UNIVERSITY OF CALIFORNIA,
IRVINE

Geometry-aware topological decompositions of meshes

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Jia Chen

Dissertation Committee:
Professor Gopi Meenakshisundaram, Chair
Professor Aditi Majumder
Professor Shuang Zhao

2019

DEDICATION

To my family

"... a man who keeps company with glaciers comes
to feel tolerably insignificant by and by."
– Mark Twain, *A Tramp Abroad*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Algorithms

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Gopi Meenakshisundaram, for his valuable advice and continuous encouragement throughout the process. I would also like to thank Prof. Aditi Majumder and Prof. Shuang Zhao for extending help whenever I needed.

I want to thank my parents, my wife Faye, and my daughter Agnes for their unequivocal support and sacrifices.

Last but not the least, I want to thank all my labmates, especially Mahdi, Hao, Nitin, Mehdi, Zahra, Yu, Ali, Cheng, Samia, Zhanhang, Jessica, Isabela and Andy, for always being there.

CURRICULUM VITAE

Jia Chen

EDUCATION

Doctor of Philosophy in Computer Science **2014 - 2019**
University of California, Irvine *Irvine, California*

Master of Science in Computer Applied Technology **2007 - 2010**
Chinese Academy of Sciences *Beijing, China*

Bachelor of Science in Software Engineering **2003 - 2007**
Beihang University *Beijing, China*

SELECT PUBLICATIONS

Jia Chen, Shan Jiang, Zachary Destefano, Sungeui Yoon, and M. Gopi. "Performance Driven Redundancy Optimization of Data Layouts for Walkthrough Applications." In Computer Graphics International (CGI). Computer Graphics and Geometry Group of ICube Laboratory (CNRS/Universit de Strasbourg), 2015.

Jia Chen, Shan Jiang, Zachary Destefano, Sungeui Yoon, and M. Gopi. "Optimally redundant, seek-time minimizing data layout for interactive rendering." The Visual Computer 33, no. 2 (2017): 139-149.

Jia Chen, James Jester, and M. Gopi. 2016. Robust segmentation of corneal fibers from noisy images. In Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP '16). ACM, New York, NY, USA, Article 58, 8 pages. DOI: https://doi.org/10.1145/3009977.3010051

Jia Chen, James Jester, and M. Gopi. 2018. Fast Computation of Tunnels in Corneal Collagen Structure. In Proceedings of Computer Graphics International 2018 (CGI 2018). ACM, New York, NY, USA, 57-65. DOI: https://doi.org/10.1145/3208159.3208175

Jia Chen, M. Gopi. "Geometry-aware Tori Decomposition." In Computer Graphics Forum (EuroGraphics). Oxford, UK and Boston, USA: Blackwell Publishing, Inc.

ABSTRACT OF THE DISSERTATION

Geometry-aware topological decompositions of meshes

By

Jia Chen

Doctor of Philosophy in Computer Science

University of California, Irvine, 2019

Professor Gopi Meenakshisundaram, Chair

Topology captures a surfaces global features invariant to local deformation, and many geometry processing applications can benefit from topological information. However, traditional topological data analysis methods, e.g., persistent homology, when applied to surfaces, suffer from their massive computation cost and their lack of exact correspondence with surface geometry. In this dissertation, we use edge cycles as a compact representation of the surfaces topology and apply it in two topological decompositions of meshes. We propose an iterative method to localize tunnel and handle cycles, which respectively capture the surface's exterior and interior spaces. We then present the tori decomposition that segments the surface into genus-1 components. We formulate the tori decomposition as a min-cut problem in the dual graph and design geometry-aware edge weights to make the decomposition fit to the geometry. We also propose a framework to decompose the surface into contractible solids. Unlike previous methods which rely on volumetric representation, we solve the problem on the surface. We find a redundant set of cycles, which form an oversegmentation of the surface, and then we apply a dynamic programming method to merge the cells to form a contractible decomposition. All of our algorithms are based on efficient surface-embedded graph algorithms, and we demonstrate their robustness on numerous models.

Chapter 1

Introduction

The recent advent of 3D acquisition technology, such as computer tomography, magnetic resonance imaging, 3D laser scanning, ultrasound, and microscopy has enabled highly accurate digitization of 3D objects. Therefore, with the vast number of digital 3D objects, there is an increasing research interest in digital geometry processing, which is concerned with mathematical models and algorithms for analyzing and manipulating geometric data. Typical operations include surface reconstruction from point samples, filtering operations for noise removal, geometry analysis, shape simplification, and geometric modeling and interactive design.

In mathematics, two tools are commonly used for studying surfaces: geometry and topology. The two are closely related but have distinct focuses. Geometry studies a surface shape's local properties, e.g., the location of each point, while topology represents the shape's global configuration. For example, a flat torus and a doughnut surface have the same global topology, but different local geometry. If we extract the same number of sample points from a smooth sphere and a smooth torus, the coordinates of the extracted sample points may happen to be the same, but the two shapes have different topologies.

Early studies of digital geometry processing focus mostly on geometry as geometric features themselves are sufficient for a lot of common tasks such as mesh smoothing or mesh defor-

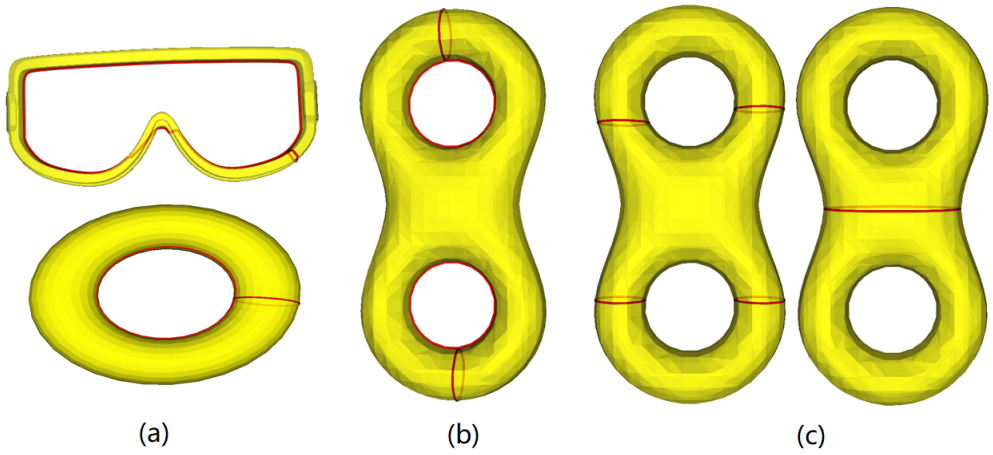

Figure 1.1: Cycles reflect both the surface's geometry and topology. (a) Representative cycles give indication about the surface's outline and local size (b) The topological relationship between cycles are related to the surface topology, for example, a genus g surface has $2g$ fundamental cycles. (c) A set of cycles split the surface, and determine the topology and geometry of the decomposed components.

mation. However, along with the advent of tools which can capture or model topologically complex meshes [1], there is an increasing demand for methods that can process the topologically complex meshes. The major challenges of processing such models are (1) the existing tools in topological data analysis, such as persistence diagram, topological barcode, don't have exact correspondence with the shape geometry, which makes it difficult to directly apply them in geometry processing problems (2) the topological analysis methods, mostly based on algebraic topology and persistence filtration, are not efficient enough for practical surface shapes.

1.1 Cycles of topological properties

In our work, we consider the cycles on a surface as a representation of the topology. The advantages of using such cycles to represent topology are as follows: (1) The cycles are part of the surface-embedded graph. While the graph is an extensively studied term, a lot

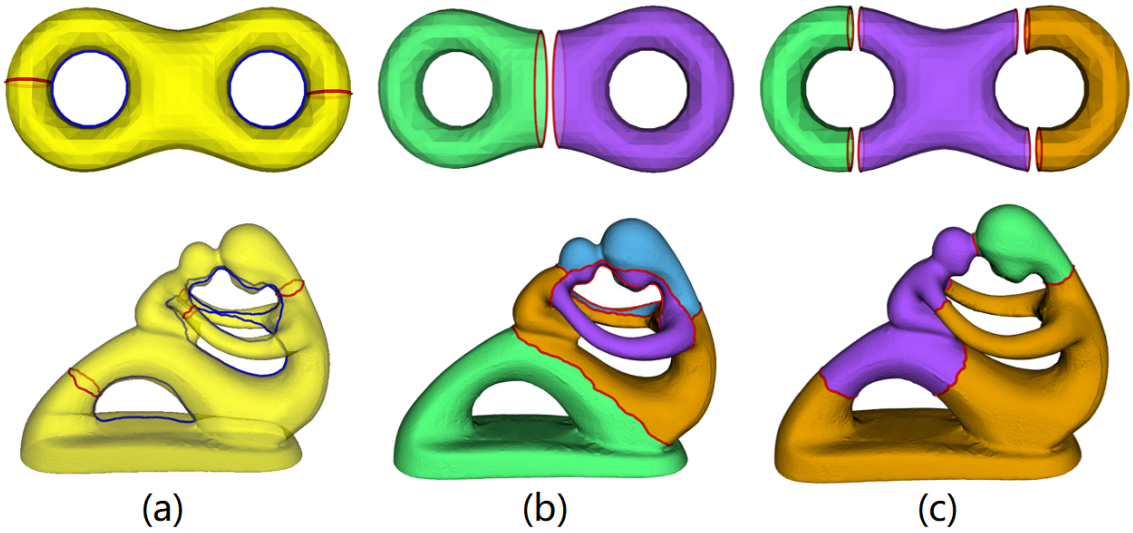

Figure 1.2: Our work presented in this dissertation. Given a 2-manifold surface with genus g (a) we localize g tunnel cycles and g handle cycles which are topologically independent (chapter 3), (b) we decompose the surface into topological tori (chapter 4), (c) we decompose the surface into contractible pieces (chapter 5).

of theories and efficient algorithms on graphs can be used to find, analyze and manipulate such cycles. (2) The cycles contain both topological and geometric properties: for geometry, geometric measurements such as curvature and length are well defined on these cycles, shown in Fig. 1.1(a); for topology, the cycles differentiates surfaces of different topology, shown in Fig. 1.1(b). Furthermore, as shown in Fig. 1.1(c), a set of cycles can split the surface into parts, and we can determine the geometry and topology of the segmented parts using these cycles.

In this dissertation, we introduce an iterative tree-cotree algorithm to localize two special kinds of cycles: tunnels and handles, as shown in Fig. 1.2(a). This method is based on solving simple graph problems; thus it is more efficient than homology-based methods[14][13]. We show its application on finding fundamental cycles in topologically complex meshes. Previous methods generate a redundant set of non-contractible and non-separating cycles and select an independent set of fundamental cycles. However, the selection procedure cannot guarantee

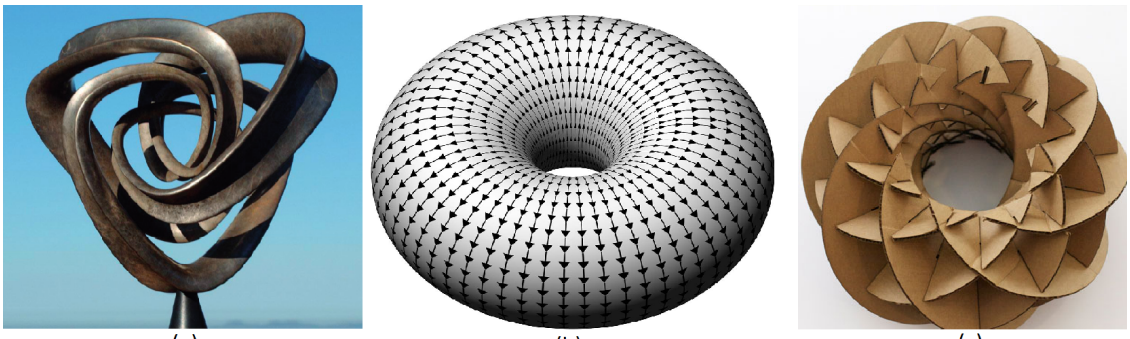

Figure 1.3: Applications of topological tori. (a) Topological tori have been extensively studied from a theoretical perspective for abstract art[44], (b) On a torus there exist vector fields without singular points. As in the case of art, multi-genus objects can be decomposed into multiple tori and the vector field can be designed for individual tori and merged. (c) There exist techniques which are especially suitable for making a torus [53]. Tori decomposition makes it possible to apply these techniques to make component tori, which then can be assembled into more complex shapes.

independence among cycles. Unlike such methods, our iterative method guarantees that the cycles generated in each iteration are independent.

1.2 Topological decompositions of meshes

Shape decomposition, or shape segmentation, is a classic problem in mesh processing. Many mesh processing algorithms such as shape matching, mesh editing, shape retrieval, and object rigging, require shape segmentation as a pre-processing step. A lot of fabrication methodologies, e.g., milling and 3D printing, require that the shape is of the particular properties. Shape decomposition may segment the shape into parts such that each of them can be manufactured using these fabrication techniques.

In this dissertation, we study two topological decompositions-tori decomposition and contractible decomposition.

Tori decomposition partitions a surface mesh with genus g into g components each of

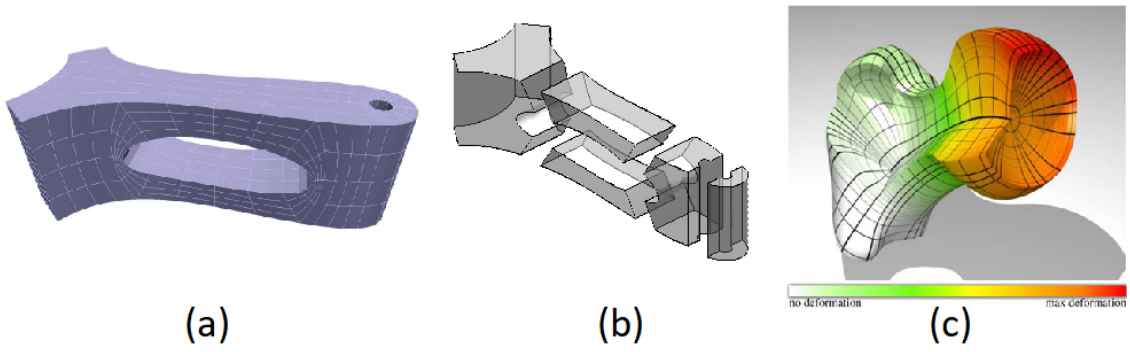

Figure 1.4: Application of contractible solids in mechanical analysis. Given (a) a mechanical part, it is required to be decomposed into contractible solids (b), such that isogeometric analysis (c) can be applied on the mechanical part [42].

which has genus-1. We make use of the computed handle and tunnel cycles to find the splitting cycles that produce such a tori decomposition. The problem is posed as a min-cut problem on the mesh's dual graph with earlier computed tunnels as source and target. The edge weights for the min-cut problem are designed for the cut to be geometry-aware. We present an implementation and demonstrate the results of our algorithm on numerous examples. The tori decomposition is of great practical significance. In 3D art, a torus can be an element for generating abstract artworks [44]. In surface matching, the relative location of the decomposed tori helps consistently match parts in different models. In vector field processing, since on a torus there exist differentiable vector fields with no singular points, tori decomposition helps to avoid singular points or to determine where the singular points should be located. In 3D printing, there is a growing interest in 3D printed coils which are more suitable for manufacturing genus-1 objects. A tori decomposition may make application of these manufacturing techniques suitable for more complicated shapes by partitioning the shapes into genus-1 components.

Contractible decomposition partitions the surfaces into contractible solids. While finding cuts in the volume is hard, we show that searching for such cuts is equivalent to finding handle cycles and separating cycles on the surface. We present a two-stage method to find such

a decomposition. First, we generate a series of handle cycles and separating cycles using the iterative tree-cotree algorithm and tori decomposition algorithm. These cycles form an oversegmentation of the surface shape. Second, we apply a dynamic programming based method to find an optimal layout composed of a subset of the oversegmentation cycles. The contractible decomposition is of practical significance as (1) each component of the result has genus-0, thus is easier to manufacture compared to topologically complex shapes (2) as shown in Fig. 1.4, such decomposition is a necessary pre-processing step for isogeometric analysis.

We organize the dissertation as follows. First, we provide relevant concepts from computational geometry and computational topology in Chapter 2. In Chapter 3, we describe the algorithm for localizing tunnel and handle cycles. Next, we elaborate on making use of the tunnel and handle cycles on the classical mesh segmentation problem: in Chapter 4, we describe methods for decomposing a surface mesh into topological tori. In Chapters 5, we explain the approach we apply to decompose a surface shape into contractible pieces. Finally, in Chapter 6 we summarize our results and discuss future research directions.

Chapter 2

Definitions and Background

2.1 Surfaces and their topological classification

Given two topological spaces X and X', a map $h : X \to X'$ is a **homeomorphism** if h is bijective and both h and its inverse are continuous. A **surface** (or 2-manifold possibly with boundary) M is a compact Hausdorff space where each point has an open neighborhood homeomorphic to either the plane R^2 or the closed halfplane. The points with neighborhood homeomorphic to the closed half-plane comprise the **boundary** of M. In computer graphics and digital geometry processing, the most common representation of the surface is **triangle mesh**, which is a collection of triangles without any particular mathematical structure. All the meshes discussed in this dissertation are triangle meshes unless specified.

A surface is **non-orientable** if it contains a subset homeomorphic to the Mobius band, otherwise **orientable**. A connected orientable manifold has exactly two different possible orientations. Any orientable surface is homeomorphic to a sphere with g handles attached and b open disks removed. The orientability is a **topological invariant** of the surface, which only depends on the surface itself, but oblivious to its triangulation. For example, orientable surfaces are not homeomorphic to non-orientable surfaces.

Let G be a graph cellularly embedded on a compact surface M. The **Euler characteristic** of

G is $v-e+f$, where v is the number of vertices, e is the number of edges, and f is the number of faces of the graph. For a closed orientable surface, it is known that $v - e + f = 2 - 2g$, where g is genus. The Euler characteristic is also a topological invariant.

A surface is **connected** if any two points of the surface are the endpoints of some path. The inclusionwise maximal connected subsets of a surface form its **connected components**. For geometry processing applications, a surface with multiple connected components can usually be processed by processing each one of the components. All the surfaces we discuss in this dissertation are connected surfaces.

Two homeomorphic surfaces are regarded as **topologically equivalent**. For example, a cube and a sphere are topologically equivalent. The topological invariants of the surface determine the homeomorphism of the surfaces, for example, two compact orientable surfaces without boundary are homeomorphic if and only if they have the same genus.

2.2 Primal graph and dual graph embedded on a surface

The **primal graph** takes a vertex v in the original mesh as a node in the graph, and the mesh edge uv as a graph edge. Its **dual graph** is formed by considering each mesh face as a node, and two nodes are connected if their corresponding faces are adjacent in the original mesh surface. As shown in Fig 2.1, a mesh vertex is dual to a face in the dual graph, and a mesh edge is dual to an edge in the dual graph.

Given the primal graph, we can find a **spanning tree**, which is a tree formed by a subset of the edges of the primal graph that incorporates all the vertices of the graph. Similarly, for the dual graph, we can get a spanning tree, called **spanning cotree**. If the edges of a map are given weights, the weight of a tree or cotree is defined to be the sum of the weights of its edges.

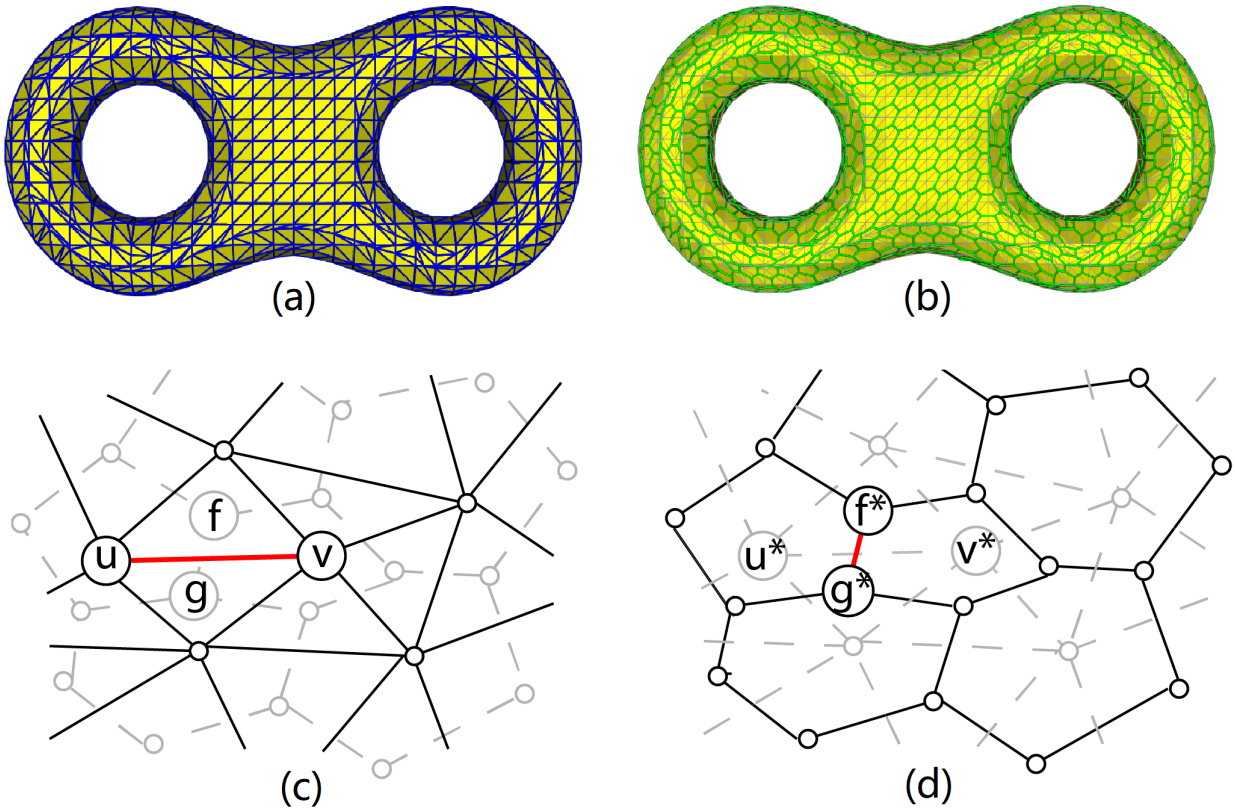

Figure 2.1: Primal graph and dual graph. (a) A primal graph embedded on a double torus, (b) A dual graph corresponding to primal graph (a), (c) An edge uv and (d) its dual $(uv)^* = f^*g^*$ are emphasized.

[21] proves that the following lemma is true, which is the basis of tree-cotree decomposition.

Lemma 1. Let a map M be given, with distinct weights on each of its edges. Then the minimum weight spanning tree of M and the maximum weight spanning cotree of M are disjoint.

