in Section 3.3 [FM11, Cor. 4.15].

Analogously to the closed surface case for $g \geq 2$, it is possible to give an explicit, minimal generating set of $2g + n$ Dehn twists for $\mathrm{PMod}(\mathcal{S}_{g,n})$, which

corresponds to an extension of the Humphries generators by $n-1$ additional twist loops that separate the punctures from each other (Fig. 3.7 (c)).

Even on simple domains, the pure mapping class group can become quite complex as the number of punctures increases. A somewhat surprising special case is the sphere with n punctures $\mathcal{S}_{0,n}$. For $n < 4$, the group $\mathrm{PMod}(\mathcal{S}_{0,n})$ remains trivial [FM11, Ch. 2.2]. However, for $n \geq 4$, the group becomes infinite, and the number of Dehn twists required to generate it is at least $\binom{n-1}{2} - 1$, which grows quadratically with n [GW17].

Practical Viability

From an algebraic perspective, compositions over a generating set of Dehn twists provide arguably the most compact and elegant representation of the mapping class group. However, so far this has not been put into practice for surface map applications in geometry processing: The idea of describing a map's topology based on successive modifications of an initial prototype radically differs from existing map generation approaches, which construct maps from scratch through cellular graph embeddings. While the translation of a Dehn twist sequence to a representative homeomorphism is already quite involved, the opposite conversion is even more challenging.

A different, still purely algebraic representation of map topology that is yet closer to practical constructions (i. e. by embeddings of paths and loops) may be derived from the action of the mapping class group on loop homotopy classes.

3.4.3 Action on the Fundamental Group

Instead of $\mathrm{Mod}(\mathcal{S})$, one can also consider the *extended mapping class group* $\mathrm{Mod}^{\pm}(\mathcal{S})$ which additionally includes homeomorphisms that reverse the orientation of $\mathcal{S}$. A simple algebraic characterization of $\mathrm{Mod}^{\pm}(\mathcal{S})$ is given by its action on homotopy classes of loops, i. e. on the fundamental group (Section 2.3.1). Consider a closed surface $\mathcal{S}$ with fundamental group $\pi_1(\mathcal{S})$. An automorphism

of $\pi_1(\mathcal{S})$ is a bijective self-map $\varphi : \pi_1(\mathcal{S}) \to \pi_1(\mathcal{S})$ that preserves the group structure, i. e. $\varphi([\alpha] \cdot [\beta]) = \varphi([\alpha]) \cdot \varphi([\beta])$. Every such automorphism φ is induced by some mapping class $[f_\varphi] \in \mathrm{Mod}^{\pm}(\mathcal{S})$ which describes the same action as φ through a self-map on $\mathcal{S}$, i. e. $\varphi([\gamma]) = [f_\varphi(\gamma)]$. This correspondence is unique up to variations of the base point: Any automorphism φ' that merely modifies the base point (with $\varphi'([\gamma]) = [\overline{\eta}] \cdot \varphi([\gamma]) \cdot [\eta]$ for some loop η) induces the same mapping class as φ. Factoring out this redundancy yields the set of outer automorphisms of $\pi_1(\mathcal{S})$, which corresponds one-to-one with elements of $\mathrm{Mod}^{\pm}(\mathcal{S})$, as stated by the Dehn-Nielsen-Baer theorem [FM11, Thm. 8.1].

In practice, a representation of $\mathrm{Mod}(\mathcal{S})$ in terms of automorphism classes of $\pi_1(\mathcal{S})$ is still somewhat cumbersome: By fixing a homotopy basis $B_\mathcal{S}$ of $\mathcal{S}$, any self-map $\varphi : \pi_1(\mathcal{S}) \to \pi_1(\mathcal{S})$ could be expressed by an assignment from each generator $B_\mathcal{S}$ to a word over the alphabet $B_\mathcal{S}$ (with reversals). However, one must additionally ensure that this mapping is bijective, unique up to conjugacy, and preserves the fundamental group structure and surface orientation, which is no easy task.

One obtains a much more manageable representation by considering the action of the mapping class group on *homology* instead of *homotopy* classes: Unlike $\pi_1(\mathcal{S})$, the homology group $H_1(\mathcal{S})$ is commutative and independent of any base point (Section 2.4.2), leading to a much simpler (although non-unique) encoding of mapping classes, known as the symplectic representation.

3.4.4 The Symplectic Representation

Let $f : \mathcal{A} \to \mathcal{B}$ be a homeomorphism between two surfaces. Consider how f acts on embedded cycles: For any cycle c on $\mathcal{A}$, its image $f(c)$ is again a cycle on $\mathcal{B}$ (see Fig. 3.8). This mapping is invariant under homology: For any homologous pair of cycles $c_1 \sim c_2$, it is $f(c_1) \sim f(c_2)$. Thus, the surface map f

implies a mapping between homology classes, called the *induced homology map* $f_* : H_1(\mathcal{A}) \to H_1(\mathcal{B})$ with

$$f_*([c]) = [f(c)]. \tag{3.1}$$

Indeed, this mapping is well-defined and constitutes a linear isomorphism between the homology groups $H_1(\mathcal{A})$ and $H_1(\mathcal{B})$ [Hat02, Cor. 2.11]. It associates each homology class of $\mathcal{A}$ with a unique corresponding homology class on $\mathcal{B}$.

Due to the linearity of the homology group, any homology map may be encoded as an integer matrix: We can choose homology bases $B_\mathcal{A} = \{a_1, \dots a_{2g}\}$ and $B_\mathcal{B} = \{b_1, \dots b_{2g}\}$ for the two surfaces $\mathcal{A}$ and $\mathcal{B}$, as described in Section 2.4.3. Then, we can use integer coefficient vectors $h_\mathcal{A} \in \mathbb{Z}^{2g}$ and $h_\mathcal{B} \in \mathbb{Z}^{2g}$ to represent homology classes $[B_\mathcal{A} h_\mathcal{A}] \in H_1(\mathcal{A})$ and $[B_\mathcal{B} h_\mathcal{B}] \in H_1(\mathcal{B})$. In this setting, we can express any homology map $f_* : H_1(\mathcal{A}) \to H_1(\mathcal{B})$ explicitly as a map $M : \mathbb{Z}^{2g} \to \mathbb{Z}^{2g}$ between coefficient vectors: For every homology class $[B_\mathcal{A} h_\mathcal{A}]$, mapped to $\mathcal{B}$ as $f_*([B_\mathcal{A} h_\mathcal{A}]) = [B_\mathcal{B} h_\mathcal{B}]$, we have $M(h_\mathcal{A}) = h_\mathcal{B}$, or equivalently

$$f_*([B_\mathcal{A} h_\mathcal{A}]) = [B_\mathcal{B} M(h_\mathcal{A})]. \tag{3.2}$$

Since f_* is linear, so is M, i. e. it is a matrix $M \in \mathbb{Z}^{2g \times 2g}$.

Note that generally, not every integer matrix M represents a homology map that is actually induced by some orientation-preserving homeomorphism $\mathrm{Homeo}(\mathcal{A}, \mathcal{B})$.

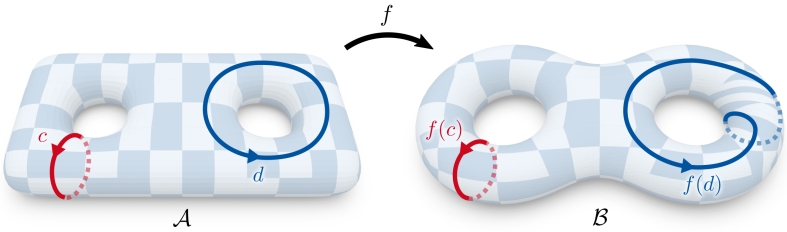


Figure 3.8: *The homeomorphism $f : \mathcal{A} \to \mathcal{B}$ carries cycles (c and d) from one surface to the other. By only considering cycles and their images up to homology, this action defines an induced homology map $f^* : H_1(\mathcal{A}) \to H_1(\mathcal{B})$.*

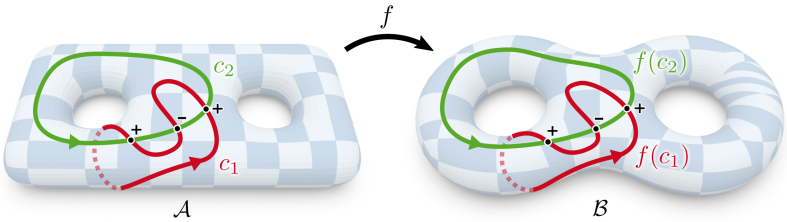


Figure 3.9: *A homeomorphism $f : \mathcal{A} \to \mathcal{B}$ preserves intersections between cycles: If c_1 and c_2 have algebraic intersection number $\omega(c_1, c_2)$ on $\mathcal{A}$, then their images must have the same intersection number on $\mathcal{B}$.*

This more restricted class of maps is characterized by an additional constraint on M, which can be derived from the fact that such maps preserve pairwise intersections of cycles: Suppose two cycles c_1, c_2 have algebraic intersection number $\omega(c_1, c_2)$ on surface $\mathcal{A}$ (as defined in Section 2.4.4). Under a map $f \in \text{Homeo}(\mathcal{A}, \mathcal{B})$, the images $f(c_1)$, $f(c_2)$ must have the same algebraic intersection number on $\mathcal{B}$ (see Fig. 3.9). This also applies to their homology classes:

$$\omega_{\mathcal{A}}([c_1], [c_2]) = \omega_{\mathcal{B}}([f(c_1)], [f(c_2)]).$$

Following Eq. (3.1), f induces a homology map f_*, i.e.

$$\omega_{\mathcal{A}}([c_1], [c_2]) = \omega_{\mathcal{B}}(f_*([c_1]), f_*([c_2])),$$

which may be expressed using coefficients as $[c_1] = [B_{\mathcal{A}} h_1]$, $[c_2] = [B_{\mathcal{A}} h_2]$:

$$\omega_{\mathcal{A}}([B_{\mathcal{A}} h_1], [B_{\mathcal{A}} h_2]) = \omega_{\mathcal{B}}(f_*([B_{\mathcal{A}} h_1]), f_*([B_{\mathcal{A}} h_2])).$$

In this representation, we can replace f_* by M via Eq. (3.2) to get

$$\omega_{\mathcal{A}}([B_{\mathcal{A}} h_1], [B_{\mathcal{A}} h_2]) = \omega_{\mathcal{B}}([B_{\mathcal{B}} M(h_1)], [B_{\mathcal{B}} M(h_2)])$$

and compute intersection forms using Eq. (2.2) as

$$\langle h_1, h_2 \rangle_{\Omega_\mathcal{A}} = \langle M(h_1), M(h_2) \rangle_{\Omega_\mathcal{B}}.$$

Requiring this for all homology classes yields the succinct constraint

$$\Omega_\mathcal{A} = M^\mathsf{T} \Omega_\mathcal{B} M. \tag{3.3}$$

Clearly, this constraint on M is necessary, as any homology map f_* induced by a homeomorphism f must preserve intersection numbers. Furthermore, it can be shown that this property is also sufficient for the existence of a corresponding homeomorphism: For any matrix $M \in \mathbb{Z}^{2g \times 2g}$ fulfilling Eq. (3.3), one can find some map $f : \mathrm{Homeo}(\mathcal{A}, \mathcal{B})$ that induces a homology map f_* represented by M [MP78, Thm. 2; FM11, Ch. 6.3.2].

Applying this matrix representation to self-homeomorphisms $\mathrm{Homeo}(\mathcal{S})$ on a single surface, constraint Eq. (3.3) becomes

$$\Omega_\mathcal{S} = M^\mathsf{T} \Omega_\mathcal{S} M. \tag{3.4}$$

Recalling that Ω_S is non-singular and anti-symmetric (Section 2.4.4), this precisely states that M is a *symplectic matrix*. Any element of the mapping class group $\mathrm{Mod}(\mathcal{S})$ thus induces a homology map represented by some symplectic matrix $M \in \mathrm{Sp}(2g, \mathbb{Z})$. This mapping $\mathrm{Mod}(\mathcal{S}) \to \mathrm{Sp}(2g, \mathbb{Z})$ is therefore known as the *symplectic representation* of the mapping class group [FM11, Ch. 6]. We also refer to Eq. (3.4), and generally to Eq. (3.3) as the *symplectic constraint*.

Note that the symplectic representation $\mathrm{Mod}(\mathcal{S}) \to \mathrm{Sp}(2g, \mathbb{Z})$ is indeed surjective [Put18], i. e. every symplectic matrix corresponds to a homology map induced by some homeomorphism, but generally not injective: For surfaces of genus $g \geq 2$, there may be maps from different mapping classes $[f] \neq [g]$ that induce the same homology map $f_* = g_*$, and thus the same matrix M. This non-uniqueness of the symplectic representation mirrors the relationship between homotopy and homology (Section 2.4.2). By enforcing commutativity,

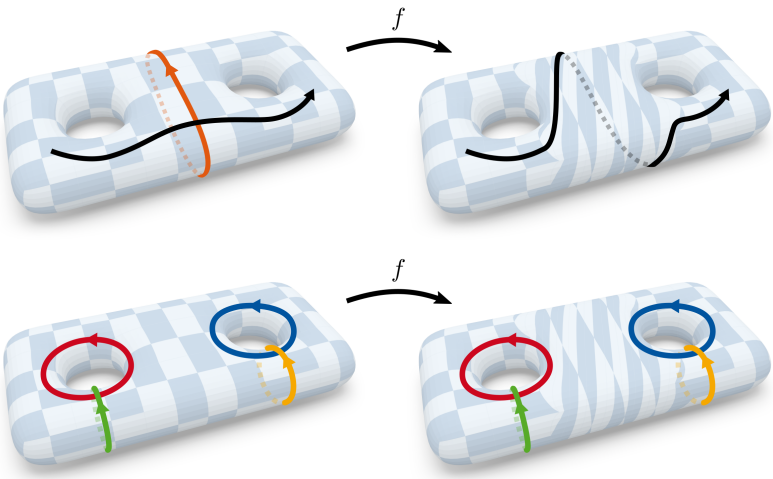


Figure 3.10: *A Dehn twist along a separating cycle (top left, orange) produces a self-homeomorphism f in a non-trivial mapping class (top right, black arrows added to emphasize the twist action). However, the same map f induces the identity mapping on cycles up to homology (bottom row). The mapping class $[f]$ is hence an element of the Torelli group.*

one obtains linearized (and therefore non-unique) approximations of topological representations: Through linearization, loop homotopy classes become homology classes (represented by integer vectors). Consequently, through the symplectic representation, mapping classes become homology maps (represented by integer symplectic matrices).

The topological ambiguities introduced by this linearization can be studied systematically in the form of the so-called *Torelli group* [FM11, Ch. 6.5; HM12]. As a subgroup of $\mathrm{Mod}(\mathcal{S})$, the Torelli group contains precisely the mapping classes that have no effect on homology, i. e. those that induce the identity map in the symplectic representation. Two mapping classes that differ by an element of the Torelli group are therefore indistinguishable in the symplectic representation.

The most prominent examples for Torelli group elements are Dehn twists along simple loops that separate $\mathcal{S}$ into two components (Fig. 3.10).

While the symplectic representation may thus fail to distinguish certain subtle topological details, it provides a concise encoding that is easy to modify, compose and interpret. Meanwhile, it still captures the most important topological aspects of each mapping class: the identification of corresponding handles and tunnels and how often cycles travel around them. In practice, the remaining degrees of freedom (which are precisely the elements of the Torelli group) tend to correspond to more exotic topological variations. For applications that require the reconstruction of a full mapping class when only a homology map is given, these ambiguities can often be resolved successfully by simple greedy decisions (e. g., preferring shortest cut paths where several alternatives exist).

In Chapter 5, we describe a novel method that leverages this symplectic representation to infer purely topological descriptions from (possibly low-quality) input maps. The extracted descriptions, in the form of homology maps, may then be applied e. g. for data transfer or the guided construction of discrete homeomorphisms of a desired topological type.

4 Shortest-Path Layout Embeddings

The methods and results presented in this chapter have previously been published in [BSK21].

We now turn to an algorithmic component that is of fundamental importance for automatic surface map generation: The construction of layout embeddings. Since compositions of embeddings from a common layout define a surface homeomorphism (Section 3.4.1), they are typically used to provide an initial map topology and geometry. As further processing of the map often no longer allows topological changes (Section 3.1.3), it is vital to find the correct map topology right from the start. Preferably, automatic layout embedding methods should therefore generate "topologically natural" embeddings.

This notion of a "natural embedding topology" is not easily formalized. Still, a useful proxy measure for embedding quality can be derived from the lengths of embedded layout edges: Embeddings that are overall short and simple tend to avoid topological artifacts such as global twists.

This principle suggests a simple incremental layout embedding strategy, which is followed by many existing methods: Given a layout with prescribed vertex positions on a target surface (Fig. 4.1 top left and top right), the layout edges are embedded one-by-one as shortest paths between the given layout vertex positions. In many cases, this simple strategy suffices to generate simple, topologically plausible embeddings. However, it requires careful planning: As embedded edges may block each others' paths, a wrong order of insertions may force later edge embeddings into accidental detours. This can occasionally result in surprisingly bad embeddings of highly unnatural topology. As previous methods carry out

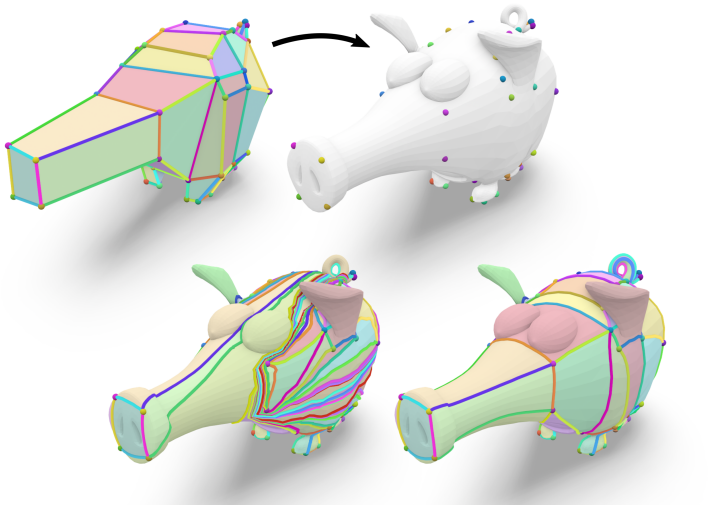


Figure 4.1: *Top left: layout. Top right: target surface with prescribed layout vertex positions. Bottom left: layout embedding generated by a greedy sequence of edge insertions. Bottom right: optimized layout embedding.*

these insertions in a greedy fashion without any backtracking, they are susceptible to such sporadic failures in practice (as in Fig. 4.1 bottom left). Our contribution is a new layout embedding algorithm that systematically prevents such errors and reliably finds short and simple embeddings (Fig. 4.1 bottom right). Effectively, our proposed method optimizes across different insertion sequences of edges which implicitly represent potentially different embedding topologies.

4.1 Layout Embeddings

In the following, we consider a *layout* $L = (V, E, F)$, i. e. a coarse abstract cell complex (Section 2.2). A *layout embedding* $M : L \to \mathcal{S}$ defines a continuous, bijective, and orientation-preserving map from the cells of L into a surface $\mathcal{S}$.

Such layout embeddings are ubiquitous in geometry processing. In the following, we briefly review their practical uses (Section 4.1.1), topological complexity (Section 4.1.2), and construction methods (Section 4.1.3).

4.1.1 Applications

We distinguish between applications that construct embeddings from one layout into multiple surfaces (typically for surface map generation), and single-surface geometry processing tasks.

Maps between Surfaces

Recall from Section 3.4.1 that we may describe a homeomorphism between two surfaces $f : \mathcal{A} \to \mathcal{B}$ as a composition of two layout embeddings $M_{\mathcal{A}} : L \to \mathcal{A}$ and $M_{\mathcal{B}} : L \to \mathcal{B}$ as $f = M_{\mathcal{B}} \circ M_{\mathcal{A}}^{-1}$.

For many design tasks, the input surfaces come with *landmark annotations*, i. e. points on the two surfaces that encode semantically corresponding locations. Given landmark annotations $a_1, \ldots, a_n$ on $\mathcal{A}$ and $b_1, \ldots, b_n$ on $\mathcal{B}$, the map f should identify these points, i. e. $f(a_i) = b_i$. A simple way to enforce this is to take a layout L whose vertices V correspond one-to-one with landmarks and to prescribe their embedding images, i. e. $M_{\mathcal{A}}(v_i) = a_i$ and $M_{\mathcal{B}}(v_i) = b_i$ for all $v_i \in V$. Under this condition, any composed map $f = M_{\mathcal{B}} \circ M_{\mathcal{A}}^{-1}$ is a homeomorphism that indeed respects the given landmark constraints.

Thus, a homeomorphism f may be generated by constructing an appropriate layout L and two landmark-constrained embeddings $M_{\mathcal{A}}$ and $M_{\mathcal{B}}$. Typically, the goal is to find a natural mapping, e. g. one that minimizes overall map distortion under the given constraints. A proper solution would entail a joint optimization of both map geometry and topology induced by the two embeddings $M_{\mathcal{A}}$ and $M_{\mathcal{B}}$. However, no such optimization method for homeomorphisms is currently known. In practice, it is therefore replaced by the comparatively much simpler task of constructing $M_{\mathcal{A}}$ and $M_{\mathcal{B}}$ as short, natural layout embeddings on $\mathcal{A}$ and $\mathcal{B}$

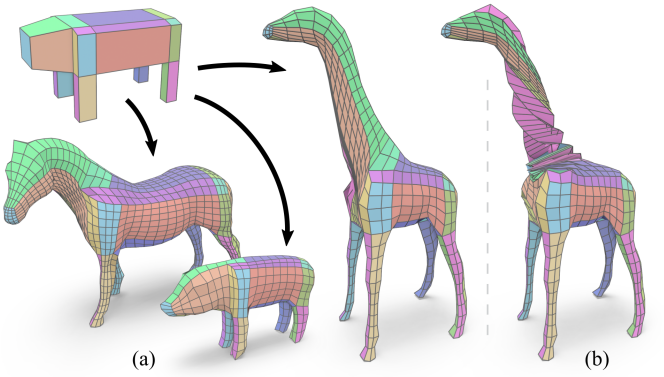


Figure 4.2: *One application scenario which relies on robust layout embedding is quad meshing with prescribed base complex. Results heavily depend on finding embeddings in the correct mapping class (a), thus avoiding unintended twists (b).*

individually. This follows the tacit assumption that a composition $f = M_{\mathcal{B}} \circ M_{\mathcal{A}}^{-1}$ of two natural (i. e. short) embeddings is also likely a natural (i. e. low-distortion) map.

Single-Surface Embeddings

The prototypical application for layout embeddings on a single surface are *chart-based parametrizations* [LSS+98; KLS03], i. e. global, continuous parametrizations onto a collection of domains that form a coarse abstract cell complex. They have been used e. g. for texturing [KSG03], texture atlas generation [PCK04] or geometry encoding [CHCH06].

Layout embeddings may also be used to define a cellular *segmentation* of a target surface. In this setting, the layout cells may be annotated with semantic information of a general structure (e. g. part labels of a design template or skeleton), which is then carried over to particular instances.

For *re-meshing* applications, a layout may define a desired mesh connectivity which is then imposed onto a given target surface via an embedding. This approach

is of particular relevance for surface quadrangulation where a coarse quad layout defines the *base complex* of a highly regular quad mesh, as illustrated in Fig. 4.2. For certain meshing tasks, it can be desirable to prescribe a (hand-crafted or synthesized) layout, allowing to explicitly model structural features (such as symmetries or repetitions) or standardized functional components in a controlled way. Numerous methods for quad layout generation [THCM04; DBG+06; CBK12; BCE+13; RRP15; ULP+15; PPM+16; SBLS18] and embedding [CK14b] have been described (see [Cam17] for an extensive survey). Besides those fully-automatic methods, there are interactive tools that guide a user in the design of (quad) layouts and their embeddings [TPSS13; CK14a; MTP+15].

For *polynomial surface fitting*, a layout can encode the control structure of a spline or subdivision surface. The approximation is then driven by an embedding that defines a direct correspondence between the parameter domain and the target surface [EH96; MK05; LRL06].

4.1.2 Topological Degrees of Freedom

Generally, different embeddings of the same layout L onto a surface $\mathcal{S}$ may be topologically distinct, i. e. non-homotopic. Since layout embeddings are

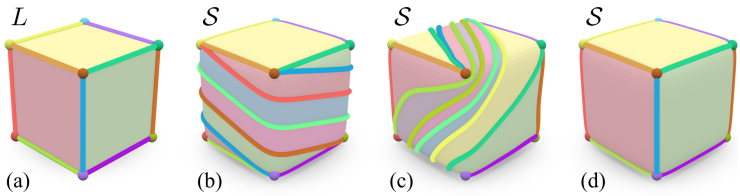


Figure 4.3: *When landmark positions are fixed, there are still infinitely many topologically distinct ways to embed a layout (a) into a target surface (b, c, d). Embeddings of shortest total path length (d) often correspond intuitively to the correct embedding topology.*

orientation-preserving homeomorphisms, we may identify all possible embedding topologies with elements of the mapping class group $\mathrm{Mod}(\mathcal{S})$ (Section 3.2).

In this chapter, we consider more specifically embeddings with fixed, prescribed positions of the layout vertices (e. g. for landmark-constrained map generation). Note that this specification of fixed points does generally *not* reduce the topological degrees of freedom that need to be resolved during embedding construction. In fact, their presence typically *increases* the topological complexity: Even for surfaces of trivial topology (e. g. spheres), the addition of landmarks creates obstacles that allow embeddings to twist and tangle around them in homotopically different ways (Section 3.4.2), as shown in Fig. 4.3. Algebraically, these degrees of freedom correspond to the mapping classes of a surface with fixed punctures, which is characterized precisely by the pure mapping class group $\mathrm{PMod}(\mathcal{S})$ (Section 3.3).

4.1.3 Incremental Construction

Instead of dealing with the complexity of the pure mapping class group explicitly, many practical methods instead follow a heuristic approach that constructs layout embeddings incrementally, by successively computing embeddings for individual layout edges as shortest paths.

The problem setup is described by the following input: A layout $L = (V, E, F)$, a (topologically compatible) target surface $\mathcal{S}$, and a set of landmarks, i. e. prescribed embedding positions on $\mathcal{S}$ for all layout vertices V. The desired result is an embedding $M : L \to \mathcal{S}$ that respects the given landmark constraints.