2.3 Paths and cycles

A **path** on surface M is a continuous map p: $[0, 1] \to M$ with its two endpoints as $p(0)$ and $p(1)$. A path is simple when the mapping is injective, except for the common endpoint in

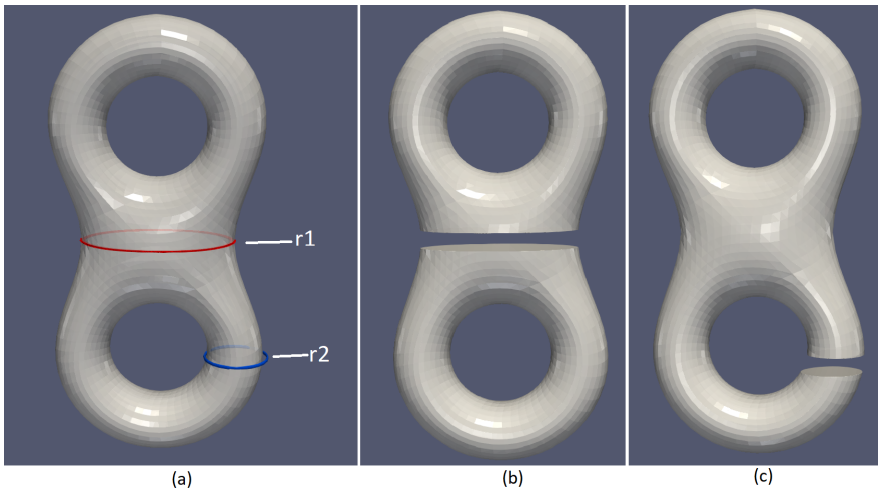

Figure 2.2: Separating and non-separating cycles. When cycle r_1 is removed from the surface, the surface is separated into two parts, thus r_1 is a separating cycle. When cycle r_2 is removed, the surface is still one connected component, thus r_2 is non-separating.

the case of loops. In discrete settings, a **cycle** is a closed path without repeated vertices.

Two paths p, q with $p(0) = q(0)$ and $p(1) = q(1)$ are called to be **homotopic** if there is a continuous function $h : [0, 1]^2 \to \sum$ such that $p(\cdot) = h(0, \cdot)$, $q(\cdot) = h(1, \cdot)$,$h(\cdot, 0) = p(0)$, and $h(\cdot, 1) = p(1)$. Two cycles α, β are called to be **freely homotopic** if there is a continuous function $g : [0, 1] \times S^1 :\to \sum$ such that $\alpha(\cdot) = g(0, \cdot)$ and $\beta(\cdot) = g(1, \cdot)$.

Separating and non-separating cycles. A cycle is **separating** if cutting the surface along the cycle gives rise to two connected components; the concept is closely related to Z_2-homology. Fig. 2.2 shows examples of separating and non-separating cycles.

Contractible and non-contractible cycles. A cycle is **contractible** if it can be continuously contracted into a point, for example, as shown in Fig. 2.3(e). Otherwise it is **non-contractible**, for example, as shown in Fig. 2.3(a)-(d). A contractible cycle is also a separating cycle. When cutting along a simple contractible cycle, a surface is segmented into two connected components, and one of them is a topological disk. Being contractible or separating is a property invariant under homotopy of cycles.

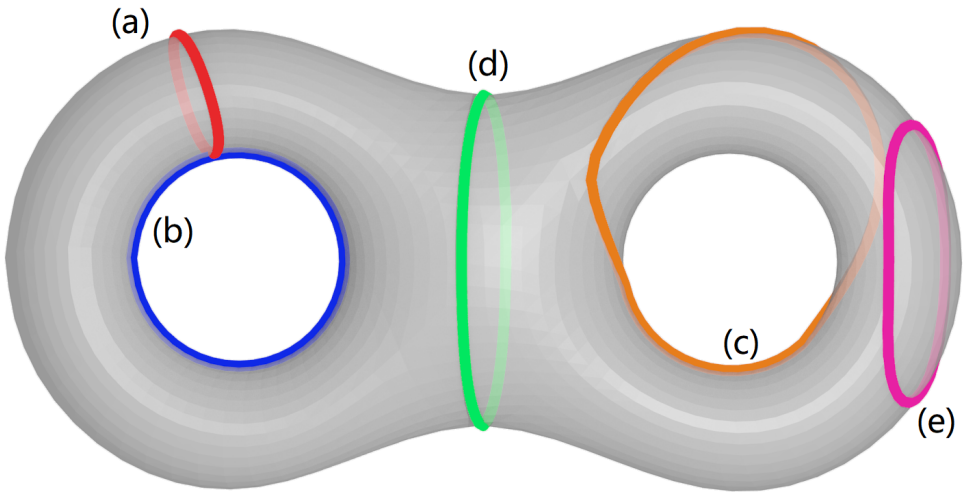

Figure 2.3: Surface cycles with topological features. (a), (b) and (c) are all non-separating and non-contractible cycles, also called fundamental cycles. (a) is a handle cycle, (b) is a tunnel cycle while (c) is neither. (d) is a separating and non-contractible cycle, also called a splitting cycle, (e) is a separating and contractible cycle.

As shown in Fig. 2.3, based on if a cycle is separating and contractible, the cycles can be categorized into three classes:

- **Contractible cycles** are simple, contractible and separating cycles. Contractible cycles are also called to be **trivial**, for example Fig. 2.3(e).
- **Splitting cycles** are simple, non-contractible and separating cycles, and such cycles divide the topology of the surface as well as the underlying graph, for example Fig. 2.3(d).
- **Fundamental cycles** are simple, non-contractible and non-separating cycles, , for example Fig. 2.3(a)-(c).

2.4 Tunnel and handle cycles

For a 2-manifold with genus g, there will be $2g$ independent fundamental cycles called the generator set (or the homology basis) that can generate all other cycles on the mesh through

algebraic addition of fundamental cycles and contractible cycles. A fundamental cycle may further be classified as a tunnel cycle, a handle cycle, or neither[14]. An orientable manifold surface M separates R^3 into two parts: *inside* I and *outside* O. A fundamental cycle is a tunnel cycle if it is contractible in O but non-contractible in M. A fundamental cycle is a handle cycle if it is contractible in I but non-contractible in M. Intuitively speaking, the handle cycles bound surfaces in I whereas tunnel cycles bound them in O. If we cut the independent set of g handle cycles and g tunnel cycles from the surface, the surface becomes a closed disk. Therefore, the fundamental cycles form a **cut graph**. For a formal treatment of the subject, readers are referred to [30].

In this dissertation, both the tori decomposition and contractible decomposition rely on the fundamental cycles: we use tunnel cycles to find splitting cycles which form a tori decomposition, and the contractible decomposition is composed by only handle cycles and separating cycles.

Chapter 3

Iterative localization of handle and tunnel cycles

Topological analysis on geometric data set has proved to work well on high-dimensional, incomplete and noisy data sets [5]. However, there are many difficulties in applying it on complex low dimensional data, such as surface meshes. (1) Computation of topological attributes involving the construction of simplicial complex, Morse complex, homology groups, etc., is expensive or even cannot be run on large-scale datasets. (2) The topological features, such as Betti number, persistence, etc., draw insights on global information but lack exact correspondence with local geometry. Thus, such topological features are not intuitive to non-expert users. In this chapter, we propose an iterative method to find a set of handle and tunnel cycles which form a compact representation of the surface's topology. Each of the cycles we find represents a class of equivalent cycles, all of which characterize the same topological properties.

The handle and tunnel cycle are useful for a lot of topology related geometry processing applications such as topology surgery, mesh parameterization[56], surface mapping[35]. A series of previous works explore to localize handle and tunnel cycles using homology[14], Reeb graph[13], or tree-cotree decomposition[17]. However, none of these methods is suitable for the large-scale models for (1) The homology[14] and Reeb graph[13] based methods are too

heavy in computation, especially when the surface is large. (2) Tree-cotree decomposition based algorithm [17] generates a redundant set of cycles, and pick $2g$ good ones from the set, in which the computed cycles are not guaranteed to be independent. In this chapter, we adopt the tree-cotree algorithm for its efficiency but improve it into an iterative method which guarantees that in each iteration the computed cycles are independent fundamental cycles.

The main contributions of this chapter are as follows:

- We propose an iterative method to localize handle and tunnel cycles. We guarantee that, in each iteration, the computed cycles are topologically independent.
- We propose a tightening method, which, instead of searching for the shortest-length loops, refine the fundamental cycles to make them good for visualization, mechanical and optical simulation.
- We analyze the properties of the refined fundamental cycles, and based on their geometric properties and relationship between each other, we cluster and label them into tunnels and non-tunnels.

3.1 Related work

[24] proposes a greedy algorithm to construct the shortest set of loops that generate the fundamental group of any oriented combinatorial 2-manifold in $O(n \cdot logn)$ time. [14] study two particular kinds of fundamental cycles: *tunnels* and *handles*, by constructing curve skeletons for both inner space and exterior space bounded by the original manifold mesh. The tunnels and handles are extremely useful for a lot of topology related problem, our algorithm in this paper also requires finding tunnels and handles as a prerequisite. The computation of the curve skeleton in [14] requires triangulating the spaces, which is a challenging task

in itself when the input mesh is large. Thus [13] makes use of Reeb graph, which can be computed efficiently, to avoid the triangulation. Moreover, by perturbating the cycle inside and outside, the winding number between the perturbated cycles can be used to decide if a specific cycle is a tunnel or handle cycle. In this paper, we use the same method for classifying the found cycles that we compute with our algorithm into tunnels and handles. Another set of work attempts to solve the problem in the surface's embedded graph and its dual graph. [21] introduces tree-cotree decomposition which is efficient in finding $2g$ fundamental cycles. Pablo et al. [17] apply this idea while designing the edge weights such that the computed fundamental cycles are aligned with principal curvature directions of a surface. These graph-based methods are efficient as the most time-consuming part of the tree-cotree decomposition is merely calculating spanning trees. Approaches studying the structure of tunnels, or similarly pores, cavities, have been proposed in various domains. [20] uses α-hulls to identify small tunnels that are not accessible by a user-defined ball rolling on the surface. [29] applies Morse theory to segment a surface mesh into simple pieces called "pants", which, up to topology, are genus 0 orientable surfaces each with three boundary components, however, although they are related, the "pants" are not directly useful for locating the tunnels. [51] and [50] study the tunnels in protein structure, and trace the pathway from the internal cavity to the protein surface. These methods rely on the fact that the tunnels in protein are mostly of a similar size. Therefore, they cannot be extended to applications where the tunnels are of various sizes.

3.2 Problem formulation

In this chapter, we aim to find handle and tunnel cycles which are both topologically correct and geometrically good. The method is expected to find exactly g handle cycles and g tunnel cycles. As shown in Fig. 3.1, the tunnel and handle cycles must be topologically

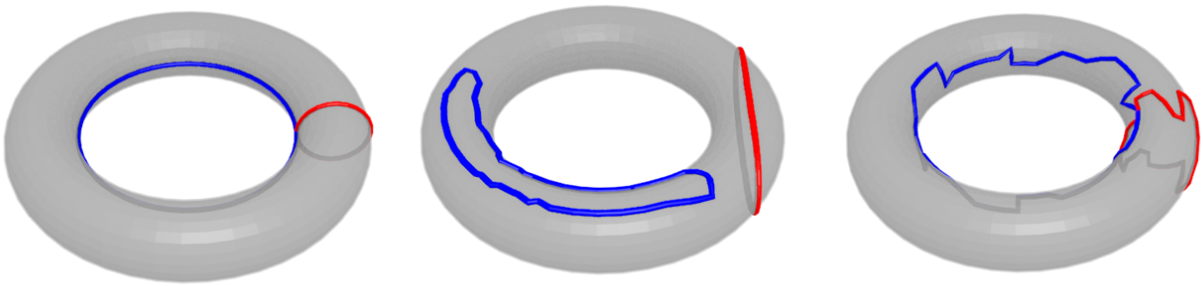

Figure 3.1: The computed fundamental cycles. (a) is considered as good as both cycles are non-contractible and non-separating. The cycles in (b) are contractible, therefore, (b) is topologically incorrect. The cycles in (c) are valid fundamental cycles, but they are unnecessarily long and noisy, so (c) is not good in geometry.

independent. Furthermore, the cycles should be in proper location and without unnecessary contours.

3.3 Localizing handle and tunnel cycles

In this chapter, we apply an iterative tree-cotree algorithm to find a set of fundamental cycles. The computed cycles may not be optimal in geometry, therefore, we further improve the cycles' geometry by tightening the computed cycles while guaranteeing the tightened cycles are homotopic to original ones. Furthermore, when there exist composite cycles, we decompose them into handle and tunnel cycles.

3.3.1 Iterative tree-cotree algorithm

The tree-cotree algorithm is based on computing spanning trees in two graphs. We first calculate a spanning tree in the primal graph, which we call a **tree**, and remove the edges in the dual graph that correspond to the edges in the tree. On the resulting subgraph of the dual graph, we compute a spanning tree in the dual graph, which we call a **cotree**. It

is known that a tree-cotree decomposition of a mesh partitions the set E of edges of a mesh into three sets $(T, C, E\backslash(T \cup C))$, where T is the set of edges in the *tree* and C is the set of mesh edges corresponding to the edges of *cotree* [21]. The set $E\backslash(T \cup C)$ contains exactly $2g$ edges, and introducing these edges into *tree* creates $2g$ cycles which are the fundamental cycles of the mesh, as shown in Fig. 3.2.

We follow [17] to determine the edge weights using principal curvature directions. As shown in Fig. 3.3, the weight of an edge is defined as the average angle the edge makes with one of the chosen principal curvature directions (say, minimum curvature direction) at its incident vertices. The minimum and maximum curvature directions have shown to represent the directions of the tunnel and handle cycles respectively well.

[17] generate $2g$ cycles using minimum curvature direction and $2g$ cycles using maximum curvature direction, and from the total $4g$ cycles [17] selects $2g$ cycles which are good in geometry. We also compute fundamental cycles over multiple runs of the algorithm using different edge weights, but we guarantee that the cycles are independent and form the basis for the homology groups. The intuition behind this idea is that we would like the algorithm to memorize some of the desired results computed in the previous iterations, and guarantee that the cycles in subsequent iterations will be independent of earlier computed and memorized cycles, as well as satisfy the new edge weights assigned for new iterations. The algorithm is shown in Algorithm 3.1. In each iteration, we get a set of candidate cycles that are alternating between using the minimum and maximum curvature directions. The good cycles are kept and are forced to be selected in subsequent iterations as well. To achieve this, we apply two techniques. First, we decide if a cycle is good using the geometry-based method introduced in [8]. Second, after each iteration, we adjust the edge weights such that edges in good cycles are chosen in any subsequent iteration to be part of the tree. Fig. 3.3 shows the major steps of our iterative method. Unlike [17], our algorithm never generates redundant cycles, so the cycles that are computed are guaranteed to be an independent set of cycles.

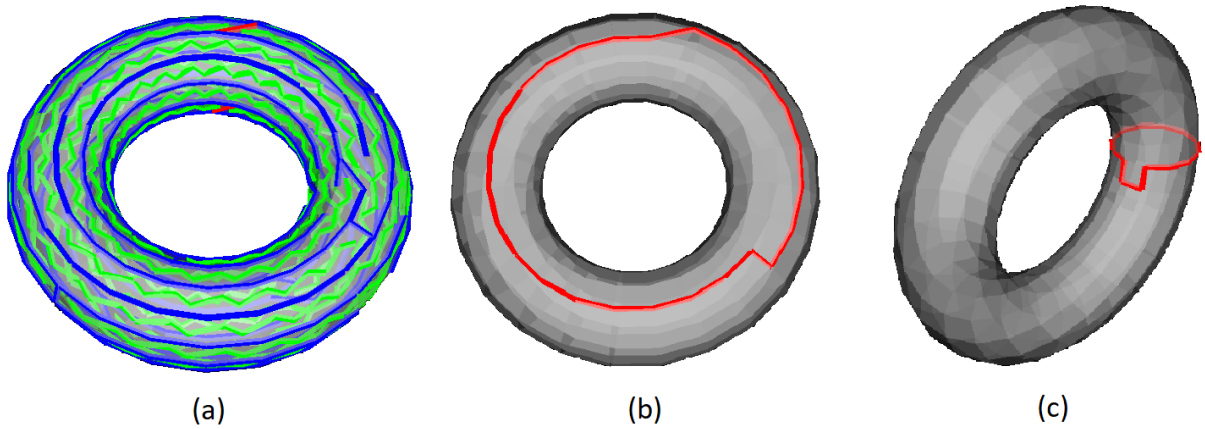

Figure 3.2: Spanning trees of the primal and dual graph. (a) The spanning tree for the primal graph is colored as blue, and the spanning tree for the dual graph is colored as green. The two leftover edges are colored as red. When adding the leftover edges into the spanning tree of the primal graph, a loop is formed for each of the leftover edges. For torus, one of the formed cycles is tunnel cycle (b), the other is handle cycle (c).

The algorithm progressively finds better cycles and terminates if all the $2g$ found cycles are good according to [8]'s criteria or the number of iterations exceeds the parameter MAX. With a sufficiently dense triangulation, out of many options, good cycles can be computed in less than MAX iterations. With very sparse triangulation, for example, as shown in Fig. 4.10, the cycles may have very few edges, may not be smooth, or may not be geometrically good, causing the algorithm to fail to converge in MAX iterations. However, such cycles, even if not good in geometry, are still topologically correct independent fundamental cycles (guaranteed by tree-cotree algorithm). In each iteration, two spanning trees are constructed, which can be computed in $O(n \cdot log(n))$ time where n is the number of vertices, thus the overall time complexity for fundamental cycles localization is $O(MAX \cdot n \cdot log(n))$, and since in our implementation we use $5g$ as MAX, the overall complexity is $O(g \cdot n \cdot log(n))$.

The $2g$ fundamental cycles computed by our method would include both tunnel and handle cycles. We classify the cycles as tunnels and handles using the linking number method introduced in [13].

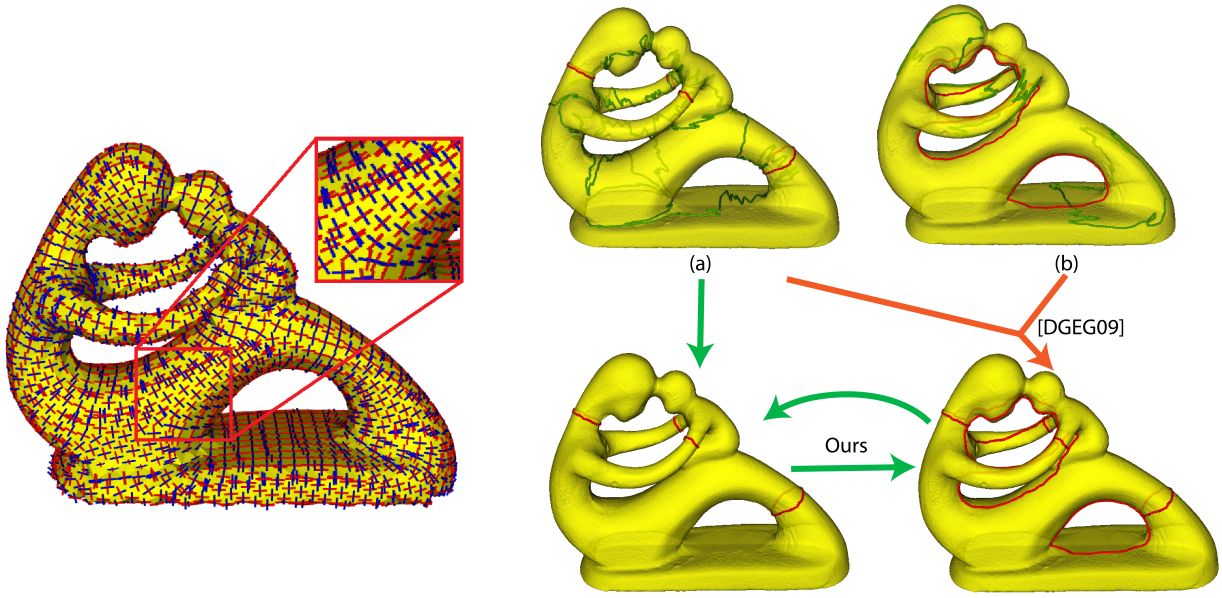

Figure 3.3: Major steps for finding handle and tunnel cycles. Left: Principal curvature directions. Red lines denote minimum curvature direction and the blue lines denote the maximum curvature direction. Right:(a) Tree-cotree algorithm results with maximum curvature direction based edge weights, with cycles of good quality highlighted in red color (b) Tree-cotree algorithm results with minimum curvature direction based edge weights. While [17] merges the two sets of cycles by picking good ones, we apply an iterative method: we alternate between two principal curvature directions and keep good cycles in the next iteration. This guarantees that all the resulting cycles are independent.

Algorithm 3.1 Iterative method for localizing the g tunnel and g handle cycles.

Initialize edge_weights_min using min curvature directions Initialize edge_weights_max using max curvature directions Initialize good_cycles as null **while** *number_of_good_cycles* ¡ $2g$ *and iteration* ¡ *MAX* **do**
    reset good_cycles to null
    if *odd iteration* **then**
        tree-cotree decomposition using edge_weights_min
    else
        tree-cotree decomposition using edge_weights_max
    end
    foreach *cycle c in found cycles* **do**
        if *c is good* **then**
            add c to good_cycles;
            assign minimum weights to all edges $\in$ c for both edge_weights_min and edge_weights_max;
        end
    end
end

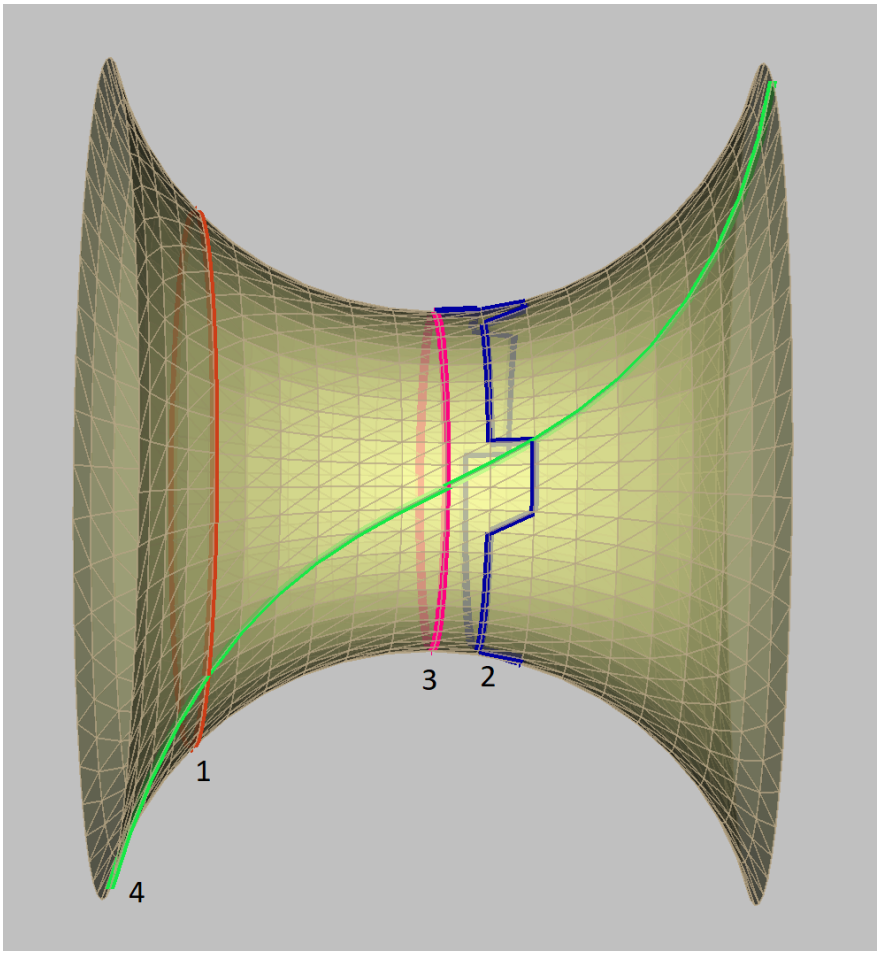

Figure 3.4: Good and bad fundamental cycles are shown on a section of a torus. Cycle 3 is the cycle with the shortest path, which is optimal for this tunnel. Cycle 2 contains unnecessary contours, so it is not desired. Cycle 4 does not have any unnecessary local contours but has a tilt orientation, which is also not suitable for mechanical and optical simulation. Cycle 1, though it is longer than the optimal cycle 3, expresses almost the same information to the viewers. Thus we consider it also as good.

3.3.2 Cycle tightening

Each computed fundamental cycle is a representative of other cycles that are topologically equivalent to it. The weighting function introduced in the last section prefers representative loops in the same slice and with the shorter distance, but artifacts may still exist, e.g., the loop might be unnecessarily long or winding. So we further refine and tighten the loop.

A few previous works explore tightening the loops. [15] applies "geodesic size" to control the

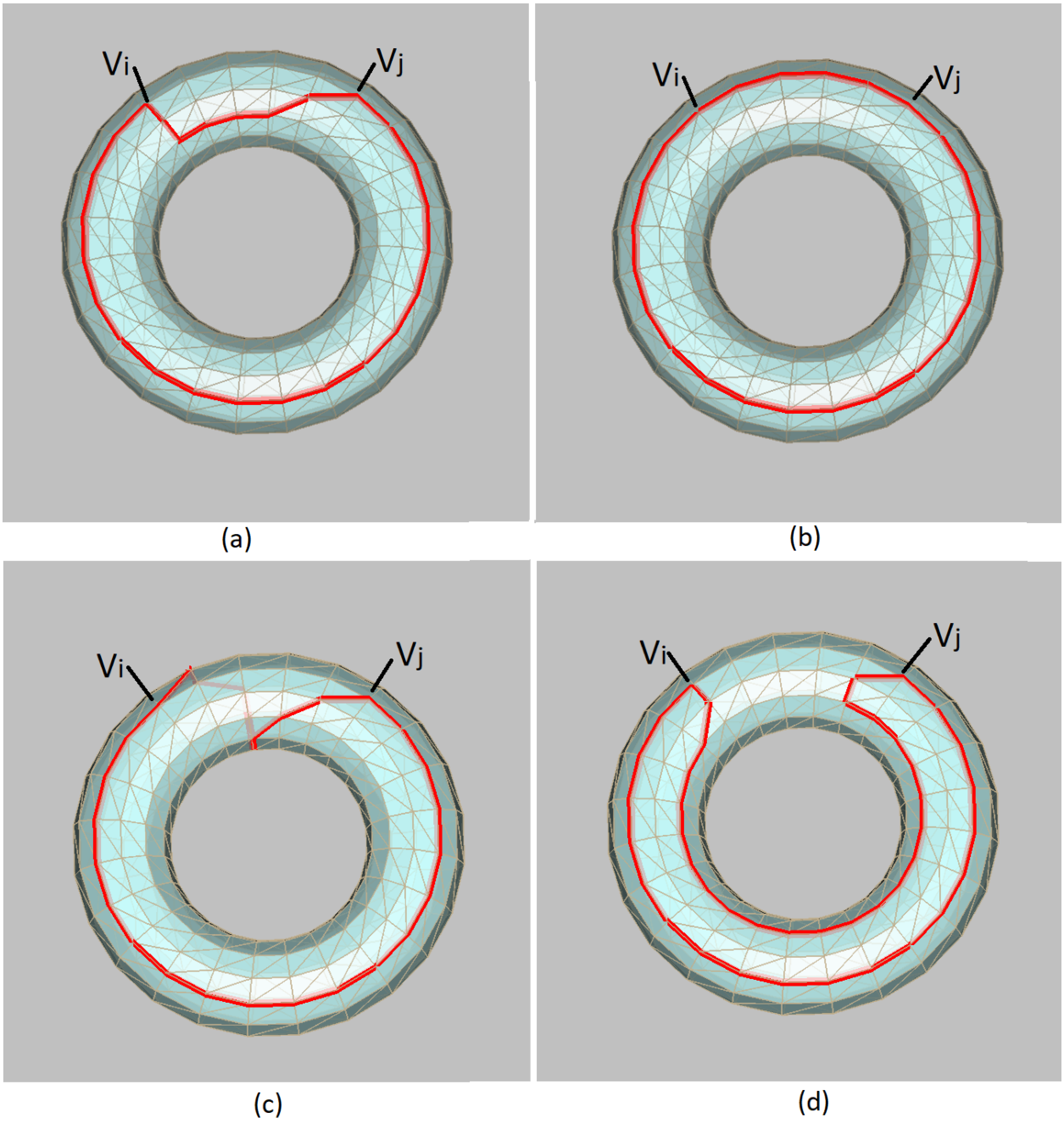

Figure 3.5: Homotopy equivalence. Given a cycle r_1 in (a), r_2 in (b) is homotopy equivalent to r_1, while cycles r_3 (c) and r_4 (d) are not as they cannot be continuously deformed to r_1. While tightening cycles, we have only homotopy equivalent cycles in the search space.

order of adding triangles into the filtration. This method works well on persistent homology-based methods, but may not be extended to the methods which do not have the process of filtration. [26] deforms the loops along the gradient of the distance field, thus suffers from the local minimum and also may not work well on noisy surfaces. [16] enumerates all the fundamental cycles, ordering them by their length, and then find the independent loops greedily. However, the enumeration of all canonical loops makes it only feasible for a small data set. [13] constructs shortest path trees at "base points", and finds independent handle loops using "annotations". All of those methods seek for loops with the shortest length, and thus searching is inevitable. Note that as this searching problem is thought to be NP-hard in general[13], the aforementioned methods all apply heuristics and do not guarantee to find the actual optimal loops. For our application, as shown in Fig. 3.4, a "good" tunnel does not have to be the shortest regarding the sum of the edge lengths. Instead, a good tunnel for visualization and mechanical simulation (1) should not have unnecessary contours, e.g., cycle 2 in Fig. 3.4 (2) should not have tilted orientation, as cycle 4 in Fig. 3.4. Therefore, in order to tighten the initial fundamental cycles into such defined good ones, instead of searching for the shortest length loops, we iteratively apply refinements as follows.

Homotopy equivalent cycles: Let's recall that each fundamental cycle we find is a representative of a class of equivalent cycles. Two cycles are *homotopy equivalent* if one of them can be continuously deformed into the other [19]. On the triangular mesh, we discretize this relationship by two rules: (1) if cycle r_i is different from r_j by only one triangle, then they are equivalent to each other (2) if r_i is equivalent to r_j, and r_i is equivalent to r_k, then r_j is equivalent to r_k.

During cycle tightening, we need to ensure that the tightened cycle is homotopy equivalent to the original cycle. Each step of the cycle tightening carefully replaces the edges in the original cycle with alternative ones. For example, in Fig. 3.5, when replacing the edges between vertices i and j with an arbitrary path from i to j, the modified loop may not be

homotopy equivalent to the original cycle. In order to avoid cases such as (c) and (d) in Fig. 3.5, we restrict the searching of alternative edges as follows (1) In each step, we search alternative edges only in *one-ring neighborhood* of the current cycle, which is composed by all incident faces or edges and neighboring vertices of the current cycle. This is to avoid case (c) in Fig. 3.5, as the path in one ring neighborhood is impossible to create new winding around the surface. (2) The one-ring neighborhood of a fundamental cycle has three edge loops, and we label them with consistent direction, as shown in Fig. 3.6. In order to avoid the case (d) in Fig. 3.5 , the alternative path is required to be consistent with the labeled edge direction. (3) Additionally, in the alternative path, no vertex is allowed to appear more than once.

Path cost definition: Given a fundamental cycle, we seek for a better alternative from all the homotopy equivalent cycles in its one ring neighbor. As discussed earlier, we consider a cycle as good when it has no unnecessary contours, and correct in orientation. The unnecessary contours can be avoided using the sum of edge lengths as path cost. However, as shown in Fig. 3.8, even if the path is the shortest in the search space, it cannot guarantee the desired orientation. We refine the orientation of the tunnel loops based on a simple observation: when a cycle is tilted, in the neighborhood of the vertex v_i on the cycle, there exist a vertex that is nearer to the centroid C of the cycle than v_i, as shown in Fig 3.8. Therefore, we apply the distance between the vertex and the centroid as a measure for orientation correction. Combining these two, for a cycle γ we have the path cost:

$$\phi_\gamma = \alpha \sum_{v_i \in \gamma} |\overrightarrow{v_i C}| + \sum_{e_i \in \gamma} |e_i| \tag{3.1}$$

where e_i is the ith edge in the cycle, and α is a user-specified coefficient which keeps the balance between edge lengths and orientation correctness.

Cycle searching as a scheduling problem: We iteratively search for a better alternative

cycle and replace the original one with it. The search space gets big when the length of the original cycle is long, thus enumeration is not feasible. To simplify the problem, we consider another problem first: given a vertex v_i in the one ring neighborhood, what is the shortest non-trivial loop which passes through v_i? As shown in Fig. 3.7, we flatten the one ring neighborhood, then the region can be considered as three assembly lines with v_i as their starting and ending points when a job is on the assembly lines, it has to follow the labeled dicrections but they can switch between the lines freely along the undirected edges. We define the cost the same way as in eq. 3.1, and the shortest schedule is corresponding to the optimal path in the original one-ring neighborhood. To choose v_i which is the starting and ending point of the schedule, we greedily pick the vertex that is nearest to the centroid in the one ring. As scheduling problem can be solved by dynamic programming in $O(n)$ time [11], the searching in each step can be done in $O(n)$ time, where n is the number of vertices in the original fundamental cycle. Note that since we search for alternative cycles only in one ring neighbor, the algorithm does not find the globally shortest path in most cases. However, as discussed earlier global shortest path searching is an NP-hard problem, and our goal is not the shortest path cycle but non-contouring and orientation correct cycle which is good for visualization and mechanical simulation.

3.3.3 Decoupling composite fundamental cycles.

Our fundamental cycle identification algorithm finds $2g$ fundamental cycles, which may be (1) tunnels, (2) handles, and (3) ones that are neither tunnel nor handle. For the third class, as it can be considered as a composition of tunnels and handles in topology, we call it a *composite cycle*. The composite cycles are not desired, they may produce visually unpleasant results, and furthermore, if we do not decouple them into pure tunnels and handles, we may miss tunnels which are expected to be found. Fig 3.9 shows the procedure for decoupling the composite cycles into tunnels and handles.

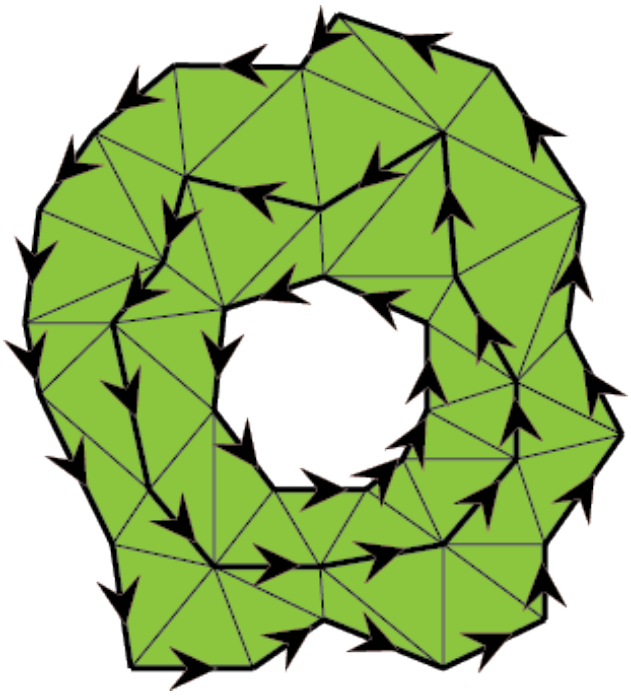

Figure 3.6: Searching for better alternative cycle in one ring neighborhood. The one ring neighborhood is composed by all the vertices adjacent to the original fundamental cycle. The alternative cycle should be consistent with the labelled directions, and have no self-intersections.

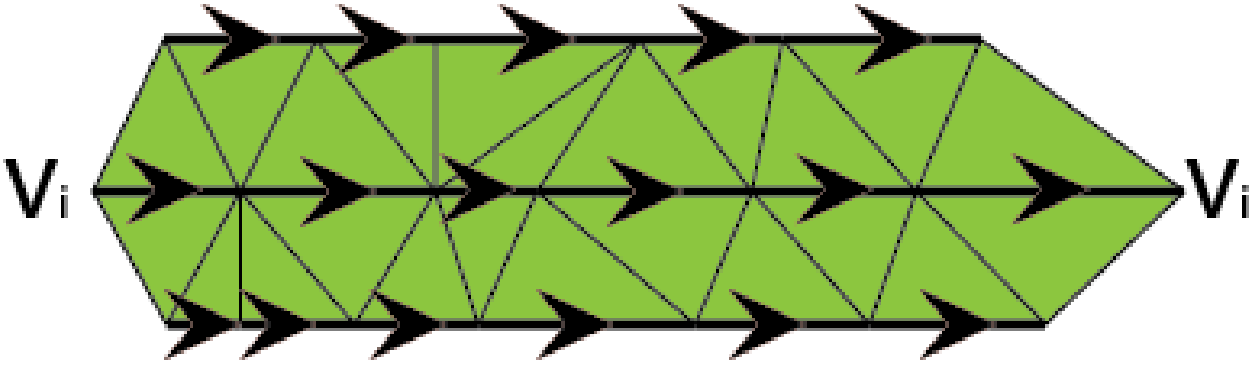

Figure 3.7: If we consider v_i as starting and ending point, and flatten the one ring neighborhood, the searching problem becomes a scheduling problem with three assembly lines. The job can be switched between assembly lines along the undirected edges, and the schedule with the lowest cost is corresponding to the optimal alternative cycle.

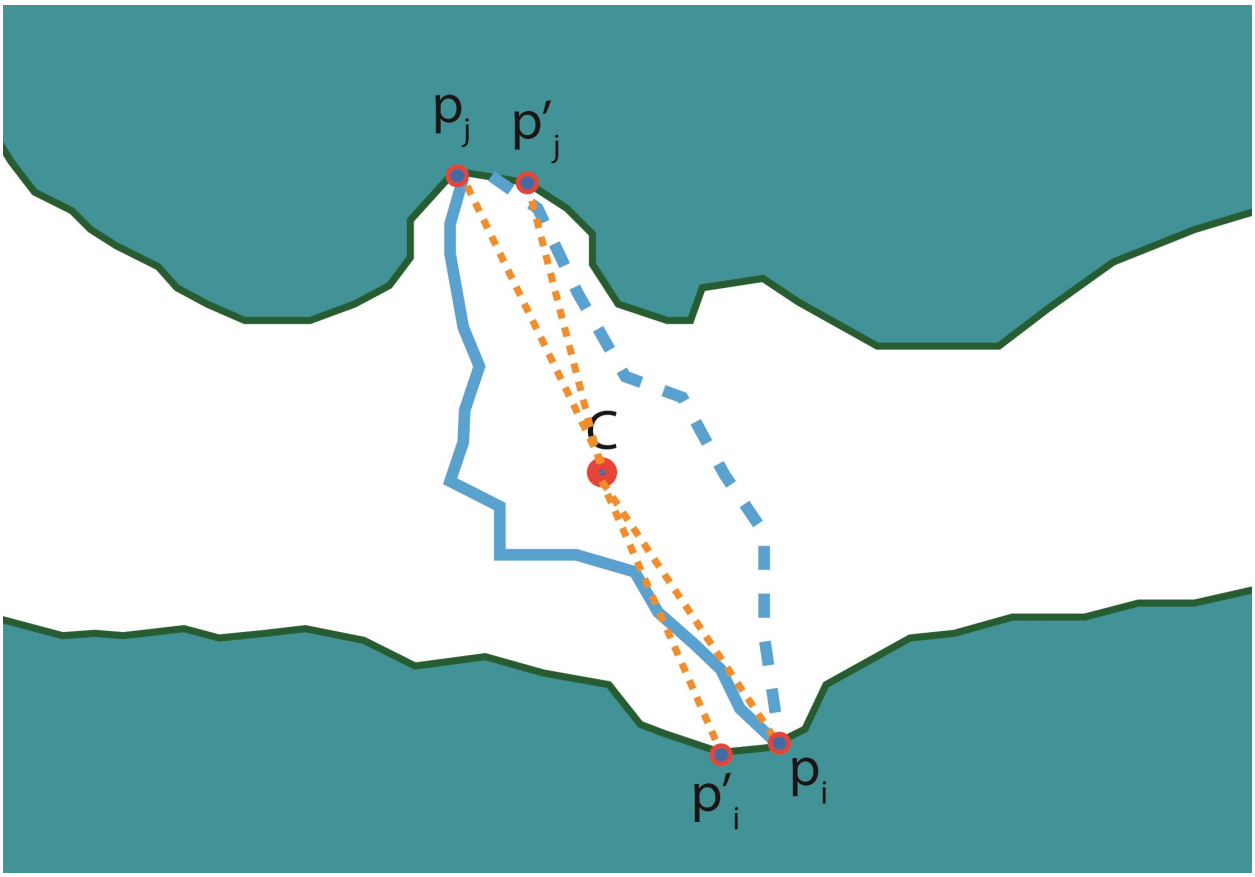

Figure 3.8: Orientation refinement. If we consider only path in one ring neighborhood, the local shortest paths chosen may have undesired orientation. By computing the distance from vertices to the centroid of the cycle, the orientation is estimated.

Step 1. Detection of the composite cycles: We observe that, if the triangulation is dense enough, a handle and a tunnel, after tightening of both cycles, will share no more than an edge. If two intersecting fundamental cycles share more than one edge after cycle tightening, one of them has to be a composite cycle as shown in Fig. 3.9. If the triangulation is sparse, a tunnel and a handle may share more than one edge, and hence we may be looking for a composite cycle. But the following steps in the algorithm can handle such false-positives in our classification.

Step 2. Replacing shared edges: We replace the shared edges between cycles with the part that is not shared. We label the edges in the two cycles into three groups (A) the edges

in cycle 1 but not in cycle 2; (B) the edges in cycle 2 but not in cycle 1; (C) the edges in both cycle 1 and 2. We may represent the original cycles we found as cycle 1=$A + C$, cycle 2=$B + C$. Then if $|A| < |B|$, then we modify cycle 2 to $A + B$, and similarly if $|A| > |B|$, we modify cycle 1 to $A + B$.

Step 3. Tightening.: After replacing the shared edges, we apply the cycle tightening introduced in the last section on the modified cycles. We iteratively run the whole procedure until no further change is made to any cycle.

Remark: It is possible that a mixed cycle does not have any intersection with any other fundamental cycle, and thus our algorithm cannot detect this case. However, such occurrences seem rare and we have not observed in our experiments.

3.4 Results

We apply our algorithm on a series of data sets, including synthesized high genus meshes and meshes extracted from corneal NLO-HRMac image slices. The synthesized meshes are smooth, but with a high genus. The meshes extracted from image slices are from cornea structures of dog, hawk, and chicken. The sizes of the meshes are shown in Table 4.1. Due to the complexity of our data sets, the 3D tetrahedralization is very expensive, thus the 3D tessellation based methods, [15] [26] are not feasible for failing in our data sets. [18] is efficient to work on a large data set, but does not have cycle refinement. The only previous work that can work on large size data sets while has cycle refinement is [13]. In this paper, we extensively compare our algorithm and [13], and evaluate the algorithm from the quality of the identified tunnels, the performance, and the robustness of the algorithm.