Cellular Graph Embeddings

An embedding $M : L \to \mathcal{S}$ defines a full homeomorphism between layout and target surface. In particular, it describes continuous parametrizations for all embedded edges and faces. However, during initial construction, this representation is often simplified in two important aspects:

- *Non-Parametric Cell Embeddings.* Instead of giving a full parametric description of edge and face embeddings, one first computes only their *images*, i. e. curves and regions on $\mathcal{S}$. The computation of edge and face parametrizations is postponed to a later algorithm stage: After all cells have been embedded, one can compute prototypical unit-speed or harmonic parametrizations for layout edges and faces, respectively.
- *Cellular Graph Embeddings.* This representation is further simplified by entirely omitting face embeddings and only specifying the images of layout vertices and edges, i. e. a cellular graph embedding of the vertex-edge-graph (Section 3.4.1). Face embeddings can be recovered later: A valid cellular graph embedding cuts $\mathcal{S}$ into a collection of disks corresponding to faces of L (which also fixes all topological degrees of freedom). Such an embedding is valid if all embedded edges are globally non-intersecting and maintain a correct cyclic ordering around layout vertices, according to L's rotation system (Section 2.2).

By postponing the computation of face embeddings and parametrizations, the core of the layout embedding process thus boils down to the computation of edge embeddings.

Edge Embeddings as Shortest Paths

While the true objective is to find a natural embedding with low parametric distortion, this property is difficult to predict during construction as cellular graph embeddings only specify edge images. However, one geometric property that *can* be measured directly is the total length of embedded edges on $\mathcal{S}$, which is often taken as a proxy measure for geometric quality: This is based on the observation that unnatural embedding topologies induce global twists around handles or punctures that force embedded paths to take excessive detours. Thus, layout embeddings consisting of overall short paths tend to correspond to natural mapping classes.

Following this intuition, many algorithms (first described by [PSS01]) construct layout embeddings *incrementally*, by successively computing individual edge embeddings as shortest paths on $\mathcal{S}$. To ensure validity, paths are constrained to avoid intersections and to approach their target vertices from the correct *sectors*, i. e. in a correct cyclic ordering with other paths.

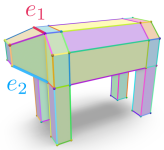
In a discrete setting where $\mathcal{S}$ is given as a triangle mesh, such paths may be computed as constrained shortest geodesics, e. g. via window propagation [MMP87; SSK+05] or fast marching [Set96; KS98]. In practice, many approaches avoid exact geodesic computations and instead compute approximate shortest paths in the triangulation of $\mathcal{S}$ via Dijkstra's algorithm [PSS01; KSG03; KS04; SAPH04; APL15]. This often requires some special handling to accommodate paths in mesh regions that are not sufficiently tessellated, either by (temporarily) allowing parallel paths [PSS01] or through adaptive local mesh refinement [KSG03; SAPH04; Liv20].

Shortest Path Metrics

Most methods compute shortest geodesics in the usual intrinsic metric of the surface [KSG03; KS04; SAPH04; APL14]. Others deliberately modify this metric in an effort to promote more natural path shapes (and homotopies): Examples are geodesics w. r. t. flat metrics obtained via planar parametrizations [APL15], or harmonic potentials that repulse paths from layout vertices [PSS01].

Ordering Heuristics

As each shortest path computation is constrained to avoid intersections with previously embedded edges, the order of insertions can potentially have a dramatic effect on the final outcome. Consider Fig. 4.4 where the layout from the inset on the left is embedded into a target surface using two different insertion sequences. In the top row, layout edge e_1 (red) is embedded before e_2 (blue). By itself, the



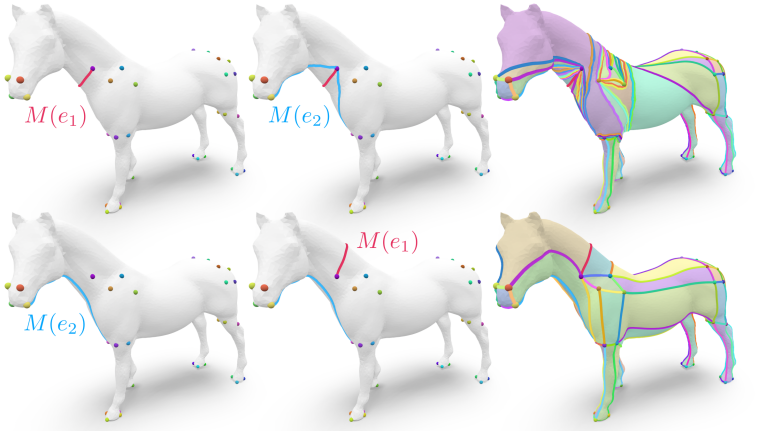


Figure 4.4: *Edges of a layout are embedded one-by-one into a target surface. Embedding edge e_1 before e_2 (first row) forces paths into excessive detours, creating a severely entangled embedding. An insertion sequence starting with a different ordering (second row) yields a short, topologically natural result.*

shortest embedding of e_1 is routed below instead of above the horse's neck, which forces the subsequent embedding of e_2 to take a detour. These detours accumulate in the following insertions, creating a topologically unexpected embedding with excessively long and entangled paths (top right). If instead e_2 is inserted before e_1 (as in the bottom row), the embedding of e_1 assumes its intended homotopy class, as do all subsequent layout edges. The resulting embedding (bottom right) is topologically natural and overall much shorter.

Aware of this issue, previous works have proposed different heuristic strategies to select a favorable insertion order of layout edges: Most methods apply a simple greedy scheme and always insert the edge that currently allows the shortest embedding [KSG03; KS04]. Additionally, [PSS01] suggest a "swirl detection" heuristic that postpones the insertion of edges if their embedding would pass by layout vertices on the wrong side. In a similar effort, [SAPH04] identify a

set of "extremal" layout vertices (based on mutual distance) and initially prefer connections between those.

A careless sequence of edge insertions may not only degrade embedding quality, it may even lead into dead-end situations where no valid completion is possible: Groups of embedded edges may form closed cycles that separate $\mathcal{S}$ into different components. This may accidentally separate the endpoints of a (not yet inserted) layout edge and thus prevent its embedding forever. Some methods such as [KSG03; KS04] counteract this problem by explicitly detecting and disallowing such separating edge insertions (e. g. through a flood fill check). Others proceed more conservatively by initially preventing the insertion of edges that form separating cycles until the embedding effectively cuts $\mathcal{S}$ into a disk, at which point all other edge insertions become topologically trivial [PSS01; SAPH04].

Prescribed vs. Generated Layouts

Many layout embedding methods assume that a prescribed layout L is given as input [MGR00; PSS01; CK14a], which may be either hand-crafted or previously automatically generated. A common variation of this task presupposes that only a set of layout vertices V and their target positions are provided. In this setting, the remaining connectivity of L is to be constructed simultaneously with its embedding into $\mathcal{S}$. This setup is especially popular for (landmark-constrained) map initialization applications where the structure of the layout itself is not of particular importance beyond its purpose as a parameter domain.

Layout connectivity generation may be integrated with the incremental layout embedding process: At each step, shortest paths between *all* unconnected layout vertices are considered as edge embedding candidates [KS04; SAPH04; APL14; APL15]. One of these paths is selected (typically the shortest one), which simultaneously defines a new edge in L and its embedding. Insertions are repeated until the embedded paths cut $\mathcal{S}$ into a disk (yielding a single-face layout L),

which may then be further refined, e. g. via triangulation. For map generation, this process may be performed on two target surfaces $\mathcal{A}$ and $\mathcal{B}$ *simultaneously*, producing a common layout L with two embeddings $M_{\mathcal{A}}$ and $M_{\mathcal{B}}$. Insertion decisions must then jointly consider the quality of both embeddings, e. g. by measuring the combined lengths of paths on both meshes.

For single-surface embeddings without a prescribed layout connectivity, there are alternative approaches beyond incremental shortest-path constructions: One idea is to generate an embedding of a single-face layout by growing a disk-shaped region on the surface [GGH02]. Another is to obtain a layout connectivity and its embedding by progressively decimating the target mesh until only layout vertices remain [LDSS99; PH03].

Post-Processing

The primary objective of the above methods is to initialize a valid and topologically natural embedding. Geometric quality is often only a secondary concern: Resulting embeddings may come with artifacts, e. g. jagged paths from discrete edge-based shortest-path computations, or uneven spacing in crowded areas. Various methods to optimize embedding geometry (in a homotopy-preserving way) have been proposed, which can be applied as a post-process: Path-based methods include active models that continuously evolve an explicit curve representation [LL02; BWK05; YSC21] or discrete methods for finding shortest homotopic geodesics [HS94; SC20]. Other methods consider the interior of patches and optimize embeddings based on mapping distortion [KS04; SAPH04; SCBK20] or curvature alignment [CK14b]. Another simple approach is re-tracing paths as straight lines in local planar parametrizations of cellular neighborhoods [PSS01].

4.1.4 Discussion

The construction of layout embeddings via incremental shortest-path insertions is simple and pragmatic. However, all previous works following this approach

rely on greedy strategies for picking an insertion sequence, basing their decisions on local path lengths and various best-first heuristics. Crucially, these methods allow no backtracking, which makes the reliance on heuristics especially risky: When heuristics fail, wrong choices in earlier stages can lead to spectacularly bad embeddings later on, exhibiting paths in unexpected homotopy classes (and of excessive length) that swirl around remote regions of the target surface (cf. Fig. 4.1).

Downstream applications that use embeddings for further processing (Section 4.1.1) can typically not recover from initializations in a wrong mapping class (Section 3.1.3). Due to their weak reliability, greedy methods for layout embedding therefore require human supervision and intervention, making them poorly suited for automatic shape processing pipelines.

4.2 Contribution

As a remedy to the issues raised above in Section 4.1.4, we suggest an alternative layout embedding method that goes beyond the greedy paradigm and therefore robustly avoids such sporadic failures.

In the remainder of this chapter, we present an algorithm that systematically searches for optimal insertion sequences yielding shortest-path embeddings of minimal total length. Our search is powered by a custom *branch-and-bound* strategy, crawling the decision tree that governs the incremental construction of embeddings. We demonstrate that—despite the vast search space—it is possible to quickly and reliably find low-cost solutions by exploiting domain knowledge in the form of bounding and pruning rules and a specialized search priority. For an additional speedup, we can safely incorporate previous greedy strategies and heuristics into this system by using their results as initial solutions, which are then improved or confirmed by our algorithm.

In particular, we make the following contributions:

- a method for embedding a given layout into a simply-connected target surface (with prescribed vertex positions) while explicitly optimizing for minimum embedding path length,
- a novel combinatorial formulation of this problem in terms of edge insertion sequences,
- a custom branch-and-bound strategy, tailored to this task.

In Section 4.3, we more precisely define the domain of our optimization: the class of *shortest-path embeddings* and its combinatorial encoding. This space is searched by our specialized branch-and-bound algorithm, which we explain in Section 4.4. We evaluate the performance of our method against several greedy heuristics on a large test dataset (Sections 4.5.1 and 4.5.2), documenting the frequency of failure cases in previous methods and our improvements. Finally, we demonstrate applications of our layout embedding technique in quad meshing (Section 4.5.3) and surface map initialization (Section 4.5.4).

4.3 Problem Setup

We consider the task of injectively embedding a given layout into a target surface with prescribed embedding positions for the layout vertices. We restrict to the class of *shortest-path embeddings*, which is only parametrized by discrete degrees of freedom and eliminates continuous choices: They are precisely those embeddings that can be obtained by successively embedding layout edges in a certain order as (non-intersecting) shortest paths. Each of them is uniquely identified by an insertion sequence of edges, which encodes topological decisions *implicitly*: Depending on the ordering, shortest edge embeddings naturally assume certain homotopy classes by avoiding intersections with earlier insertions (Fig. 4.5).

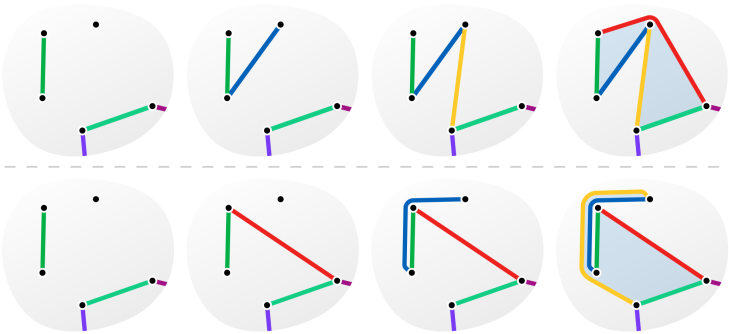


Figure 4.5: *Starting from a partial embedding (left), three additional edges are successively inserted as shortest paths. Depending on the insertion order, we get topologically different embeddings.*

4.3.1 Notation

We use the following notation throughout the rest of this chapter:

A *layout* $L = (V_L, E_L, F_L)$ is an abstract cell complex (Section 2.2) which, by itself, it carries no geometric information. Recall that the connectivity of L is defined by a rotation system (ρ, ν). For every vertex $v \in V_L$, repeated applications of $\nu \circ \rho$ enumerate a set of incident edges in a specific order. We denote this cyclic ordering of edges around vertex v by σ_v. The *target surface* $\mathcal{S}$ is an orientable, simply-connected (genus 0 or disk) surface.

Following previous works, we describe embeddings of L into $\mathcal{S}$ as *cellular graph embeddings* (Section 3.4.1), i. e. as images of the layout graph (V_L, E_L).

A *layout embedding* $M : L \to \mathcal{S}$ is therefore an injective map that assigns layout vertices and edges to points and paths on the target surface $\mathcal{S}$ (Fig. 4.6). Specifically:

- Every vertex $v \in V_L$ is mapped to a distinct point $M(v) \in \mathcal{S}$.

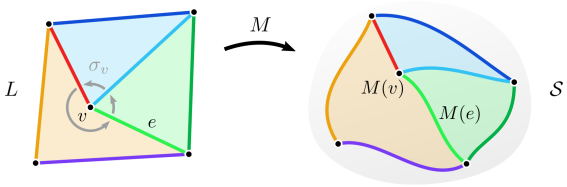


Figure 4.6: *Vertices and edges of a layout L are mapped to points and paths on a target surface $\mathcal{S}$ via an embedding M. Embeddings are injective and preserve the cyclic ordering σ_v of edges around each layout vertex v.*

- Every edge $e \in E_L$ connecting layout vertices v_1 and v_2 is embedded as a continuous path $M(e)$ on $\mathcal{S}$ with $M(v_1)$ and $M(v_2)$ as endpoints. Embedded edges do not intersect and only touch embedded vertices at their endpoints.
- The cyclic order of incident paths around each embedded vertex $M(v)$ respects the ordering σ_v defined by the layout's rotation system.

We also consider *partial* embeddings where only a subset of layout edges is mapped. We denote this subset of *embedded edges* by $E(M) \subseteq E_L$ and the set of *unembedded edges* as its complement $\overline{E}(M) = E_L \setminus E(M)$. A *complete* embedding cuts the target surface into cellular patches which correspond to the faces F_L of the layout.

4.3.2 Shortest-Path Embeddings

Given a partial embedding M and an unembedded layout edge $e \in \overline{E}(M)$, we define $p(M, e)$ as the *shortest path* on $\mathcal{S}$ that yields a valid extension of M. Such a path does not intersect already embedded edges and respects the cyclic ordering of incident embedded edges at its endpoints.

An *insertion sequence* $s = e_1 \cdots e_n \in E_L^*$ is a (partial) permutation of layout edges. It describes the construction of an embedding via successive addition of edges embedded as shortest paths: Starting from an "empty" embedding M_0

that only maps vertices to their prescribed positions, we successively construct embeddings $M_1, \ldots, M_n$, corresponding to the elements of s. Each M_k is identical to the previous M_{k-1}, but additionally includes an embedding of e_k, defined as $M_k(e_k) = p(M_{k-1}, e_k)$. The final embedding M_n includes all edges in s. We call it the *shortest-path embedding* of s, denoted by $M(s)$.

Note that different insertion sequences of the same set of layout edges can result in different shortest-path embeddings (Fig. 4.5): In general, $M(s) \neq M(\pi(s))$ for a permutation π.

We define the cost $c(M(e))$ of a path as its length on $\mathcal{S}$. The cost $c(M)$ of a (partial) embedding is the sum of costs of all embedded edges. It can happen that in an embedding M, no shortest path $p(M, e)$ exists for some edge e due to blocking by other paths. Consequently, it is possible that an insertion sequence s does not imply a valid shortest-path embedding. We consider such invalid paths and embeddings to have infinite cost.

4.3.3 Objective

We address the following problem: Given a layout L, a target surface $\mathcal{S}$, and an initial embedding M_0 that assigns target positions on $\mathcal{S}$ for all layout vertices, find the insertion sequence s that produces a shortest-path embedding $M(s)$ of minimum cost, i. e.

$$\operatorname*{argmin}_{s \in E_L^*} c(M(s)).$$

4.3.4 Discrete Representation

In practice, we use discrete representations of target surfaces and embeddings. A target surface $\mathcal{S}$ is given by a triangle mesh $(V_\mathcal{S}, E_\mathcal{S}, F_\mathcal{S})$. We represent an embedding by mapping directly to mesh elements: Each layout vertex $v \in V_L$ maps to a target vertex $M(v) \in V_\mathcal{S}$. Each layout edge $e \in E_L$ maps to a path of target edges $M(e) \subset E_\mathcal{S}$, connecting the corresponding endpoints.

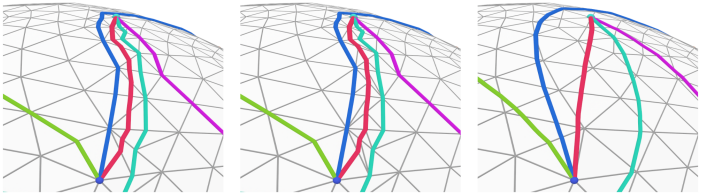


Figure 4.7: *A new edge embedding (red) is computed as a shortest intersection-free path on $\mathcal{S}$. During tracing (a), the path may travel across edge midpoints. Upon insertion (b), midpoints are incorporated into $\mathcal{S}$ via local refinement. After the embedding is complete, we apply path smoothing to the entire embedding (c).*

In this representation, shortest paths $p(M, e)$ can be found via Dijkstra's algorithm. As described in several prior works (e. g. [PSS01]), the search is constrained to maintain injectivity of the embedding: New paths must not cross or touch previously inserted edges. Additionally, when tracing paths between vertices with incident embedded edges, those vertices may only be approached from sectors that are consistent with the cyclic orderings σ_v defined by the layout.

Following [SAPH04] and others, we adaptively refine the target mesh to accommodate paths in mesh regions that are not sufficiently tessellated: During path tracing, we allow paths to tentatively travel across edge midpoints of the mesh $\mathcal{S}$ (Fig. 4.7 (a)). When a path is added to an embedding, we insert the required edge midpoints into $\mathcal{S}$ via local edge splits (Fig. 4.7 (b)).

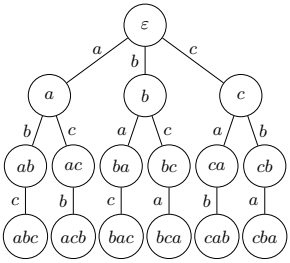
Path generation is a fully modular component of our algorithm. The above method is easy to implement and avoids excessive mesh refinement; it can however deviate from true geodesics (especially on poorly triangulated target meshes). If this is a concern, it can be easily replaced by a more costly but exact geodesic computation method [CLPQ20].

As a consequence of this discrete representation, resulting shortest paths can appear slightly jagged (due to the tessellation of the underlying mesh). A variety of (homotopy-preserving) post-processes can be applied to remove such discretization

artifacts and improve the shape of embeddings (cf. Section 4.1.3). We use the path smoothing method described by [PSS01] (retracing each edge path in a harmonic parametrization of the incident faces, Fig. 4.7 (c)) in our visualizations.

4.4 Branch-and-Bound Optimization

The incremental construction of a layout embedding via successive shortest path insertions consists of a sequence of decisions: Which layout edge should be embedded next? We can describe this process by a decision tree in which nodes represent intermediate partial embedding states and each edge indicates an insertion decision. Each node is uniquely identified by an insertion sequence s, encoding the (partial) embedding $M(s)$. (In the following, we will use the terms *node*, *embedding state*, and *insertion sequence* interchangeably.) The root node is the empty insertion sequence ε, corresponding to the initial state where only vertex correspondences are given and no edges are embedded. The outgoing edges of each interior node s represent possible extensions of the partial embedding $M(s)$: Each unembedded layout edge $e \in \overline{E}(M(s))$ is a candidate for insertion, leading to a child node identified by the sequence se. Therefore, every node s is a prefix of all of its descendants s', we write $s \sqsubseteq s'$. The $|E_L|!$ leaf nodes of the tree correspond to full embedding sequences, which are potential solutions to our problem (cf. Section 4.3.3).



Already for layouts with a moderate number of edges, the factorial size of this decision tree makes an exhaustive search for the optimal solution infeasible. In the following, we describe our branch-and-bound strategy that finds solutions while only exploring a small fraction of the search tree. Effective bounding allows our

method to prune the search space and to precisely judge the optimality of the solutions it discovers.

4.4.1 Algorithm

Essentially, our branch-and-bound algorithm is a tree search that crawls the decision tree along a front propagating from the root. During the entire search, it keeps track of an *incumbent solution* s^*: an insertion sequence representing the best (i. e., lowest-cost) solution encountered so far. The cost of the current incumbent defines a *global upper bound* $c^\top = c(M(s^*))$. At any time, this upper bound is greater than (or equal to) the cost of the true optimal solution of the problem. It decreases monotonically with each update, approaching the optimum.

We can use greedy heuristic methods (cf. Section 4.5 for details) to produce an initial incumbent and upper bound. If no heuristic initialization is used, we start with $c^\top = \infty$.

Besides global upper bound estimates, a vital element of any branch-and-bound method is the ability to additionally compute lower bound cost estimates for individual states of the decision tree. For each state of the tree, identified by a partial insertion sequence s, we define a *local lower bound* $c_\perp(s)$ fulfilling the following property: Any state s' in the subtree rooted at s yields an embedding with a cost of at least $c_\perp(s)$, i. e.

$$c_\perp(s) \leq c(M(s')) \quad \forall s \sqsubseteq s'.$$

Lower bounds are typically obtained via a relaxation of the remaining sub-problem. We explain the computation of lower bounds in our setting in Section 4.4.3.

Available information on upper and lower bounds can be exploited to prune the decision tree. Any time a state s has a lower bound $c_\perp(s)$ that exceeds the current global upper bound $c^\top$, we can safely ignore it (and its entire subtree) in our search: By definition of $c_\perp(s)$, all solutions in the subtree of s will be more costly than $c^\top$ and therefore cannot improve the incumbent s^*.

The order in which states (nodes of the search tree) are visited is controlled by a function that assigns a priority $P(s)$ to any state s. We discuss our design of the priority function $P(s)$ in Section 4.4.6. Potential states to be visited are stored in a *priority queue* Q sorted by P. Initially, this queue only contains the root node, i. e. $Q = \{\varepsilon\}$.

The core of our branch-and-bound algorithm is a loop that consumes the priority queue Q. In each iteration, the state s with the highest priority is extracted from Q and processed as follows:

1. *Update*: Whenever the insertion sequence s yields a complete embedding $M(s)$, it is a potential solution (i. e., a leaf node of the decision tree). If its cost $c(M(s))$ improves the current incumbent solution s^*, we record s as the new incumbent and update the global cost upper bound $c^\top$ accordingly.
2. *Branch*: If s corresponds to a partial embedding, we enumerate all states that can be reached from s by inserting an additional layout edge $e \in \overline{E}(M(s))$ (child nodes of s in the decision tree) and add them to Q for future exploration.
3. *Bound*: For each new state s' that is added to Q, we perform a bounds check: If the cost lower bound $c_\perp(s')$ is larger than the current global upper bound $c^\top$, it can be pruned: Instead of adding s' to Q, we simply discard it.

A typical behavior for Q is to initially keep growing until an upper bound $c^\top$ is reached that facilitates enough pruning to reverse the growth of Q. The algorithm terminates when Q is exhausted.

4.4.2 Optimality Gap

At any time during the execution of the algorithm, Q contains a set of states along the current propagation front of the tree search. By examining the lower bounds of states in Q, we can compute a *global lower bound*

$$c_\perp = \min_{s \in Q} c_\perp(s)$$

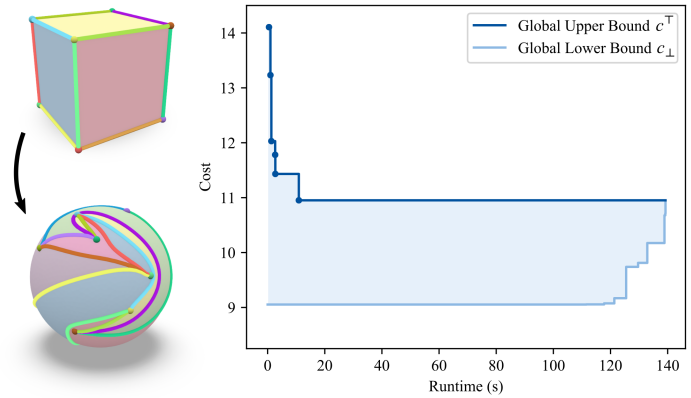


Figure 4.8: *Progression of global upper bound $c^\top$ and global lower bound $c_\perp$ over the course of an optimization (here: embedding a cube layout into a sphere with random target vertex positions). The optimal solution is already found after 11 s. Optimality is proven after 139 s.*

which bounds the cost of all unseen solutions further down the tree. The difference between the two global bounds $c^\top - c_\perp$, called the *optimality gap*, quantifies by how much the incumbent solution could possibly improve during the remainder of the search. Figure 4.8 visualizes how the optimality gap gradually closes as global upper and lower bounds converge over the course of an optimization.

When the algorithm terminates, we know the incumbent s^* is a global optimum. Instead of waiting for termination, the algorithm can also be stopped prematurely. In that case, s^* may not be optimal, but we can give an estimate of its quality: We know that if there is a better solution, its cost could at most be better by the current optimality gap. This mechanism gives us the flexibility to run the branch-and-bound algorithm on a time limit.

Likewise, we can choose to terminate the algorithm early if the current optimality gap falls below an acceptable threshold. When the relative optimality gap $\frac{c^\top - c_\perp}{c^\top}$ falls below a certain tolerance fraction α, we stop execution. The same tolerance

can be applied for all pruning operations, which further reduces the search space. We use $\alpha = 1\,\%$ in all of our experiments.

In the following, we discuss customizations of this general branch-and-bound setup, which are specifically tailored to the structure of our layout embedding problem.

4.4.3 Lower Bounds

Given a state s, corresponding to a partial embedding $M(s)$, we want to compute $c_\perp(s)$: A lower bound on the cost of any full embedding $M(s')$ that can be reached from s by embedding all remaining layout edges $\overline{E}(M(s))$.

One valid lower bound is the cost of the current partial embedding $c(M(s))$ (Fig. 4.9 (a)): All edges already embedded in $M(s)$ are identical in any extension $M(s')$ and the insertion of the remaining edges in $M(s')$ will only incur additional cost. By itself, this lower bound is fairly ineffective because it entirely ignores the potential cost of embedding the remaining part of the layout.

We can obtain a tighter bound by also simulating the insertion cost of the remaining edges in a setting that relaxes the constraints on a valid embedding:

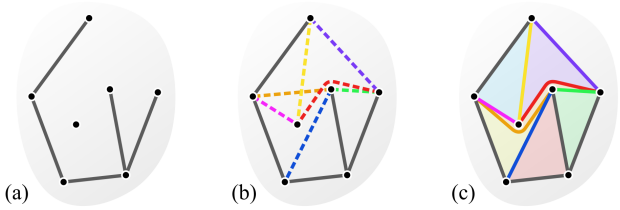


Figure 4.9: *We compute a lower bound for the cost of any valid completion of a partial embedding (a): Besides measuring the length of embedded edges (solid gray lines), we compute candidate paths (dashed lines) for unembedded layout edges (b). These are compatible with the partial embedding but possibly mutually conflicting. The sum of these path lengths is a lower bound for the cost of any embedding where all conflicts are resolved (c).*

For each unembedded edge $e \in \overline{E}(M(s))$, we compute a *candidate path*, i. e. its shortest-path embedding $p(M(s), e)$. While each candidate by itself is compatible with the current state $M(s)$, the set of all candidates will likely not constitute a valid embedding (Fig. 4.9 (b)): It is possible that candidates mutually intersect or violate each other's cyclic ordering requirements. Still, due to each candidate path being individually shortest, we know that any modification to resolve these conflicts would only increase cost (Fig. 4.9 (c)). Therefore, the sum of candidate path costs serves as a lower bound for the cost of any valid completion. In combination with the cost of already embedded edges, we arrive at the following cost lower bound for a state s:

$$c_\perp(s) = c(M(s)) + \sum_{e \in \overline{E}(M(s))} c(p(M(s), e)).$$

By definition (Section 4.3.2), we consider path costs to be infinite where no valid path exists. In a situation where one of the candidate paths has no valid embedding (due to being blocked by different, already inserted paths), the lower bound $c_\perp(s)$ becomes infinite, thereby identifying the current state as a "dead end" and excluding it from further processing.

Besides this generic pruning based on upper and lower bounds, we employ additional specialized pruning rules to reduce redundant computations in different branches of the decision tree.

4.4.4 Detecting Redundant States

It is possible that two different insertion sequences s_1, s_2 lead to an identical (partial) embedding $M(s_1) = M(s_2)$. In that case, the entire subtrees of s_1 and s_2 are identical as well. If we have already visited s_1 (or know that we will visit it), we can safely ignore s_2 and its entire subtree in our search.

Whenever we consider adding a new state s to the priority queue Q for exploration, we compute a lightweight hash of its embedding $h(M(s))$ and look it up in

a table H of known embeddings. If $h(M(s))$ is already in H, we simply discard s. Otherwise, we add s to Q and insert its hash $h(M(s))$ into H.

In our discrete setting, an embedding hash $h(M)$ can be computed as follows: For each layout edge $e \in E_L$, we form a string w_e by concatenating the vertex positions along the corresponding embedded edge $M(e)$. If e is not embedded in M, we use a blank symbol $w_e = \#$. We concatenate all strings w_e in a canonical order and compute a hash of the resulting word to obtain $h(M)$.

The above strategy detects redundant states when different branches of the decision tree have arrived at an identical embedding. In the following, we describe additional rules that can predict when the exploration of certain subtrees will only lead to redundant results so we can prune proactively.

4.4.5 Delaying Non-Conflicting Insertions

In a state s, each unembedded edge $e \in \overline{E}(M(s))$ is associated with a candidate path $p(M(s), e)$. Inserting an edge e will lead to a new state in which other edges have different candidate paths in general. However, there are some edges (called *non-conflicting edges*) whose insertion will not change the candidate paths of any other edges. When inserting only non-conflicting edges, the outcome will hence *not* depend on their order: For a set of n non-conflicting edges, all of their $n!$ insertion sequences will lead to an identical embedding. We detect non-conflicting edges and avoid the redundant computation of exploring all their permutations and thus significantly reduce the effective branching degree of our search tree.

A non-conflicting set of unembedded edges $\overline{C} \subseteq \overline{E}(M(s))$ is a set for which all insertion sequences lead to the same embedding; i. e. $M(s\pi(\overline{C}))$ is identical for any permutation π. For each state, we define $\overline{C}(s)$ as the maximal non-conflicting set and its complement, the set of conflicting edges, as $C(s) = \overline{E}(M(s)) \setminus \overline{C}(s)$. We consider the following two cases:

- If $C(s)$ is empty, all remaining unembedded edges are the non-conflicting edges $\overline{C}(s)$. In this case, the remaining edges can be inserted in any order,

all leading to the same solution. Instead of exploring further child states, we immediately insert the remaining edges in an arbitrary order and terminate the search in this branch.

- If $C(s)$ is not empty, there are insertion decisions that matter: Every insertion of an edge from $C(s)$ has consequences for the embedding of at least one other edge, so we must consider these states in our search. In contrast, insertions of edges from $\overline{C}(s)$ have no immediate effect on other edges, so inserting them at this point would only introduce redundant branching. We therefore skip all insertions from $\overline{C}(s)$. Essentially, this delays the insertion of non-conflicting edges until they either become conflicting in some later state (at which point they are considered as an insertion option), or until only non-conflicting edges remain, which are then inserted simultaneously (see case above).

An extreme effect of this pruning strategy can be observed when the initial network of candidate paths at the root state ε is already conflict-free (which can in fact happen on simple inputs). In that case, $C(\epsilon)$ is empty and our algorithm immediately terminates, returning the optimal result: An arbitrary insertion sequence of all edges.

This ruleset indeed only excludes redundant states, as can be seen by the following argument: Consider a state s and a pair of edges $c \in C(s)$ and $n \in \overline{C}(s)$. Since n is non-conflicting, its insertion will neither change the candidate path of any other edge when inserted, nor will its candidate path be changed by the insertion of any other edge into s, i. e.

$$p(M(s), c) = p(M(sn), c),$$
$$p(M(s), n) = p(M(sc), n),$$

which implies $M(snc) = M(scn)$, and thus

$$M(sncs') = M(scns') \tag{4.1}$$

for any remaining sequence s'. Our conflict-avoiding pruning only visits states $v \in V$ with insertion sequences of the form $v = s_c s_n$ where

- s_c is a *conflicting sequence*, i. e. for each prefix $s'_c e \sqsubseteq s_c$, it is $e \in C(s'_c)$,
- s_n is a *non-conflicting sequence*, i. e. for each prefix $s_c s'_n e \sqsubseteq s_c s_n$, it is $e \in \overline{C}(s_c s'_n)$.

We show that for every unvisited state $s \notin V$, there is an embedding sequence $v \in V$ such that $M(s) = M(v)$. We can construct v from s by iteratively pushing non-conflicting insertions towards the end: Suppose $s \notin V$. Then, there is a decomposition $s = s_c s_n n c s'$ such that s_c is a (possibly empty) conflicting sequence, s_n is a (possibly empty) non-conflicting sequence, $n \in \overline{C}(s_c s_n)$, and $c \in C(s_c s_n)$. By Eq. (4.1), we have $M(s_c s_n n c s') = M(s_c s_n c n s')$. Repeating this operation yields $M(s_c s_n n c s') = M(s_c c s_n n s')$. As $c \in C(s_c s_n)$, so is $c \in C(s_c)$, hence $s_c c$ is a conflicting sequence. If $s_n n s'$ is a non-conflicting sequence, then $s_c c s_n n s' \in V$ (*q. e. d.*). Otherwise, we repeat the same argument for the remaining part until s' becomes empty.

Detecting Non-Conflicting Edges

For each unembedded edge $e_i \in \overline{E}(M(s))$, we have already computed a candidate path $p_i = p(M(s), e_i)$ as part of the lower bound estimate of state s

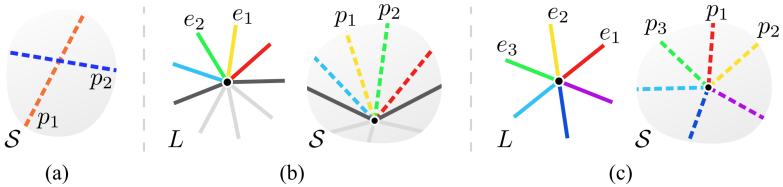


Figure 4.10: *There are three types of conflicts between candidate paths (dashed lines): (a) path intersections, (b) wrong path ordering within a sector created by already embedded edges (gray lines), (c) wrong path ordering around a vertex without sectors.*

(Section 4.4.3). We can determine the sets of conflicting and non-conflicting edges $C(s)$, $\overline{C}(s)$ based on the current configuration of candidate paths p_i: Clearly, if two candidate paths p_1 and p_2 intersect (Fig. 4.10 (a)), the edges e_1 and e_2 are in conflict and we can add them to $C(s)$.

Additional conflicts can arise from the cyclic ordering requirements of paths around embedded vertices (Section 4.3): A path p_i may affect incident candidate paths at its endpoints, requiring them to approach the vertex from a different direction if p_i were inserted earlier. We detect such conflicts by verifying the current ordering of candidate paths around each embedded vertex $M(v)$. We distinguish two cases:

- If the layout vertex v has incident edges that are already embedded, the corresponding outgoing paths divide the region around $M(v)$ into sectors (Fig. 4.10 (b)). All candidate paths incident to $M(v)$ are already in their correct sector (otherwise their embedding would be invalid), but their relative ordering within a sector may be incompatible with the ordering σ_v imposed by the layout. In this case, it suffices to check conflicts based on a linear ordering: If an edge e_1 comes before another edge e_2 (in a counterclockwise sense) within a layout sector, the candidate path p_1 must come before p_2 in the embedded sector. Otherwise, e_1 and e_2 are conflicting and are added to $C(s)$.

- If $M(v)$ has no incident embedded edges, there are no sectors, so we need to verify the cyclic ordering of all candidate paths (Fig. 4.10 (c)): For every triplet $(e_1, e_2, e_3) \in \sigma_v$ of layout edges around v (in counterclockwise order), the corresponding candidate paths p_1, p_2, p_3 must have the same cyclic ordering at $M(v)$ in the embedding. Otherwise, we consider the edges e_1, e_2, e_3 as conflicting and add them to $C(s)$.

4.4.6 Priority

A common strategy to explore the search tree is to prioritize states with a small cost lower bound $c_\perp(s)$, motivated by the expectation to potentially find the lowest-cost solution there. In our observations, this strategy performs poorly: During early stages of the search, lower bounds can be quite similar across many states. With lower bounds monotonically increasing along branches of the decision tree, this priority leads to an approximate breadth-first traversal where large portions of the tree are expanded before reaching any leaf nodes that produce potential solutions. Since significant pruning of states can only happen after suitable upper bounds have been found, this search strategy essentially explores almost the entire decision tree, which is practically impossible.

We suggest a different exploration priority that aims to produce upper bounds early on (to facilitate pruning) while still steering towards states with promising lower bounds (to find potentially better solutions). The idea is to favor the pursuit of branches that have already made significant progress towards a solution.

For a given state s, we quantify this progress as follows: After computing candidate paths for the unembedded edges of $M(s)$, we apply the classification into conflicting and non-conflicting edges $C(s)$ and $\overline{C}(s)$ (Section 4.4.5). We use the number of conflicting edges $|C(s)|$ to judge how far s is from a complete solution: Starting from s, we estimate that $|C(s)|$ decisions are needed to resolve all remaining conflicts. We combine this information with the cost lower bound of s to define

$$P(s) = |C(s)| \cdot c_\perp(s)$$

and prioritize states where $P(s)$ is smallest.

4.4.7 Implementation Notes

During optimization, we need to represent different partial embeddings on the target triangle mesh $\mathcal{S}$. In each iteration, we extract a state s from the queue Q

and reconstruct the embedding $M(s)$ by starting from the original mesh $\mathcal{S}$ and inserting the sequence s, locally refining the mesh as needed (cf. Section 4.3.4).

During our search, we build a lightweight representation of the decision tree explored so far, which allows us to cache intermediate results and avoid costly recomputations of paths. Whenever we produce a new state se from a parent state s, we save the newly computed path $p(M(s), e)$ at the corresponding edge of the decision tree. We can later reconstruct the embedding $M(s)$ by collecting all cached paths along the branch from ε to s and inserting them in that order.

The computation of candidate paths (Section 4.4.3) can be cached in a similar way: Initially, we compute candidate paths for all edges independently and store them in the root node ε. For all subsequent states, we only need to recompute candidate paths for the edges that are affected by a new insertion. In each child state se, those are precisely the edges that were in conflict with e in the parent state s (Section 4.4.5). It suffices to only cache the set of updated candidate paths in each state: To reconstruct the full set of candidate paths, we follow the tree towards its root until a cached path is found for all edges.

In all our examples, the total memory consumed by the queue Q (Section 4.4.1), the hash table H (Section 4.4.4), and the hierarchical cache of shortest paths never exceeded 200 MB.

4.5 Results and Applications

In the following we evaluate the performance of our method, compare against greedy approaches with different heuristics, and demonstrate application scenarios that benefit from the robustness of our method.

Greedy Ordering Heuristics

We compare against three different greedy methods that base their decisions (which edge to embed next) on the heuristics discussed in Section 4.1.3, specifically those

that appear in [PSS01; KSG03; SAPH04]. While [PSS01] addresses our exact problem setting, we adapt the ideas presented in [KSG03] and [SAPH04] from the *variable*-layout setting to our *fixed*-layout setting:

- [PSS01] use a "swirl detection" heuristic to delay path insertions that are likely to cause topological artifacts: Edge embeddings are postponed if opposite layout vertices of the inserted edge are located on the wrong side of the tentative path, based on a proximity check. Further, embedded paths are traced using a custom metric that pushes paths away from landmark vertices. To prevent dead ends, insertions that close a cycle of edges are delayed until a spanning tree of the entire layout is embedded.
- Instead of relying on a conservative spanning tree heuristic, [KSG03] (and [KS04]) only prevent insertions that actually lead to a blocking of future paths.
- [SAPH04] also employ the swirl detection and spanning tree heuristics described by [PSS01]. In addition, their method prefers the insertion of edges connecting "extremal vertices" with a large average geodesic distance to other landmarks.

As described in Section 4.4.1, such greedy orderings are convenient to quickly derive global upper bounds on the total embedding length. We initialize our branch-and-bound algorithm by running all three greedy variants and choosing the best result as upper bound.

4.5.1 Layout Embedding in Shape Collections

Applications that establish correspondences within shape collections may require embedding a common layout into a range of models. This layout can carry semantic information, guide remeshing algorithms, or act as a parametrization base domain. Especially if the layout was created before all instances in the collection were known or when the collection contains outliers, insertion of new instances can be

challenging and requires a robust embedding algorithm. We simulate this situation by generating one layout per class of the SHREC'07 dataset and embedding it into all models of that class.

Dataset

From the SHREC'07 dataset [GBP07] with sparse landmark correspondences provided by [KLF11], we select classes of genus 0 meshes with at least 3 landmark annotations (15 classes in total). For each class, we generate a triangular layout via constrained decimation of its first model: We incrementally perform halfedge collapses until only the landmark vertices (between 7 and 36 per class) remain. The resulting coarse mesh connectivity defines our prototypical template layout.

Experiment

For each model, we run our algorithm as well as the three greedy methods based on [PSS01], [KSG03] and [SAPH04]. For visual clarity we apply the path smoothing operator of [PSS01] to all embeddings. Quantitative results (e. g. total embedding lengths) are reported prior to this post process.

Discussion

In Fig. 4.11 we present a selection of resulting embeddings, while Fig. 4.12 shows quantitative results of the entire experiment: Per object class, we arrange all models within this class on the horizontal axis and plot the total embedding lengths achieved by the four methods on the vertical axis; i. e. each column represents a problem instance and each dot represents a solution by one of the algorithms. For each problem instance we mark the global lower bound $c_\perp$, computed by our method, with a horizontal bar (light blue). This means our solver proved that no embedding sequence shorter than this bound can exist.

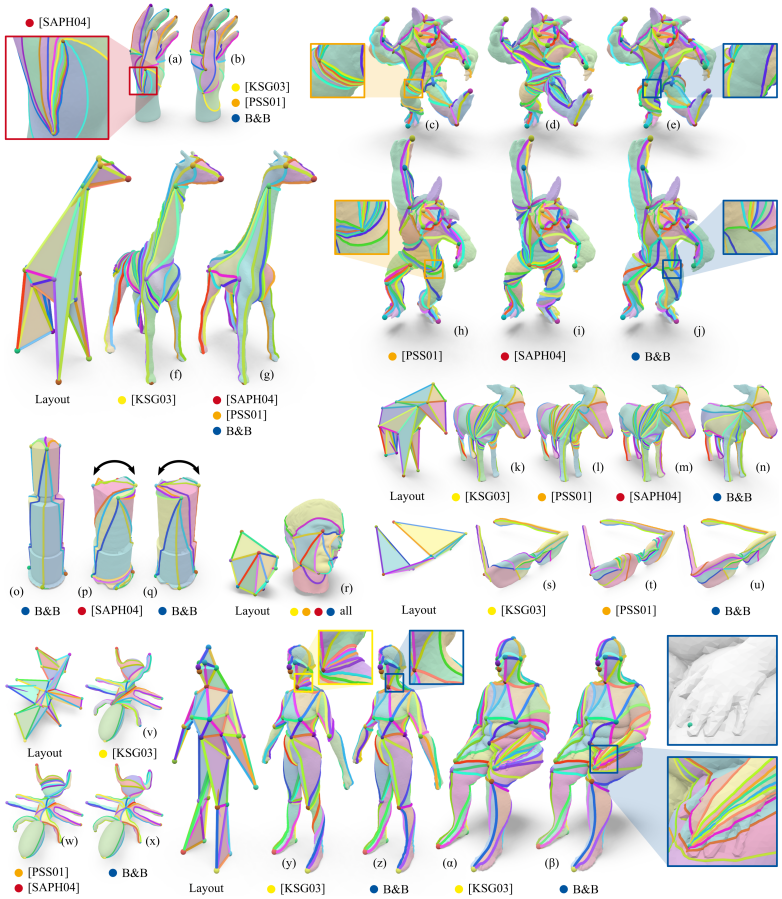


Figure 4.11: *Selected results of our evaluation in Fig. 4.12: Automatically generated layouts are embedded into models of the SHREC'07 dataset. We compare greedy methods based on the heuristics in [PSS01], [KSG03], and [SAPH04] against our branch-and-bound method. While in some cases (e. g. (f)) all methods achieve the desired result, and in some instances at least one greedy method succeeds ((b), (k)), there are many inputs on which all greedy strategies fail ((c-e), (g-i) (l-n), (r-u)). If embedding algorithms fail, defects can be severe, with paths forced into unwanted homotopy classes.*

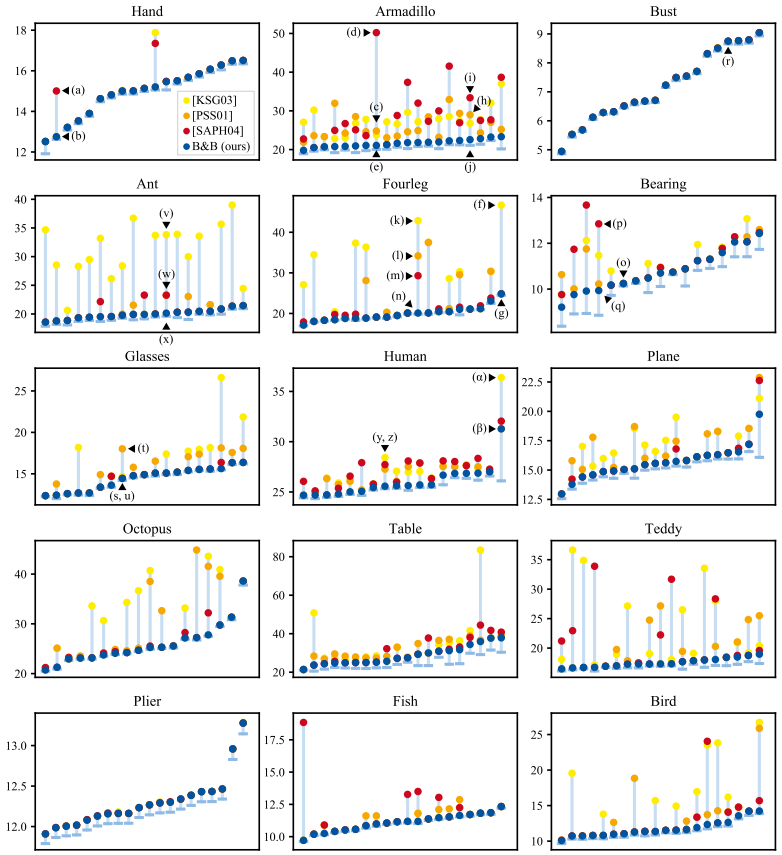


Figure 4.12: *Quantitative evaluation of our SHREC'07 experiment, in which we embed a common layout into the objects of each class. Corresponding qualitative results are reported in Fig. 4.11. Per class, we horizontally arrange all problem instances as columns and plot total embedding lengths on the vertical axis. Each dot represents a solution by one of the algorithms: either greedy—based on [KSG03] (yellow), [PSS01] (orange), [SAPH04] (red)—or by our branch-and-bound method (blue). Global lower bounds proven by our method are marked in light blue. For a few classes all methods perform equally well, but in most instances our branch-and-bound optimization achieves considerable improvements.*

As we employ all three greedy heuristics to find an initial upper bound, the embeddings computed by our method (blue dots in Fig. 4.12) are always shorter or equal to those of the greedy methods.

On favorable examples (e. g. Fig. 4.11 (r)), all four methods find the same embedding. In those cases (including the entire classes Bust and Plier, cf. Fig. 4.12), our algorithm quickly proves that a greedy embedding sequence is indeed optimal up to the prescribed threshold ($\alpha = 1\,\%$ in all examples).

In Fig. 4.12 we observe that each greedy method, while performing well in some instances, fails to generate shortest embeddings in a significant number of cases. The results shown in Fig. 4.11 demonstrate that these cases coincide with topologically unexpected embeddings containing excessive swirls.

While in some cases at least one of the greedy methods yields the desired result (e. g. Fig. 4.11 (b), (g), (r)), there are plenty of cases in which all three fail (e. g. Fig. 4.11 (c-e), (h-j), (k-m), (v-x)). Therefore, an algorithm running all three heuristics and choosing the best result does not provide a satisfactory solution. In contrast, we did not find unnecessarily long paths or swirls in any of the embeddings produced by our branch-and-bound method.

In many cases ($35\,\%$ of the dataset, e. g. Fig. 4.12 (b), (o), (r), (z)) our algorithm confirms optimality of the solution (up to the threshold of $1\,\%$) within the given time frame (5 minutes in all examples). When the algorithm exhausts this time budget, small gaps are usually reported. For $83\,\%$ of the dataset, optimality has been proven up to $5\,\%$.

Only in a few particularly challenging cases our algorithm was stopped with a large remaining gap (e. g. $10\,\%$ in (q), $17\,\%$ in (β)). Fig. 4.11 (q) poses a challenge as two landmark positions are swapped in the original dataset. Example (β) is difficult because it is the only model in the Human class where the arms are merged with upper body and legs (see zoom-in). While the task of embedding the given layout into this particular model is semantically questionable, our method still handles this outlier gracefully. Even though optimality is only proven up to $17\,\%$,

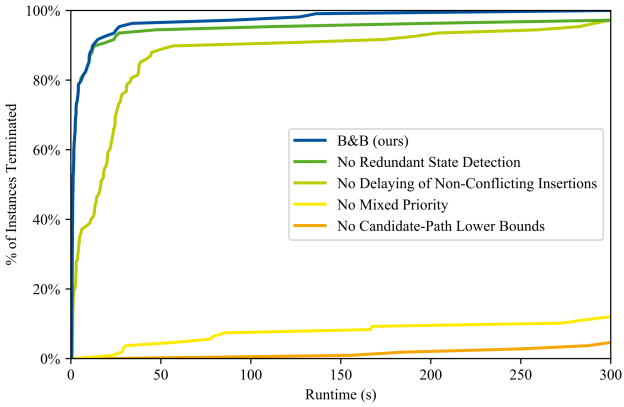


Figure 4.13: *Ablation study: We run our algorithm in different configurations (each disabling one of the features discussed in Sections 4.4.3 to 4.4.6) on a subset of the SHREC'07 dataset. For each configuration, we report the percentage of instances which terminated within a given time.*

we argue that the solution is indeed the desired one since we could not find any unnecessarily long paths or swirls upon visual inspection.

Leaving the difficult task of proving optimality aside, we observe that our algorithm produces good embeddings very quickly. In $60\,\%$ of the dataset, the best solution of each problem instance was already found after 10 seconds. Only in $2\,\%$ of cases the final result was found after more than 3 minutes. All timings were measured using a single-threaded implementation on a desktop computer with an Intel Core i7-8700K CPU.

Ablation Study

We evaluate the performance impact of the branch-and-bound customizations discussed in Section 4.4. Based on the previous experiment, we select all instances of the SHREC'07 dataset where the search was completed within 5 minutes and re-run our algorithm in different configurations, each disabling one of its features.