Quality of the found tunnels: Fig. 3.13 shows the tunnels found by our algorithm

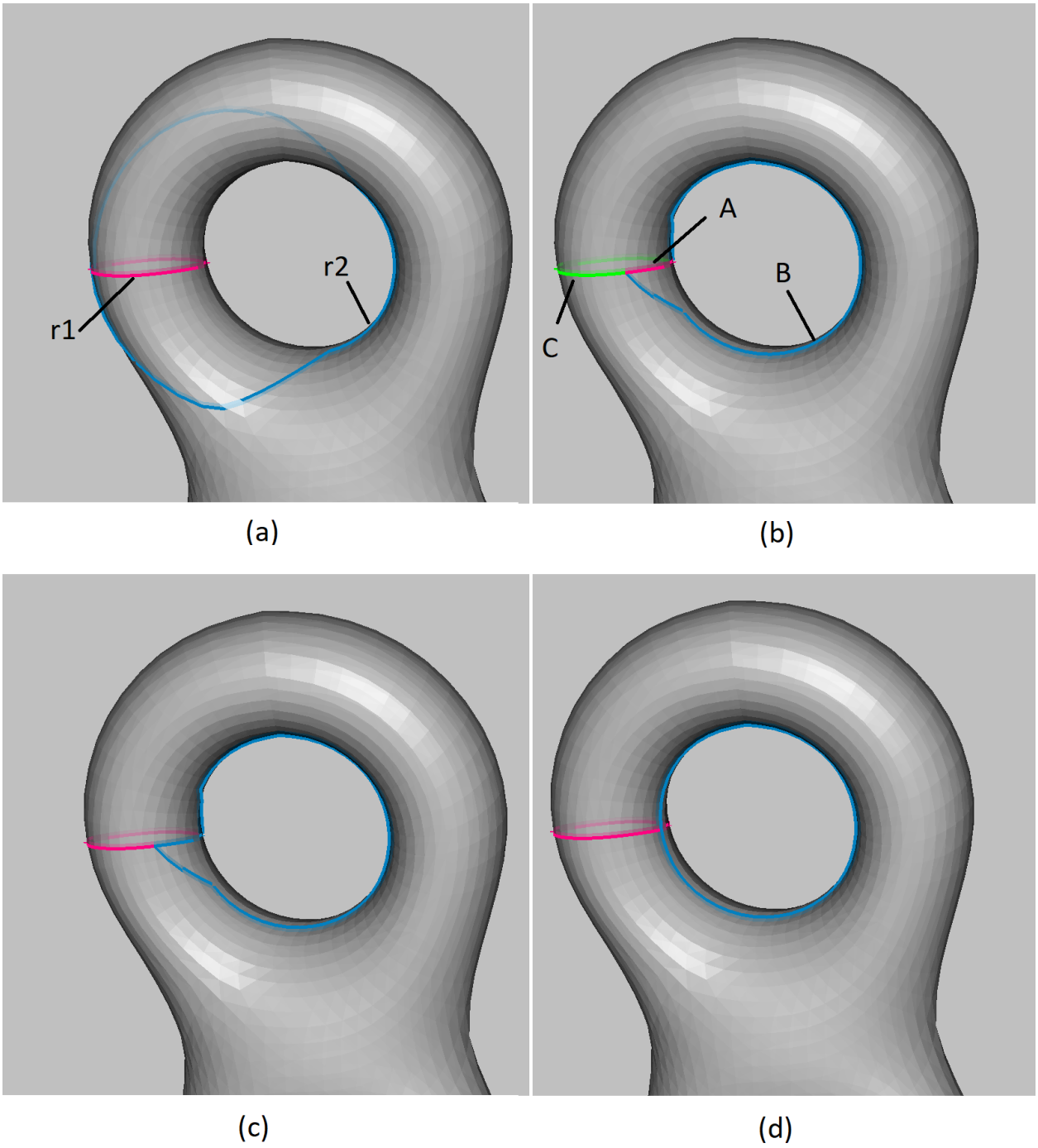

Figure 3.9: Decoupling composite fundamental cycles. (a)The fundamental cycle identification algorithm may find cycles which are neither tunnels or handles, such as r2. (b) After tightening, this composite cycle shares common edges with adjacent fundamental. r1 is composed by A and C, and r2 is composed by B and C (c) The composite cycle can be decoupled by replacing r2=B+C with r2=A+B (d) The shape of decoupled cycle can be further refined.

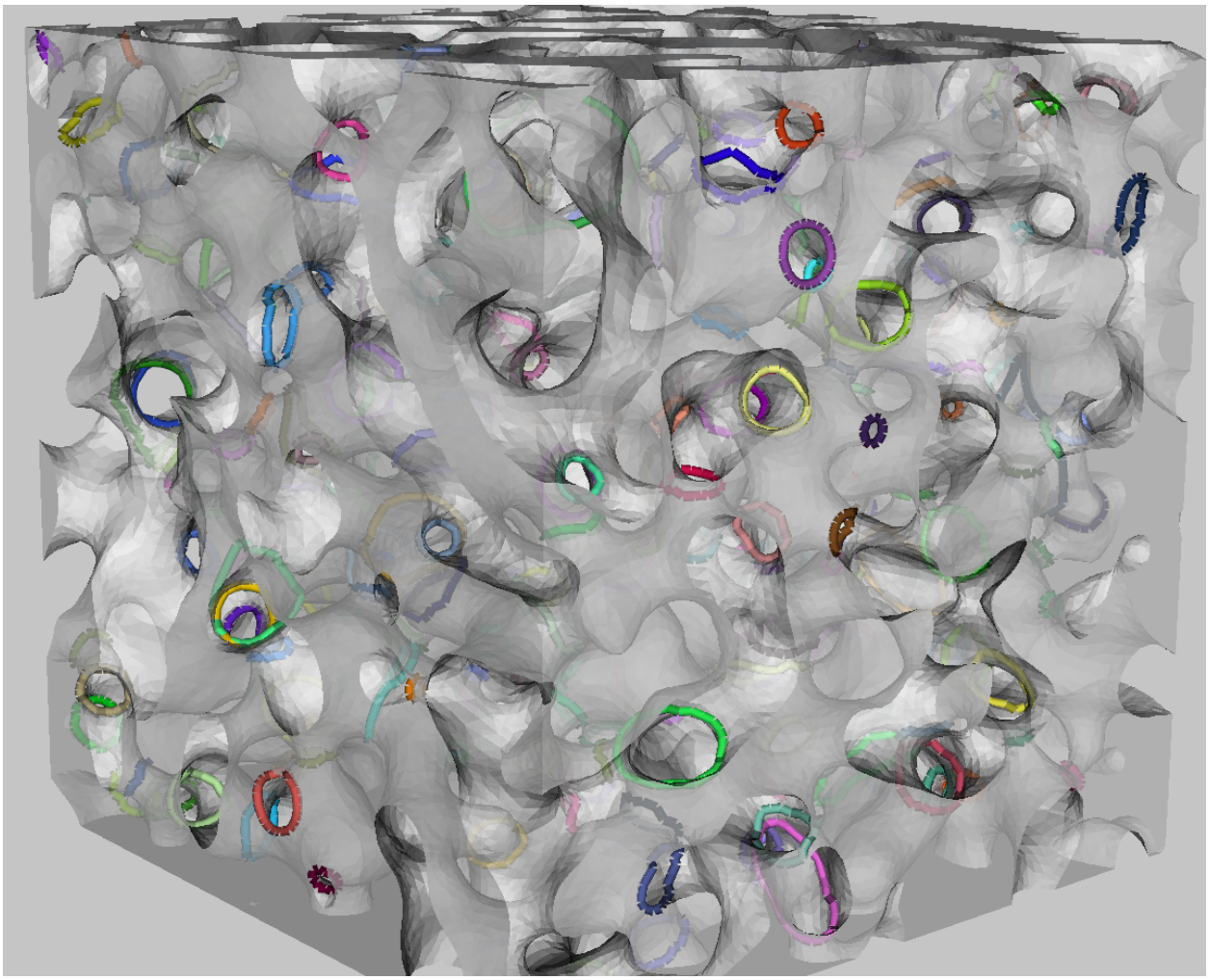

Figure 3.10: Tunnels found from a synthesized mesh with 404 randomly generated tunnels in various sizes.

and by [13]. As [13] seeks for shortest path loop, the tunnels found by [13] are generally smaller in size. However, for visualization and mechanical simulation purpose, the result of ours is comparable with theirs. Fig. 3.12 shows [13]'s and our result of a model which has two interlocking tunnels. [13] relies on constructing a Reeb graph from the surface, which may introduce problems when two tunnels cross with each other. Our method is based on the tree-cotree decomposition of the surface, therefore is not oblivious to such topological relationship.

Performance: Table 4.1 shows the timing result. [13]'s performance is greatly affected by

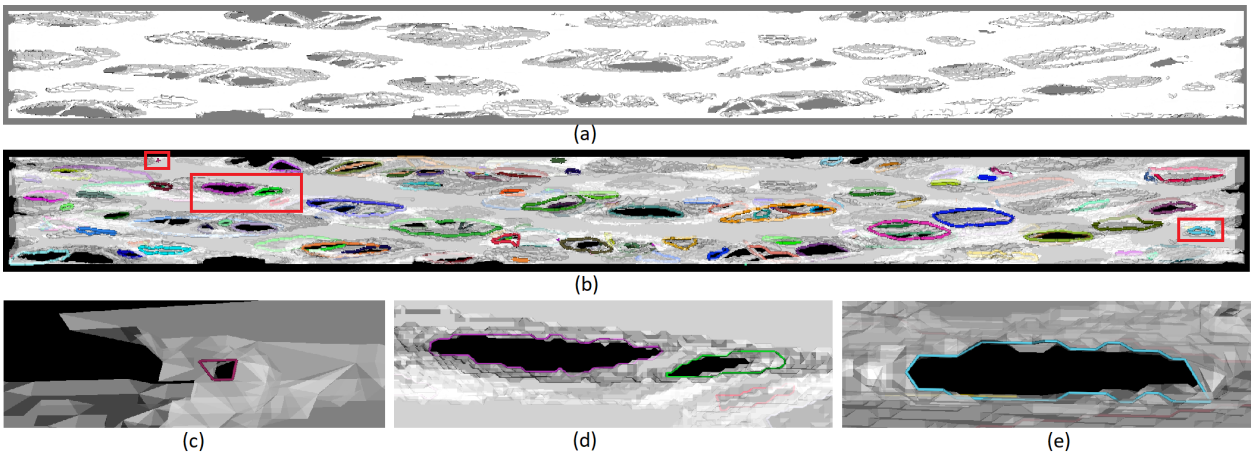

Figure 3.11: Tunnels found by our algorithm from hawk data set. (a) manifold mesh extracted from corneal images. (b) tunnels found by our algorithm (c-e) zoom in view of selected regions, captured from a different point of view for visualizing nearby geometry.

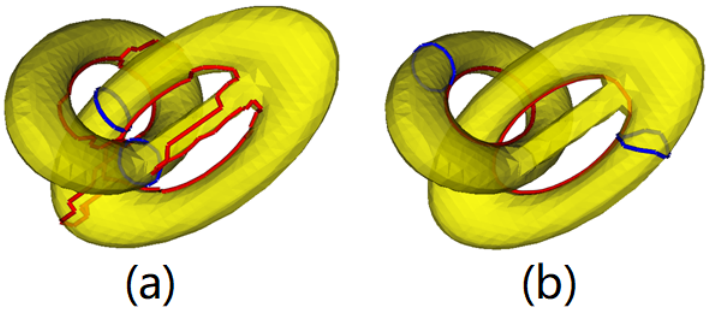

Figure 3.12: Comparison of [13]'s and our result on interlocking tunnel model. (a) [13]'s result. (b) ours. [13] relies on constructing a Reeb graph, which may introduce problems when two tunnels cross with each other, while our method does not have such limitation.

the random direction along which it generates Reeb Graph, for a fair comparison, we run the program several times and choose the shortest timing. For chicken and synthesized_2 data sets, [13] fails to identify the fundamental cycles as these meshes have a high genus and complex structure, in which a good Reeb graph for identifying fundamental cycles is difficult to build.

Robustness: [13] is based on the construction of the Reeb graph, and the quality of the Reeb graph determines if all the fundamental cycles can be identified. As shown in Table 4.1, for chicken and synthesized_2 data sets, due to the complexity and the high genus of the models,

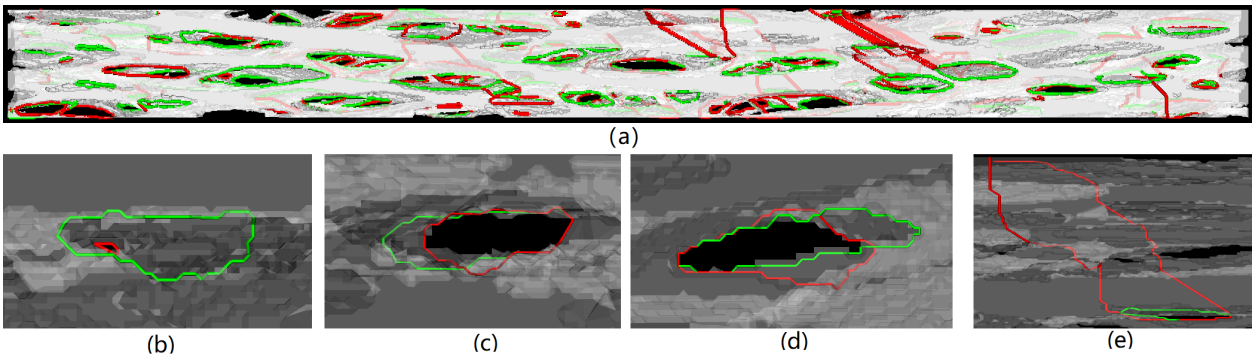

Figure 3.13: (a) Our vs. [13]'s results on hawk data set, ours are colored as green while [13]'s are colored as red. (b-d) Zoom in views of selected tunnels. Some of the tunnels found by our algorithm are larger in size compared to [13], but this does not have big negavtive impact on visualization and mechanical simulation purpose. (e) Due to numerical issues of matrix inversion, [13] may introduce excessively long tunnels, while our algorithm is stable in all cases.

Model details			[13] (sec)			Our algorithm(sec)		
Data set	triangles	genus	step 1-5	tightening	total	cycle	tightening	total
synthesized_1	117,824	192	180	1,811	1,991	178	268	446
synthesized_2	231,138	404	N/A	N/A	N/A	561	878	1,439
dog	782,962	53	205	3,546	3,751	283	937	1,220
hawk	793,432	148	360	N/A	N/A	538	1,429	1,967
chicken	994,136	562	N/A	N/A	N/A	849	1,926	2,775

Table 3.1: Timing results of our algorithm and [13]'s. Synthesized_1 and synthesized_2 are smooth meshes randomly generated with a large number of tunnels. The others are manifold meshes extracted from NLO-HRMac image slices. Tightening column includes time for both tightening and classification. "N/A" means the algorithm failed to finish in two hours after ten times trials.

no good Reeb graph can be constructed, thus [13] is unable to identify fundamental cycles on these data sets. Furthermore, [13] relies on inversion of a linking number matrix, and we observe that the numerical issue related to the matrix inversion may introduce excessively long tunnels. As every step of our algorithm is based on basic geometry operation, our algorithm does not have such issues.

3.5 Discussion

We have introduced an algorithm to extract tunnel and handle cycles from a manifold mesh. The algorithm is designed to deal with the complex and noisy structure. We apply an iterative tree-cotree algorithm to find fundamental cycles, and in each iteration, the computed cycles are guaranteed to be topologically independent. On the computed fundamental cycles, we present algorithms to tighten cycles and resolve cases of composite cycles. To identify the tunnels from fundamental cycles which include both tunnels and non-tunnels, we present a classification algorithm based on the fundamental cycles' geometric properties and their relationship between each other. Finally, we demonstrate the results of our algorithm on various models. The results show that the algorithm is applicable to various complex model and is robust to geometric noise.

Chapter 4

Tori decomposition

Decomposing a 3D shape into smaller and simpler parts is a fundamental yet challenging problem in mesh processing. As most mesh processing methods have limitations on the input model's geometric and topological complexities, similar to a divide-and-conquer strategy, partitioning a complex shape into smaller pieces is often the basis and prerequisite for further processing. The result of such shape decompositions can be used in various fields of computer graphics such as shape deformation [52], geometric modeling [9], and shape editing [55].

The shape decomposition techniques can be categorized into two classes: geometry-based methods and topology-based methods. The geometry-based methods, e.g. polycube decomposition [39], cylinder decomposition [58], etc., seek to partition the surface into geometrically simpler components. Such methods take predefined primitive shapes such as planes, spheres, cylinders or cubes as their target components and apply either a top-down or a bottom-up approach to fit parts of the surface shape to the primitives. However, these techniques do not guarantee any topological property of the decomposed components. The topology-based methods segment the shape into prescribed topological components, e.g. topological disks [23][14] [13], topological pants [36], or stars [54], but most of the works do not take geometry into account. Our work falls into the class of the topology-based methods, but we desire the result to be not only of prescribed topology but also good in geometry.

In this chapter, we introduce a novel topology based decomposition called tori decomposition to partition a surface mesh with genus g into g components each of which has genus-1. The torus is the simplest topologically non-trivial shape. Besides that, the tori decomposition is of great practical significance. In 3D art, a torus can be an element for generating abstract artworks [44]. In surface matching, the relative location of the decomposed tori helps consistently match parts in different models. In vector field processing, since on a torus there exist differentiable vector fields with no singular points, tori decomposition helps to avoid singular points or to determine where the singular points should be located. In 3D printing, there is a growing interest in 3D printed coils which are more suitable for manufacturing genus-1 objects. A tori decomposition may make application of these manufacturing techniques suitable for more complicated shapes by partitioning the shapes into genus-1 components.

The problem of finding tori decomposition is equivalent to seeking $g-1$ splitting cycles such that when cut along these splitting cycles, the surface is decomposed into g topologically nontrivial components. Computing the shortest splitting cycle on a given surface is NP-hard [7], so instead of directly searching for shortest splitting cycles, we formulate the problem as finding a minimum-weight graph cut in the dual graph of the surface mesh. We first find the tunnel and handle cycles on the surface, and then we consider the tunnel cycles as terminals for finding a minimum-weight cut. We analyze each possible result of min-cut on a surface-embedded graph and design our algorithm to guarantee that the cycles we compute are all splitting cycles. The weights assigned to the edges of the dual graph for the min-cut problem are designed to produce geometrically pleasant cuts. The method is fully automatic-not requiring the users to provide any seed points.

In summary, our main contributions are as follows:

- We introduce a new topology decomposition of geometric manifold models called tori-decomposition and present an analysis of this problem

- We formulate the geometry aware tori decomposition problem and show results in various high genus models

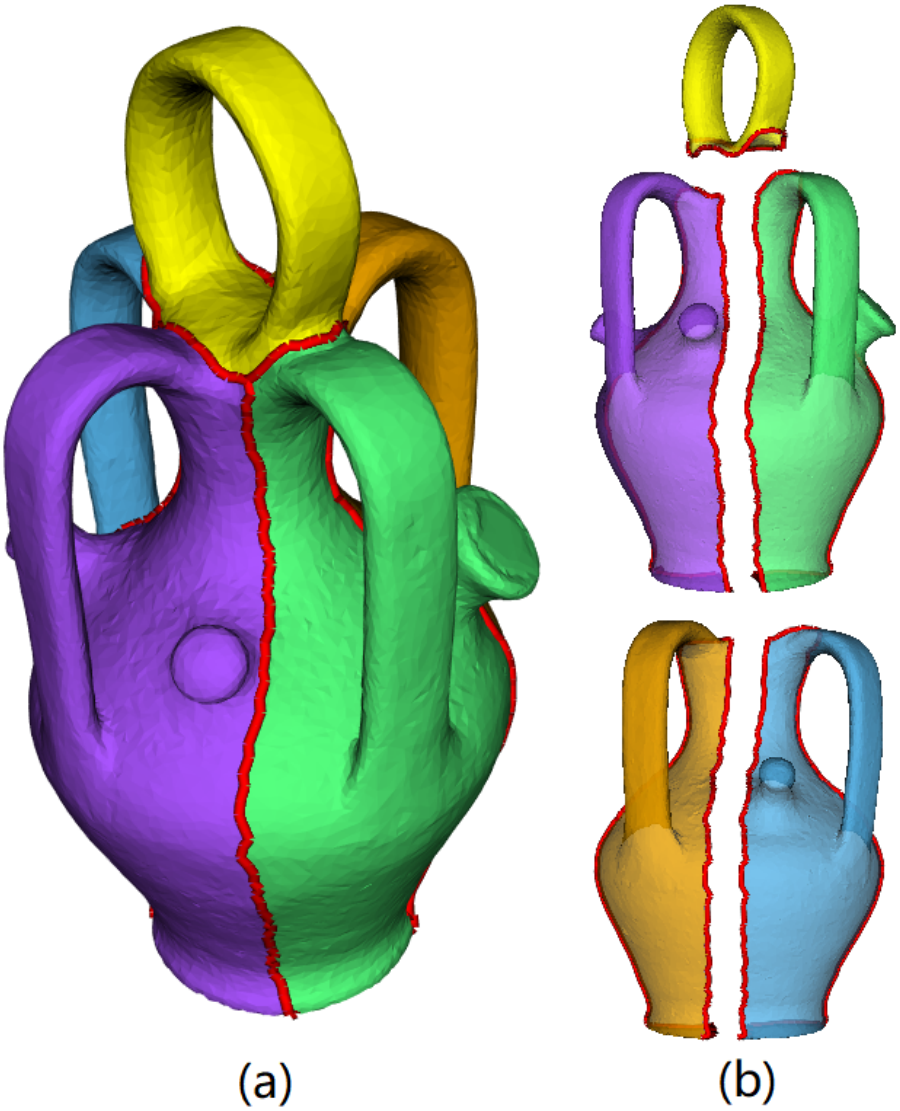

Figure 4.1: Tori decomposition generated by our method, each of the decomposed component has genus 1.

4.1 Related work

Shape decomposition is a well-studied problem with extensive surveys, e.g. [45]. Many methods are designed to segment the shape based on shape semantics or primitive fitting. Our work has a different focus: instead of seeking semantically or geometrically simple components, our goal is to decompose a shape into *topologically simple but non-trivial* components. Here we review previous works on mesh segmentation, fundamental cycle localization, and topological decomposition of surface shapes with a focus on those related to our problem.

4.1.1 Mesh segmentation

Depending on the application of the segmentation, mesh segmentation can be categorized into two classes. The first class, which is often called primitive fitting[2] or shape approximation[10], is mainly used for reverse engineering purposes. This category of segmentation cuts the mesh into patches, and each patch is matched with one of the predefined primitive shapes such as planes, spheres, or cylinders. The second class segments organic meshes into meaningful parts using higher-level constraints such as *minima rule*[34] and symmetry. The goal of these methods is to be consistent with human perception of shape geometry. Our approach falls into the second category since we consider higher-level information-not just the geometry, but also the shape's topology.

A series of segmentation works are based on *minima rule* or *part salience* from cognitive theory, which states that human perception usually divides a surface into parts along the concave discontinuity of the tangent plane. [34] applies salience to find candidate contours. [3] constructs a set of concavity-sensitive scalar fields to locate concave creases and seams. Statistics-driven automatic segmentation algorithms have also been proposed, which attempt to solve the segmentation problem by learning semantic information from human-labeled training data [40]. More complete surveys on mesh segmentation techniques can be found in

[45] and [43]. However, none of these methods consider topology. A few works such as [49] use the term "topology" to denote the hierarchical relationship between parts of the shape, which is not the focus of our work.

4.1.2 Topological shape decomposition

Although many 3-manifold decomposition works exist, most of the theoretical findings on the 3-manifold cannot be directly applied to 2-manifold in 3D. Here we review only the work on the decomposition of 2D surfaces in 3D. Topological decomposition of surface meshes includes pants decomposition [36][57], punctured tori decomposition [7], etc. Li et al. [36] segment a surface mesh into a set of pants patches where each pants patch is a genus-zero surface with three boundaries and applies the segmentation methods for finding a consistent surface mapping among a set of surfaces. Zhang and Li [57] traverse different classes of pants decompositions and search for the optimal one using a pre-defined geometric criterion that includes length, symmetry, and concaveness. [38] exhaustively partitions a solid into a set of tubular parts using a curve skeleton. The work that is most relevant to our problem is that of Chambers [7] that presents a theoretical approach to constructing splitting cycles which split the surface into two disconnected surfaces of prescribed topology and proposes a greedy algorithm that leads to a decomposition into punctured tori. However, the cycles computed by [7] need not be short or geometry-aware. There is also no practical implementation of this theoretical approach. Our method is a geometry-aware topology-based decomposition of a surface mesh into multiple tori components.

4.2 Problem formulation

4.2.1 Tori decomposition

Definition: Tori decomposition decomposes a surface of non-trivial topology into multiple components, each of which is a punctured torus.

A few previous works in topology have defined similar decompositions. For example, Heegaard splitting [6] decomposes a compact oriented 3-manifold that results from dividing it into two handlebodies, but the result cannot be easily extended to 2-manifold surfaces. A definition of tori decomposition similar to ours is given in [7], in which the tori decomposition is defined as a set of simple, pairwise disjoint cycles that split M into g punctured tori. The requirement of cycles being disjoint often leads to bad geometry of the individual torus component, which is counterintuitive for human perception. Thus, in this chapter, we remove this requirement.

4.2.2 Relationship between pants decomposition and torus decomposition

A **pants decomposition** partitions a surface into components each of which has genus 0, and three boundaries [57][36]. Intuitively, tori decomposition is strongly related to pants decomposition. Imagine that out of the three boundaries of the pants, two are the same, we may form a torus by merging the two common boundaries together, resulting in tori decomposition. So if we get a pants decomposition, we may be able to convert it into a tori decomposition. However, as shown in Fig. 4.2, this is not always true, Fig. 4.2 shows two different pants decompositions of a double-torus, one of them can be converted into tori decomposition while not the other. In fact, any topologically non-trivial shape with genus ¿ 1 has multiple ways of pants decomposition, but very few of them can be converted into a tori decomposition. However, conversely, every tori decomposition can be further segmented

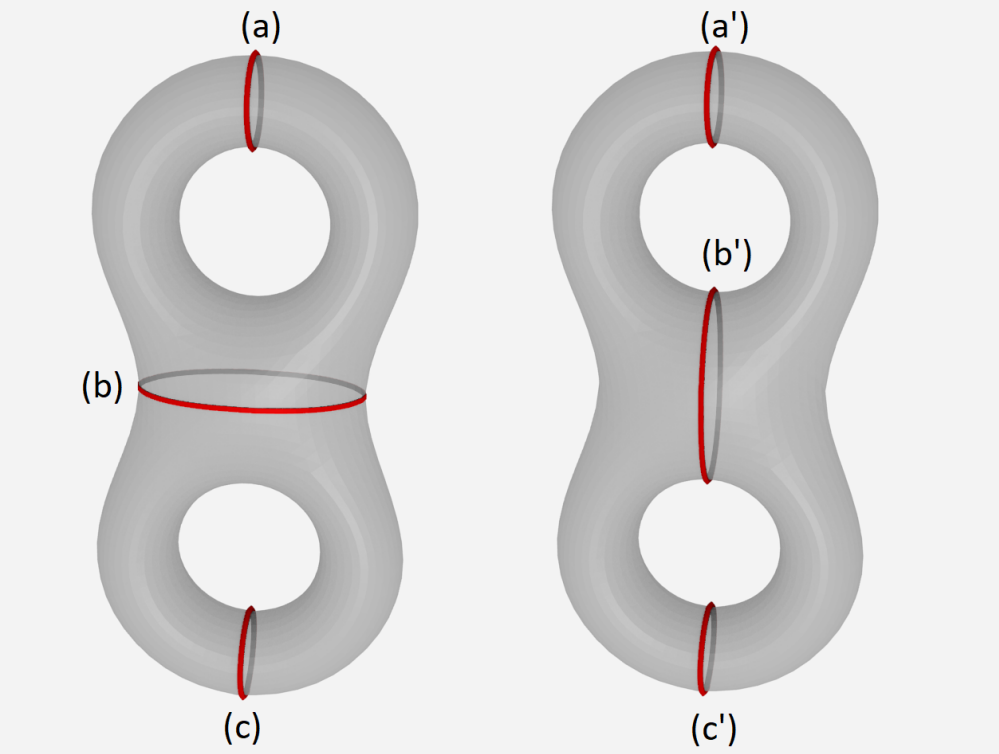

Figure 4.2: Cut along the red cycles, both left and right figures result in valid pants decomposition where the surface is split into two pants components, each with three boundary cycles. The left decomposition can be converted to tori decomposition by repairing the cut (a) and (c), but for the decomposition on the right no such conversion exists.

into pants using the A-move, and S-move introduced in [31]. Thus, a tori decomposition has all the advantages and potential applications of a pants decomposition. In summary, our tori decomposition can always be converted into a pants decomposition, but not all pants decompositions can be converted into a tori decomposition.

4.3 Geometry-aware tori decomposition

The tunnel cycles as computed above capture the topological features of the interior space, while the handle cycles capture mostly the topological features of the exterior space. These cycles are simple, non-contractible and non-separating. But to get a tori decomposition,

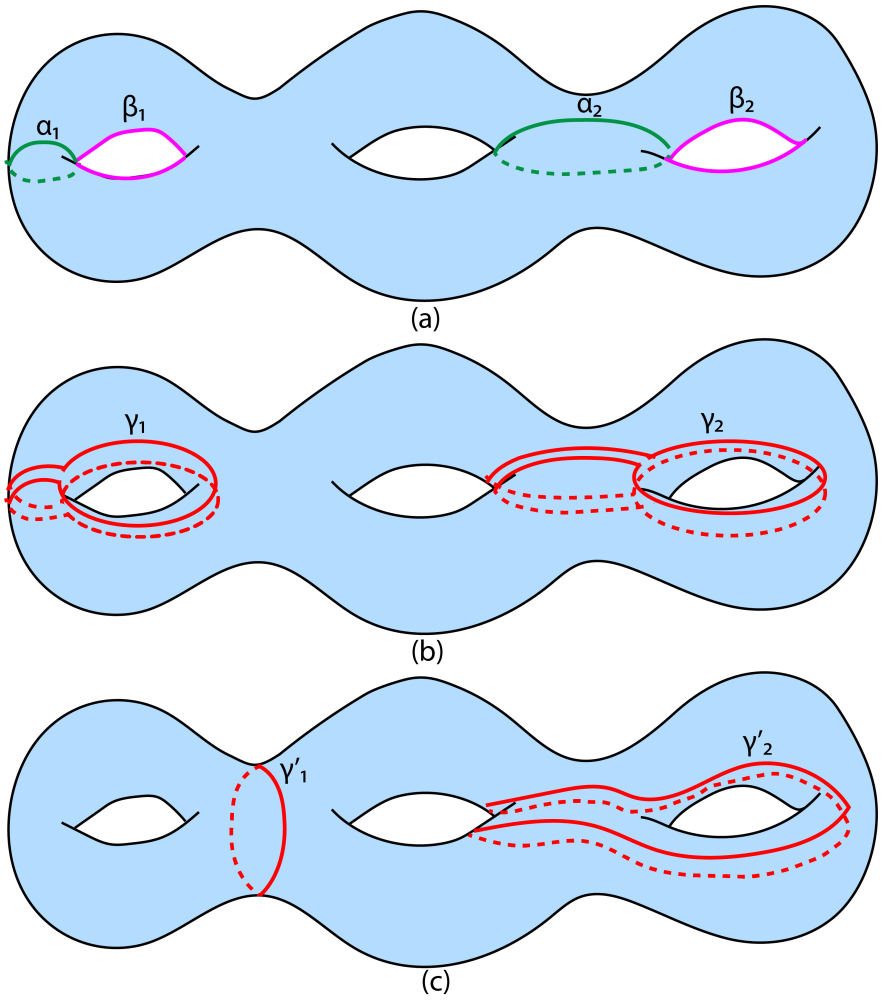

Figure 4.3: A well-known technique to compose splitting cycle from fundamental cycles: (a) given a pair of intersecting fundamental cycles α and β, (b) the cycle $\gamma = \alpha \cdot \beta \cdot \bar{\alpha} \cdot \bar{\beta}$ is guaranteed to be a splitting cycle [7]. Note that the result cycles are all unnecessarily long and tightening them is not a trivial task. (c) Tightened result. Even if we tighten the generated splitting cycles (γ_1' is the shortest cycle homotopic to γ_1 and γ_2' is the shortest cycle homotopic to γ_2), some of the cycles e.g. γ_2' are still not desirable.

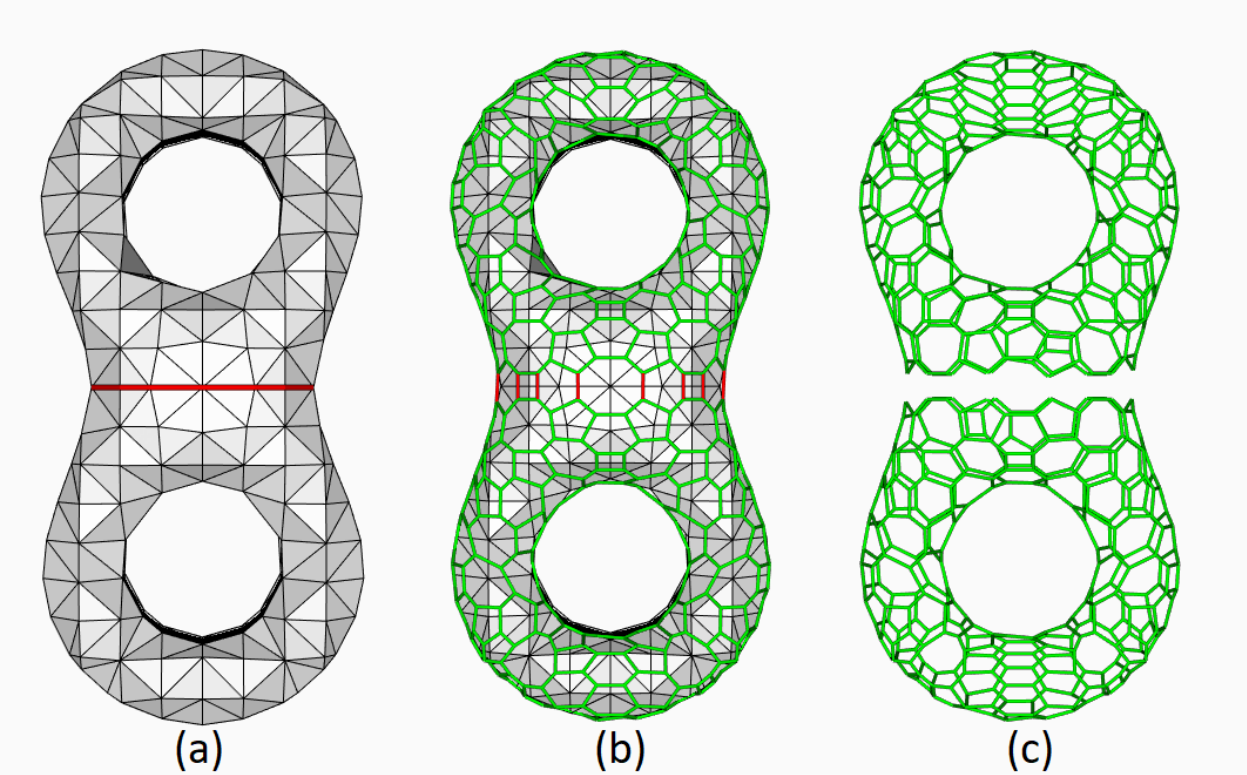

Figure 4.4: A splitting cycle and its dual cut. (a) a surface and a splitting cycle which splits it into two genus-1 pieces (b) the dual graph with the edges dual to the splitting cycle colored in red (c) if the edges dual to the splitting cycle is removed from the dual graph, the graph is split into two parts. In other words, the splitting cycle is dual to a cut in the dual graph.

we need to find *splitting cycles*, which are simple, non-contractible and separating. The easiest way known for composing a splitting cycle (not requiring that it is the shortest) is to combine two intersecting fundamental cycles: given two fundamental cycles α and β which intersect with each other, the cycle $\gamma = \alpha \cdot \beta \cdot \bar{\alpha} \cdot \bar{\beta}$ is a splitting cycle[7], where $\cdot$ denotes concatenation and $\bar{\alpha}$ denotes the reversed path of α. However, as shown in Fig. 4.3(b), the result of this operation is always unnecessarily long. Fig. 4.3(c) suggests that this is not just a geometry problem: even if we tighten each of the found cycles, i.e. find the shortest cycle homotopic to it, the tightened cycle may still not be optimal, e.g. γ_2' shown in the figure. (If one curve can be continuously deformed to another on the surface then the curves are said to be homotopic to each other.) Chambers [7] proves that finding the shortest splitting

cycle on a combinatorial surface is NP-hard. We pose the problem of computing splitting cycles into a set of single-source single-target min-cut problems.

4.3.1 Graph min-cuts as splitting cycles

Min-cut is a classical problem in graph theory and has been widely applied in mesh segmentation and mesh analysis [28]. The general approach is to select a set of K seed nodes in an edge-weighted graph and then find the minimum cost cut (min-cut) that partitions the seeds. This can be converted into a classical network flow problem with well-known polynomial-time solutions for $K = 2$, and approximation algorithms for $K > 2$ since multi-way min-cut is an NP-hard problem. The min-cut algorithm is more appropriate for our problem as (1) its resulting cycles are guaranteed to be simple, i.e. no repeated vertices, (2) it naturally leads to cycles with good geometry. For example, the cycles are shortest when taking the lengths as edge weights. But unlike the cases in planar graphs, when applying min-cut on meshes, trivial cuts may exist. For example, a cut which surrounds a source vertex and its infinitesimal neighborhood is a valid cut, but it is trivial as the resulting region is too small. Previous works e.g. [33] constrain cuts to lie within a "fuzzy" area to avoid the problem of making trivial cuts that encompass geometrically small size regions. In this chapter, we not only avoid geometrically trivial cuts but also avoid topologically trivial cuts.

Observation 1. A splitting cycle on the surface is dual to a cut in the dual graph, as shown in Fig. 4.4. Hence we run the min-cut algorithm on the dual graph of the mesh.

Observation 2. A cut in dual graph is not necessarily dual to a single splitting cycle. For example, as shown in Fig. 4.5, a cut in the dual graph may be dual to cases such as Fig. 4.5(b) and Fig. 4.5(c). 4.5(b) is not a splitting cycle as it is trivial, 4.5(c) contains multiple cycles, and those cycles are not splitting cycles. So we analyze possible cases of min-cut results and extend the min-cut algorithm to ensure that all the cycles we generate

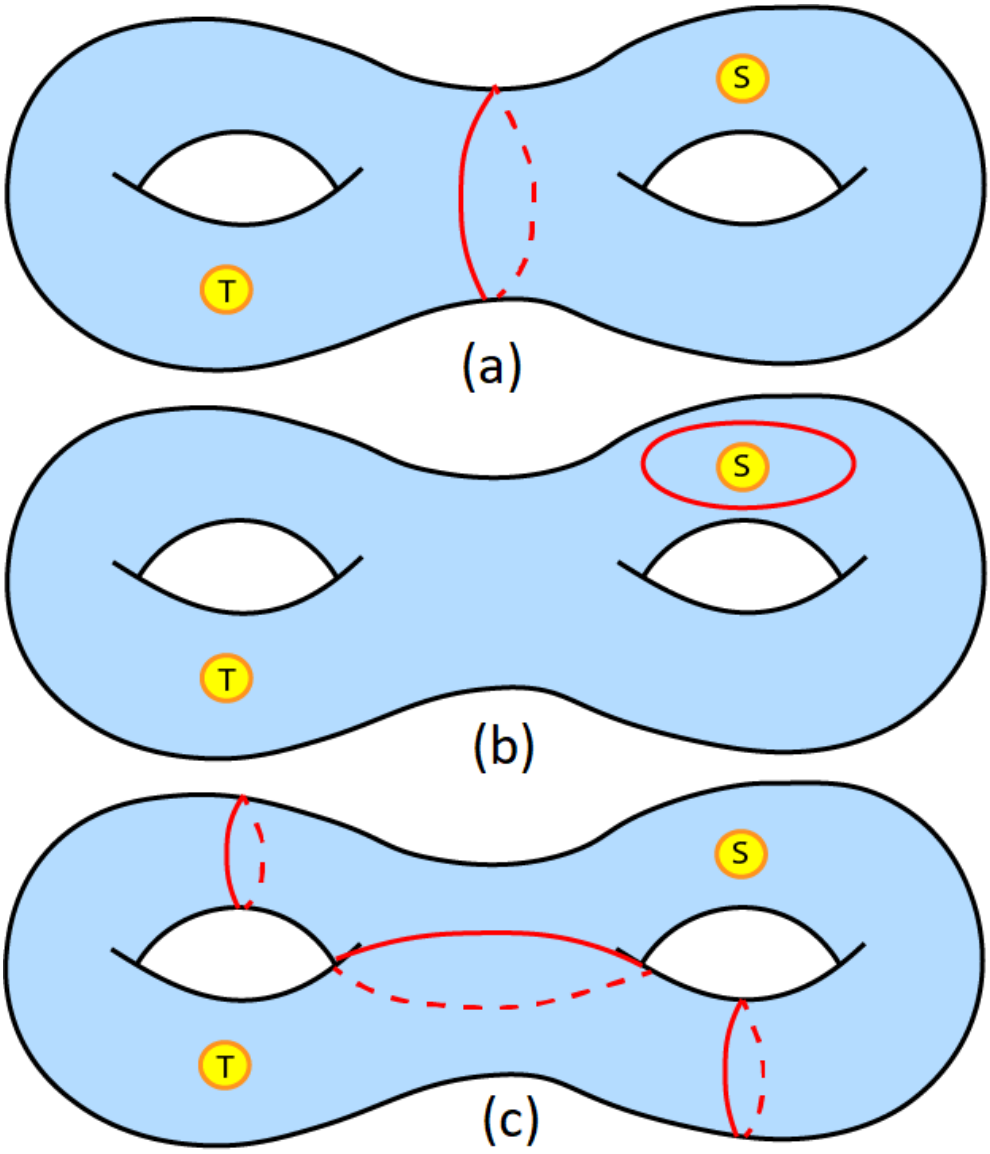

Figure 4.5: Three types of graph-cut results on embedded graphs: (a) splitting, for which the cut is a splitting cycle, (b) contractible, for which the cut is a contractible cycle, and (c) decomposable, for which the cut is composed of multiple non-separating and non-contractible cycles. Note that unlike the case for a planar graph, a cut on surface-embedded graph may be composed of multiple cycles.

are splitting cycles.

4.3.2 Finding a splitting cycle

Min-cut finds a cut c which consists of a set of edges in the dual graph. For simplicity, we also refer to the cycles formed by the mesh edges corresponding to c as **cut**. Recall that the tori decomposition that we seek is composed of $g-1$ splitting cycles. Since the multi-way min-cut problem is NP-hard, we iteratively find $g-1$ splitting cycles one at a time. So in this subsection, we study a simpler problem: finding a cut (1) that splits a genus g surface M into two pieces: one with genus 1, the other with genus $g-1$ (2) in which all the cycles in the cut are splitting cycles. In a surface-embedded graph such as a manifold triangle mesh, if we choose a set of vertices S as the source, and another set of vertices T as the target, min-cut would give a region P_s belonging to S, a region P_t belonging to T, and a set of cycles C which is the boundary between P_s and P_t. The result may fall into three types [22]:

Case 1. The boundary is a splitting cycle, as shown in Fig. 4.5(a);

Case 2. The boundary is a contractible cycle, as shown in Fig. 4.5(b);

Case 3. The boundary is composed of non-contractible cycles, as shown in Fig. 4.5(c).

For our purpose of partitioning the mesh, case 1 is always good, case 2 is always bad, and case 3 is good only when its cycles are not only non-contractible but also splitting. We design our algorithm to avoid bad cases:

Avoiding case 2. If a separating cycle is contractible, then one of the two regions of the mesh separated by the cycle is a topological disk. Therefore, to avoid that situation, we always use tunnel cycles as sources and targets, which ensures that each separated part is nontrivial.

Lemma 1. Let $t_1, ..., t_g$ be the tunnel cycles of a 2-manifold, where the set of vertices in

t_i is the source and the set of vertices in the rest of the tunnels $t_1, ..., t_{i-1}, t_{i+1}, ..., t_g$ is the target, for the min-cut problem. If the min-cut has only one cycle component c, then c is a splitting cycle and both sides of c are topologically non-trivial.

Proof: We can prove it by contradiction. Assume that P_s is topologically trivial, which is to say P_s is a topological disk, then every cycle inside P_s is contractible, which contradicts with the fact that $t_i \in P_s$ is a non-contractible cycle. Similarly, the assumption that P_t is topologically trivial contracts with the fact that $t_1, ..., t_{i-1}, t_{i+1}, ..., t_g \in P_t$ are non-contractible cycles. Therefore, both P_s and P_t have to be topologically non-trivial, hence c is also non-contractible. Further, since c is corresponding to a min-cut, so it is a separating cycle, and therefore is a splitting cycle.