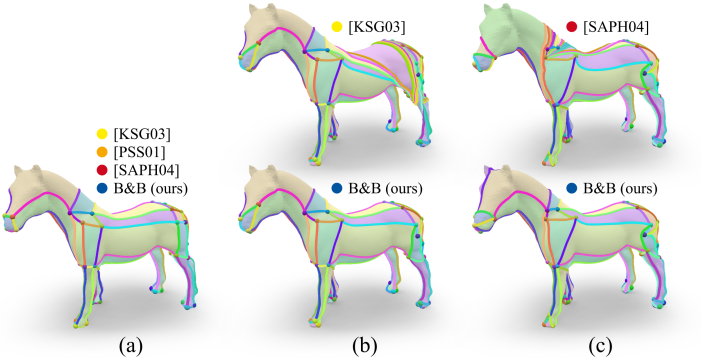


Figure 4.14: *Starting from their initial position on the target surface, we move landmarks along random trajectories and compute embeddings (a, b, c). Corresponding embedding costs are plotted in Fig. 4.15. Already for quite small perturbations (b, c), greedy methods (e. g. [KSG03; SAPH04]) can produce unfavorable topological changes. In comparison, embeddings generated by our method (B&B) remain in the most adequate mapping class (identical for (a, b, c)).*

To rule out the influence of different heuristics, we do not initialize our algorithm with a greedy solution in this experiment. Results are reported in Fig. 4.13: The plot shows the relative number of instances that have terminated (i. e. found and proved an optimal solution up to $1\,\%$) within a given time. Disabling the detection of redundant states via hashing (dark green, Section 4.4.4) causes some runs to exceed the 5 minute time limit but has nearly no effect on simple inputs, where runtimes are close to our fully-featured algorithm (blue). The impact of our proactive pruning by delaying non-conflicting insertions (Section 4.4.5) is more pronounced: When disabled (light green), performance deteriorates across almost all examples. With a traditional best-first search priority (yellow, instead of our mixed priority, Section 4.4.6), or a simplistic computation of lower bounds (orange, ignoring the cost of candidate paths, Section 4.4.3), only very few runs terminate within the time limit.

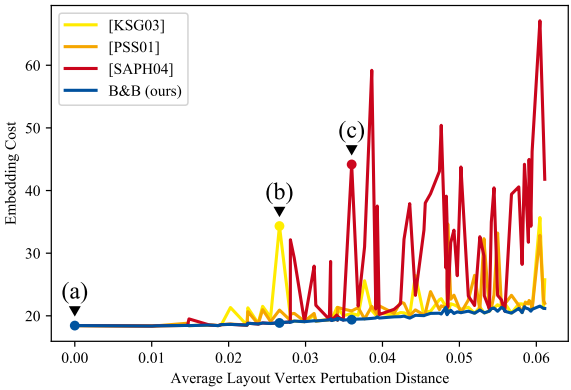


Figure 4.15: *Embedding costs from the layout vertex perturbation experiment in Fig. 4.14, with corresponding labels (a, b, c). Sudden changes in embedding topology manifest as discontinuous increases in embedding cost. The cost of our optimized embeddings (blue) only increases gradually.*

4.5.2 Robustness with Respect to Landmark Positions

In an additional experiment (Fig. 4.14) we compare the resilience of all four algorithms with respect to a perturbation of landmark positions on the target surface. Initially, we embed the layout using a favorable set of landmark positions. We then move each landmark along a random walk on the surface and recompute embeddings at different travel distances. We find that greedy methods are quite sensitive towards such changes in landmark positions. Already for mild perturbations, all three greedy methods introduce swirls leading to high embedding costs. Further, we observe that for an (approximately) continuous motion of target vertices, the length of greedy embeddings changes discontinuously, as path homotopy classes are often switched accidentally. In contrast, the lengths of our branch-and-bound embeddings increases gradually and all paths remain in their initial homotopy class for the duration of this experiment.

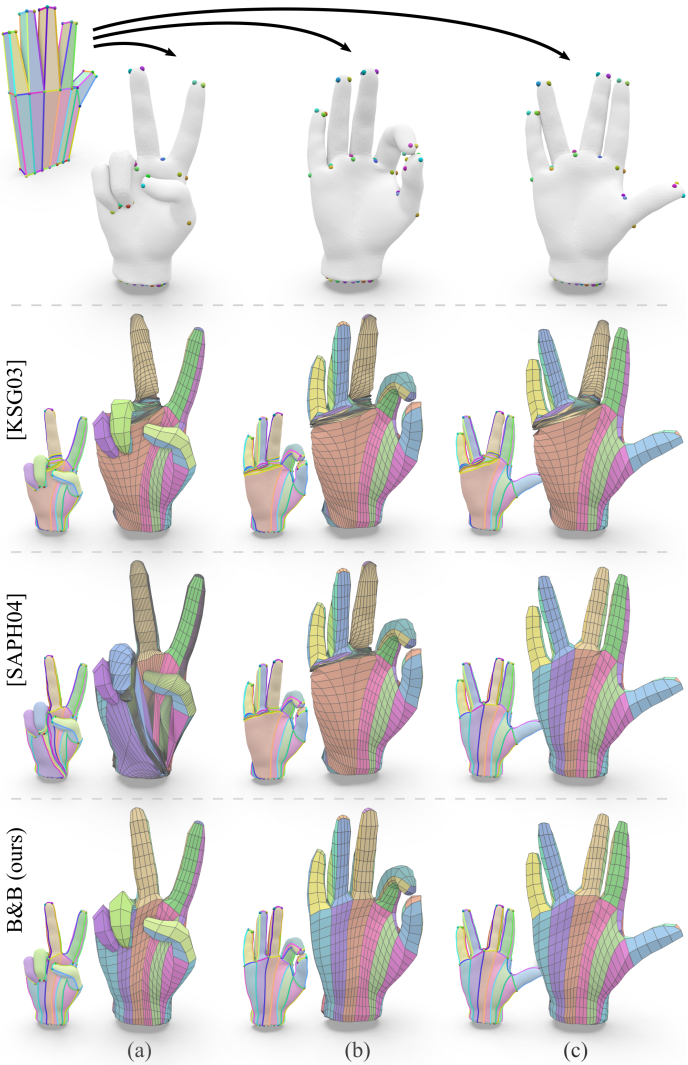


Figure 4.16: *We embed a layout template into multiple target meshes (top row) to compute quad meshes (a, b, c) with prescribed base complex. In contrast to greedy methods (middle rows), our branch-and-bound algorithm finds natural embeddings, allowing us to extract clean quad meshes (bottom row).*

4.5.3 Quad Meshing with Prescribed Base Complex

Generating meshes with a high-quality base complex is a challenging task in parametrization-based quad meshing. The problem becomes even more difficult when the same resulting base complex is required across multiple target shapes. In many methods (e. g. [BZK09; BCE+13; RRP15]), the base complex arises as a result of the remeshing process on a single shape. An alternative approach is to reverse this dependency and explicitly model the base complex a priori; either manually [TPSS13; CK14a] or automatically [CBK12; ULP+15]. Several quad meshing methods (e. g. [BCE+13; CBK15; ESCK16]) offer control over the resulting base complex by constraining pairs of prescribed vertices to the same iso-parameter line. However, formulating such constraints requires knowledge of the desired path homotopy, which is usually not available when the target base complex was designed on a different model. Obtaining this required homotopy information amounts to the exact same problem as embedding a prescribed layout (base complex) into a target surface.

We illustrate the general approach of quad meshing with a prescribed base complex at the example of a simplistic quad meshing pipeline: We (1) embed a given quad layout into multiple target surfaces, (2) apply the path smoothing operator from [PSS01], (3) choose a number of subdivisions per dual loop, (4) Tutte-embed each patch to a planar rectangle, and (5) extract a quad mesh via regular re-sampling. In Fig. 4.16 we run this pipeline on three hand models and compare results using our embedding method to typical failure cases of greedy embeddings. While all resulting quad meshes share the same base complex, some meshes based on greedy embeddings exhibit extreme distortion due to paths and patches in undesirable homotopy classes. Such cases demonstrate that robustness of an embedding algorithm is essential when used as a step in a geometry processing pipeline. Our branch-and-bound algorithm produced the expected embedding homotopy in all our experiments.

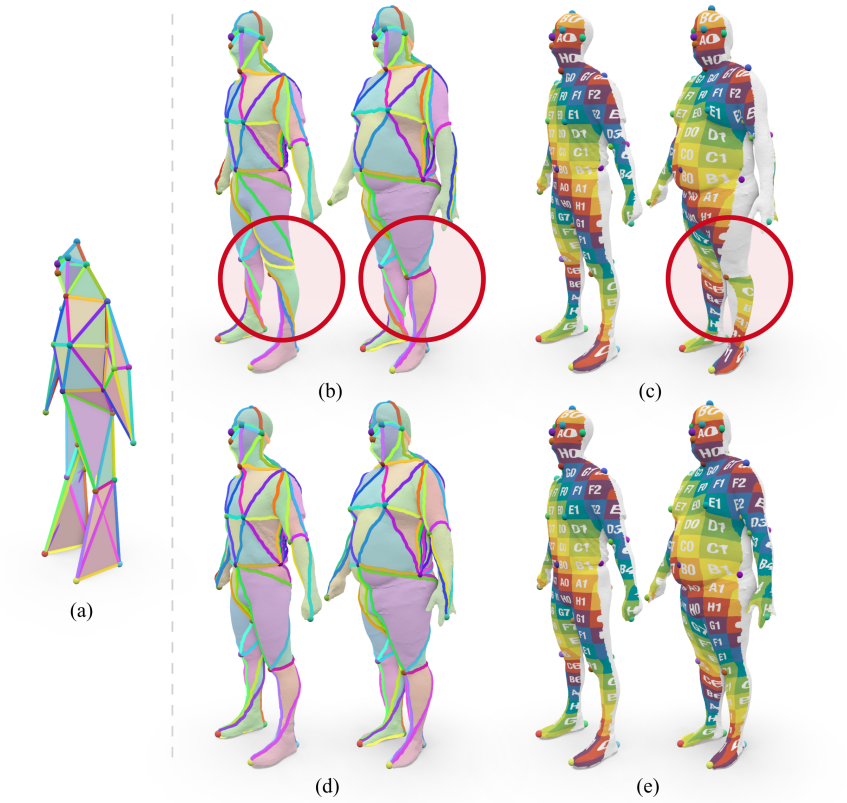


Figure 4.17: *We compute inter-surface maps by embedding an automatically generated layout (a) into two target models. Greedy embeddings (top row, here via [KSG03]) are likely to include paths in different homotopy classes across target models (see e. g. leg in (b)). Subsequent continuous map optimization (here using [SCBK20]) cannot leave the initial mapping class implied by the embeddings. We visualize the optimized map (c) by projecting a texture to the front-facing part of the first model and transferring it to the second model. Our method successfully generates shortest embeddings (d) which lead to the desired mapping class and allow continuous optimization to reach the expected result (e).*

4.5.4 Surface Map Initialization

Bijective maps between surfaces can be initialized via compatible layout embeddings on two models (Section 4.1). The resulting map topology (i. e. its mapping class) is then a composition of the mapping classes of the two embeddings. If landmark correspondences are used as hard point-to-point constraints, these are elements of the pure mapping class group of the surfaces punctured at the landmarks (Section 4.1.2). Continuous optimization algorithms reducing map distortion, by their very nature, cannot switch between mapping classes (Section 3.1.3). Therefore, it is crucial to compute an initial map that is already of the desired topology.

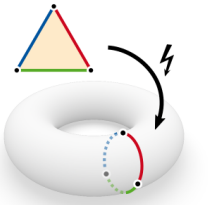
To demonstrate this importance, we initialize surface maps for a pair of models from the experiment in Section 4.5.1; once via greedy embeddings and once via our branch-and-bound embeddings (see Fig. 4.17). We then optimize both maps using [SCBK20] and visualize the results via texture transfer. The greedy pair (b) shows paths in different homotopy classes on the left leg: Some paths (e. g. yellow) are twisted around the leg of the first model, but not around the leg of the second model. As a result of this discrepancy, the optimized map (c) exhibits a twist, where regions from the back of the leg (untextured) are mapped to the front of the leg. The optimization [SCBK20] is inevitably trapped in this map homotopy class and cannot resolve the situation. In contrast, the embeddings produced by our optimal (up to $1\,\%$) insertion sequences solely contain paths in the expected homotopy classes (d) and lead to the desired bijective map (e).

4.6 Limitations and Future Work

While our layout embedding algorithm provides a robust alternative to previous greedy methods, there are still certain limitations, which we discuss below.

4.6.1 Surface Topology

The layout embedding method as it was described in this chapter is limited to simply-connected (i. e., genus 0 or disk-topology, Fig. 4.18) domains. On other input topologies, our algorithm will still find an optimal shortest-path embedding of all edges. However, it is no longer guaranteed that such an embedding is *cellular*, i. e. that all layout faces have disk-homeomorphic embeddings (see inset). A potential remedy is to impose further topological constraints on candidate paths, e. g. by enforcing that edges which close a separating cycle in the layout graph are embedded as separating paths (similar to [LPVV01; SAPH04], though finding *shortest* separating paths is more involved). The higher-genus setting requires additional restrictions to anticipate and avoid insertion decisions that run into dead-ends, e. g. when a separating path splits layout and target surface into topologically incompatible parts (which cannot occur in the simply-connected setup).



4.6.2 Global Optimality of Shortest-Path Embeddings

Our algorithm finds the shortest embedding that can be produced by a sequence of successive shortest path insertions (Section 4.3.3). Is such an embedding always optimal? At least for general graphs, there are known examples of optimal embeddings that are not the result of any insertion sequence [CHKL13] (see inset). However, none of these counterexamples directly apply to our more constrained problem setting (requiring a connected graph and fixed cyclic orderings around vertices). We are neither aware of a proof nor a counterexample of optimality in this setting and leave this question for future research.

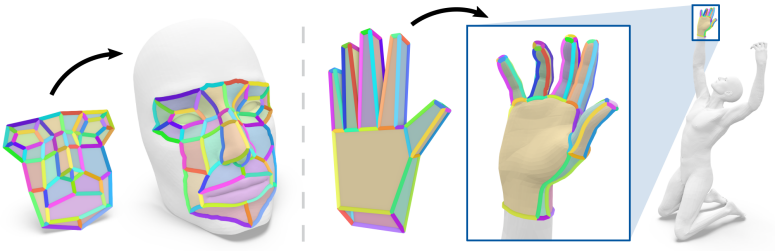


Figure 4.18: *Our algorithm can embed disk-topology layouts (face, hand) into target surfaces of disk topology (left) or genus 0 (right).*

4.6.3 Pruning Efficiency

One source of redundancy that remains undetected by our proactive pruning strategy (Section 4.4.5) arises when disjoint subsets of mutually conflicting edges can be resolved independently. In this case, insertions in different conflict components can be interleaved, leading to a number of sequences which can only later be identified as redundant (Section 4.4.4). A possible solution would be to solve component sub-problems separately. However, this is only viable if their individual results are guaranteed to remain non-conflicting, which is difficult to predict.

4.6.4 Shortest-Path Metrics

It would be interesting to explore objectives beyond total embedded path length. While different path metrics (e. g. favoring smoothness or feature alignment) can be readily integrated in our framework, a more challenging extension is the formulation of patch-based quality measures (e. g. mapping distortion), requiring new techniques for computing lower bounds.

4.6.5 Prescribed vs. Generated Layouts

A different direction is the application of our branch-and-bound optimization to more loosely constrained tasks such as shortest cut graph generation or compatible

triangulation (without a prescribed layout). Here, one has to deal with a much higher branching degree, as decisions concerning the layout connectivity and its embedding need to be considered simultaneously. The even more challenging task of automatically embedding layouts *without* prescribed vertex locations is another interesting unsolved problem.

4.7 Summary

We have introduced a novel algorithm to solve a fairly traditional problem: the computation of short, landmark-constrained layout embeddings. Effectively, our optimization strategy searches across different mapping classes, implicitly encoded as insertion sequences.

The construction of surface homeomorphisms from compatible layout embeddings follows the tacit premise that the natural map topology may be obtained as a composition of two natural layout embeddings, and that the total length of embedded edges is a suitable measure for embedding naturality. As demonstrated

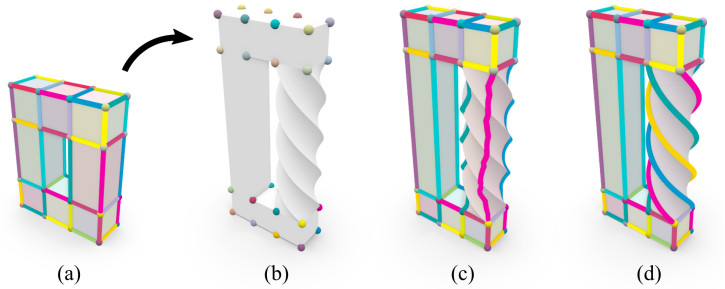


Figure 4.19: *Embedding a layout (a) into a target surface (b). Automatically generating an embedding via shortest-path insertions (c) may ignore implied topological features: Here, the intuitively correct topology (d) contains a deliberate twist around a handle.*

by our experiments in Section 4.5, these assumptions are indeed reasonable for a wide range of input models and applications.

There are however certain scenarios where total embedded edge length as a quality prior does not match human intuition, e. g. in Fig. 4.19. Here, the twisted profile of the rectangular frame also suggests a topological twist around the handle (as in Fig. 4.19 (d)), but this is ignored by a shortest-path embedding (Fig. 4.19 (c)).

In such cases, a topologically correct reconstruction requires additional control: Here, we want to deliberately avoid tracing paths in their shortest homotopy class and instead prescribe a specific topology (which needs to be globally consistent for the entire embedding). In the following chapter, we explore how to generate such global topological descriptions that allow to guide the construction of maps and embeddings.

5 Surface Map Homology Inference

The methods and results presented in this chapter have previously been published in [BSCK21].

As described in the previous chapter, a standard way to initialize a homeomorphism between two surfaces is via compatible layout embeddings. In that setting, the map's geometry and topology arises from shortest-path connections between corresponding landmarks, which are taken as hard constraints. Note that this approach presumes an accurate and reliable placement of landmarks on the two surfaces: One misplaced landmark could dramatically change the generated map topology. Furthermore, since the topology of the resulting map is only deduced from shortest-path connections between landmarks, it may easily miss topological features if landmarks are not strategically placed (as in Fig. 4.19). Therefore, this annotation task almost always requires significant human interaction or supervision.

It is an ongoing research effort to replace this largely manual annotation process by automatic correspondence detection methods. Indeed, in recent years, increasingly capable shape matching algorithms have been proposed which estimate correspondences between a given pair of shapes with no (or minimal) additional input. Promising approaches include model-driven matching techniques based on local shape descriptors and distortion priors, or supervised learning methods trained to identify high-level features. We review some of these in Section 5.1.

Many of these automatic methods ultimately produce a mostly dense (e. g. densely sampled or fuzzy) identification of the two surfaces, albeit with some degree of inaccuracy or uncertainty. In many cases, this result already closely

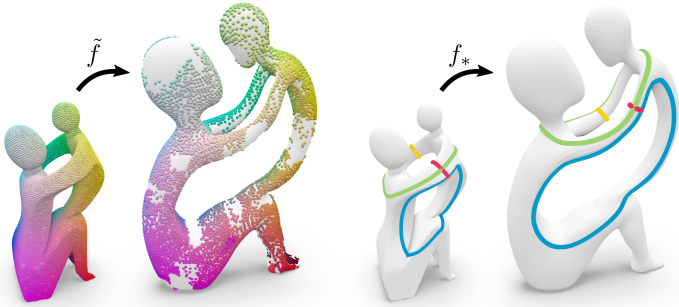


Figure 5.1: *(a): a sampled map $\tilde{f}$ between two surfaces, ridden with noise, outliers, non-injective overlaps, and gaps. (b): from this input, we infer a purely topological description in form of a homology map f^*.*

resembles or approximates a surface map which indicates a specific map topology. However, while this implied topology of the map may be quite apparent upon visual inspection, as in Fig. 5.1 (a), it is not directly accessible in automatic computations: As the approximate mapping is represented in a way that is neither truly bijective nor continuous, it is not a homeomorphism and therefore — from a technical standpoint — inherently ambiguous about map topology.

In this chapter, we propose a method that overcomes this ambiguity and infers the implied map topology from a non-homeomorphic input map (Fig. 5.1 (b)). Intuitively, this topological inference corresponds to a homeomorphic completion of the input map that consolidates ambiguities and corrects local map defects. In contrast to many previous topological initialization methods (which also includes the landmark-constrained approaches from the previous chapter), we do *not* seek a natural matching between two input shapes. Instead, we aim to extract the specific map topology encoded in a *given* input map (with very modest requirements on map representation and quality), as illustrated in Fig. 5.2.

Our inference employs the symplectic representation (Section 3.4.4) to produce a purely topological description in the form of a *homology map*, i. e. a one-to-one

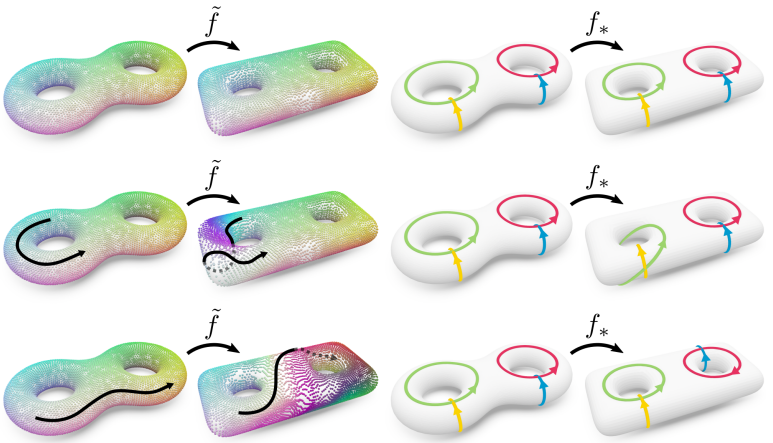


Figure 5.2: *Left: three densely sampled input maps between the same pair of shapes, describing topologically different maps, as highlighted by black curves and their images. Right: our inferred homology maps capture the implied map topology of each case.*

mapping between homology classes (of cycles) on the source and target shape. We can leverage the duality between homology and cohomology (Section 2.5.2) to derive an extremely robust and error-tolerant inference formulation in the cohomology setting. The resulting homology map is linear and can be encoded by a small integer matrix (of size $2g \times 2g$ for a pair of genus g surfaces).

We demonstrate the method's support for a large variety of input map representations and its high level of robustness to a wide range of input map imperfections, as commonly encountered in geometry processing applications. The homology map's utility is exemplified in the context of surface data transfer in a homologically correct manner and the recovery and the construction of true surface homeomorphisms from approximate input.

We begin with a review of common practical map representations, automatic shape matching techniques, and their viability for topological guidance (Section 5.1). We then explain our encoding of surface map topology, based on

the symplectic representation (Section 5.3), which is the basis for our inference algorithm (Section 5.4).

5.1 Approximate Surface Maps

Our proposed homology inference method can extract a topological description from a given input map that exemplifies a desired mapping class. Such input maps may be obtained from a variety of sources: They may be generated by fully automatic shape matching algorithms, as a by-product of some mesh processing or conversion task, or provided by a user, e. g. by manual annotation or registration. Commonly, these exemplary input maps do not come in the form of a proper (discrete) homeomorphism (cf. Section 3.1.2), but describe surface-to-surface correspondences only in some approximate, incomplete, or ambiguous way. We briefly discuss the most relevant practical approaches for representing and generating such maps.

5.1.1 Surface Map Representations

Input maps come in many different forms. The most common representations can be roughly classified into three categories: sampled maps, fuzzy maps, and extrinsic registrations.

Sampled Maps

Sample-based representations approximate a map between two surfaces $\mathcal{A}$ and $\mathcal{B}$ through a finite set of corresponding point pairs $\{(a_i, b_i)\}_i \subset \mathcal{A} \times \mathcal{B}$. If $\mathcal{A}$ and $\mathcal{B}$ are triangle meshes, the points a_i and b_i may be represented as barycentric points within triangles or simply by mesh vertices, giving rise to vertex-to-vertex, or vertex-to-surface maps.

By construction, sampled representations do not describe a full map $\mathcal{A} \to \mathcal{B}$ but only a sparse assignment. The degree of sparsity may vary, ranging from

a handful of landmarks, over unevenly distributed feature correspondences, to densely sampled vertex-to-surface maps. In addition, sampled maps can come with defects such as noise, outliers, undersampling, or non-injectivities.

From a purely theoretical perspective, sampled maps can never unambiguously identify a map topology, even in the absence of such defects: No matter how dense the sampling, one can always find an infinite number of homeomorphisms that interpolate the sample points but twist between them in topologically distinct ways (cf. Sections 3.3 and 3.4.2). Our homology inference algorithm resolves this ambiguity by extracting a map topology that corresponds to a smooth harmonic extension of the sampled map.

Fuzzy Maps

Functional maps [OBS+12] characterize a map by its action on scalar fields, i. e. how it transports real-valued functions from $\mathcal{A}$ to $\mathcal{B}$. By encoding scalar distributions as linear combinations over a set of canonical basis fields, one obtains a discrete linear map representation which can be conveniently represented by a real-valued matrix. A typical basis choice is a finite set of eigenfunctions of the Laplace-Beltrami operator, which approximate functions as smooth signals.

Due to this reduced-order encoding, functional maps by themselves may only transport low-frequency functions directly. While it is possible to recover sharp, point-wise mappings from this representation [RMC15], their continuity and bijectivity of course depends on the resolution and accuracy of the underlying functional map.

Functional maps are relevant to our work in two regards: As a common map interchange format, they may act as input for our homology inference. In a more abstract sense, our homology inference formulation itself bears a high-level resemblance to the functional maps setting: Whereas a functional map represents a surface map by its transport of scalar fields between two shapes, a homology map represents a map topology by its transport of homology classes (Section 5.3.1).

Both spaces (scalar fields and homology classes) admit a linear encoding, which enables a similar least-squares inference formulation (Section 5.4.6).

A related representation are soft maps [SNB+12] which explicitly quantify a degree of uncertainty or ambiguity in the mapping. They encode shape matchings as probability measures, describing the likelihood of correspondence between surface regions.

Extrinsic Registrations

Another common way to express correspondence between two shapes is via an extrinsic alignment or deformation that brings matching regions of the two surfaces into close spatial proximity. In simple cases, this may be achieved through a rough rigid registration or affine transformation; more generally, through non-rigid registrations that allow arbitrary deformations.

Such registrations only provide an approximate geometric alignment of the two shapes. A proper surface map may be derived from this representation by computing closest-point projections from the deformed surface $\mathcal{A}$ onto the target shape $\mathcal{B}$. As projections can introduce gaps, discontinuities, or overlaps, the resulting mapping is, again, generally not a homeomorphism.

5.1.2 Surface Map Generation

Model-Based Shape Matching

Many automatic shape matching algorithms aim to find dense correspondences based on empirical models for natural surface maps [KZHC11]: Typical objectives are a preference for low mapping distortion or a (descriptor-based) alignment of geometrically similar surface regions.

In a sampled map representation, such a matching may be phrased as an assignment problem [ASP+04], i. e. a search for a (partial) permutation between given sets of sample points on both surfaces, based on affinities defined in terms

of unary and pairwise potentials. This problem has been tackled by different relaxations and computational approaches [SY11; KKBL15; DML17; VLB+17], among them evolutionary algorithms [Sah18; EEB20], sampling-based consensus methods [MGP06; LF09], or simulated annealing [HLC20]. Other formulations maintain a fixed correspondence between samples and instead optimize their mapped positions, e. g. for map harmonicity [ESB19].