Avoiding non-separating cycles in case 3. For a tori decomposition, we seek splitting cycles, which are non-contractible and separating. When the cut is composed of multiple component cycles, min-cut guarantees that each of its component cycles is non-contractible. Otherwise, the cut will not be minimum[22]. Therefore, to get splitting cycles, we just need to ensure the cycles are separating. If there are non-separating cycles, we enhance the set of source or target cycles with vertices in the non-separating min-cut cycles, and re-run the min-cut algorithm, as described in Algorithm 4.1. The intuition behind the idea is that the existence of the non-separating cycles indicates that either P_s or P_t needs to grow further. When P_s has genus 0, as shown in the top of Fig. 4.6(1), we add cycle c into S. When P_s has genus larger than 1, as shown in the bottom of Fig. 4.6(1), we add c into T. When P_s has genus 1, P_s is a torus which is desired and we do nothing to the cycles. As shown in Fig. 4.6(4), this operation converts cuts with non-separating cycles into cuts composed of only separating cycles. To determine if the cycle is separating we simply check how many connected components are produced when the cycle is removed from the surface, and the genus can be calculated using the Euler characteristic. In each iteration, if there exist non-separating cycles, at least one vertex is added to either set S or T, so the algorithm

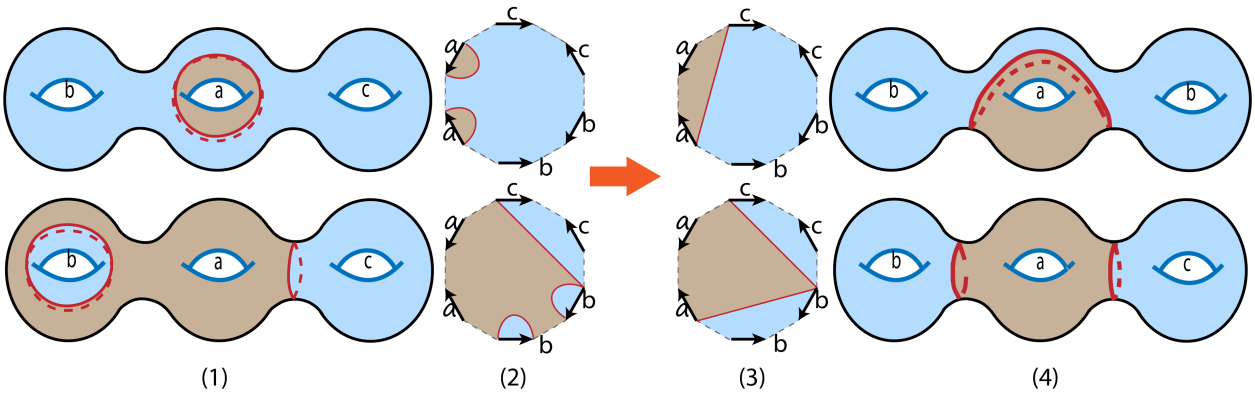

Figure 4.6: Assume that the tunnel cycle a is the source and the tunnel cycles b and c are targets. (1) shows two undesirable results where some of the cut cycles are not splitting cycles, (4) shows two desirable results where the cut cycles are all splitting cycles. (2) and (3) show the fundamental polygonal (12-sided polygon) representation of the four respective cases shown. Pairs of polygonal edges as labeled represent (the curves homotopic to) the tunnel cycles (and the unlabeled pairs of dashed edges represent the handle cycles). Consider the red curves in (2). If exactly one curve around either of the tunnels a or b is cut, there is a copy of the same tunnel appearing in both pieces of the cut polygon. In other words, the original model in (1) is not split when cut along that curve, and hence these individual curves do not represent splitting cycles. However, in (3), if the polygon is cut along the red line to the polygon is split, there is no tunnel appearing on both pieces of the polygon, which shows that the original model in (4) is also split. (3) visually shows the existence of splitting cycles between one tunnel and the rest of the tunnels. For cases in (2), the region around a (top figure) is grown by making the curves around a as the source, or the region around b (bottom figure) is grown by making the curves around tunnel b as the target, to achieve one of the two cases in (3).

terminates in $O(n)$ iterations where n is the number of vertices in the surface. As the time complexity for min-cut is $O(n^2 \cdot log(n))$, the overall time complexity for finding one splitting cycle is $O(n^3 \cdot log(n))$. But since in practice the number of iterations is small, we may even consider it as a constant value, which leads to $O(n^2 \cdot log(n))$ overall time complexity. With a sufficiently dense triangulation (enough triangles/edges between every two tunnels), there are many separating cycle candidates for the algorithm to find a valid splitting cycle which is both separating and non-contractible. However, very sparse, hand-crafted triangulations, e.g. the one shown in Fig. 4.10(c), can be avoided by dynamic re-triangulation during run time.

4.3.3 Finding all splitting cycles

In the previous subsection, we discussed finding one cut that partitions the surface into a punctured torus and a part with genus $g-1$. To decompose the surface into multiple tori, we need to find $g-1$ such cuts. Fig. 4.7 shows this process. Each step is analogous to cutting a torus from the shape, which introduces a new boundary to the original mesh. To deal with the boundary in the follow-up iterations, for each such boundary, we add an extra virtual vertex v and connect v to all the vertices on this boundary. For all the new added edges from v, we assign edge weight 0, as we would like the cuts in later iterations to go through them freely. If we would like a result similar to [7], where the cuts are disjoint, we may just set the edge weights for them to a large value. As discussed in the previous subsection, while the worst case and amortized time complexity for finding one splitting cycle are $O(n^3 \cdot log(n))$ and $O(n^2 \cdot log(n))$, the worst case and amortized complexity for finding all the splitting cycles are $O(g \cdot n^3 \cdot log(n))$ and $O(g \cdot n^2 \cdot log(n))$.

Algorithm 4.1 Finding a cut composed of only splitting cycles that separates tunnel t_i from the other tunnels.

```
Initialize S as {t_i}, T as {t_1, ..., t_{i-1}, t_{i+1}, ..., t_g}
Initialize P_s and P_t as ∅ // The segmented regions belonging to S and T
Initialize C = ∅ // Set of boundary cycles between segmented regions
while C = ∅ or C contains non-separating cycles do
    {C, P_s, P_t} = min-cut(S, T)
    foreach cycle c in C do
        if c is non-separating then
            if genus of P_s ¡ 1 then
                add vertices in c to S
            else
                add vertices in c to T
            end
        end
    end
    if C is unchanged as last iteration then
        break
    end
end
```

4.3.4 Computation of edge weights

To make the splitting cycles generated by min-cut to be mesh geometry-aware, we take a few other geometric measures, excluding edge lengths, into account:

Symmetry. As shown in Fig. 4.8, if we consider only the edge lengths as edge weights for the algorithm, there might exist multiple optimal cuts, and the result may not be symmetric even on the symmetric surface. Therefore, we would like the cycle to lie almost equidistant between tunnels. In other words, we would like the splitting cycles to be as far as possible from the tunnels. We apply the geodesic heat method [12] to compute the distance field using all the tunnels as the source of the heat. We choose this method as (1) its amortized time complexity is roughly linear and (2) the distance computed is geodesic, thus oblivious to mesh triangulation. Other geodesic distance computation methods should work too.

Minima rule. Humans often perceive that the shapes are segmented along concave regions, which is known as *minima rule*[34]. Therefore, if possible, we would like the cut to go along the concave region of the mesh. For each edge, we determine its minima rule energy using the average of its two incident vertices' minimum curvature $\kappa(v)$, where $\kappa(v)$ is normalized among the whole mesh.

In summary, for edge $e = (v_i, v_j)$, its weight is defined as

$$w_e = l_e + \alpha(dist_i + dist_j) + \beta(\kappa(v_i) + \kappa(v_j)) \tag{4.1}$$

where l_e is the normalized length of e, $dist_i$ is v_i's value in the geodesic distance field computed with vertices on tunnels as source, α and β are predefined coefficients. Each of the terms can be computed in linear time, so amortized time complexity for edge weight computation is $O(n)$, where n is the number of vertices.

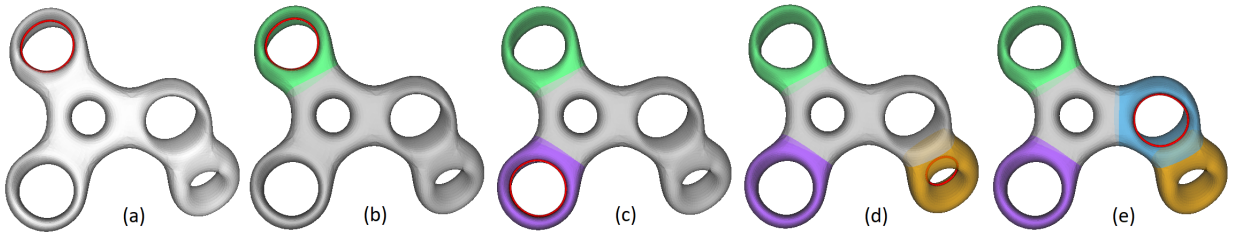

Figure 4.7: Steps to finalize each region. Each time a tunnel cycle is chosen as source, and the other tunnel cycles as target, the min-cut found in each step would split the surface further.

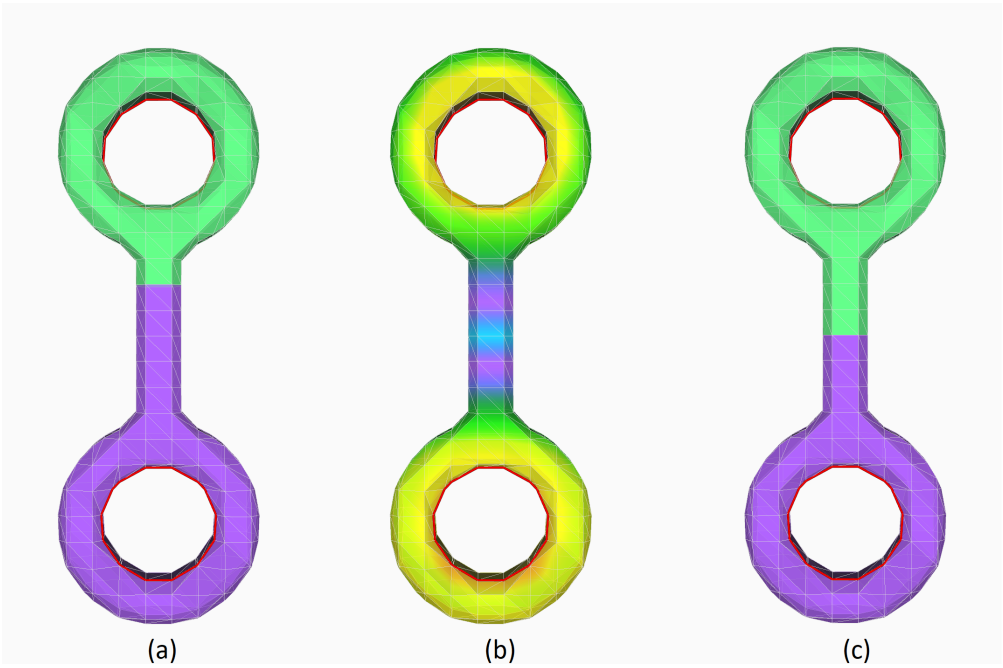

Figure 4.8: The role of distances to tunnels in determining optimal splitting cycle. (a) optimal splitting cycle found using edge lengths as edge weights. Note that all the cross sections of the central cylinder have the same diameters, so any one of them may be picked by the algorithm as the optimal splitting cycle. (b) distance map formed by using tunnels as a source. (c) optimal splitting cycle found using distance aware edge weights.

4.4 Results

We implemented our algorithm and tested on several models with a wide variety of geometric and topological complexity. Specifically, for max-flow/s-t cut, we adopt [4]'s implementation, which is not optimal in theoretical time complexity, but faster than most of the other implementations in practice. Fig. 4.11 shows our results on various meshes. (a)-(d) are models from public data sets, (e)-(g) are created to test our method on extreme topology: one of the tunnels in (e) has a knot, (f) has two tunnels interlocking with each other, and (g) is a high-genus object with genus 64. Our method is able to correctly segment the shape in all these cases.

Imperfect input. To demonstrate that our algorithm is robust to various defects in the input meshes, we consider two kinds of imperfect inputs: surface with boundaries (holes) and surface with noise. Fig. 4.9 shows our result for mesh with boundary, as the two key steps tree-cotree decomposition and min-cut both works with meshes with boundary, no special precautions are necessary to handle them in Fig. 4.9(b) we perturb the mesh vertices to introduce noise, and since none of the steps in our method require the mesh to be smooth, the result is oblivious to such kind of artifacts.

Results from different triangulations. Fig. 4.10 shows our results from different triangulations. Fig. 4.10(a) and (b) suggest that, since the splitting curves always follow mesh edges, the geometric quality and smoothness of the boundaries are affected by the coarseness of the triangulation. This can be improved by re-triangulating or up-sampling the surface. The boundary smoothing technique introduced in [52] can also be applied to post-process the boundaries. Fig. 4.10(c) shows that even when the triangulation is extremely sparse, the decomposition result of our approach is still a valid tori decomposition, and the topological correctness of our approach is not affected by the triangulation.

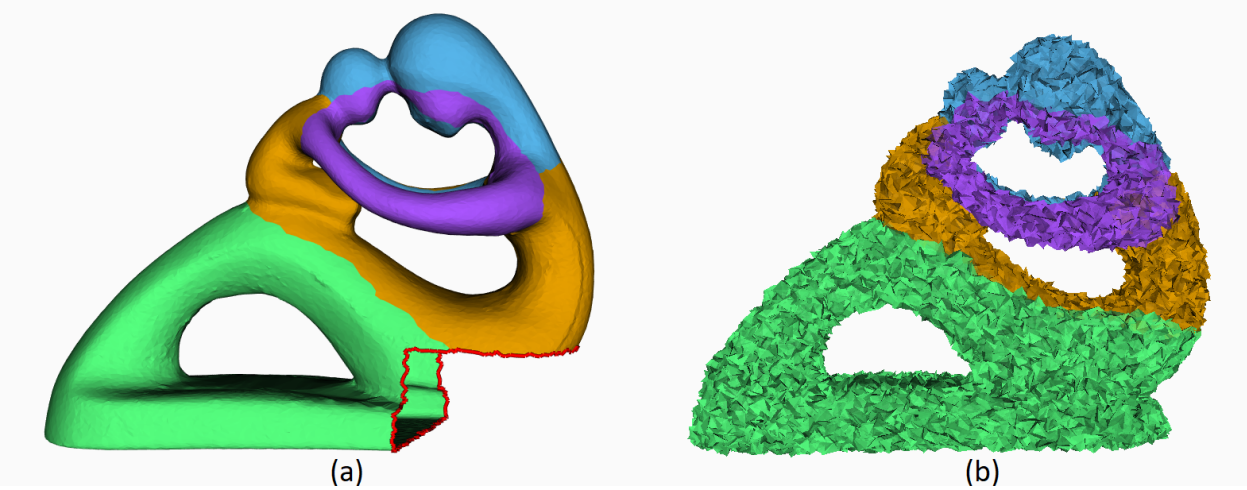

Figure 4.9: Imperfect input: (a) mesh with boundaries (boundary labelled in red) (b) mesh with vertex perturbation noise.

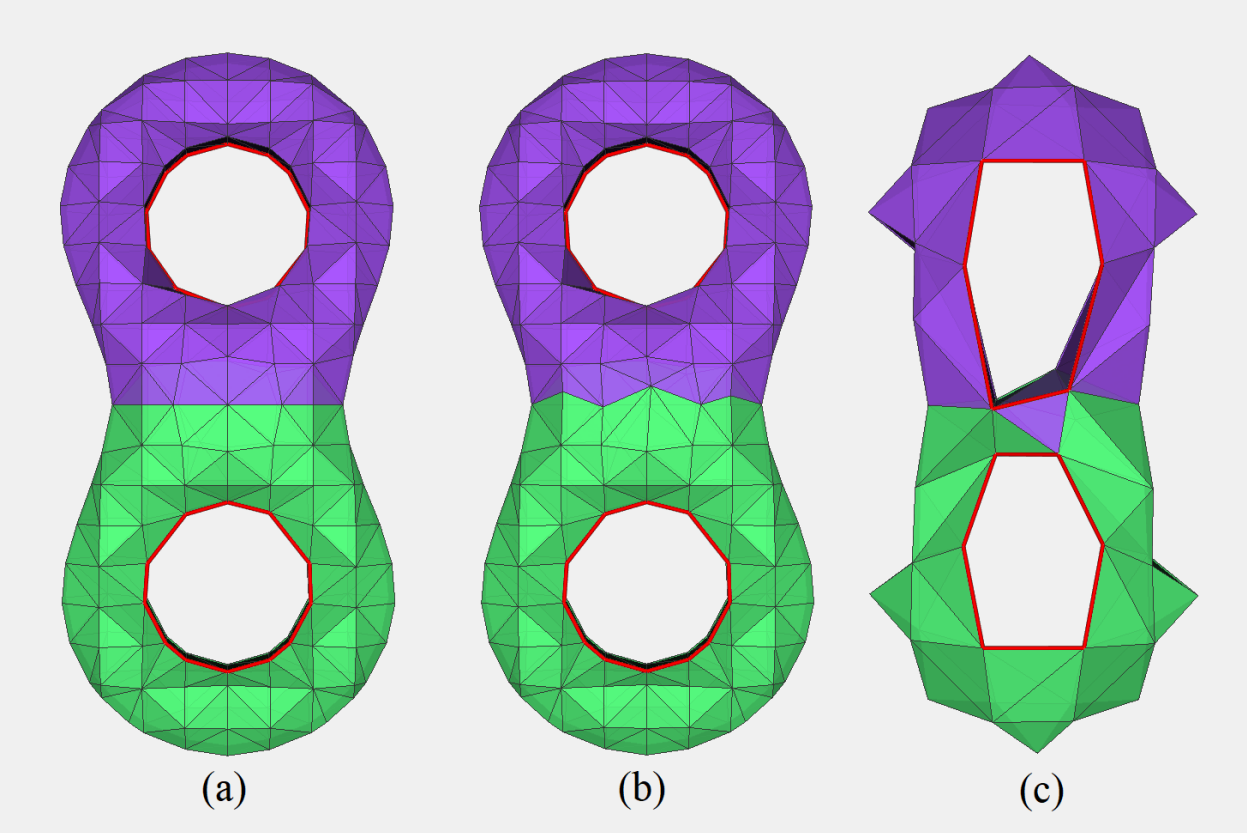

Figure 4.10: Tori decomposition results from different triangulations. The smoothness of the splitting cycles is affected by the triangulation (a)(b), but even when the triangulation is extremely sparse (c), the components found by our method are still topological tori.

Figure 4.11: Tori decomposition results of our method, with tunnel cycles highlighted in (b)-(f). One of the tunnels in (e) is knotted and (f) contains tunnels interlocking with each other. (g) has genus 64.

Performance. We test our algorithm on various mesh models. Table 4.1 shows the performance statistics of our approach, where t_{pre} is the time for pre-processing the mesh, including principal curvature calculation, distance field generation etc., t_{cycles} is the time for calculating the fundamental cycles, t_{decomp} is the time for computing the final tori decomposition, and t_{total} is the total time spent. We remark that for distance field generation, we follow [12] to prefactor a pair of sparse linear systems. The timing reported here does not include this pre-computation.

Model	#Tri	g	t_{pre}	t_{cycles}	t_{decomp}	t_{total}
Fig. 4.1	29,734	5	0.253	0.387	0.524	1.164
Fig. 4.11(a)	2,420	5	0.036	0.031	0.053	0.120
Fig. 4.11(b)	220,390	2	0.548	1.532	1.396	3.476
Fig. 4.11(c)	63,454	8	0.329	2.585	4.204	7.118
Fig. 4.11(d)	62,540	5	0.363	2.386	4.193	6.942
Fig. 4.11(e)	20,094	3	0.223	0.837	0.103	1.163
Fig. 4.11(f)	4,696	2	0.121	0.437	0.033	0.591
Fig. 4.11(g)	342,064	64	0.984	32.245	67.378	100.607
Fig. 4.9(a)	50,000	4	0.268	1.299	1.171	2.738
Fig. 4.9(b)	50,000	4	0.271	1.345	1.402	3.018

Table 4.1: Performance statistics (in seconds). t_{pre} is the time for pre-processing, t_{cycles} is the time for localizing fundamental cycles, t_{decomp} is the time to generate the final tori components, and t_{total} is the total time spent.

4.5 Discussion

In this chapter, we developed a tori decomposition framework to partition a manifold surface mesh into topologically non-trivial components, i.e. each of the components has genus-1. Our tori decomposition is based on first iteratively finding all the g tunnel and g handle cycles on the surface, and then rather than directly finding optimal splitting cycles on the surface, we formulate the problem as finding minimum cuts in the surface's dual graph. Unlike the planar graphs, min-cuts in a surface-embedded graph may not always produce a single splitting cycle, and hence we design our algorithm to avoid the undesired cases. This result

generates splitting cycles with small edge weights, and we assign the edge weights to reflect the mesh geometry. Experimental results suggest that our framework is efficient and robust on numerous examples.

We would like to investigate several improvements to our method. The geometric quality of the boundaries between the decomposed components depends on the quality of the triangulation of the input mesh. This can be improved by re-triangulating the surface or post-processing the boundaries. Another limitation of our method is that we always decompose the shape into g components, which may not be exactly consistent with how humans perceive the shape. It would be interesting to apply our method along with perception based mesh segmentation methods to determine the desired number of components.

Chapter 5

Contractible decomposition

In this chapter, we aim to decompose the surface shape into topological contractible components. This process reduces the number of tunnels, thus simplifies the topological complexity of the surface. Therefore, many applications in 3D printing[25] can benefit from such decomposition. Besides that, such decomposition is a necessary pre-processing step for isogeometry based mechanical analysis[42]. For example, there exist numerous constructions of volumetric spline models that represent contractible solids. Each of the components that our method generates can be dealt with by these existing methods.

Previous methods depend on a volumetric representation of the interior volume to segment the shape into contractible solids[42][48]. While the searching space is too big to search arbitrary cuts inside the volume, these methods search only planar cuts, and thus they can only deal with the straight tunnels, and the number of resulting pieces is often unnecessarily high. In this chapter, instead of directly working on the volume, we adopt an approach purely based on the surface. There exist numerous previous works that can segment a surface into topological disks, e.g., cut locus[47] and homotopy generators[24]. Although topological disks are contractible, they are not the desired result of this chapter. As shown in Fig. 5.1, the contractible components we seek are surface patches that when their boundaries are filled, the solid bounded inside the surface is contractible.

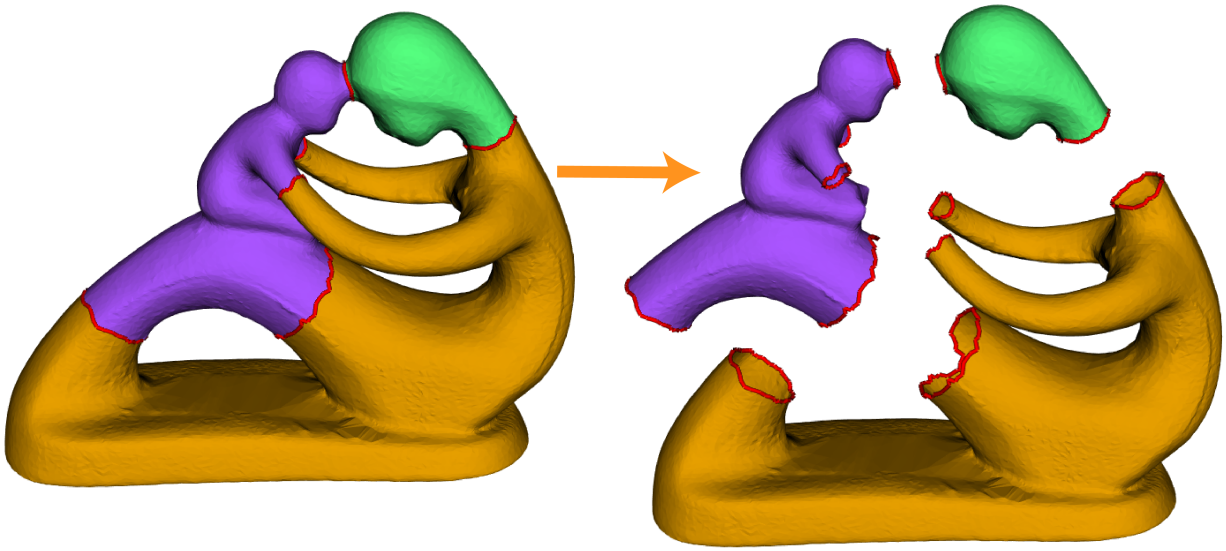

Figure 5.1: Contractible decomposition (a) generated by our method, each of the components (b) has genus-0, and the two sides of each cut cycle belong to different components.

In summary, our main contributions are:

- We formulate the problem of decomposing a volume as finding a combination of handle cycles and separating cycles on the volume's surface;
- Based on an initial set of fundamental cycles, we find a set of cycles which oversegment the shape into contractible solids;
- Given the desired number of components m, we propose a method based on dynamic programming to find a contractible decomposition composed of m components, and the resulting pieces are good in geometry.

5.1 Related work

The previous work most relevant to ours is [48] which automatically subdivides 3D solids into contractible pieces. Using a combination of volume and surface Reeb graphs as a

skeletal representation of the volume, [48] finds the location of cuts where the solid can be split. However, since linear Morse functions are used, their method works only for straight tunnels, and the number of resulting pieces may be unnecessarily high.

Many research efforts are devoted to the decomposition of solid objects or their surfaces into convex or nearly convex pieces. The typical methods for generating such decompositions are to repeatedly split the surface in concave surface areas[34] [37] or to grow certain surface regions with similar characteristics[41]. The convex pieces are contractible solids, but the number of pieces generated by such methods is always more than needed.

A series of geometry-based decomposition methods decompose the shape into predefined primitives. [58] forms over-complete covers of the input shape and decomposes a surface into generalized cylinders. The primitive shape used by these methods are mostly contractible shape, so in many cases, their segmentation results happen to be contractible decompositions. However, since essentially they do not take the topology into account, so the results are not guaranteed to be contractible decompositions.

5.2 Problem formulation

5.2.1 3-manifold topology

In this chapter, we consider the volume bounded by the surface, which is a 3-manifold. Thus, this problem is closely related to a few topics in 3-manifold topology, e.g., handlebodies and Heegaard splittings[30][32]. Here we introduce some of the theorems in 3-manifold topology that are related to our problem.

Definition 5.1. *Let B_1,..., B_n be a collection of closed 3-balls and let D_1, ..., D_m,D'_1,...,D'_m be a collection of pairwise disjoint disks in $\bigcup \partial B_i$. For each $i \leq m$, let $\phi_i : D_i \to D'_i$ be a homeomorphism. Let H be the result of gluing along ϕ_1, then gluing along ϕ_2, and so on.*

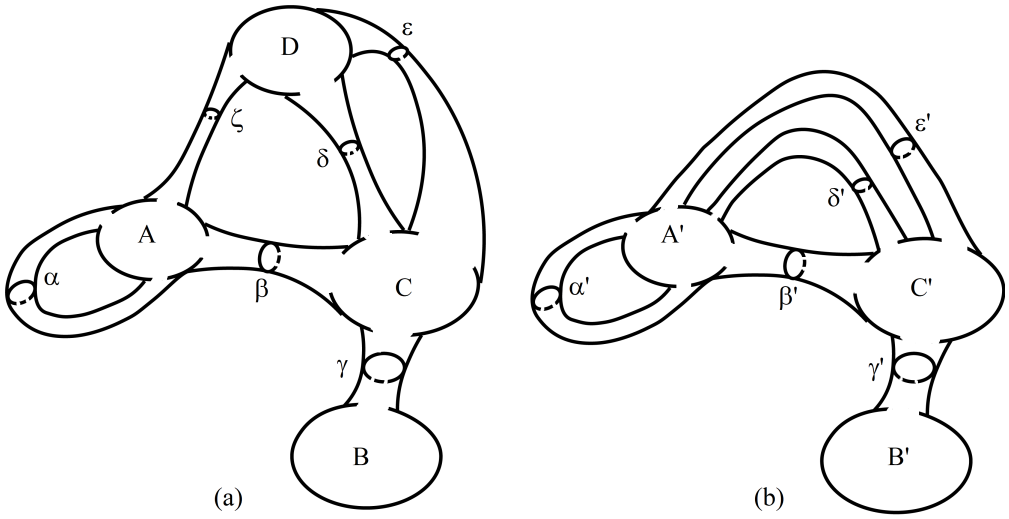

Figure 5.2: Handlebody is a collection of 3-balls, glued together along disks. (a) would be denoted $H^{4,6}$, and (b) would be denoted $H^{3,5}$. (a) is homotopic to (b). Note that the union of the disks forms a contractible decomposition.

After the final gluing, if H is connected then H is a handlebody.

In other words, a handlebody H is a 3-manifold with a connected boundary $\partial H^{n,m}$. $\partial H^{n,m}$ is a 2-manifold which we are familiar in Chapter 2, and we can compute the genus of $\partial H^{n,m}$ from n and m:

$$g = m + 1 - n \tag{5.1}$$

Note that the boundary ∂D_i of a disk D_i is the handle cycle since it is contractible in the interior space. The following theorem is the basis of proving a series of lemmas in this chapter; see [32] for a line of proof.

Theorem 5.1. *Two handlebodies are homeomorphic if and only if their boundaries have the same genus.*

Theorem 5.2. *A handlebody of the form $H^{n,m}$, where $n > 1$, is always homeomorphic to a handlebody of the form $H^{n-1,m-1}$.*

Based on these theorems and the relationship between 3-manifold and its boundary, we may

derive a few lemmas as follows.

Lemma 5.3. *Given an oriented connected 2-manifold surface M, we can always find a cut, composed of only handle cycles, that cuts the surface into pieces.*

Proof: Instead of the 2-surface, let us consider the interior space of M, which is a 3-manifold. A 3-manifold can always be represented as a handlebody, as shown in Fig. 5.2. The union of the topological disks splits the 3-manifold into a set of 3-balls, thus forms a contractible decomposition. The boundaries of the disks are either separating cycles or handle cycles. However, if we remove separating cycles from the union, for example, cycle γ in Fig. 5.2. Therefore, we can find a union which contains only handle cycles to form a contractible decomposition.

Lemma 5.4. *Given a 2-manifold surface of genus-g, we need at least $g+1$ cycles to form a contractible decomposition.*

Proof: Note that lower bound of the number of required cut cycles is the same as m of the corresponding handlebody. According to theorem 5.2, $g = m - n + 1$, thus $m = g + n - 1$ while for a contractible decomposition, the number of components is at least 2, which is, $n \geq 2$, and we have $m \geq g + 1$.

Lemma 5.5. *Given a surface of genus g, if the contractible decomposition contains X cycles, then the surface is decomposed into $X - g + 1$ pieces.*

Proof: Note that the number of cut cycles X is equal to m of the corresponding handlebody, and thus we can derive directly from theorem 5.2 that the number of components is $X-g+1$.

5.2.2 Contractible decomposition

In this chapter, we would like to seek a contractible decomposition in which the volume bounded by each segmented surface patch is contractible. The resulting decomposition

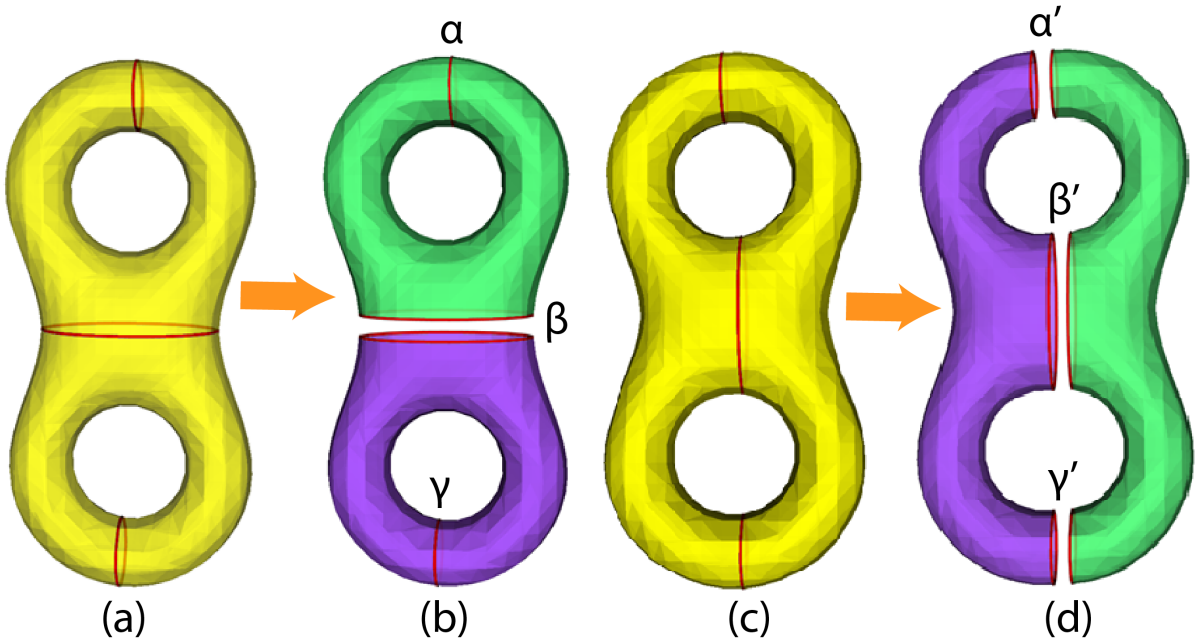

Figure 5.3: We seek decomposition in which each component has genus-0, and also the two sides of each cut cycle belong to different components. The cut cycles in (a) can segment the surface into pieces in (b), and if we consider only the surface, both pieces are contractible. However, note that cycles α and γ are not separable (cannot be separated physically), so this is not a valid contractible decomposition. Conversely, (c) is a valid contractible decomposition. The components it segments into, shown in (d), both have genus-0, and each cut cycle can be physically separated.

should be both topologically correct and geometrically proper. To guarantee topological correctness (every decomposed piece is a contractible solid), the decomposition must satisfy the following three conditions:

Condition 1. Each of the decomposed surface patches should bound a volume. To satisfy this condition, as discussed in the previous section, the cut surfaces of the 3-manifold volume must be topological disks, and the cut cycles of the corresponding 2-manifold surface must be either handle cycles or separating cycles.

Condition 2. Each of the bounded volumes should be contractible. This condition can be verified by checking if each of the decomposed surface patches has genus-0.

Condition 3. The bounded volumes can be separated apart. This condition may look trivial in the first glance, but consider Fig. 5.3(a) for example. The cycles in Fig. 5.3(a) split the surface into surface patches, and each surface patch has genus-0, thus is contractible. However, for cycle α, its two sides both belong to the same component, thus cycle *alpha* cannot separate the upper genus-1 component. In Fig. 5.3(a), both the two decomposed components have genus-1. Therefore, to guarantee that the components can be separated, we require that each tunnel is covered by at least two decomposed components.

In summary, we formulate the contractible decomposition problem as finding a set of simple cycles embedded on the surface. These cycles are either handle cycles or separating cycles, and when cut along, the surface is decomposed into genus-0 surface patches, and each tunnel should be covered by at least two decomposed components.

5.3 Geometry-aware contractible decomposition

In this chapter, as shown in Fig. 5.4, we propose a two-stage method to generate a number of cycles which segments the shape into contractible solids.

First, we find an independent set of fundamental cycles using the iterative tree-cotree algorithm introduced in chapter 3, and then based on these cycles, we find more handle cycles and separating cycles whose union segments the surface into contractible solids. We call this union of cycles an *oversegmentation* of the surface, as the number of components is larger than desired.

Second, based on the oversegmentation, we apply a dynamic programming method to form a contractible decomposition which satisfies (1) The number of components should be the number as specified by the user (2) The decomposition is topologically correct, or in other words, every decomposed component is contractible (3) The overall geometric quality of the

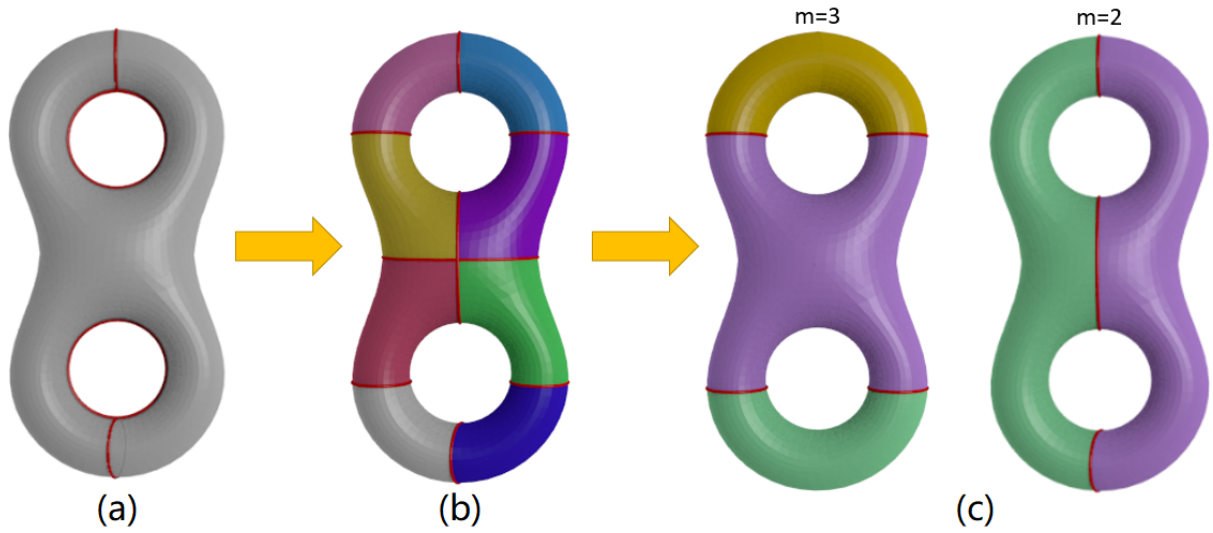

Figure 5.4: Major steps of finding a geometry-aware contractible decomposition. (a) We find an independent set of fundamental cycles using the iterative tree-cotree algorithm introduced in chapter 3 (b) we generate a set of cycles which, when cut along, oversegment the surface into contractible solids (c) given a user specified number of components m, we find a geometrically good decomposition with m components.

components should be good.

5.3.1 Generation of an oversegmentation

We follow four steps to generate an oversegmentation of the surface mesh as following.

(1) As shown in Fig. 5.5(b), we apply the iterative tree-cotree algorithm introduced in chapter 3 to generate an independent set of fundamental cycles. The handles will be in the set of final cut cycles. The tunnel cycles are not cut cycles but will be used to find more handle cycles and separating cycles.

(2) We obtain a tori decomposition following chapter 4 as shown in Fig. 5.5(c). If two tori components A and B are adjacent to each other and the tunnels in them are α and β respectively, we may find a shortest cycle which intersect with both α and β, similar to the method in [57], as follows: First, we cut the surface along α and β. After the cut, α becomes

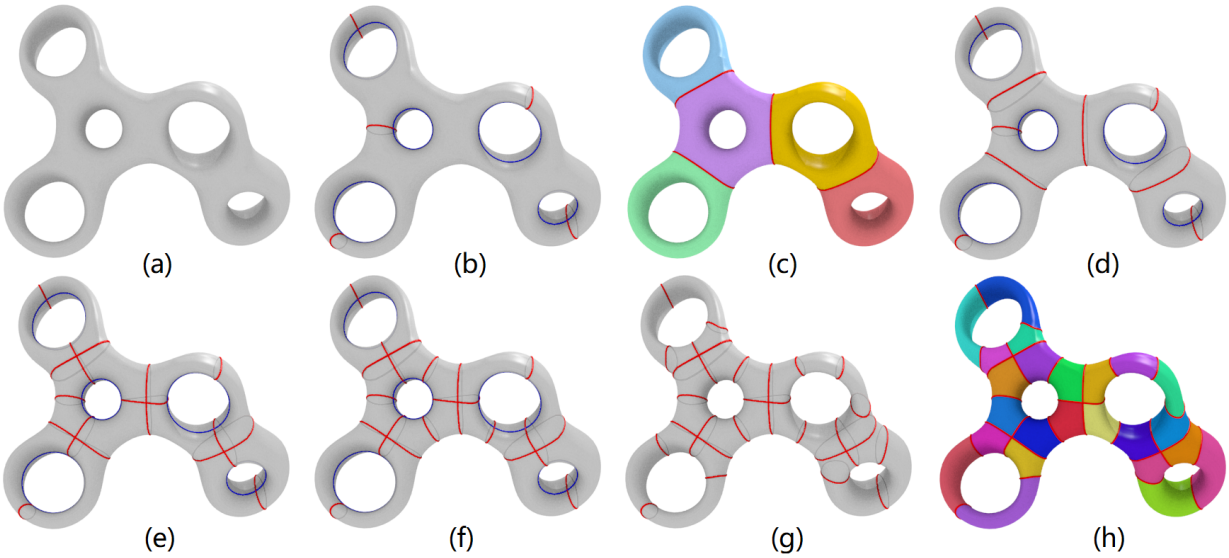

Figure 5.5: Generation of an oversegmentation of the surface mesh.

two boundaries b_1 and b_2 and a vertex α_i on α is split into vertex b_{1i} on b_1 and vertex b_{2i} on b_2. β becomes two boundaries b_3 and b_4, and a vertex β_j on β is split into a vertex b_{3j} on b_3 and a vertex b_{4j} on b_4. Second, for each i and j, we find the shortest path p_{ij} from b_{1i} to b_{3j}, and the shortest path p'_{ij} from b_{2i} to b_{4j}. Then the shortest cycle is the shortest union of p_{ij} and p'_{ij}.

(3) The cycles found in step (2) split each tunnel into several pieces, and we check each piece if the piece intersects with an existing handle. If not, we try to find a handle

(4) We make copies of type B handles, and move them as long as the size of the copied handle is not changed much (above a predefined threshold).

After the steps above, we get an oversegmentation, in which each piece is a contractible solid. The number of components in this oversegmentation is necessarily large, and we will merge the components to obtain a decomposition of the desired number of components.

5.3.2 Geometric quality of a decomposition

To find a decomposition in which the pieces are geometrically good, we need an evaluation function to measure the geometric quality of a decomposition. Intuitively, for the desired decomposition, (1) the components should have an overlapping region as small as possible, for example in Fig. 5.6, the decompositions (a) and (b) both have two components, but we prefer (a) as it has a smaller overlapping region (2) the narrow shapes are not desired. for example in Fig. 5.6, we prefer (c) as (d)'s components are relatively narrower than (c)'s components. Based on these considerations, we formulate the quality function as:

$$L_Q(D) = \sum_{p \in D} V(p) + \alpha \sum_{p \in D} \sigma^2(p) \tag{5.2}$$

For a decomposition D, we calculate the bounding volume V for each component p, and the variance σ^2 of p's principal axes. The cost of quality L_Q is defined as the sum of bounding volumes and the sum of variance, and for a high-quality contractible decomposition, we would like L_Q to be small.

5.3.3 Finding an optimal decomposition

Based on the oversegmentation, we apply a dynamic programming based method to find an optimal decomposition. As shown in Fig. 5.7, starting from any cell, we label all the cells in breadth-first search order. In each step, we classify one cell, and the cell may be either merged into an existing component or create a new component. We observe that for each step, only the quality of the components adjacent to the current cell can be updated, or in other words, we just need to update a small portion of the quality function.

We keep track of the topological properties: the number of components C, and the number of resolved tunnels T. A tunnel is considered as resolved if the tunnel is covered at least two

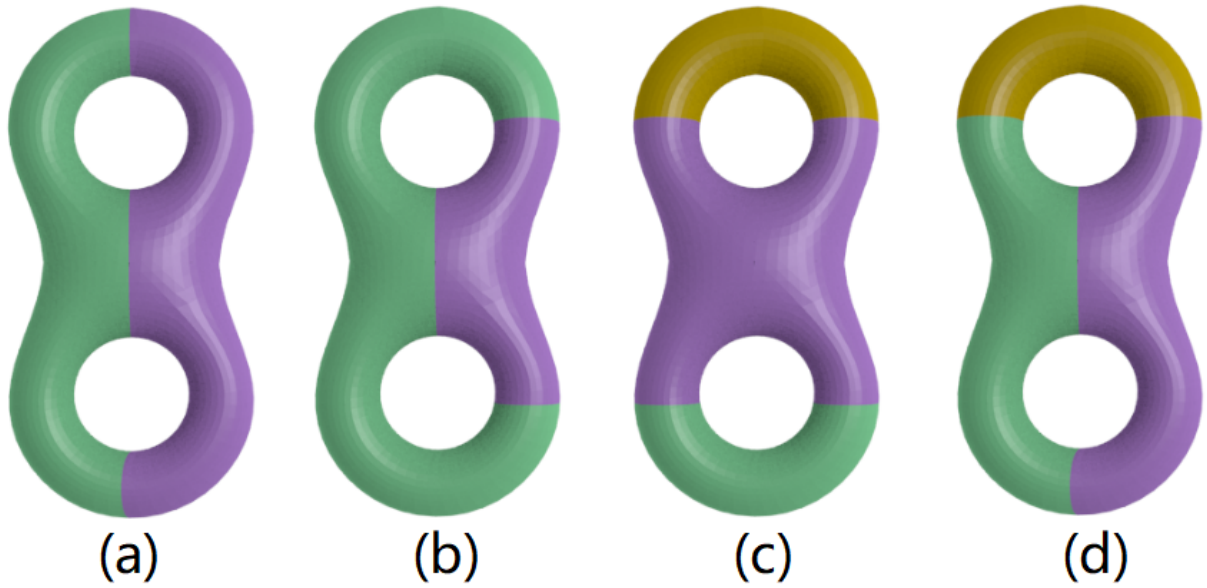

Figure 5.6: Geometric quality of contractible decompositions. Decompositions (a) and (b) both have two components, and our method prefers (a) as the total bounding volume of (a) is smaller than (b). Decompositions (c) and (d) both have three components, and our method prefers (c) as the pieces in (c) have smaller variance among their axes.

components. For a genus-g object with the number of components specified by the user as m, a valid final decomposition should have $C = m$ and $T = g$. For each (C, T) pair in each step, we keep the top-K scored results, where K is a user-specified constant. When K is infinitely large, the dynamic programming method would enumerate all the possible layout. However, the computational cost may become prohibitive if the object is topologically complex. To be able to handle these inputs, one could trade off optimality for efficiency by using a smaller K.

During the process of dynamic programming, we may filter out the invalid cases. For example, if the user requires the number of components is m, then we may eliminate all the cases where C is larger than m. Also, if a tunnel is entirely covered by one component, the tunnel will not be resolved in the future, so the decomposition is not valid, and we may eliminate it safely.