In the functional maps setting, shape matchings may be computed by finding a matrix that optimally aligns a given set of corresponding descriptor functions, e. g. in a least-squares sense [OBS+12], often combined with some operator-based regularization. Countless modifications and improvements of this setup have been suggested [OCB+17], more recently: techniques to promote map continuity [PSO18] and orientation-preservation [RPWO18], for improved regularization [RPWO19] and iterative refinement [MRR+19; HRWO20].

There are also extrinsic methods that produce shape matchings represented by coarse, large-scale deformations [ZSC+08]. Fine-grained registrations can be achieved by optimizing for surface alignment, which is often combined with additional measures e. g. mesh regularity [YLSL10] or extrinsic bending energy [EHA+19].

Data-Driven Shape Matching

Purely model-based methods reach their limits where a matching is not directly indicated by geometric properties, e. g. for highly dissimilar, non-isometric shape pairs. In these cases, natural correspondences are often only implied by a semantic identification of surface regions, which is extremely difficult to model universally.

In response, there is an increasing number of data-driven methods that can learn to imitate class-specific shape correspondences from a given training dataset [XKH+16; Sah20]. Examples are the supervised learning of vertex-to-vertex maps [RRW+14; BMRB16; LDCK18]), functional maps [COC14; LRR+17], or mutual deformations [WCMN19; JHTG20; YAK+20].

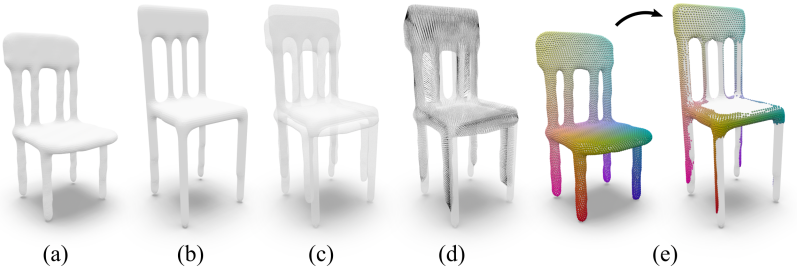


Figure 5.3: *Two shapes (a, b) are alined by a rough rigid registration (c). Closest-point projections (d) produce a sampled surface map (e) which roughly matches corresponding features but contains various local defects (gaps, non-injectivities).*

Manual Registration

For somewhat similar shapes (such as Fig. 5.3 (a, b)), a very rough correspondence may be initialized from a simple extrinsic alignment or coarse deformation [LSLC05; KS06], which can be constructed manually with minimal effort (Fig. 5.3 (c)), and converted to a sampled map via closest-point projections (Fig. 5.3 (d, e)). While locally, the quality of such ad-hoc generated maps may be extremely poor, they often still correctly capture certain global properties of the map, i. e. the matching of extremities or topological features.

5.1.3 Discussion

Virtually all of the above shape matching methods generally produce non-homeomorphic maps, partly due to inherent limitations of the underlying map representations, partly due to ambiguities or inaccuracies in the inference process.

Meanwhile, certain processing tasks depend on high-quality input maps in the form of actual homeomorphisms: A common task is the lossless transfer of highly intricate surface data, e. g. parametrizations, texture coordinates with seams, or embedded curves and path networks — which requires a truly continuous map.

Injectivity-preserving map optimization methods [SAPH04; SBCK19; SCBK20] also necessarily operate in this setting.

This compatibility gap between shape matching and homeomorphic map processing algorithms calls for an automatic method to recover valid homeomorphisms from approximate, imperfect surface maps. A first step towards such a reconstruction is a detection of the intended map topology which gracefully handles input map defects and resolves ambiguities. In this regard, a landmark-based layout embedding approach as in Chapter 4 is not suitable as it takes point correspondences as hard constraints. Instead, we propose a global field-based topology inference formulation that forgives local errors and consolidates incomplete or contradictory measurements.

5.2 Contribution

We introduce a method that takes a topologically ambiguous surface map as input and infers an unambiguous topological description in the form of a homology map. The method exhibits the following properties:

- It supports a large variety of input map types, from dense vertex-to-vertex and vertex-to-triangle maps, over sparse correspondences, to soft or functional maps, and extrinsic registrations.
- It exhibits a favorable level of resilience to a wide range of map imperfections (e. g. noise, outliers, undersampling, uncertainty, fuzzyness, or local non-injectivity).
- It prevents degeneration of the homology map regardless of input map quality.

Technically, the method's robustness in the face of imperfect maps hinges on three main ingredients:

- Instead of attempting to directly transfer and compare *local* and *high-frequency* information (like surface paths or cycles) via an unreliable map,

we carefully choose *global* and *low-frequency* operations, reducing sensitivity to imperfections. Only maximally smooth data is mapped, and compared globally.

- Differential information is mapped effectively without reference to the map's unreliable differential, by encoding it as coordinate-free scalar information.
- Further resilience to low-quality and even misguiding input is achieved by incorporating a global topological consistency constraint, restricting the inference process to the subspace of homology maps that are compatible with homeomorphisms.

Computationally, our inference task in its core boils down to solving an integer quadratic program of modest size, $O(g^2)$ for genus g surfaces, which can be efficiently handled by off-the-shelf branch-and-bound solvers. Before we delve into the details of our inference algorithm (Section 5.4), we review some theoretical foundations of our method, i. e. the underlying homology-based representation of map topology.

5.3 The Symplectic Representation

In this chapter, we operate exclusively with the symplectic representation of the mapping class group (Section 3.4.4), which describes the topology of surface maps through their action on homology classes of cycles (or, as we will see, equivalently: cohomology classes of cocycles). In the following, we recall all relevant concepts and settle the notation for the rest of this chapter.

Throughout, we consider two closed surfaces, $\mathcal{A}$ and $\mathcal{B}$, both of genus g, which are connected by a surface map $f : \mathcal{A} \to \mathcal{B}$. For now, we will assume that f is a smooth orientation-preserving homeomorphism between the two shapes which unambiguously defines a map topology. Later on, we will consider the case where f is replaced by an approximate, possibly degraded, and topologically ambiguous non-homeomorphism $\tilde{f}$.

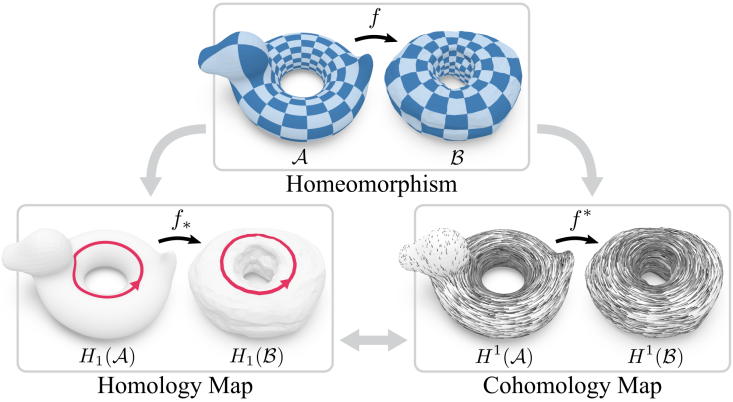


Figure 5.4: *A homeomorphism f between surfaces $\mathcal{A}$, $\mathcal{B}$ (top) induces a homology map f_* (bottom left) that maps between homology classes (equivalence classes of cycles) of $\mathcal{A}$ and $\mathcal{B}$, and a corresponding cohomology map f^* (bottom right) that maps between cohomology classes (equivalence classes of cocycles).*

5.3.1 Induced Homology Maps

The objects of homology are cycles, i. e. linear combinations of oriented loops (see Section 2.4). Since f is a homeomorphism, we can use it to transfer such cycles from $\mathcal{A}$ to $\mathcal{B}$: If c is a cycle on $\mathcal{A}$, then the image $f(c)$ is again a cycle on $\mathcal{B}$.

This action of f on cycles also induces a *homology map* $f_* : H_1(\mathcal{A}) \to H_1(\mathcal{B})$, defined by

$$f_*([c]) = [f(c)], \tag{5.1}$$

which is a well-defined isomorphism between the homology groups $H_1(\mathcal{A})$ and $H_1(\mathcal{B})$ of the two surfaces (Section 3.4.4). It associates every homology class of $\mathcal{A}$ with a unique homology class of $\mathcal{B}$ (Fig. 5.4, bottom left).

5.3.2 Induced Cohomology Maps

Cohomology considers cocycles, i. e. closed 1-forms on surfaces (see Section 2.5). Let $x \in T\mathcal{A} \to \mathbb{R}$ be a cocycle on $\mathcal{A}$. We can again use the map f to transfer this cocycle to surface $\mathcal{B}$: For any point $p \in \mathcal{B}$ and tangent vector $t \in T_p\mathcal{B}$, the value of the mapped cocycle is computed by pulling back t through the inverse of the map differential $\mathrm{d}f$ and then evaluating the original cocycle x at the preimage of p on $\mathcal{A}$, formally [Lee13, Ch. 11.3]:

$$x_{f^{-1}(p)}\left(\left(\mathrm{d}_{f^{-1}(p)}f\right)^{-1}(t)\right). \tag{5.2}$$

This mapping fully defines the image of x on $\mathcal{B}$, which we will denote by $f(x)$ in the following. It is again a cocycle $T\mathcal{B} \to \mathbb{R}$.

In analogy to Eq. (5.1), this action of the map f on cocycles also induces a *cohomology map* $f^* : H^1(\mathcal{A}) \to H^1(\mathcal{B})$ by

$$f^*([x]) = [f(x)], \tag{5.3}$$

which indeed describes a well-defined map that associates each cohomology class of $\mathcal{A}$ with a cohomology class of $\mathcal{B}$ [Lee13, Prop. 17.2]. If f is a homeomorphism, the induced cohomology map f^* is, in fact, an isomorphism between the cohomology groups of the two surfaces, $H^1(\mathcal{A})$ and $H^1(\mathcal{B})$ [Lee13, Thm. 17.11].

Note how the cocycle mapping according to Eq. (5.2) requires a well-defined map differential. Thus, to induce a valid bidirectional cocycle mapping, the map f must be a *diffeomorphism*, i. e. both f and its inverse f^{-1} must be smooth functions. However, in practice, most input maps are non-smooth or even non-continuous. We handle such non-diffeomorphic inputs with an alternative cocycle mapping formulation (Section 5.4.5) that avoids any direct reference to the (potentially unavailable or unreliable) map differential.

5.3.3 Homology Maps Between Bases

In practice, we use a coefficient-based representation to describe homology classes on $\mathcal{A}$ and $\mathcal{B}$. For each surface, we pick a homology basis, i. e. a generating set of cycles $B_\mathcal{A} = \{a_1, \dots, a_{2g}\}$ and $B_\mathcal{B} = \{b_1, \dots, b_{2g}\}$, as introduced in Section 2.4.3. Then, we may encode any homology class as an integer linear combination of basis cycles: The coefficient vectors $h_\mathcal{A} \in \mathbb{Z}^{2g}$ and $h_\mathcal{B} \in \mathbb{Z}^{2g}$ represent the homology classes $[B_\mathcal{A} h_\mathcal{A}] \in H_1(\mathcal{A})$ and $[B_\mathcal{B} h_\mathcal{B}] \in H_1(\mathcal{B})$.

In this encoding, a homology map $f_* : H_1(\mathcal{A}) \to H_1(\mathcal{B})$ becomes a linear map $M : \mathbb{Z}^{2g} \to \mathbb{Z}^{2g}$ between coefficient vectors, i. e. a matrix $M \in \mathbb{Z}^{2g \times 2g}$. For every $h_\mathcal{A} \in \mathbb{Z}^{2g}$, it is

$$f_*([B_\mathcal{A} h_\mathcal{A}]) = [B_\mathcal{B} M h_\mathcal{A}]. \tag{5.4}$$

As stated in Section 3.4.4, any matrix M that fulfills the *symplectic constraint*

$$\Omega_\mathcal{A} = M^\mathsf{T} \Omega_\mathcal{B} M \tag{5.5}$$

encodes a topologically valid homology map, i. e. one that is induced by some orientation-preserving homeomorphism $f \in \mathrm{Homeo}(\mathcal{A}, \mathcal{B})$. The integer matrices $\Omega_\mathcal{A} \in \mathbb{R}^{2g \times 2g}$ and $\Omega_\mathcal{B} \in \mathbb{R}^{2g \times 2g}$ are the inner product matrices of the algebraic intersection forms for homology classes encoded w. r. t. $B_\mathcal{A}$ and $B_\mathcal{B}$ (as defined in Section 2.4.4).

5.3.4 Cohomology Maps Between Bases

Analogously, we can represent cohomology classes on $\mathcal{A}$ and $\mathcal{B}$ using coefficients w. r. t. cohomology bases, defined by sets of cocycles $B^\mathcal{A} = \{a^1, \dots, a^{2g}\}$ and $B^\mathcal{B} = \{b^1, \dots, b^{2g}\}$, as defined in Section 2.5.3. We assume that $B^\mathcal{A}$ and $B^\mathcal{B}$ are *dual cohomology bases* to the given homology bases $B_\mathcal{A}$ and $B_\mathcal{B}$ in the sense that

$$\int_{a_i} a^j = \delta_{ij} \qquad \text{and} \qquad \int_{b_i} b^j = \delta_{ij}$$

for any i and j. We will discuss how to construct such dual bases in practice in Section 5.4.3.

Mirroring Eq. (5.4) for homology maps, we can represent any cohomology map f^* as an integer matrix $\overline{M} \in \mathbb{Z}^{2g \times 2g}$, defined by

$$f^*([B^{\mathcal{A}}h]) = [B^{\mathcal{B}}\overline{M}(h)]. \tag{5.6}$$

The dual choice of homology and cohomology bases implies a direct relation between the matrix representations of the homology and cohomology maps M and $\overline{M}$. This follows from the fact that a homeomorphism f preserves integrals: On surface $\mathcal{A}$, we can compute the integral of a cocycle x along a cycle c. Then, the images of x and c on $\mathcal{B}$ under a homeomorphism f will integrate to the same value:

$$\int_c x = \int_{f(c)} f(x),$$

which, due to homology and cohomology invariance (Section 2.5.2), also holds for classes, i. e.

$$\int_{[c]} [x] = \int_{[f(c)]} [f(x)].$$

We apply Eqs. (5.1) and (5.3) on the right-hand side to substitute for the induced homology and cohomology maps:

$$\int_{[c]} [x] = \int_{f_*([c])} f^*([x]).$$

We now express $[c]$ and $[x]$ using coefficients $h_c \in \mathbb{Z}^{2g}$ and $h_x \in \mathbb{Z}^{2g}$ w. r. t. the given bases $B_{\mathcal{A}}$ and $B^{\mathcal{A}}$ as $[c] = [B_{\mathcal{A}}h_c]$ and $[x] = [B^{\mathcal{A}}h_x]$. Similarly, on the right-hand side, we replace the homology and cohomology maps by their matrix representations M and $\overline{M}$, as defined in Eqs. (5.4) and (5.6), and get

$$\int_{[B_{\mathcal{A}}h_c]} [B^{\mathcal{A}}h_x] = \int_{[B_{\mathcal{B}}Mh_c]} [B^{\mathcal{B}}\overline{M}h_x].$$

Due to the duality of $B_{\mathcal{A}}$ to $B^{\mathcal{A}}$ and $B_{\mathcal{B}}$ to $B^{\mathcal{B}}$, these integrals can be computed simply as standard dot products on the coefficient vectors, according to Eq. (2.7) from Section 2.5.3. We obtain

$$\langle h_c, h_x \rangle = \langle M h_c, \overline{M} h_x \rangle,$$

which must hold for all h_c and h_x. We can therefore rearrange to

$$\overline{M} = M^{-\mathsf{T}}, \tag{5.7}$$

implying we can convert between matrix representations of homology and cohomology maps by simply taking the inverse transpose.

Recall that any valid homology matrix M must conform to the symplectic constraint (5.5), i. e. $\Omega_{\mathcal{A}} = M^{\mathsf{T}} \Omega_{\mathcal{B}} M$. With Eq. (5.7), we can simply rephrase this constraint in terms of the cohomology matrix $\overline{M}$ as

$$\overline{M} \Omega_{\mathcal{A}} \overline{M}^{\mathsf{T}} = \Omega_{\mathcal{B}}, \tag{5.8}$$

which we call the *dual symplectic constraint*. This ability to effortlessly convert between homology and cohomology representations of the same map topology is the foundation of our inference method.

5.4 Map Homology Inference

Equipped with this background on homology and cohomology aspects of maps between surfaces, we are now ready to precisely state the problem we address and the algorithmic approach we propose.

5.4.1 Problem Statement

Our method takes as input:

- A pair of closed surfaces $\mathcal{A}$, $\mathcal{B}$, both of genus g, represented by triangle meshes $(V_{\mathcal{A}}, E_{\mathcal{A}}, F_{\mathcal{A}})$, $(V_{\mathcal{B}}, E_{\mathcal{B}}, F_{\mathcal{B}})$, i. e. cell complexes with piecewise affine embeddings in $\mathbb{R}^3$.

- A map $\tilde{f}$ that loosely resembles a homeomorphism $f : \mathcal{A} \to \mathcal{B}$.

We support any map representation that can be used to transfer scalar quantities from $\mathcal{A}$ to $\mathcal{B}$ in some way. This includes sharp dense point-wise maps (e. g. vertex correspondences) and non-sharp representations such as functional maps or soft maps. Sparse maps given by a finite set of corresponding surface points $(p_{\mathcal{A}}, p_{\mathcal{B}}) \in \mathcal{A} \times \mathcal{B}$, e. g. samples or landmarks, can be used as well via interpolation (Section 5.4.5).

Homology bases $B_{\mathcal{A}}$ and $B_{\mathcal{B}}$ of $\mathcal{A}$ and $\mathcal{B}$, represented by $2g$ cycles of oriented edges per mesh, may be provided as input or be generated automatically, e. g. using the tree-cotree method [EW05], as described in Section 2.3.4.

From this input, we infer a homology map $f_* : H_1(\mathcal{A}) \to H_1(\mathcal{B})$, represented by a matrix $M \in \mathbb{Z}^{2g \times 2g}$ that maps between $B_{\mathcal{A}}$ and $B_{\mathcal{B}}$ such that

- the inference of f_* is guided by the input map $\tilde{f}$;
- there is a true homeomorphism $f : \mathcal{A} \to \mathcal{B}$ that induces f_*.

In particular, if the imperfect input map $\tilde{f}$ is close to a homeomorphism f, we expect the homology map inferred from $\tilde{f}$ to be the one induced by f. In any case, no matter how bad the input, we ensure that the homology map will never degenerate topologically.

5.4.2 Algorithm Overview

Since $\tilde{f}$ is possibly sparse and non-injective, we cannot use it directly to map cycles (i. e., homology representatives) from $\mathcal{A}$ to $\mathcal{B}$, which would immediately allow to determine the matrix M of the desired induced homology map f_*. One could try mapping some points of a cycle from $\mathcal{A}$ to $\mathcal{B}$, to the extent $\tilde{f}$ permits, and reconstruct a cycle from the images on $\mathcal{B}$. A major downside of such a *local* approach is its sensitivity to local imperfections such as outliers or noise in the map and consequently its heavy dependence on the choice of cycle. Our aim is to use a *global* approach.

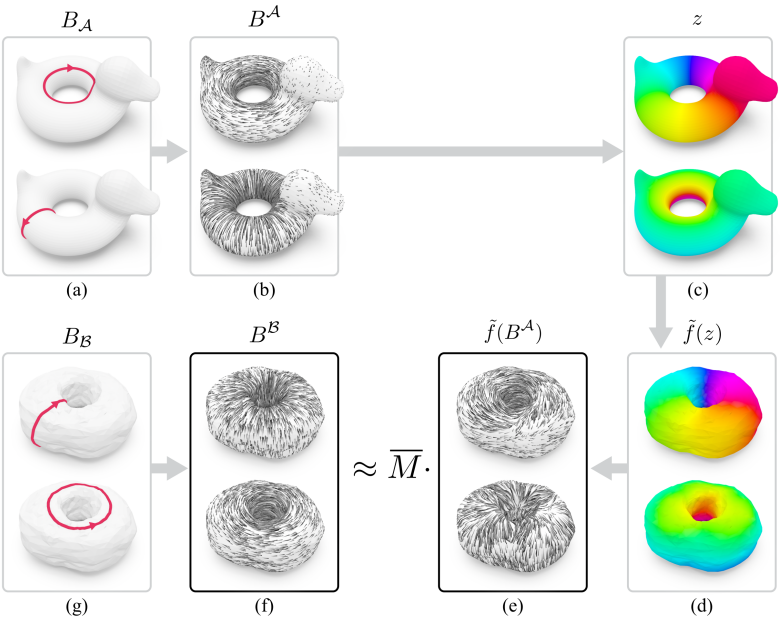


Figure 5.5: *From a homology basis of surface $\mathcal{A}$ (a), we compute a dual cohomology basis of cocycles (b). We encode these cocycles on $\mathcal{A}$ as periodic scalar fields z (c) which are then transferred via the input map $\tilde{f}$ onto $\mathcal{B}$ (d), where we differentiate to obtain cocycles again (e). We find an integer matrix $\overline{M}$ that best matches the transferred cocycles against combinations of cohomology basis representatives on $\mathcal{B}$ (f).*

To this end, instead of local high-frequency information, we map global low-frequency information via $\tilde{f}$. Concretely, we transport maximally smooth cocycles (acting as cohomology representatives). As the input map $\tilde{f}$ cannot be assumed to be differentiable (as is required to directly map differential information like cocycles, i. e. 1-forms), we take a detour via an encoding as (periodic) scalar fields — that are easily mapped by a wide variety of types of commonly used imperfect maps.

Algorithmically, our approach can be summarized as follows (see Fig. 5.5 for a visual overview):

1. create a cohomology basis $B^{\mathcal{A}}$ of $2g$ smooth cocycles (Section 5.4.3)
2. encode $B^{\mathcal{A}}$ as periodic scalar potentials (Section 5.4.4)
3. map them to $\mathcal{B}$ using $\tilde{f}$ with interpolation (Section 5.4.5)
4. decode the mapped potentials to yield cocycles on $\mathcal{B}$ (Section 5.4.4)
5. collectively match them against a canonical representation of $B^{\mathcal{B}}$, subject to symplectic constraint (5.8), to infer a cohomology map f^*, represented by a matrix $\overline{M}$ (Section 5.4.6)

Finally, we convert to $M = \overline{M}^{-\mathsf{T}}$ to obtain a representation of the desired homology map f_*, compatible with some homeomorphism f as a consequence of constraint (5.8), equivalent to (5.5).

5.4.3 Harmonic Cohomology Representatives

By Hodge theory [War83, Ch. 6], for each cohomology class $[x]$ there exists a unique representative cocycle x that is *harmonic*, i. e. $\Delta x = 0$, thus as smooth as possible. We adopt this choice to construct canonical representatives on $\mathcal{A}$ and $\mathcal{B}$, see Fig. 5.5 (b).

For each cycle a_j of our (input) homology basis $B_{\mathcal{A}}$, we compute a corresponding dual cohomology representative $a^j \in B^{\mathcal{A}}$ that is the unique 1-form that fulfills the three constituting properties

$$\mathrm{d}a^j = 0 \qquad (a^j \text{ closed}), \tag{5.9}$$

$$\Delta a^j = 0 \qquad (a^j \text{ harmonic}), \tag{5.10}$$

$$\textstyle\int_{a_i} a^j = \delta_{ij} \quad \text{for all } i \qquad (a^j \text{ dual to } a_j). \tag{5.11}$$

In our discrete setting, we encode each cocycle as a column vector $a^j = \mathbb{R}^{|E_{\mathcal{A}}|}$ of scalar values assigned to oriented mesh edges (cf. Section 2.5.4). Then, as

described by [GY02; GY03], properties (5.9)–(5.11) discretize to a linear system of equations, which can be solved simultaneously for all a^j. The result is a set of discrete harmonic cohomology generators, encoded as columns of a matrix $B^{\mathcal{A}} \in \mathbb{R}^{|E_{\mathcal{A}}| \times 2g}$. We repeat the same process on $\mathcal{B}$ to compute $B^{\mathcal{B}} \in \mathbb{R}^{|E_{\mathcal{B}}| \times 2g}$. Note that (5.9) and (5.11) imply that these cocycles are *integral*, i. e., they have integer integrals along all cycles (cf. Section 2.5.2), in particular 1 along respective dual basis cycles.

5.4.4 Periodic Encoding

Encoding

Given an integral cocycle a^j (discretized as an assignment $a^j : E_{\mathcal{A}} \to \mathbb{R}$), we compute a real-valued scalar potential $\phi^j : V \to \mathbb{R}$, defined on vertices, by integrating a^j along edges. Starting at some root vertex, we propagate ϕ^j by summing up integrated values of a^j along an arbitrary spanning tree. We then define a complex field $z^j : V \to \mathbb{C}$ as

$$z^j = \mathrm{e}^{\phi^j 2\pi \mathrm{i}},$$

i. e., we use ϕ^j as the argument of a unit complex number, see Fig. 5.5 (c).

While ϕ^j depends on the choice of tree and has discontinuities (across the tree's cut locus), z^j is *continuous* and *unique*, i. e., independent of the tree and its root (up to a constant phase shift). This is due to the fact that a^j is closed and integral, implying that ϕ^j has integer jumps at all discontinuities, which may be shown as follows:

Consider a cocycle a^j on $\mathcal{A}$ such that $\int_c a^j \in \mathbb{Z}$ for any cycle c. According to the Poincaré lemma [Lit17, Thm. 19], on any contractible domain, there exists a 0-form (i. e., scalar field) ϕ that has a^j as its gradient, i. e. $a^j = \mathrm{d}\phi$. If $\mathcal{A}$ is of genus $g > 0$, its universal cover C is contractible [Lüc08, Sec. 2.2]. We can lift a^j to C as $\hat{a}^j$, which is a cocycle on C. Therefore, there exists some 0-form $\phi : C \to \mathbb{R}$ on the universal cover with $\mathrm{d}\phi = \hat{a}^j$. Then, ϕ is periodic in the sense

that $\phi(c_1) = \phi(c_2) + n$ with $n \in \mathbb{Z}$ for any pair of points $c_1, c_2 \in C$ with identical projections $p(c_1) = p(c_2)$: As $p(c_1) = p(c_2)$, there is a path γ connecting c_1 and c_2 in C. Via Stokes' theorem, $\phi(c_2) - \phi(c_1) = \int_{\partial\gamma} \phi = \int_\gamma \mathrm{d}\phi = \int_\gamma \hat{a}^j$. Since $\hat{a}^j$ is a lift of a^j, it is $\int_\gamma \hat{a}^j = \int_{p(\gamma)} a^j$. The projection $p(\gamma)$ onto $\mathcal{A}$ is a closed curve, i. e. a cycle, hence $\int_{p(\gamma)} a^j = n$ is an integer.

Decoding

A cocycle $\hat{a}^j$ is easily recovered from the complex field z^j: On each edge $e_{pq} \in E$, we compute the value of $\hat{a}^j$ as the smallest angle difference between the arguments of z_p^j and z_q^j:

$$\hat{a}_{pq}^j = \frac{1}{2\pi} \arg\left(z_q^j / z_p^j\right). \tag{5.12}$$

where $\arg(z) \in [-\pi, \pi)$ is the argument of $z \in \mathbb{C}$. Note that the recovery is exact ($\hat{a}^j = a^j$) as long as $|a_{pq}^j| < \frac{1}{2}$ for each edge e_{pq}.

5.4.5 Periodic Potential Mapping

If the input map representation $\tilde{f}$ directly allows the transfer of scalar fields from $V_\mathcal{A}$ to $V_\mathcal{B}$ (as is the case e. g. for functional maps), we can immediately apply it to transfer the periodic potential z^j of each cohomology representative a^j from

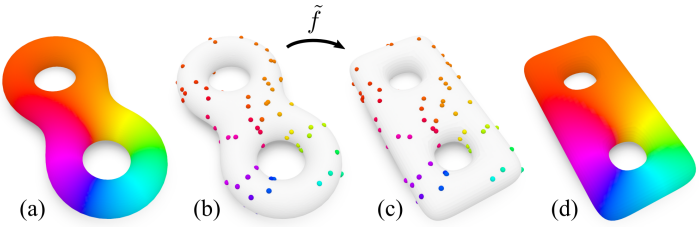


Figure 5.6: *Transport of a smooth periodic potential under a sparse map $\tilde{f}$ between surfaces: Samples from a complex field (a) on $\mathcal{A}$ are mapped via point-to-point correspondences (b-c) and harmonically extended to a dense, smooth field on $\mathcal{B}$ via interpolation (d).*

$\mathcal{A}$ to $\mathcal{B}$ as $\tilde{f}(z^j) \in V_\mathcal{B} \to \mathbb{C}$, see Fig. 5.5 (d). Otherwise, if $\tilde{f}$ only defines sparse correspondences $(p_\mathcal{A}, p_\mathcal{B}) \in \mathcal{A} \times \mathcal{B}$, we define $\tilde{f}(z^j)$ on $\mathcal{B}$ as a field $\tilde{z}^j$ that smoothly interpolates sparse mapped values (Fig. 5.6). One simple way is to obtain $\tilde{z}^j$ as a least-squares minimizer of a set of interpolation and smoothness terms: For each correspondence $(p_\mathcal{A}, p_\mathcal{B})$, we define a fitting term $\|\tilde{z}^j(p_\mathcal{B}) - z^j(p_\mathcal{A})\|^2$ (using barycentric interpolation if $p_\mathcal{B}$ lies inside a face). We add the harmonicity term $\|\Delta\, \tilde{z}^j\|^2$ (discretized by the cotangent Laplacian [PP93] on the triangle mesh $\mathcal{B}$) as a regularizer with a small weight, 10^{-3} in our experiments.

We then decode the periodic potential $\tilde{f}(z^j)$ for each j and denote the resulting set of cocycles by $\tilde{f}(B^\mathcal{A}) \in \mathbb{R}^{|E_\mathcal{B} \times 2g|}$, see Fig. 5.5 (e). While for maps of very low quality or high distortion and sparsity the period ambiguity of (5.12) might locally be resolved in a way that does match the intent, the global nature of the following procedure gracefully deals with such sporadic effects.

5.4.6 Symplectic Cohomology Matching

We can now infer a cohomology map f^* (represented by $\overline{M} \in \mathbb{Z}^{2g \times 2g}$) from the above defined action of $\tilde{f}$ on $B^\mathcal{A}$. According to (5.3), the cohomology map f^* is induced by the images of cocycles from $\mathcal{A}$ under $\tilde{f}$. In particular, for any basis element $a^j \in B^\mathcal{A}$, it is

$$f^*([a^j]) = [\tilde{f}(a^j)]. \tag{5.13}$$

In coefficients: $[a^j] = [B^\mathcal{A} e_j]$, where $e_j \in \mathbb{Z}^{2g}$ is a standard unit vector (where all entries are 0 except for a 1 in the j-th component). Using (5.6), we can represent f^* by $\overline{M}$, so (5.13) becomes

$$[B^\mathcal{B} \overline{M} e_j] = [\tilde{f}(a^j)].$$

Therefore, for any basis cycle $b_i \in B_\mathcal{B}$, we have

$$\int_{b_i} [B^\mathcal{B} \overline{M} e_j] = \int_{b_i} [\tilde{f}(a^j)].$$

With $b_i = B_{\mathcal{B}} e_i$ and Eq. (2.7) the left-hand side becomes $\langle e_i, \overline{M} e_j \rangle = \overline{M}_{ij}$. On the right-hand side, we can drop the brackets and obtain

$$\overline{M}_{ij} = \int_{b_i} \tilde{f}(a^j), \quad \text{or in matrix form:} \quad \overline{M} = B_{\mathcal{B}} \tilde{f}(B^{\mathcal{A}}). \tag{5.14}$$

Having computed the images $\tilde{f}(a^j)$ of basis cocycles (Section 5.4.5), we can use (5.14) to immediately read off the coefficients of $\overline{M}$, representing the induced cohomology map f^*.

For input maps $\tilde{f}$ that are dense and reasonably close to a homeomorphism, (5.14) will recover its induced cohomology map. However, in the presence of map defects like non-injectivities or undersampling of topological features, it is possible that the interpolated images $\tilde{f}(a^j)$ "snap" into unintended cohomology classes, become linearly dependent, or degenerate to trivial exact cocycles. By additionally enforcing the symplectic constraint (5.8), it can be ensured that, regardless of such deficiencies, the inferred cohomology map is compatible with a true homeomorphism. This can be achieved by satisfying (5.14) in the constrained least squares sense:

$$\begin{aligned} \min_{\overline{M}} \quad & \left\| \overline{M} - B_{\mathcal{B}} \tilde{f}(B^{\mathcal{A}}) \right\|_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \overline{M} \Omega_{\mathcal{A}} \overline{M}^{\mathsf{T}} = \Omega_{\mathcal{B}}, \quad \overline{M} \in \mathbb{Z}^{2g \times 2g}. \end{aligned} \tag{5.15}$$

In practice, we observe that a variation of this formulation is much more robust in particular to low-quality input maps $\tilde{f}$: Instead of comparing cycle integrals of cocycles (as effectively done in (5.15)), we infer $\overline{M}$ by directly and globally measuring similarity of these cocycles. This corresponds to a minor change to (5.15):

$$\begin{aligned} \min_{\overline{M}} \quad & \left\| B^{\mathcal{B}} \overline{M} - \tilde{f}(B^{\mathcal{A}}) \right\|_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \overline{M} \Omega_{\mathcal{A}} \overline{M}^{\mathsf{T}} = \Omega_{\mathcal{B}}, \quad \overline{M} \in \mathbb{Z}^{2g \times 2g}, \end{aligned} \tag{5.16}$$

i. e. a multiplication with $B^{\mathcal{B}}$ from the left (and using $B^{\mathcal{B}} B_{\mathcal{B}} = I$, as implied by Eqs. (2.6) and (2.7)). This is a quadratic integer problem under quadratic equality

constraints. The solution variables are the $(2g)^2$ integer entries of $\overline{M}$. The matrix equality constraint (5.8) can be encoded coefficient-wise; due to antisymmetry of $\Omega_{\mathcal{A}}$ and $\Omega_{\mathcal{B}}$ (which are constant matrices computed from the intersection patterns of the chosen bases $B_{\mathcal{A}}$, $B_{\mathcal{B}}$, see Section 2.4.4), it suffices to encode the strictly upper triangular part. Due to the small problem size, (5.16) can be efficiently optimized by off-the-shelf branch-and-bound solvers. We use Gurobi [Gur21] in our experiments. After a cohomology map $\overline{M}$ has been computed, we obtain the desired homology map as $M = \overline{M}^{-\mathsf{T}}$.

Compared to the original problem (5.15), the reformulation in (5.16) can be vastly more resilient as it no longer depends on integrals along (arbitrary) generator cycles but only on global measurements. Thus, even in the presence of local map defects, the correct cohomology can often still be inferred from correct alignment of cocycles in reliable regions. The symplectic constraint (5.8) on $\overline{M}$ further restricts the search space by ruling out homology maps incompatible with any homeomorphism, globally regularizing the process.

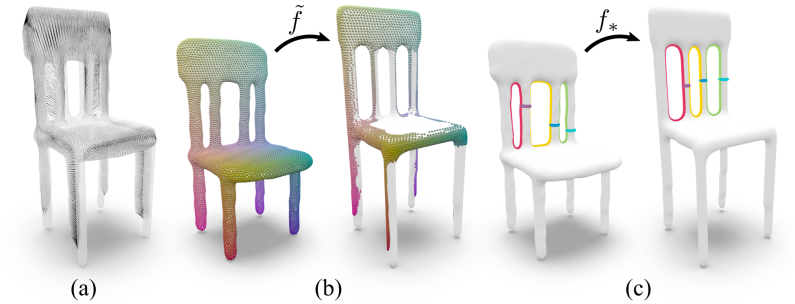


Figure 5.7: *A sampled surface map, generated via rigid registration and closest point projections (a), as in Fig. 5.3. The resulting map $\tilde{f}$ is discontinuous and heavily undersamples or double-covers certain regions of the target surface (b). Still, our inference (c) produces the correct homology map f_*.*

5.5 Results and Applications

We demonstrate the applicability of our homology inference method for different input map representations (Section 5.5.1) and its robustness to various map defects (Section 5.5.2). The computed homology maps can be used to transfer data in a homologically correct manner (Section 5.5.3) and to topologically control the construction of compatible surface decompositions, e. g. cut graphs or layout embeddings (Sections 5.5.4 and 5.5.5), and therefore proper homeomorphisms.

To visualize homology classes we will show smooth representative cycles, selected for clarity and visibility.

5.5.1 Input Map Representations

We show examples of homology maps inferred from different types of input maps: densely sampled (Fig. 5.2) or functional maps (Fig. 5.8), and sparse (Fig. 5.9) or partial correspondences (Fig. 5.16).

Due to its high tolerance to map defects, our inference method can be used to quickly generate valid topological correspondences from ad-hoc inputs. One example are vertex-to-surface maps obtained by roughly aligning two shapes in ambient space (e. g. by a rigid transformation) and generating correspondences via simple closest-point projections from one surface to the other (Fig. 5.7 (a)). While

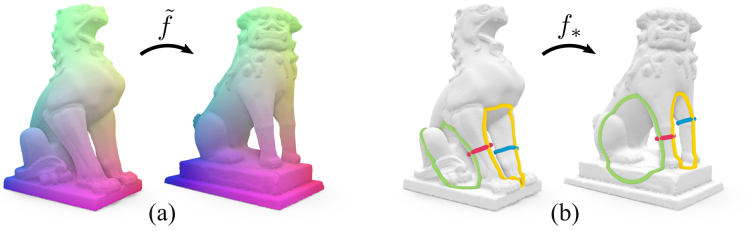


Figure 5.8: *(a): A functional map $\tilde{f}$ represented by 50×50 coefficients. (b): The homology map f_* obtained from $\tilde{f}$ by our inference.*

such maps are typically discontinuous and non-injective (Fig. 5.7 (b)), they often suffice for the inference of the desired homology map (Fig. 5.7 (c)).

5.5.2 Robustness

In Fig. 5.10 we evaluate the robustness of our method with respect to map imperfections such as noise, outliers, and sparsity. Starting from a dense high-quality input map, we progressively degrade the quality of the (sampled) map until the inferred result switches from the original homology map to a different one. First, we simulate noise by displacing map samples on the target surface along random geodesic paths. Our method still tolerates an average geodesic displacement of 24% of the bounding box diagonal and only switches to a different homology at 32%. Second, we examine resilience towards outliers by randomly permuting a subset of samples. Due to the global nature of our objective (5.16), map inference succeeds even in the presence of 60% outliers. Third, we observe robustness towards very sparse inputs: dropping all but 0.5% (12 samples) of the input map still yields the correct homology in this experiment. Figure 5.11 correspondingly reports the cohomology matrices $\overline{M}$ obtained in these experiments. To demonstrate the effect of the symplectic constraint (5.5), we also show the

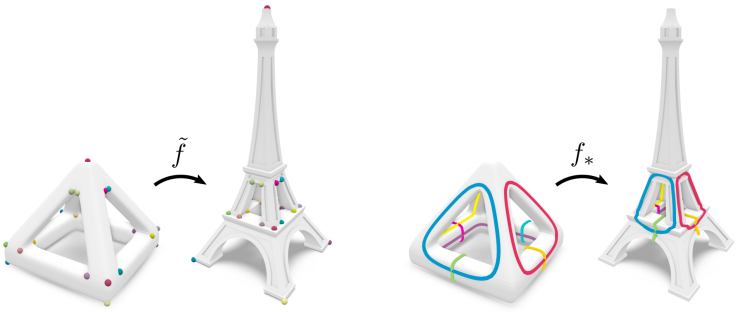


Figure 5.9: *Homology map f_* computed from an input map $\tilde{f}$ represented by a very sparse set of 29 landmark correspondences on a pair of genus 4 surfaces.*

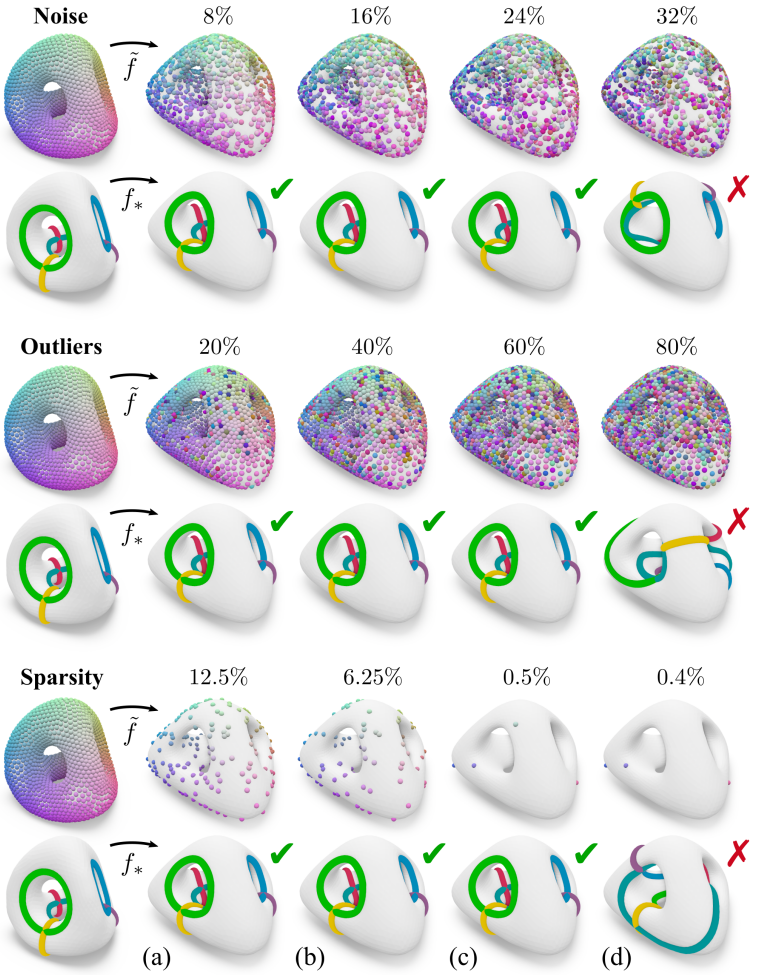


Figure 5.10: *Robustness towards different input map defects. Noise: samples are randomly displaced by an average geodesic distance. Outliers: a subset of samples is randomly shuffled. Sparsity: only a small subset of samples is kept (11 samples in (c), 10 in (d)). Our method infers the correct homology map in columns (a-c) and only reports a different (but valid) homology in column (d).*

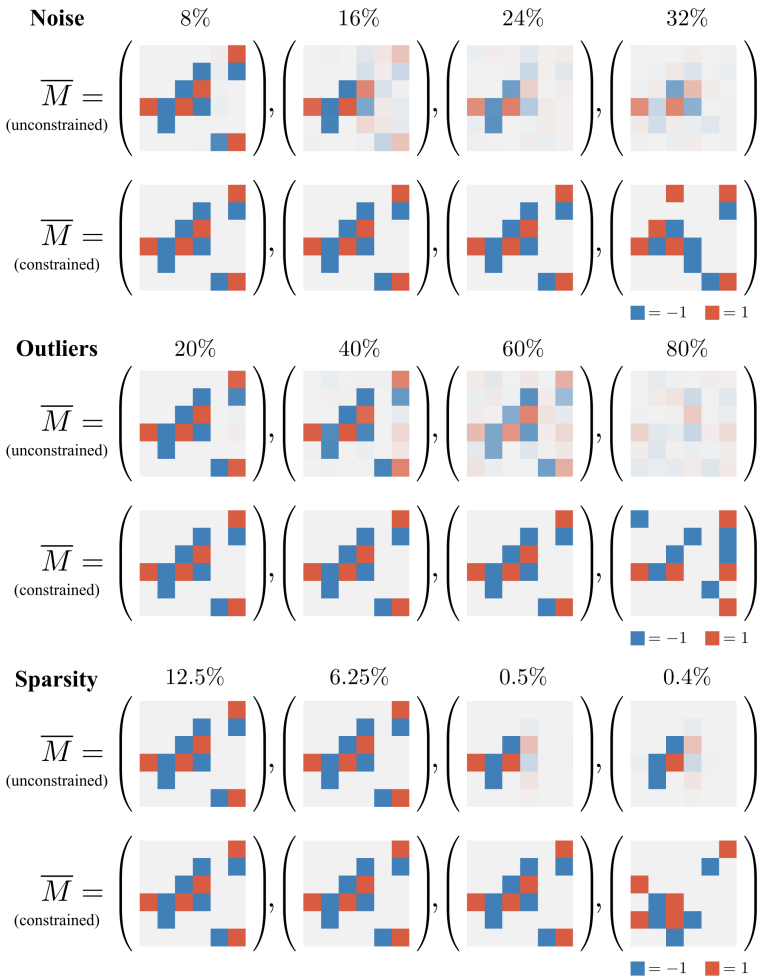


Figure 5.11: *Cohomology matrices resulting from our optimization. Columns correspond to Fig. 5.10. Per category: Top rows show the real-valued solutions to the unconstrained problem. Bottom rows show our results, subject to symplectic and integer constraints, which are always valid (co)homology maps. Matrix entries suffering from missing or ambiguous information in the unconstrained case can be faithfully completed via the symplectic constraint.*

Figure 5.12: *Due to non-surjectivity and severe misalignment in the input $\tilde{f}$ (a), the interpolated mapping of periodic potentials (b) may yield non-closed 1-forms. In such cases, an integral-based homology inference via* (5.15) *may fail (c) as it considers integrals along representative paths only; the global similarity-based inference* (5.16) *can compensate for such errors and yields the expected result (d).*

real-valued solutions to an unconstrained version of the optimization problem (5.16).

This high resilience to input defects is specific to the similarity-based inference formulation (5.16). In contrast, the integration-based inference formulation (5.15) is highly sensitive to mapping and interpolation errors, as demonstrated in Fig. 5.12. Besides the ability to cope with various kinds of sampling errors, we also demonstrate robust inference in the presence of large distortions and maps far from isometry in Figs. 5.9, 5.13 (b), 5.12, and 5.16.

In Figs. 5.1, 5.2, and 5.17 we emphasize that our method extracts topological information *from an input map* rather than aiming to find a natural matching between two shapes. In all three examples, our inferred homology maps faithfully reproduce the intentional twists indicated by the input maps. Finally, Fig. 5.14 displays the successful handling of a challenging combination of higher-genus surfaces and low-quality input map.

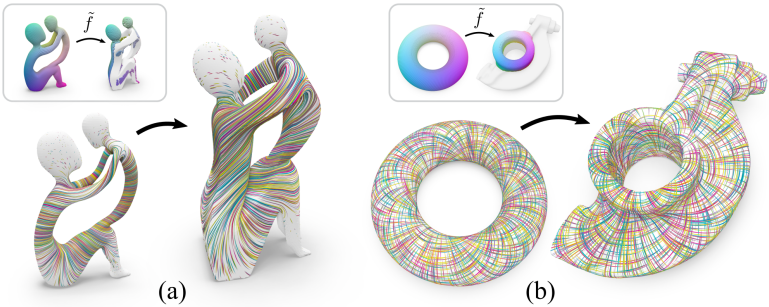


Figure 5.13: *Left: Transfer of harmonic vector fields (via transfer of real-valued coefficients between cohomology bases). Right: Transfer of direction field turning numbers along homology generators.*

5.5.3 Homological Data Transfer

Induced (co)homology maps can be used to directly transfer data between the (co)homology bases of corresponding objects. One example are harmonic tangent vector fields, which can be expressed as real-valued linear combinations of harmonic fields in a cohomology basis. Fig. 5.13 (a) demonstrates the transfer of a harmonic vector field from one surface to another via the cohomology map f^*, producing fields with the same flow around corresponding handles.

Figure 5.13 (b) shows a regular cross field on a genus 1 surface. Its global structure can be encoded as turning numbers along basis cycles. Using a homology map f_*, we can transport these turning numbers to reconstruct a field with the same global behavior on another object.

5.5.4 Compatible Cut Graph Generation

One way to construct a homeomorphism is by cutting two surfaces into disks along compatible cut graphs and overlaying them in a common domain. Figure 5.15 shows the construction of cut graphs rooted in a single vertex: Cuts are generated

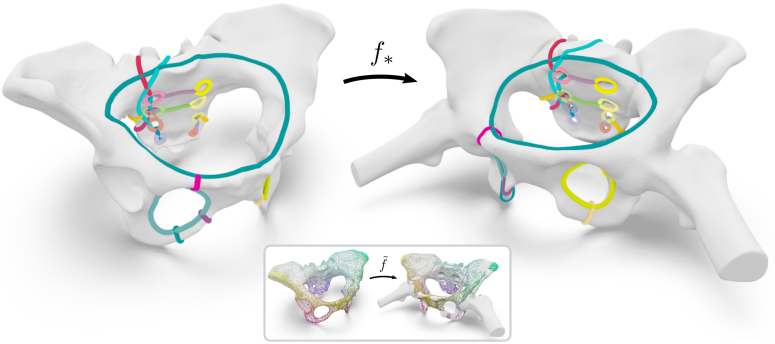


Figure 5.14: *Homology map f^* inferred from a low-quality map (inset) on a topologically complex surface of genus 12.*

by successively embedding $2g$ non-separating shortest paths that maintain the same cyclic order around the root vertex on both surfaces.

While this process produces cut graphs of compatible connectivity, the topology of embedded cut paths arises from a sequence of greedy decisions, which in general cannot be expected to correspond semantically between the two shapes (Fig. 5.15 (a)).

However, if semantic correspondence information is given in form of an input map $\tilde{f}$, we can use the induced homology map f_* to further constrain the topology of matching paths during cut graph construction: Specifically, for each cut path in homology class $[c]$ on $\mathcal{A}$, we find a matching cut path on $\mathcal{B}$ as a shortest path in homology class $f_*([c])$. Shortest paths under homology constraints can be computed using Dijkstra's algorithm on the universal cover of the target surface $\mathcal{B}$, i. e. on a graph with nodes $V_\mathcal{B} \times \mathbb{Z}^{2g}$, and counting intersections with basis cycles $B_\mathcal{B}$. Figure 5.15 (b) shows the result: Matching cut paths respect topological correspondences specified by the input map and wrap around handles in the same way on both surfaces. Cutting and overlaying both surfaces yields a homeomorphism with the intended map topology (Fig. 5.15 (c)). It provides a

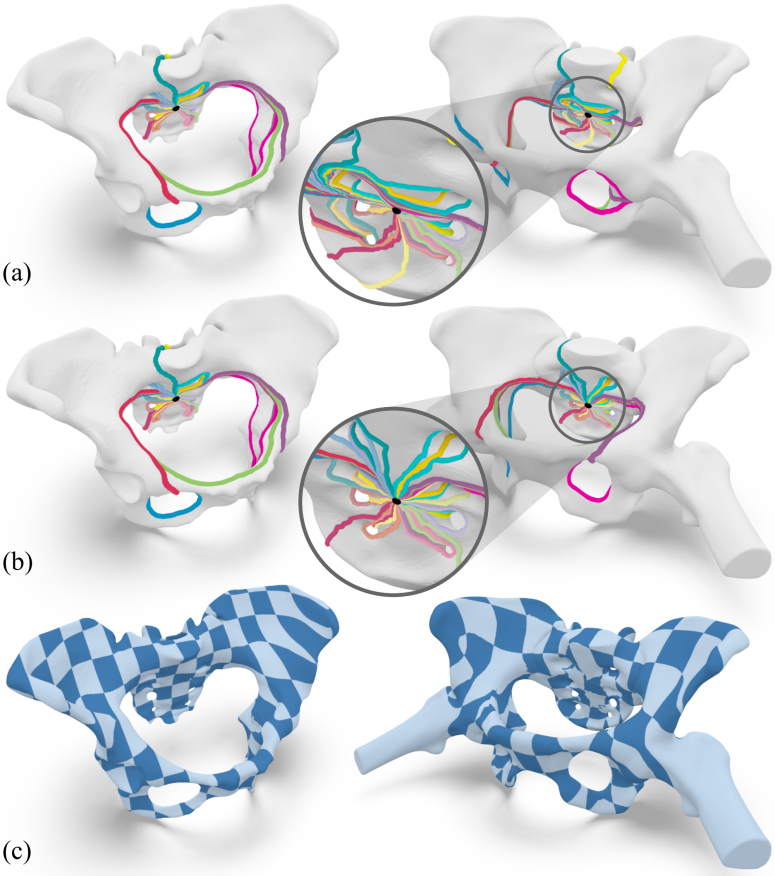


Figure 5.15: *Generating compatible cut graphs via greedy shortest path embeddings (a) offers little control over the resulting map topology. Given a homology map (Fig. 5.14), we can constrain paths on both surfaces to matching homology classes (b), here yielding a homeomorphism with the desired topology (c).*

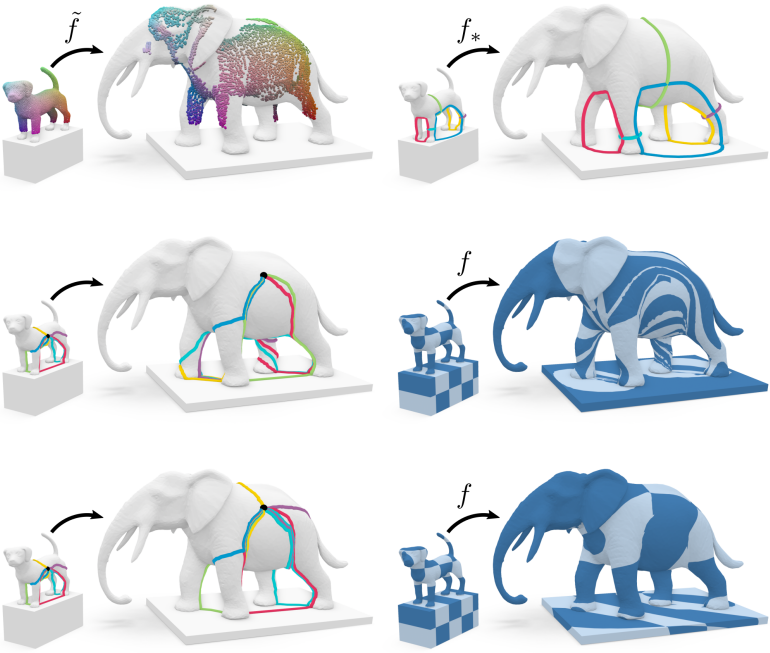


Figure 5.16: *Sampled input map (top left) and inferred homology map (top right). Constructing a homeomorphism from compatible cut graphs without any further guidance (middle left) is highly susceptible to topological artifacts (middle right). Using the inferred map to construct a cut graph from homology-constrained path embeddings (bottom left), we obtain a homeomorphism in the desired mapping class (bottom right), ready for geometric optimization.*

starting point for a subsequent continuous optimization of map geometry — which is necessarily restricted to the initial topology. Figure 5.16 shows another example.