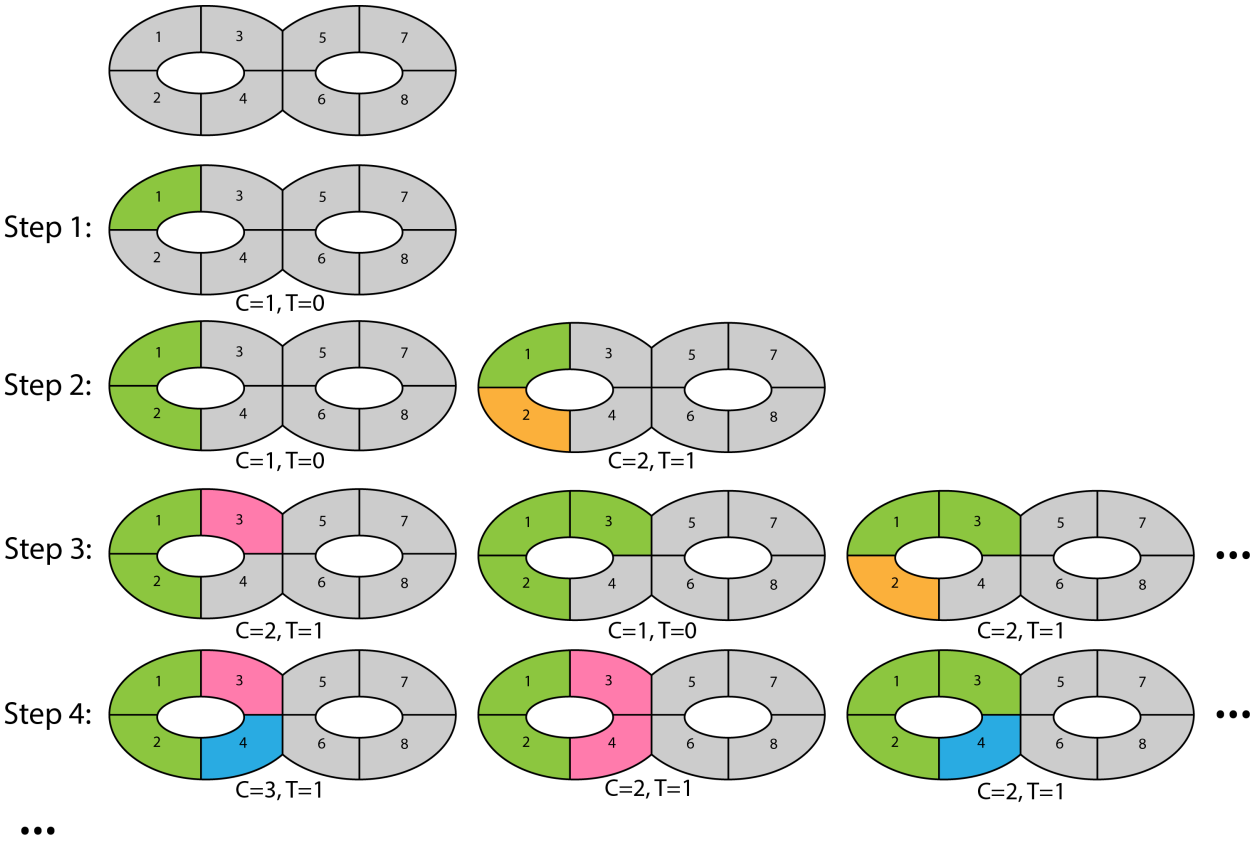

Figure 5.7: Dynamic programming steps for finding optimal contractible decomposition. We classify one cell at a time, and the cell may either be merged into an existing component or create a new component. For each layout in each step, we calculate its topological property represented by (components, resolved tunnels) pair. We keep top-K scored layouts in each topological type.

5.4 Results

We test the ability of our algorithm on a variety of examples. Fig. 5.8 shows our results on two models with a different prescribed number of components.

As shown in Fig. 5.9, we compare our method with shape diameter function [46] and approximate convex decomposition [27]. Both of the two methods are geometry-based methods. For the chair model in the first row, [46] is able to accurately locate the cylindrical structures in the shape, but one of its decomposed components has genus-1 and thus is not contractible,

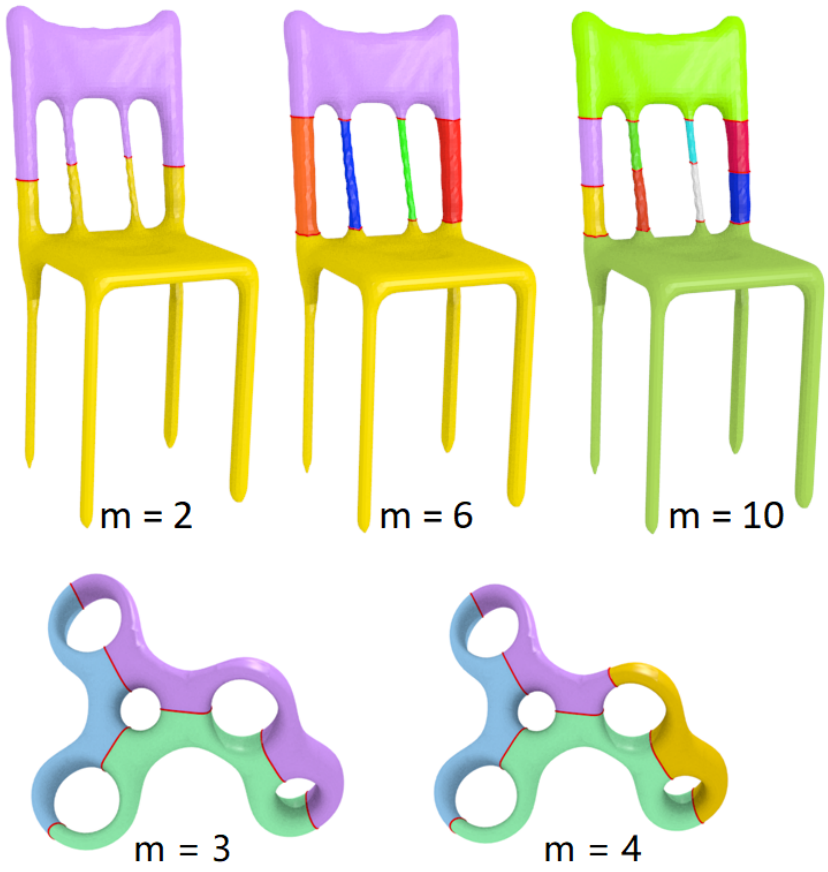

Figure 5.8: Results of our algorithm with various prescribed number of components.

[27] segments the shape into approximate convex components, but the number of components is necessarily large, and some of the geometrical details are lost in the process. Our method can segment the shape into pieces each of which is a contractible solid. For the stand model in the second row, [46] does not find the two holes in the center of the shape and consider the two sides as a separate part, [27] completely misses the two holes and converts the original genus-2 shape into a genus-0 object. Our method is able to locate both of the two tunnels and properly segment the shape into contractible solids.

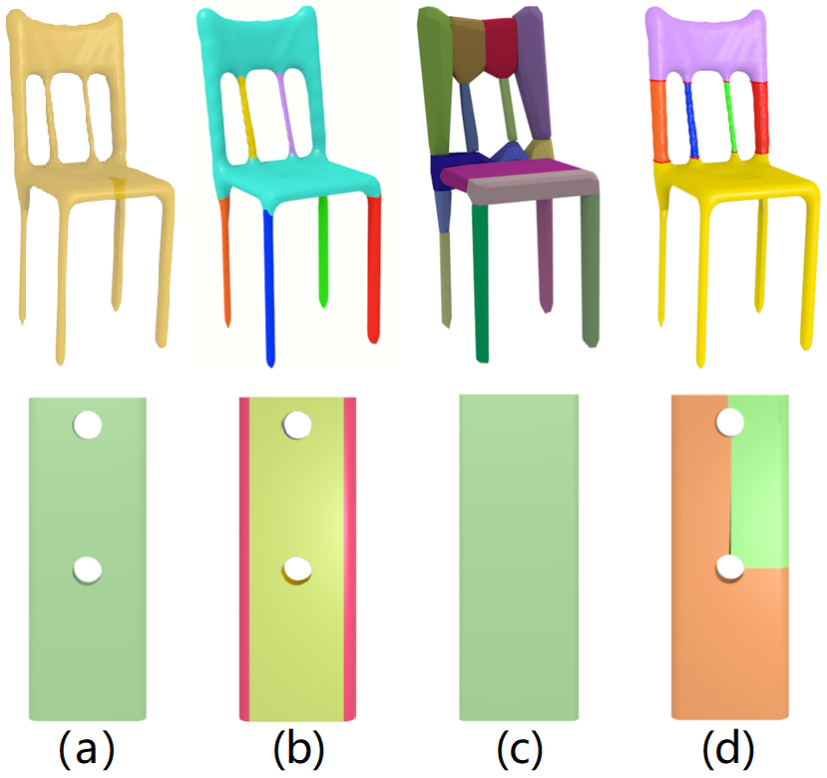

Figure 5.9: Comparison with previous methods. (a) input meshes (b) shape diameter function [46] (c) approximate convex decomposition [27] (d) ours.

5.5 Discussion and future work

In this chapter, we present an algorithm to decompose the inner volume of a surface shape into contractible solids. We analyze the topological relationship between the 2-manifold surface and the 3-manifold volume inside it and reveal that the problem of finding cuts of the volume is equal to finding handle cycles and separating cycles on the surface. Therefore, unlike previous methods which rely on the volumetric representation, our approach is completely on the surface. We solve the problem in two stages, first generating an oversegmentation of the surface by finding a series of handle cycles and separating cycles, then optimally merging the oversegmentation cells into the desired number of components.

Our approach has a few limitations. First, our algorithm is based on an oversegmentation

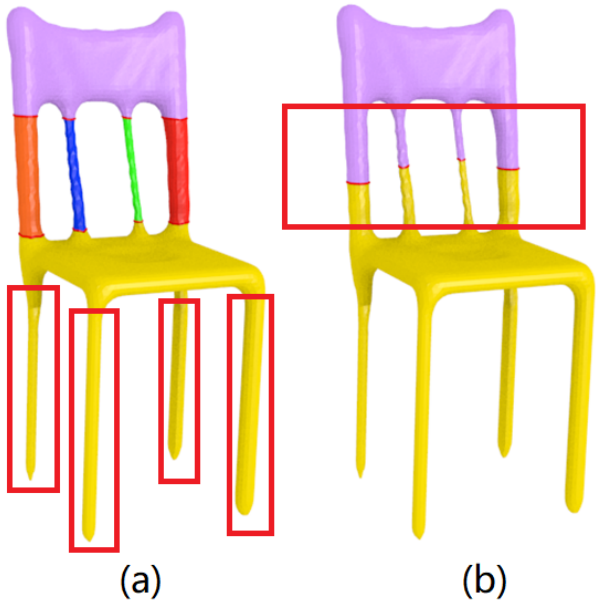

Figure 5.10: Limitations of our method. (a) Our method cannot segment the finger-like parts (b) The locations of the cut cycles may not be consistent.

formed by a series of handle cycles and separating cycles, which are both topologically and geometrically significant. However, our algorithm fails to segment the finger-like parts as shown in Fig. 5.10(a). The segmentation boundaries of the finger-like parts are contractible cycles, which are trivial in topological but may be important in geometry and human perception of the shape. As a future work, we would like to investigate integrating our method with geometry-based mesh segmentation methods to take these contractible cycles into account. Second, as shown in 5.10(b), the locations of the cut boundary may not be consistent, and the results can be improved by post-processing the cycles.

Chapter 6

Conclusion

In this dissertation, we presented a geometry-aware approach to finding two particular kinds of cycles which are of topological features: handle and tunnel cycles. These cycles are important as handle cycles capture the interior space of the surface while tunnel cycles capture the exterior space. Our approach formulated the problem as a surface-embedded graph problem and applied an iterative method to find a topologically independent set of handle and tunnel cycles, thus is more efficient than existing methods. We applied the computed cycles on two topological decompositions of surface shapes: tori decomposition and contractible decomposition. Tori decomposition splits a shape into genus-1 pieces, and contractible decomposition splits a shape into genus-0 solids. For tori decomposition, we formulated the problem as finding min-cuts in the dual graph and designed the edge weights such that the computed tori are good in geometry. For contractible decomposition, we first found a series of handle cycles and separating cycles to form an oversegmentation of the shape and then merged the oversegmented cells into a desired number of components in a dynamic programming manner. We presented results on a variety of models to show that our approach guarantees the topology of the resulting pieces but also is geometry-aware.

All the questions studied in this dissertation leave considerable room for future research. For handle and tunnel localization, edge weighing schemes other than the principal curvature directions may help get robust results. It would also be interesting to test our algorithm on

non-orientable surfaces.

For tori decomposition, we would like to investigate several improvements to our method. The geometric quality of the boundaries between the decomposed components depends on the quality of the triangulation of the input mesh. This can be improved by re-triangulating the surface or post-processing the boundaries. Another limitation of our method is that we always decompose the shape into g components, which may not be exactly consistent with how humans perceive the shape. It would be interesting to apply our method along with perception based mesh segmentation methods to determine the desired number of components.

For contractible decomposition, our method cannot locate and segment the finger-like parts, and this can be improved by integrating our approach with geometry-based methods. The locations of the found cut cycles can be improved by post-processing. Furthermore, the quality function we are currently does not involve enough human perception and machine learning based methods may work better for determining the quality.

To sum up, there are many avenues for future work regarding the problems addressed in this dissertation, and we hope that some of them will lead to exciting results.

Bibliography

[1] E. Akleman, V. Srinivasan, E. MANDAL, J. CHEN, Z. MELEK, and E. LANDRENEAU. Topmod: Interactive topological mesh modeler. 2008.

[2] M. Attene, B. Falcidieno, and M. Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer*, 22(3):181–193, 2006.

[3] O. K.-C. Au, Y. Zheng, M. Chen, P. Xu, and C.-L. Tai. Mesh segmentation with concavity-aware fields. *IEEE Transactions on Visualization and Computer Graphics*, 18(7):1125–1134, 2012.

[4] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1124–1137, 2004.

[5] G. Carlsson, A. Zomorodian, A. Collins, and L. J. Guibas. Persistence barcodes for shapes. *International Journal of Shape Modeling*, 11(02):149–187, 2005.

[6] A. J. Casson and C. M. Gordon. Reducing heegaard splittings. *Topology and its Applications*, 27(3):275–283, 1987.

[7] E. W. Chambers. Computing interesting topological features. Technical report, 2009.

[8] J. Chen, J. Jester, and M. Gopi. Fast computation of tunnels in corneal collagen structure. In *Proceedings of Computer Graphics International 2018*, CGI 2018, pages 57–65, New York, NY, USA, 2018. ACM.

[9] X. Chen, B. Zhou, F. Lu, L. Wang, L. Bi, and P. Tan. Garment modeling with a depth camera. *ACM Transactions on Graphics (TOG)*, 34(6):203, 2015.

[10] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. In *ACM Transactions on Graphics (ToG)*, volume 23, pages 905–914. ACM, 2004.

[11] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.

[12] K. Crane, C. Weischedel, and M. Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics (TOG)*, 32(5):152, 2013.

[13] T. K. Dey, F. Fan, and Y. Wang. An efficient computation of handle and tunnel loops via reeb graphs. *ACM Transactions on Graphics (TOG)*, 32(4):32, 2013.

[14] T. K. Dey, K. Li, J. Sun, and D. Cohen-Steiner. Computing geometry-aware handle and tunnel loops in 3d models. *ACM Transactions on Graphics (TOG)*, 27(3):45, 2008.

[15] T. K. Dey, K. Li, J. Sun, and D. Cohen-Steiner. Computing geometry-aware handle and tunnel loops in 3d models. In *ACM Transactions on Graphics (TOG)*, volume 27, page 45. ACM, 2008.

[16] T. K. Dey, J. Sun, and Y. Wang. Approximating cycles in a shortest basis of the first homology group from point data. *Inverse Problems*, 27(12):124004, 2011.

[17] P. Diaz-Gutierrez, D. Eppstein, and M. Gopi. Curvature aware fundamental cycles. In *Computer Graphics Forum*, volume 28, pages 2015–2024. Wiley Online Library, 2009.

[18] P. Diaz-Gutierrez, D. Eppstein, and M. Gopi. Curvature aware fundamental cycles. In *Computer Graphics Forum*, volume 28, pages 2015–2024. Wiley Online Library, 2009.

[19] A. Dold. *Lectures on algebraic topology*, volume 200. Springer Science & Business Media, 2012.

[20] J. El-Sana and A. Varshney. Controlled simplification of genus for polygonal models. In *Visualization'97., Proceedings*, pages 403–410. IEEE, 1997.

[21] D. Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 599–608. Society for Industrial and Applied Mathematics, 2003.

[22] J. Erickson, K. Fox, and A. Nayyeri. Global minimum cuts in surface embedded graphs. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1309–1318. Society for Industrial and Applied Mathematics, 2012.

[23] J. Erickson and S. Har-Peled. Optimally cutting a surface into a disk. *Discrete & Computational Geometry*, 31(1):37–59, 2004.

[24] J. Erickson and K. Whittlesey. Greedy optimal homotopy and homology generators. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1038–1046. Society for Industrial and Applied Mathematics, 2005.

[25] F. A. Fanni, G. Cherchi, A. Muntoni, A. Tola, and R. Scateni. Fabrication oriented shape decomposition using polycube mapping. *Computers & Graphics*, 77:183–193, 2018.

[26] X. Feng and Y. Tong. Choking loops on surfaces. *IEEE transactions on visualization and computer graphics*, 19(8):1298–1306, 2013.

[27] M. Ghosh, N. M. Amato, Y. Lu, and J.-M. Lien. Fast approximate convex decomposition using relative concavity. *Computer-Aided Design*, 45(2):494–504, 2013.

[28] A. Golovinskiy and T. Funkhouser. Randomized cuts for 3d mesh analysis. In *ACM transactions on graphics (TOG)*, volume 27, page 145. ACM, 2008.

[29] M. Hajij, T. Dey, and X. Li. Segmenting a surface mesh into pants using morse theory. *Graphical Models*, 88:12–21, 2016.

[30] A. Hatcher. Algebraic topology. 2002. *Cambridge UP, Cambridge*, 606(9), 2002.

[31] A. Hatcher, P. Lochak, and L. Schneps. On the teichmuller tower of mapping class groups. *Journal fur die Reine und Angewandte Mathematik*, pages 1–24, 2000.

[32] J. Johnson. Notes on heegaard splittings. *Cited in*, page 2, 2006.

[33] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 954–961, New York, NY, USA, 2003. ACM.

[34] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H.-P. Seidel. Mesh scissoring with minima rule and part salience. *Computer Aided Geometric Design*, 22(5):444–465, 2005.

[35] X. Li, Y. Bao, X. Guo, M. Jin, X. Gu, and H. Qin. Globally optimal surface mapping for surfaces with arbitrary topology. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):805–819, 2008.

[36] X. Li, X. Gu, and H. Qin. Surface mapping using consistent pants decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):558–571, 2009.

[37] J.-M. Lien and N. M. Amato. Approximate convex decomposition of polyhedra and its applications. *Computer Aided Geometric Design*, 25(7):503–522, 2008.

[38] M. Livesu, M. Attene, G. Patané, and M. Spagnuolo. Explicit cylindrical maps for general tubular shapes. *Computer-Aided Design*, 90:27–36, 2017.

[39] M. Livesu, N. Vining, A. Sheffer, J. Gregson, and R. Scateni. Polycut: monotone graph-cuts for polycube base-complex construction. *ACM Transactions on Graphics (TOG)*, 32(6):171, 2013.

[40] J. Lv, X. Chen, J. Huang, and H. Bao. Semi-supervised mesh segmentation and labeling. In *Computer Graphics Forum*, volume 31, pages 2241–2248. Wiley Online Library, 2012.

[41] L. Papaleo and L. De Floriani. Semantic-based segmentation and annotation of 3d models. In *International Conference on Image Analysis and Processing*, pages 103–112. Springer, 2009.

[42] M. Pauley, D.-M. Nguyen, D. Mayer, J. Špeh, O. Weeger, and B. Jüttler. The isogeometric segmentation pipeline. In *Isogeometric analysis and applications 2014*, pages 51–72. Springer, 2015.

[43] R. S. Rodrigues, J. F. Morgado, and A. J. Gomes. Part-based mesh segmentation: A survey. In *Computer Graphics Forum*, volume 37, pages 235–274. Wiley Online Library, 2018.

[44] C. H. Séquin. Topological tori as abstract art. *Journal of Mathematics and the Arts*, 6(4):191–209, 2012.

[45] A. Shamir. A survey on mesh segmentation techniques. In *Computer graphics forum*, volume 27, pages 1539–1556. Wiley Online Library, 2008.

[46] L. Shapira, A. Shamir, and D. Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer*, 24(4):249, 2008.

[47] R. Sinclair and M. Tanaka. Loki: Software for computing cut loci. *Experimental Mathematics*, 11(1):1–25, 2002.

[48] B. Strodthoff and B. Jüttler. Automatic decomposition of 3d solids into contractible pieces using reeb graphs. *Computer-Aided Design*, 90:157–167, 2017.

[49] J. Tierny, J.-P. Vandeborre, and M. Daoudi. Topology driven 3d mesh hierarchical segmentation. In *null*, pages 215–220. IEEE, 2007.

[50] N. R. Voss and M. Gerstein. 3v: cavity, channel and cleft volume calculator and extractor. *Nucleic acids research*, 38(suppl_2):W555–W562, 2010.

[51] E. Yaffe, D. Fishelovitch, H. J. Wolfson, D. Halperin, and R. Nussinov. Molaxis: efficient and accurate identification of channels in macromolecules. *Proteins: Structure, Function, and Bioinformatics*, 73(1):72–86, 2008.

[52] Y. Yang, W. Xu, X. Guo, K. Zhou, and B. Guo. Boundary-aware multidomain subspace deformation. *IEEE transactions on visualization and computer graphics*, 19(10):1633–1645, 2013.

[53] L. Yao. Making a torus. `http://fab.cba.mit.edu/classes/863.11/people/lining.yao/design%20Projects/project1.html`, 2018. [Online; accessed 29-Jan-2018].

[54] W. Yu and X. Li. Computing 3d shape guarding and star decomposition. In *Computer Graphics Forum*, volume 30, pages 2087–2096. Wiley Online Library, 2011.

[55] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Transactions on Graphics (TOG)*, 23(3):644–651, 2004.

[56] W. Zeng, X. Li, S.-T. Yau, X. Gu, et al. Conformal spherical parametrization for high genus surfaces. *Communications in Information & Systems*, 7(3):273–286, 2007.

[57] K. Zhang and X. Li. Searching geometry-aware pants decomposition in different isotopy classes. *Geometry, Imaging and Computing*, 1(3):367–393, 2014.

[58] Y. Zhou, K. Yin, H. Huang, H. Zhang, M. Gong, and D. Cohen-Or. Generalized cylinder decomposition. *ACM Trans. Graph.*, 34(6):171–1, 2015.