Note however, that path homology constraints do not uniquely determine topology: There can be topologically distinct paths in the same homology class (e. g. by including twists around separating cycles, further discussed in Section 5.6.5). Our homology-constrained shortest-path computation effectively resolves these remaining ambiguities in a greedy manner by picking the shortest option.

5.5.5 Layout Transfer

Similarly to the above cut graph scenario, a homeomorphism can also be established by embedding a given layout structure into two target surfaces. Again, the topology of the resulting map is decided by a sequence of path embeddings. Choosing shortest paths (as commonly done) thus renders the resulting topology solely dependent on surface geometry.

Our framework adds explicit control over the map topology by supplying additional information via the input map. In Fig. 5.17, a twist in one of the

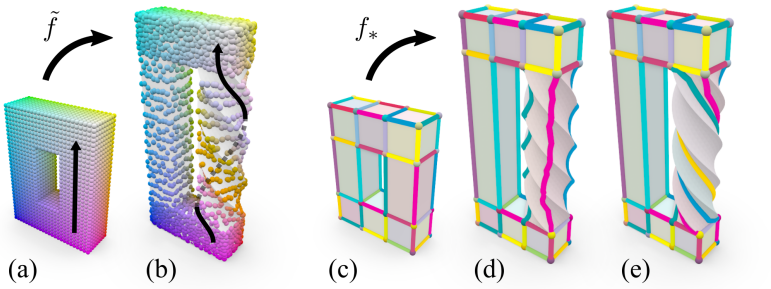


Figure 5.17: *(a-b): Input map $\tilde{f}$ between surfaces $\mathcal{A}$ and $\mathcal{B}$ indicating a twist. (c): Coarse layout embedded on $\mathcal{A}$. (d): Embedding the same layout structure on $\mathcal{B}$ via shortest paths does not capture the desired twist. (e): Via the inferred homology map f_*, we constrain the resulting layout embedding to the expected topology.*

surfaces is also indicated in the map $\tilde{f}$ and is successfully reproduced in the layout embedding by constraining shortest paths on the target surface to the inferred map homology.

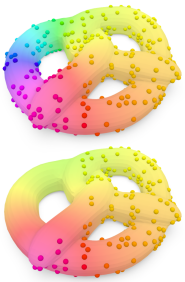
5.5.6 Computational Cost

We compute harmonic cohomology bases (Section 5.4.3) by solving the linear system (5.9)–(5.11), via sparse Cholesky decomposition. In all our examples (meshes of up to 31k vertices) this step never took longer than 1 second.

The remainder of the run time is dominated by solving the integer quadratic program (5.16) (symplectic cohomology matching, Section 5.4.4), whose number of variables is quadratic in the genus. In practice, we observe that the computational cost depends largely on the quality of the input map. In our experiments, we use a time limit of 60 seconds for the branch-and-bound solver, which is only reached by challenging inputs such as in Fig. 5.14 (where the returned best-so-far solution is already the semantically correct one), or by the artificially degraded maps in Fig. 5.10 columns (c) and (d). In all other experiments, the problem was solved in less than 9 seconds, in many cases (Figs. 5.1, 5.2, 5.5, 5.13, 5.17) below 1 second.

5.6 Limitations and Future Work

5.6.1 Ambiguous Input Maps



For very sparse input maps $\tilde{f}$, the transport of cocycles via interpolation of scalar fields (Section 5.4.5) deteriorates if topological features are undersampled (e. g. entire handles not covered by $\tilde{f}$). Depending on the number and location of samples, harmonic interpolation may not recover the full periodic signal, as illustrated in the inset. This can result in cocycles mapped to potentially unintended cohomology classes. Nevertheless, due to the enforced symplectic consistency and

the regularizing effect of its global nature, our inference still often yields the intended or expected overall homology map in such cases. Note that one cannot easily argue about the result being *correct* or *incorrect*, because the input map, especially in sparse settings, is inherently ambiguous topologically.

5.6.2 Strongly Non-Isometric Input Maps

Formulation (5.16) relies on matching images of harmonic fields from $\mathcal{A}$ under $\tilde{f}$ against harmonic fields on $\mathcal{B}$. This measure is ideal for maps $\tilde{f}$ that preserve harmonicity, such as isometries, because in this case it effectively matches cohomology classes via canonical representatives. While our experiments show that (5.16) performs robustly even for maps far from isometry, it could be interesting to explore whether an adapted measure that explicitly accounts for non-isometry can offer theoretical or practical benefits.

5.6.3 Tailored Optimization

In our prototype implementation, we employ an off-the-shelf branch-and-bound solver [Gur21] for the inference problem (5.16), a quadratic integer program under quadratic equality constraints. As indicated by our experiments, the usage of such a black-box solver is quite efficient for input pairs of small to moderate genus. However, scaling this method to higher genera, e. g. for objects with dozens or hundreds of handles, may require specialized solution strategies. A custom solver for (5.16) might exploit the specific structure of this formulation, e. g. the isomorphism of the solution space to the symplectic group.

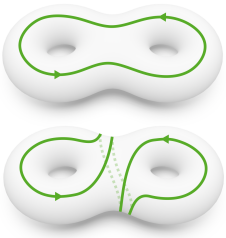
5.6.4 Surfaces with Boundaries

We have focused our attention on closed surfaces. Extension to surfaces with boundary will be a worthwhile direction for future work. It will require generalized

notions and generalized constructions of (co)homology bases. As a special case, this may also allow considering homology with respect to punctured surfaces.

5.6.5 Homology vs. Homotopy

In Sections 5.5.4 and 5.5.5, we have shown how a homology map can be used to constrain the topology of paths, e. g., for the construction of cut graphs, layout embeddings, or homeomorphisms. However, homology does not fully determine path topology in general: For surfaces of genus $g \geq 2$, there are homologous paths that are non-*homotopic*, i. e. that cannot be continuously deformed into one another (differing, e. g., by Dehn twists along separating cycles like in the inset). Homology is sometimes said to be only a "linear approximation" of homotopy. This gap between homology and homotopy extends to the topology of maps, where it is captured by the quite intricate Torelli group [FM11, Ch. 6; HM12]. As already mentioned in Section 3.4.4, elements of the Torelli group are homologically indistinguishable: The twisted input map in Fig. 5.18 induces the same homology map as the "identity map" (Fig. 5.2 top). The direct inference of a homotopy map is therefore an attractive goal. The non-commutativity of the Torelli group and the consequent fact that homotopy classes cannot be dealt with by means of linear algebra poses a key challenge in this regard.



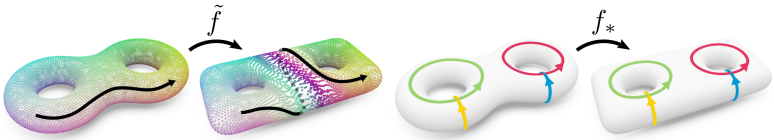


Figure 5.18: *An input map with a Dehn twist around a separating cycle, a topological feature that cannot be captured by homology. The inferred homology map is identical to the one in Fig. 5.2 top.*

5.7 Summary

We have presented the first automatic method to extract topological map descriptions from example inputs. Due to its high error tolerance and broad compatibility with different map encodings, it opens up a viable alternative to traditional map generation approaches: Instead of relegating difficult topological choices to user input, it is now possible to infer such decisions based on the results of automatic shape matching techniques, even if those cannot produce strictly valid homeomorphisms on their own.

6 Conclusion and Outlook

This thesis was primarily motivated by a lack of practically dependable methods for the construction of surface homeomorphisms: The few previously existing techniques in this area either depend on significant manual support, or are prone to topological initialization errors. In this work, we have examined the combinatorial intricacies of homeomorphism construction through the lens of algebraic topology; specifically, by studying the mapping class group, which describes the entire design space of possible map topologies and their interrelations. We have discussed several practical mapping class representations suitable for typical mesh processing settings.

Based on these representations, we have proposed novel algorithms that greatly expand the current capabilities for creating topologically natural surface map initializations, achieving superior robustness and capacity for automation. They approach the problem of map topology generation from two different angles:

- For the construction of homeomorphisms from landmark-constrained layout embeddings, we propose a robust alternative to previous heuristic-based methods. Our new algorithm reliably finds plausible and natural embedding topologies through a systematic branch-and-bound search for overall short layout embeddings.

- Leveraging the symplectic representation of the mapping class group, we obtain a linearized encoding of map topology in the form of homology maps. In this setting, we introduce a novel homology inference scheme that

generates topological descriptions from example inputs with very lenient quality and correctness requirements.

In combination, these advances serve to accelerate and automate the map generation process in two regards:

For the construction of maps from user-supplied landmarks, our reliable layout embedding algorithm methodically avoids the failure cases of previous ad-hoc methods, and thus greatly reduces the requirement for human supervision and intervention.

Meanwhile, the new ability to extract valid map topology descriptions just by analyzing approximate, imperfect maps can potentially eliminate the need for manual map constructions altogether: Our topology inference allows to delegate the demanding task of finding a semantically valid correspondence to specialized automatic shape matching methods, while being extremely tolerant about the encoding and quality of the intermediate result. Effectively, this connection allows topology-driven homeomorphism initializations to directly benefit from the capabilities of contemporary automatic shape matching methods, and from all future advances in this field.

We wish to conclude with some remarks on future directions for research. This includes potential extensions and further applications we envision for the methods and ideas proposed in this thesis, and a few considerations on still unsolved problems.

Homeomorphic Reconstruction

We have demonstrated the topology-guided initialization of homeomorphisms based on the results of our inference algorithm (Chapter 5). In this process, the input map only serves as a topological example; the homeomorphism is constructed without reference to the input map geometry.

A related task is the homeomorphic reconstruction of approximate input maps, which aims to find a true homeomorphism that closely follows the actual geometry

of a given input map, as far as possible. This involves an extension to a dense, continuous map, which necessarily requires an understanding of the intended map topology. Our inference method is particularly suitable for this task as it can globally analyze the input map and resolve local ambiguities.

Extending the input map, e. g. by connecting samples, will initialize a continuous but potentially non-bijective reconstruction in the desired map homotopy class. The remaining local inversions might then be resolved by homotopic deformations, along the lines of previous injectivity-promoting parametrization methods [FL16; DAZ+20; GKK+21; DKZ+21], but no extension of this idea to maps between surfaces has been described yet.

Topological Constraints for Shape Matching

The symplectic constraint employed by our homology inference method (Chapter 5) represents a simple necessary condition for topological map validity. While our work only applies this constraint while extracting the topology of a *previously* generated map, we envision further applications to promote topological validity already *during* map generation, i. e. for shape matching. This could complement similar measures targeting map continuity or reversibility, which have been successfully applied for the inference of functional maps [PSO18; RPWO18; RPWO19; RMWO21]. Generally, the functional maps setting appears to be particularly suited for the integration of homology-based constraints, due to its similarly linearized map representation and the ability to predict the resulting map differential [ABCO13], which enables the mapping of cocycles.

Partial Maps

In this work, we have focused primarily on bijective maps which put entire surfaces into correspondence. For this category, the mapping class group gives a complete description of all possible map topologies.

In practice, many applications also employ non-bijective *partial maps* that only cover certain subregions of the surfaces, e. g. where only parts of objects are relevant or available [PBB13; SY14; RCB+17]. Partial maps may still cover features such as handles or tunnels, hence their construction is faced with similar topological decisions as for full bijections. Unfortunately, in this case, the mapping class group no longer provides a suitable characterization of the topological design space. Whether the existing theoretical framework can be adapted to accommodate partial maps, especially between meshes of incompatible genus, is an open question. At least our matrix-based representation of homology maps (Chapter 5) allows a fairly natural extension to this setting, where partial homology maps or maps between objects of different genus would simply correspond to rank-deficient or rectangular matrices. Further research must show whether these modifications generalize to a full, non-linear description of partial map *homotopy*, and which algebraic properties of the mapping class group are retained in this process.

Incremental Topological Optimization

Contemporary homeomorphism optimization methods improve maps through sequences of purely isotopic (i. e. continuous and homeomorphism-preserving) updates. Obviously, this confines all such optimizations to their initial mapping class, putting a substantial responsibility on a topologically correct initialization. While our work has therefore mainly focused on strengthening the reliability of map initialization methods, a complementary research direction would be to liberate the subsequent homeomorphism optimization from its topological restrictions.

In this regard, our characterization of mapping classes (Chapter 3) suggests a potential extension, based on incremental map modifications: Indeed, by composing an existing homeomorphism with a non-trivial element of the mapping class group, one again obtains a valid but topologically different homeomorphism. An attractive candidate for such operators are Dehn twists (Section 3.4.2), as they are

local, relatively easy to construct, and — most importantly — can be combined to create any possible map topology.

As such operations generally introduce further map distortion, this implies an interleaved optimization scheme; alternating between discrete topology modifications and continuous distortion minimization to remedy their geometric side effects. However, at this point, there exists no simple strategy telling when and how to apply such topological updates: Due to the non-linear, non-convex nature of distortion-based map quality objectives, it is extremely difficult to judge the benefits of any potential topological modification before performing an expensive distortion minimization of the result.

These considerations point to a more fundamental question, pertaining to any topological optimization of homeomorphisms: Can we compute useful bounds or estimates for global map quality measures, depending only on the map topology? The objective of total embedded path length used by our layout embedding optimization (Chapter 4) may be seen as one example for an approximate measure. The development of similar estimates for different topology representations and quality measures would be a worthwhile effort, and could ultimately support a truly unified optimization scheme for both map topology and geometry.

Bibliography

[AKL17] Noam Aigerman, Shahar Z. Kovalsky, and Yaron Lipman. “Spherical Orbifold Tutte Embeddings”. In: *ACM Transactions on Graphics* 36.4 (2017) (cited on pages 8, 48, 49).

[AL16] Noam Aigerman and Yaron Lipman. “Hyperbolic Orbifold Tutte Embeddings”. In: *ACM Transactions on Graphics* 35.6 (2016) (cited on pages 8, 48, 49).

[APL14] Noam Aigerman, Roi Poranne, and Yaron Lipman. “Lifted Bijections for Low Distortion Surface Mappings”. In: *ACM Transactions on Graphics* 33.4 (2014) (cited on pages 8, 48, 49, 74, 76).

[APL15] Noam Aigerman, Roi Poranne, and Yaron Lipman. “Seamless Surface Mappings”. In: *ACM Transactions on Graphics* 34.4 (2015) (cited on pages 8, 48, 49, 74, 76).

[Ale00] Marc Alexa. “Merging Polyhedral Shapes with Scattered Features”. In: *The Visual Computer* 16.1 (2000) (cited on pages 2, 48).

[ASP+04] Dragomir Anguelov, Praveen Srinivasan, Hoi-Cheung Pang, Daphne Koller, Sebastian Thrun, and James Davis. “The Correlated Correspondence Algorithm for Unsupervised Registration of Nonrigid Surfaces”. In: *Advances in Neural Information Processing Systems*. 2004 (cited on page 118).

[ABCO13] Omri Azencot, Mirela Ben-Chen, Frédéric Chazal, and Maks Ovsjanikov. “An Operator Approach to Tangent Vector Field Processing”. In: *Computer Graphics Forum*. Vol. 32. 5. 2013 (cited on page 153).

[BMBZ02] Henning Biermann, Ioana Martin, Fausto Bernardini, and Denis Zorin. “Cut-and-Paste Editing of Multiresolution Surfaces”. In: *ACM Transactions on Graphics* 21.3 (2002) (cited on page 3).

[BWK05] Stephan Bischoff, Tobias Weyand, and Leif Kobbelt. “Snakes on Triangle Meshes”. In: *Bildverarbeitung für die Medizin*. Springer, 2005 (cited on page 77).

[BCE+13] David Bommes, Marcel Campen, Hans-Christian Ebke, Pierre Alliez, and Leif Kobbelt. “Integer-grid Maps for Reliable Quad Meshing”. In: *ACM Transactions on Graphics* 32.4 (2013) (cited on pages 71, 105).

[BZK09] David Bommes, Henrik Zimmer, and Leif Kobbelt. “Mixed-Integer Quadrangulation”. In: *ACM Transactions on Graphics* 28.3 (2009) (cited on page 105).

[BSCK21] Janis Born, Patrick Schmidt, Marcel Campen, and Leif Kobbelt. “Surface Map Homology Inference”. In: *Computer Graphics Forum* 40.5 (2021) (cited on pages 12, 113).

[BSK21] Janis Born, Patrick Schmidt, and Leif Kobbelt. “Layout Embedding via Combinatorial Optimization”. In: *Computer Graphics Forum* 40.2 (2021) (cited on pages 12, 67).

[BMRB16] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. “Learning Shape Correspondence with Anisotropic Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 29. 2016 (cited on page 119).

[BSBK02] Mario Botsch, Stefan Steinberg, Stefan Bischoff, and Leif Kobbelt. "OpenMesh: A Generic and Efficient Polygon Mesh Data Structure". In: *Proceedings of OpenSG Symposium*. 2002 (cited on page 15).

[Bre93] Glen E. Bredon. *Topology and Geometry*. Springer, 1993 (cited on pages 13, 14, 34, 39).

[BB08] Alexander M. Bronstein and Michael M. Bronstein. "Regularized Partial Matching of Rigid Shapes". In: *European Conference on Computer Vision*. Springer. 2008 (cited on page 46).

[Cam17] Marcel Campen. "Partitioning Surfaces into Quadrilateral Patches: A Survey". In: *Computer Graphics Forum* 36.8 (2017) (cited on page 71).

[CBK12] Marcel Campen, David Bommes, and Leif Kobbelt. "Dual Loops Meshing: Quality Quad Layouts on Manifolds". In: *ACM Transactions on Graphics* 31.4 (2012) (cited on pages 71, 105).

[CBK15] Marcel Campen, David Bommes, and Leif Kobbelt. "Quantized Global Parametrization". In: *ACM Transactions on Graphics* 34.6 (2015) (cited on page 105).

[CK14a] Marcel Campen and Leif Kobbelt. "Dual Strip Weaving: Interactive Design of Quad Layouts using Elastica Strips". In: *ACM Transactions on Graphics* 33.6 (2014) (cited on pages 71, 76, 105).

[CK14b] Marcel Campen and Leif Kobbelt. "Quad Layout Embedding via Aligned Parameterization". In: *Computer Graphics Forum* 33.8 (2014) (cited on pages 71, 77).

[CJGQ05] Christopher Carner, Miao Jin, Xianfeng Gu, and Hong Qin. "Topology-Driven Surface Mappings with Robust Feature Alignment". In: *VIS 05. IEEE Visualization, 2005.* IEEE. 2005 (cited on page 48).

[CHCH06] Nathan A. Carr, Jared Hoberock, Keenan Crane, and John C. Hart. "Rectangular Multi-Chart Geometry Images". In: *Proceedings of the Eurographics Symposium on Geometry Processing '06*. 2006 (cited on page 70).

[CHKL13] Timothy M. Chan, Hella-Franziska Hoffmann, Stephen Kiazyk, and Anna Lubiw. "Minimum Length Embedding of Planar Graphs at Fixed Vertex Locations". In: *International Symposium on Graph Drawing*. Springer. 2013 (cited on page 108).

[CFB16] Xue Chen, Jieqing Feng, and Dominique Bechmann. "Mesh Sequence Morphing". In: 35.1 (2016) (cited on pages 2, 3).

[COC14] Etienne Corman, Maks Ovsjanikov, and Antonin Chambolle. "Supervised Descriptor Learning for Non-Rigid Shape Matching". In: *European Conference on Computer Vision*. 2014 (cited on page 119).

[CGDS13] Keenan Crane, Fernando de Goes, Mathieu Desbrun, and Peter Schröder. "Digital Geometry Processing with Discrete Exterior Calculus". In: *ACM SIGGRAPH 2013 Courses*. 2013 (cited on page 44).

[CLPQ20] Keenan Crane, Marco Livesu, Enrico Puppo, and Yipeng Qin. *A Survey of Algorithms for Geodesic Paths and Distances*. 2020. arXiv: `2007.10430` (cited on page 83).

[DS95] Tamal K. Dey and Haijo Schipper. "A New Technique to Compute Polygonal Schema for 2-Manifolds with Application to Null-Homotopy Detection". In: *Discrete and Computational Geometry* 14.1 (1995) (cited on page 28).

[DYT05] Huong Quynh Dinh, Anthony Yezzi, and Greg Turk. "Texture Transfer During Shape Transformation". In: *ACM Transactions on Graphics* 24.2 (2005) (cited on page 3).

[DBG+06] Shen Dong, Peer-Timo Bremer, Michael Garland, Valerio Pascucci, and John C. Hart. “Spectral Surface Quadrangulation”. In: *Proceedings of SIGGRAPH*. 2006 (cited on page 71).

[DAZ+20] Xingyi Du, Noam Aigerman, Qingnan Zhou, Shahar Z. Kovalsky, Yajie Yan, Danny M. Kaufman, and Tao Ju. “Lifting Simplices to Find Injectivity”. In: *ACM Transactions on Graphics* 39.4 (2020) (cited on pages 9, 153).

[DKZ+21] Xingyi Du, Danny M. Kaufman, Qingnan Zhou, Shahar Z. Kovalsky, Yajie Yan, Noam Aigerman, and Tao Ju. “Optimizing Global Injectivity for Constrained Parameterization”. In: *ACM Transactions on Graphics* 40.6 (2021) (cited on pages 9, 153).

[DML17] Nadav Dym, Haggai Maron, and Yaron Lipman. “DS++: A Flexible, Scalable and Provably Tight Relaxation for Matching Problems”. In: *ACM Transactions on Graphics* 36.6 (2017) (cited on page 119).

[ESCK16] Hans-Christian Ebke, Patrick Schmidt, Marcel Campen, and Leif Kobbelt. “Interactively Controlled Quad Remeshing of High Resolution 3D Models”. In: *ACM Transactions on Graphics* 35.6 (2016) (cited on pages 3, 105).

[EH96] Matthias Eck and Hugues Hoppe. “Automatic Reconstruction of B-spline Surfaces of Arbitrary Topological Type”. In: *Proceedings of SIGGRAPH*. 1996 (cited on page 71).

[EEB20] Michal Edelstein, Danielle Ezuz, and Mirela Ben-Chen. “ENIGMA: Evolutionary Non-Isometric Geometry Matching”. In: *ACM Transactions on Graphics* 39.4 (2020) (cited on pages 46, 119).

[ELC19] Marvin Eisenberger, Zorah Lähner, and Daniel Cremers. “Divergence-Free Shape Correspondence by Deformation”. In: *Computer Graphics Forum* 38.5 (2019) (cited on page 46).

[Epp03] David Eppstein. “Dynamic Generators of Topologically Embedded Graphs”. In: *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*. 2003 (cited on page 27).

[EH04] Jeff Erickson and Sariel Har-Peled. “Optimally Cutting a Surface into a Disk”. In: *Discrete & Computational Geometry* 31.1 (2004) (cited on page 27).

[EW05] Jeff Erickson and Kim Whittlesey. “Greedy Optimal Homotopy and Homology Generators”. In: *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*. 2005 (cited on pages 23, 27, 28, 128).

[ESD+22] Isaac Esteban, Patrick Schmidt, Audrey Desgrange, Morena Raiola, Susana Temiño, Sigolène M. Meilhac, Leif Kobbelt, and Miguel Torres. “Pseudodynamic Analysis of Heart Tube Formation in the Mouse Reveals Strong Regional Variability and Early Left–Right Asymmetry”. In: *Nature Cardiovascular Research* 1.5 (2022) (cited on page 2).

[EHA+19] Danielle Ezuz, Behrend Heeren, Omri Azencot, Martin Rumpf, and Mirela Ben-Chen. “Elastic Correspondence between Triangle Meshes”. In: *Computer Graphics Forum* 38.2 (2019) (cited on pages 46, 119).

[ESB19] Danielle Ezuz, Justin Solomon, and Mirela Ben-Chen. “Reversible Harmonic Maps between Discrete Surfaces”. In: *ACM Transactions on Graphics* 38.2 (2019) (cited on pages 46, 119).

[FM11] Benson Farb and Dan Margalit. *A Primer on Mapping Class Groups*. Princeton University Press, 2011 (cited on pages 35, 51, 55–60, 63, 64, 148).

[FL16] Xiao-Ming Fu and Yang Liu. “Computing Inversion-Free Mappings by Simplex Assembly”. In: *ACM Transactions on Graphics* 35.6 (2016) (cited on pages 9, 153).

[GKK+21] Vladimir Garanzha, Igor Kaporin, Liudmila Kudryavtseva, François Protais, Nicolas Ray, and Dmitry Sokolov. “Foldover-Free Maps in 50 Lines of Code”. In: *ACM Transactions on Graphics* 40.4 (2021) (cited on page 153).

[GW17] Tyrone Ghaswala and Rebecca R. Winarski. “The Liftable Mapping Class Group of Balanced Superelliptic Covers”. In: *New York Journal of Mathematics* 23 (2017) (cited on page 59).

[GBP07] Daniela Giorgi, Silvia Biasotti, and Laura Paraboschi. *SHape REtrieval Contest 2007: Watertight Models Track*. 2007 (cited on page 97).

[GGH02] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. “Geometry Images”. In: *ACM Transactions on Graphics* 21.3 (2002) (cited on page 77).

[GY02] Xianfeng Gu and Shing-Tung Yau. “Computing Conformal Structure of Surfaces”. In: *Communications in Information and Systems* 2.2 (2002) (cited on page 131).

[GY03] Xianfeng Gu and Shing-Tung Yau. “Global Conformal Surface Parameterization”. In: *Proceedings of the Eurographics Symposium on Geometry Processing*. 2003 (cited on page 131).

[Gur21] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2021. URL: `http://www.gurobi.com` (cited on pages 135, 147).

[Hat02] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002 (cited on pages 13, 19, 21, 22, 37, 42, 61).

[HM12] Allen Hatcher and Dan Margalit. “Generating the Torelli Group”. In: *L’Enseignement Mathématique* 58.1 (2012) (cited on pages 64, 148).

[HRM05] Geremy Heitz, Torsten Rohlfing, and Calvin R. Maurer Jr. “Statistical Shape Model Generation Using Nonrigid Deformation of a Template Mesh”. In: *Medical Imaging 2005: Image Processing*. 2005 (cited on page 2).

[HS94] John Hershberger and Jack Snoeyink. “Computing Minimum Length Paths of a Given Homotopy Class”. In: *Computational Geometry* 4.2 (1994) (cited on page 77).

[HPG+21] Amir Hertz, Or Perel, Raja Giryes, Olga Sorkine-Hornung, and Daniel Cohen-Or. “SAPE: Spatially-Adaptive Progressive Encoding for Neural Optimization”. In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021 (cited on page 3).

[Hir03] Anil N. Hirani. “Discrete Exterior Calculus”. PhD thesis. California Institute of Technology, 2003 (cited on page 44).

[HLC20] Benjamin Holzschuh, Zorah Lähner, and Daniel Cremers. “Simulated Annealing for 3D Shape Correspondence”. In: *International Conference on 3D Vision*. Vol. 2. 2020 (cited on page 119).

[HRWO20] Ruqi Huang, Jing Ren, Peter Wonka, and Maks Ovsjanikov. “Consistent ZoomOut: Efficient Spectral Map Synchronization”. In: 39.5 (2020) (cited on page 119).

[HAWG08] Qi-Xing Huang, Bart Adams, Martin Wicke, and Leonidas J. Guibas. “Non-Rigid Registration under Isometric Deformations”. In: *Computer Graphics Forum* 27.5 (2008) (cited on page 46).

[Hum79] Stephen P. Humphries. “Generators for the Mapping Class Group”. In: *Topology of Low-Dimensional Manifolds*. Springer, 1979 (cited on page 58).

[JHTG20] Max Jiang, Jingwei Huang, Andrea Tagliasacchi, and Leonidas Guibas. “ShapeFlow: Learnable Deformations Among 3D Shapes”. In: *Proceedings of Neural Information Processing Systems*. 2020 (cited on page 119).

[JSP17] Zhongshi Jiang, Scott Schaefer, and Daniele Panozzo. “Simplicial Complex Augmentation Framework for Bijective Maps”. In: *ACM Transactions on Graphics* 36.6 (2017) (cited on page 49).

[KZHC11] Oliver van Kaick, Hao Zhang, Ghassan Hamarneh, and Daniel Cohen-Or. “A Survey on Shape Correspondence”. In: *Computer Graphics Forum* 30.6 (2011) (cited on pages 4, 118).

[KKBL15] Itay Kezurer, Shahar Z. Kovalsky, Ronen Basri, and Yaron Lipman. “Tight Relaxation of Quadratic Matching”. In: *Computer Graphics Forum* 34.5 (2015) (cited on page 119).

[KLS03] Andrei Khodakovsky, Nathan Litke, and Peter Schröder. “Globally Smooth Parameterizations with Low Distortion”. In: *ACM Transactions on Graphics* 22.3 (2003) (cited on page 70).

[KLF11] Vladimir G. Kim, Yaron Lipman, and Thomas Funkhouser. “Blended Intrinsic Maps”. In: *ACM Transactions on Graphics* 30.4 (2011) (cited on page 97).

[KS98] Ron Kimmel and James A. Sethian. “Computing Geodesic Paths on Manifolds”. In: *Proceedings of the National Academy of Sciences* 95.15 (1998) (cited on page 74).

[KS04] Vladislav Kraevoy and Alla Sheffer. “Cross-Parameterization and Compatible Remeshing of 3D Models”. In: *ACM Transactions on Graphics* 23.3 (2004) (cited on pages 2, 7, 8, 48, 49, 74–77, 96).

[KS06] Vladislav Kraevoy and Alla Sheffer. “Mean-Value Geometry Encoding”. In: *International Journal of Shape Modeling* 12.1 (2006) (cited on page 120).

[KSG03] Vladislav Kraevoy, Alla Sheffer, and Craig Gotsman. “Matchmaker: Constructing Constrained Texture Maps”. In: *ACM Transactions on Graphics* 22.3 (2003) (cited on pages 7, 48, 70, 74–76, 96–99, 102, 106).

[KBK13] Lars Krecklau, Janis Born, and Leif Kobbelt. “View-Dependent Realtime Rendering of Procedural Facades with High Geometric Detail”. In: *Computer Graphics Forum* 32.2 (2013).

[LM18] Francis Lazarus and Arnaud de Mesmay. *Computational Topology*. Lecture notes. 2018 (cited on pages 13, 15, 25–27).

[LPVV01] Francis Lazarus, Michel Pocchiola, Gert Vegter, and Anne Verroust. “Computing a Canonical Polygonal Schema of an Orientable Triangulated Surface”. In: *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*. 2001 (cited on pages 28, 108).

[LDSS99] Aaron W. F. Lee, David Dobkin, Wim Sweldens, and Peter Schröder. “Multiresolution Mesh Morphing”. In: *Proceedings of SIGGRAPH*. 1999 (cited on pages 2, 48, 77).

[LSS+98] Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. “MAPS: Multiresolution Adaptive Parameterization of Surfaces”. In: *Proceedings of SIGGRAPH*. 1998 (cited on page 70).

[Lee11] John M. Lee. *Introduction to Topological Manifolds*. Second Edition. Springer, 2011 (cited on pages 13, 14, 18, 33).

[Lee13] John M. Lee. *Introduction to Smooth Manifolds*. Second Edition. Springer, 2013 (cited on pages 37, 41, 124).

[LL02] Yunjin Lee and Seungyong Lee. “Geometric Snakes for Triangular Meshes”. In: *Computer Graphics Forum* 21.3 (2002) (cited on page 77).

[LSP08] Hao Li, Robert W. Sumner, and Mark Pauly. “Global Correspondence Optimization for Non-Rigid Registration of Depth Scans”. In: *Computer Graphics Forum* 27.5 (2008) (cited on page 46).

[LRL06] Wan Chiu Li, Nicolas Ray, and Bruno Lévy. “Automatic and Interactive Mesh to T-Spline Conversion”. In: *Proceedings of the Eurographics Symposium on Geometry Processing ’06*. 2006 (cited on page 71).

[LBG+08] Xin Li, Yunfan Bao, Xiaohu Guo, Miao Jin, Xianfeng Gu, and Hong Qin. “Globally Optimal Surface Mapping for Surfaces with Arbitrary Topology”. In: *IEEE Transactions on Visualization and Computer Graphics* 14.4 (2008) (cited on page 29).

[Lic64] W. B. R. Lickorish. “A Finite Set of Generators for the Homeotopy Group of a 2-Manifold”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 60.4 (1964) (cited on page 57).

[LDCK18] Isaak Lim, Alexander Dielen, Marcel Campen, and Leif Kobbelt. “A Simple Approach to Intrinsic Correspondence Learning on Unstructured 3D Meshes”. In: *Proceedings of the European Conference on Computer Vision Workshops*. 2018 (cited on pages 46, 119).

[LF09] Yaron Lipman and Thomas Funkhouser. “Möbius Voting for Surface Correspondence”. In: *ACM Transactions on Graphics* 28.3 (2009) (cited on pages 46, 119).

[LSLC05] Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or. “Linear Rotation-Invariant Coordinates for Meshes”. In: *ACM Transactions on Graphics* 24.3 (2005) (cited on page 120).

[LRR+17] Or Litany, Tal Remez, Emanuele Rodolà, Alex Bronstein, and Michael Bronstein. “Deep Functional Maps: Structured Prediction for Dense Shape Sorrespondence”. In: *IEEE International Conference on Computer Vision*. 2017 (cited on page 119).

[Lit17] Daniel Litt. “The Poincaré Lemma and de Rham Cohomology”. In: *The Harvard College Mathematics Review* 1.2 (2017) (cited on page 131).

[Liv20] Marco Livesu. “Scalable Mesh Refinement for Canonical Polygonal Schemas of Extremely High Genus Shapes”. In: *IEEE Transactions on Visualization and Computer Graphics* (2020) (cited on pages 29, 74).

[Lüc08] Wolfgang Lück. “Survey on Aspherical Manifolds”. In: *European Congress of Mathematics*. Amsterdam, 2008 (cited on page 131).

[MCK+17] Manish Mandad, David Cohen-Steiner, Leif Kobbelt, Pierre Alliez, and Mathieu Desbrun. “Variance-Minimizing Transport Plans for Inter-Surface Mapping”. In: *ACM Transactions on Graphics* 36.4 (2017) (cited on page 46).

[MTP+15] Giorgio Marcias, Kenshi Takayama, Nico Pietroni, Daniele Panozzo, Olga Sorkine-Hornung, Enrico Puppo, and Paolo Cignoni. “Data-Driven Interactive Quadrangulation”. In: *ACM Transactions on Graphics* 34.4 (2015) (cited on page 71).

[MK05] Martin Marinov and Leif Kobbelt. “Automatic Generation of Structure Preserving Multiresolution Models”. In: *Computer Graphics Forum* 24.3 (2005) (cited on page 71).

[MGR00] Stephen R. Marschner, Brian Guenter, and Sashi Raghupathy. “Modeling and Rendering for Realistic Facial Animation”. In: *Eurographics Workshop on Rendering Techniques*. 2000 (cited on page 76).

[MP78] William H. Meeks and Julie Patrusky. “Representing Homology Classes by Embedded Circles on a Compact Surface”. In: *Illinois Journal of Mathematics* 22.2 (1978) (cited on page 63).

[MRR+19] Simone Melzi, Jing Ren, Emanuele Rodolà, Abhishek Sharma, Peter Wonka, and Maks Ovsjanikov. “ZoomOut: Spectral Upsampling for Efficient Shape Correspondence”. In: *ACM Transactions on Graphics* 38.6 (2019) (cited on page 119).

[MMP87] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. “The Discrete Geodesic Problem”. In: *SIAM Journal on Computing* 16.4 (1987) (cited on page 74).

[MGP06] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. “Partial and Approximate Symmetry Detection for 3D Geometry”. In: *ACM Transactions on Graphics* 25.3 (2006) (cited on pages 46, 119).

[Mun18] James R. Munkres. *Elements of Algebraic Topology*. CRC Press, 2018 (cited on page 13).

[MDW08] Brent C. Munsell, Pahal Dalal, and Song Wang. “Evaluating Shape Correspondence for Statistical Shape Analysis: A Benchmark Study”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.11 (2008) (cited on page 2).

[OBS+12] Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. “Functional Maps: A Flexible Representation of Maps between Shapes”. In: *ACM Transactions on Graphics* 31.4 (2012) (cited on pages 46, 117, 119).

[OCB+17] 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 2017 Courses*. 2017 (cited on pages 46, 119).

[PBDS13] Daniele Panozzo, Ilya Baran, Olga Diamanti, and Olga Sorkine-Hornung. “Weighted Averages on Surfaces”. In: *ACM Transactions on Graphics* 32.4 (2013) (cited on pages 3, 46).

[PPM+16] Nico Pietroni, Enrico Puppo, Giorgio Marcias, Roberto Scopigno, and Paolo Cignoni. “Tracing Field-Coherent Quad Layouts”. In: *Computer Graphics Forum* 35.7 (2016) (cited on page 71).

[PP93] Ulrich Pinkall and Konrad Polthier. “Computing Discrete Minimal Surfaces and Their Conjugates”. In: *Experimental Mathematics* 2.1 (1993) (cited on page 133).

[PBB13] Jonathan Pokrass, Alexander M. Bronstein, and Michael M. Bronstein. “Partial Shape Matching Without Point-Wise Correspondence”. In: *Numerical Mathematics: Theory, Methods and Applications* 6.1 (2013) (cited on page 154).

[PSO18] Adrien Poulenard, Primoz Skraba, and Maks Ovsjanikov. “Topological Function Optimization for Continuous Shape Matching”. In: 37.5 (2018) (cited on pages 119, 153).

[PH03] Emil Praun and Hugues Hoppe. “Spherical Parametrization and Remeshing”. In: *ACM Transactions on Graphics* 22.3 (2003) (cited on pages 48, 77).

[PSS01] Emil Praun, Wim Sweldens, and Peter Schröder. “Consistent Mesh Parameterizations”. In: *Proceedings of SIGGRAPH*. 2001 (cited on pages 2, 3, 7, 48, 74–77, 83, 84, 96–99, 105).

[PCK04] Budirijanto Purnomo, Jonathan D. Cohen, and Subodh Kumar. “Seamless Texture Atlases”. In: *Proceedings of the Eurographics Symposium on Geometry Processing ’04*. 2004 (cited on page 70).

[Put18] Andrew Putman. *The Symplectic Representation of the Mapping Class Group is Surjective*. 2018. URL: `https://www3.nd.edu/~andyp/notes/SymplecticSurjective.pdf` (cited on page 63).

[RRP15] Faniry H. Razafindrazaka, Ulrich Reitebuch, and Konrad Polthier. “Perfect Matching Quad Layouts for Manifold Meshes”. In: *Computer Graphics Forum* 34.5 (2015) (cited on pages 71, 105).

[RMWO21] Jing Ren, Simone Melzi, Peter Wonka, and Maks Ovsjanikov. “Discrete Optimization for Shape Matching”. In: *Computer Graphics Forum*. Vol. 40. 5. 2021 (cited on page 153).

[RPWO19] Jing Ren, Mikhail Panine, Peter Wonka, and Maks Ovsjanikov. “Structured Regularization of Functional Map Computations”. In: 38.5 (2019) (cited on pages 119, 153).

[RPWO18] Jing Ren, Adrien Poulenard, Peter Wonka, and Maks Ovsjanikov. “Continuous and Orientation-Preserving Correspondences via Functional Maps”. In: *ACM Transactions on Graphics* 37.6 (2018) (cited on pages 119, 153).

[RCB+17] Emanuele Rodolà, Luca Cosmo, Michael M Bronstein, Andrea Torsello, and Daniel Cremers. “Partial Functional Correspondence”. In: *Computer Graphics Forum*. Vol. 36. 1. 2017 (cited on page 154).

[RMC15] Emanuele Rodolà, Michael Moeller, and Daniel Cremers. “Regularized Pointwise Map Recovery from Functional Correspondence”. In: *Computer Graphics Forum* 36.8 (2015) (cited on pages 46, 117).

[RRW+14] Emanuele Rodolà, Samuel Rota Bulò, Thomas Windheuser, Matthias Vestner, and Daniel Cremers. “Dense Non-Rigid Shape Correspondence Using Random Forests”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014 (cited on page 119).

[Rot73] Joseph Rotman. “Covering Complexes with Applications to Algebra”. In: *The Rocky Mountain Journal of Mathematics* 3.4 (1973) (cited on page 25).

[Sah18] Yusuf Sahillioğlu. “A Genetic Isometric Shape Correspondence Algorithm with Adaptive Sampling”. In: *ACM Transactions on Graphics* 37.5 (2018) (cited on page 119).

[Sah20] Yusuf Sahillioğlu. “Recent Advances in Shape Correspondence”. In: *The Visual Computer* 36.8 (2020) (cited on pages 4, 9, 119).

[SY11] Yusuf Sahillioğlu and Yücel Yemez. “Coarse-to-Fine Combinatorial Matching for Dense Isometric Shape Correspondence”. In: *Computer Graphics Forum* 30.5 (2011) (cited on page 119).

[SY14] Yusuf Sahillioğlu and Yücel Yemez. “Partial 3-D Correspondence from Shape Extremities”. In: *Computer Graphics Forum*. Vol. 33. 6. 2014 (cited on page 154).

[SSGH01] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. “Texture Mapping Progressive Meshes”. In: *Proceedings of SIGGRAPH*. 2001 (cited on page 3).

[SBB+22] Patrick Schmidt, Janis Born, David Bommes, Marcel Campen, and Leif Kobbelt. “TinyAD: Automatic Differentiation in Geometry Processing Made Simple”. In: *Computer Graphics Forum* 41.5 (2022).

[SBCK19] Patrick Schmidt, Janis Born, Marcel Campen, and Leif Kobbelt. “Distortion-Minimizing Injective Maps Between Surfaces”. In: *ACM Transactions on Graphics* 38.6 (2019) (cited on pages 8, 48, 49, 121).

[SCBK20] Patrick Schmidt, Marcel Campen, Janis Born, and Leif Kobbelt. “Inter-Surface Maps via Constant-Curvature Metrics”. In: *ACM Transactions on Graphics* 39.4 (2020) (cited on pages 5, 8, 48, 49, 77, 106, 107, 121).

[SAPH04] John Schreiner, Arul Asirvatham, Emil Praun, and Hugues Hoppe. “Inter-Surface Mapping”. In: *ACM Transactions on Graphics* 23.3 (2004) (cited on pages 8, 48, 74–77, 83, 96–99, 102, 108, 121).

[Set96] James A. Sethian. “A Fast Marching Level Set Method for Monotonically Advancing Fronts”. In: *Proceedings of the National Academy of Sciences* 93.4 (1996) (cited on page 74).

[SC20] Nicholas Sharp and Keenan Crane. “You Can Find Geodesic Paths in Triangle Meshes by Just Flipping Edges”. In: *ACM Transactions on Graphics* 39.6 (2020) (cited on page 77).

[SS15] Jason Smith and Scott Schaefer. “Bijective Parameterization with Free Boundaries”. In: *ACM Transactions on Graphics* 34.4 (2015) (cited on page 48).

[SNB+12] Justin Solomon, Andy Nguyen, Adrian Butscher, Mirela Ben-Chen, and Leonidas Guibas. “Soft Maps between Surfaces”. In: *Computer Graphics Forum* 31.5 (2012) (cited on pages 47, 118).

[SBLS18] Tommaso Sorgente, Silvia Biasotti, Marco Livesu, and Michela Spagnuolo. “Topology-Driven Shape Chartification”. In: *Computer Aided Geometric Design* 65 (2018) (cited on page 71).

[Sti93] John Stillwell. *Classical Topology and Combinatorial Group Theory*. Second Edition. Springer, 1993 (cited on pages 13, 34, 50).

[SSK+05] Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J. Gortler, and Hugues Hoppe. “Fast Exact and Approximate Geodesics on Meshes”. In: *ACM Transactions on Graphics* 24.3 (2005) (cited on page 74).

[TPSS13] Kenshi Takayama, Daniele Panozzo, Alexander Sorkine-Hornung, and Olga Sorkine-Hornung. “Sketch-Based Generation and Editing of Quad Meshes”. In: *ACM Transactions on Graphics* 32.4 (2013) (cited on pages 71, 105).

[THCM04] Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. “Polycube-Maps”. In: *ACM Transactions on Graphics* 23.3 (2004) (cited on page 71).

[TDN+11] Julien Tierny, Joel Daniels II, Luis G. Nonato, Valerio Pascucci, and Claudio T. Silva. “Inspired Quadrangulation”. In: *Computer-Aided Design* 43.11 (2011) (cited on page 3).

[Tur92] Greg Turk. “Re-Tiling Polygonal Surfaces”. In: *Proceedings of SIGGRAPH*. 1992 (cited on page 48).

[TSSH15] Christoph von Tycowicz, Christian Schulz, Hans-Peter Seidel, and Klaus Hildebrandt. “Real-Time Nonlinear Shape Interpolation”. In: *ACM Transactions on Graphics* 34.3 (2015) (cited on page 2).

[ULP+15] Francesco Usai, Marco Livesu, Enrico Puppo, Marco Tarini, and Riccardo Scateni. “Extraction of the Quad Layout of a Triangle Mesh Guided by its Curve Skeleton”. In: *ACM Transactions on Graphics* 35.1 (2015) (cited on pages 71, 105).

[VY90] Gert Vegter and Chee K. Yap. “Computational complexity of combinatorial surfaces”. In: *Proceedings of the Sixth Annual Symposium on Computational Geometry*. 1990 (cited on page 28).

[VLB+17] Matthias Vestner, Zorah Lähner, Amit Boyarski, Or Litany, Ron Slossberg, Tal Remez, Emanuele Rodolà, Alex Bronstein, Michael Bronstein, and Ron Kimmel. “Efficient Deformable Shape Correspondence via Kernel Matching”. In: *IEEE Int. Conf. 3D Vision*. 2017 (cited on page 119).

[VLR+17] Matthias Vestner, Roee Litman, Emanuele Rodolà, Alex Bronstein, and Daniel Cremers. “Product Manifold Filter: Non-rigid Shape Correspondence via Kernel Density Estimation in the Product Space”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017 (cited on page 46).

[WCMN19] Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. “3DN: 3D Deformation Network”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019 (cited on page 119).

[War83] Frank W. Warner. *Foundations of Differentiable Manifolds and Lie Groups*. Springer, 1983 (cited on page 130).

[XKH+16] Kai Xu, Vladimir G. Kim, Qixing Huang, Niloy Mitra, and Evangelos Kalogerakis. “Data-Driven Shape Analysis and Processing”. In: *SIGGRAPH ASIA Courses*. 2016 (cited on page 119).

[YFC+18] Yang Yang, Xiao-Ming Fu, Shuangming Chai, Shi-Wei Xiao, and Ligang Liu. “Volume-Enhanced Compatible Remeshing of 3D Models”. In: *IEEE Transactions on Visualization and Computer Graphics* 25.10 (2018) (cited on page 2).

[YZL+20] Yang Yang, Wen-Xiang Zhang, Yuan Liu, Ligang Liu, and Xiao-Ming Fu. “Error-Bounded Compatible Remeshing”. In: *ACM Transactions on Graphics* 39.4 (2020) (cited on page 2).

[YLSL10] I-Cheng Yeh, Chao-Hung Lin, Olga Sorkine, and Tong-Yee Lee. “Template-Based 3D Model Fitting Using Dual-Domain Relaxation”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.8 (2010) (cited on pages 46, 119).

[YAK+20] Wang Yifan, Noam Aigerman, Vladimir G. Kim, Siddhartha Chaudhuri, and Olga Sorkine-Hornung. “Neural Cages for Detail-Preserving 3D Deformations”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2020 (cited on page 119).

[YSC21] Christopher Yu, Henrik Schumacher, and Keenan Crane. “Repulsive Curves”. In: *ACM Transactions on Graphics* 40.2 (2021) (cited on page 77).

[ZSC+08] Hao Zhang, Alla Sheffer, Daniel Cohen-Or, Quan Zhou, Oliver van Kaick, and Andrea Tagliasacchi. “Deformation-Driven Shape Correspondence”. In: *Computer Graphics Forum* 27.5 (2008) (cited on page 119).

[ZCZ+18] Jiaran Zhou, Marcel Campen, Denis Zorin, Changhe Tu, and Claudio T. Silva. “Quadrangulation of Non-Rigid Objects using Deformation Metrics”. In: *Computer-Aided Geometric Design* 62 (2018) (cited on page 2).

[ZSH00] Malte Zöckler, Detlev Stalling, and Hans-Christian Hege. “Fast and Intuitive Generation of Geometric Shape Transitions”. In: *The Visual Computer* 16.5 (2000) (cited on page 2).

Index

Publications

[SBB+22] Patrick Schmidt, Janis Born, David Bommes, Marcel Campen, Leif Kobbelt: “TinyAD: Automatic Differentiation in Geometry Processing Made Simple”. In: *Computer Graphics Forum* 41.5, 2022. Presented at the Eurographics Symposium on Geometry Processing 2022.

[BSCK21] Janis Born, Patrick Schmidt, Marcel Campen, Leif Kobbelt: “Surface Map Homology Inference”. In: *Computer Graphics Forum* 40.5, 2021. Presented at the Eurographics Symposium on Geometry Processing 2021.

[BSK21] Janis Born, Patrick Schmidt, Leif Kobbelt: “Layout Embedding via Combinatorial Optimization”. In: *Computer Graphics Forum* 40.2, 2021. Presented at Eurographics 2021.

[SCBK20] Patrick Schmidt, Marcel Campen, Janis Born, Leif Kobbelt: “Inter-Surface Maps via Constant-Curvature Metrics”. In: *ACM Transactions on Graphics* 39.4, 2020. Presented at SIGGRAPH 2020.

[SBCK19] Patrick Schmidt, Janis Born, Marcel Campen, Leif Kobbelt: “Distortion-Minimizing Injective Maps Between Surfaces”. In: *ACM Transactions on Graphics* 38.6, 2019. Presented at SIGGRAPH Asia 2019.

[KBK13] Lars Krecklau, Janis Born, Leif Kobbelt: “View-Dependent Realtime Rendering of Procedural Facades with High Geometric Detail”. In: *Computer Graphics Forum* 32.2, 2013. Presented at Eurographics 2013.

Statement of Originality

Many of the ideas that ultimately contributed to the methods and results presented in this dissertation come from the countless fruitful discussions with Prof. Dr. Leif Kobbelt, Prof. Dr. Marcel Campen, and my co-authors and colleagues from the *Visual Computing Institute* at *RWTH Aachen*. My specific contributions to the relevant works for this thesis are listed below.

[BSK21] I am the main author of this publication. I conceived the core idea of the algorithm and wrote most parts of the paper. In my development of the prototype implementation of the method, I was assisted by Patrick Schmidt, who also conducted some of the experiments. Sasa Lukic helped with data preprocessing for additional experiments and maintenance of the software prototype, which is available from `https://github.com/jsb/LayoutEmbedding`.

[BSCK21] I am the main author of this publication. I conceived the core idea of the algorithm and wrote most parts of the paper. In my development of the prototype implementation of the method, I was assisted by Patrick Schmidt, who also conducted some of the experiments. The source code of this implementation is available from `https://github.com/jsb/HomologyInference`.