DOI: 10.1111/cgf.13641
EUROGRAPHICS 2019 / P. Alliez and F. Pellacini
(Guest Editors)
Volume 38 (2019), Number 2

Geometry Aware Tori Decomposition

Jia Chen[1] and M. Gopi[1]

[1]University of California, Irvine

Abstract

This work presents a shape decomposition algorithm to partition a complex high genus surface into simple primitives, each of which is a torus. First, using a novel iterative algorithm, handle and tunnel fundamental cycles on the surface are progressively localized. Then, the problem of computing the splitting cycles that produce such a tori decomposition 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.

1. Introduction

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 [YXG*13], geometric modeling [CZL*15], and shape editing [YZX*04].

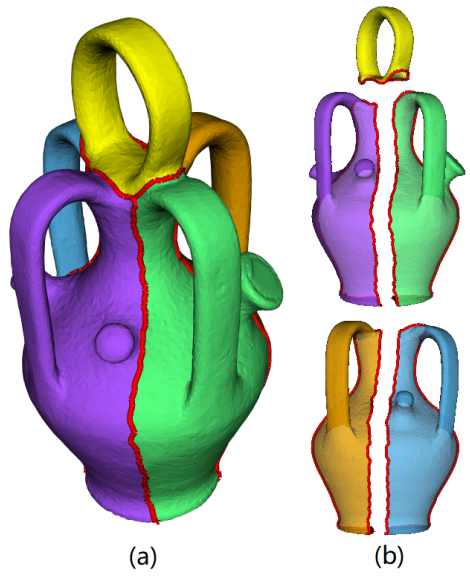

Figure 1: *Tori decomposition (a) generated by our method, each of the decomposed components (b) has genus* 1.

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 [LVS*13], cylinder decomposition [ZYH*15], 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 [EHP04] [DLSCS08] [DFW13], topological pants [LGQ09], or stars [YL11], 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 paper, 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 [Séq12]. 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

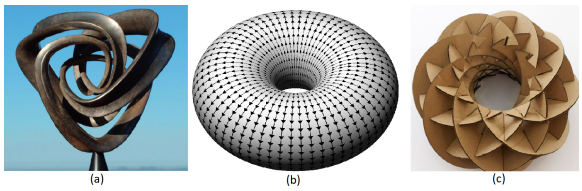

Figure 2: *Applications of topological tori. (a) Topological tori have been extensively studied from a theoretical perspective for abstract art [Séq12], (b) On a torus there exist vector fields without singular points. As in the case of art, multi-genus objects can be decomposed into multiple tori and the vector field can be designed for individual tori and merged. (c) There exist techniques which are especially suitable for making a torus [Yao18]. Tori decomposition makes it possible to apply these techniques to make component tori, which then can be assembled into more complex shapes.*

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 [Cha09], 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 propose a fast iterative method for handle and tunnel cycle localization and use it for tori decomposition
- We formulate the geometry-aware tori decomposition problem and show results in various high genus models

We begin with a brief review of the relevant literature in Section 2, followed by the introduction and analysis of tori decomposition problem in Section 3. In Section 4, we present an iterative method to localize the handle and tunnel cycles, and in Section 5, we describe our geometry-aware tori decomposition algorithm. Specifically, we use the handle-tunnel computation and formulate the splitting cycle generation problem as a min-cut problem in the dual graphs. Our algorithm guarantees that the computed cuts are geometry-aware and split the model into topologically nontrivial components. Finally, we discuss relevant implementation details and show the results of our implementation.

2. Related work

Shape decomposition is a well-studied problem with extensive surveys, e.g. [Sha08]. 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.

2.1. 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 [AFS06] or shape approximation [CSAD04], 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* [LLS*05] 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. [LLS*05] applies salience to find candidate contours. [AZC*12] 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 [LCHB12]. More complete surveys on mesh segmentation techniques can be found in [Sha08] and [RMG18]. However, none of these methods consider topology. A few works such as [TVD07] use the term "topology" to denote the hierarchical relationship between parts of the shape, which is not the focus of our work.

2.2. 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 [LGQ09] [ZL14], punctured tori decomposition [Cha09], etc. Li et al. [LGQ09] 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 [ZL14] traverse different classes of pants decompositions and search for the optimal one using a pre-defined geometric criterion that includes length, symmetry, and concaveness. [LAPS17] exhaustively partitions a solid into a set of tubular parts using a curve skeleton. The work that is most relevant to our

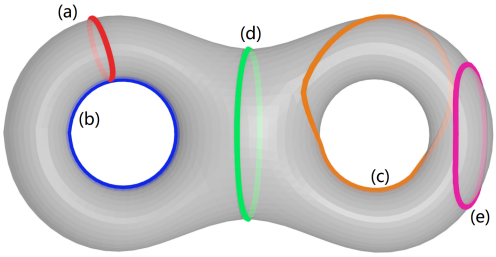

Figure 3: *Various classes of cycles on the surface: (a), (b) and (c) are all non-separating and non-contractible cycles, also called fundamental cycles. (a) is a handle cycle, (b) is a tunnel cycle while (c) is neither. (d) is a separating and non-contractible cycle, also called a splitting cycle, (e) is a separating and contractible cycle.*

problem is that of Chambers [Cha09] 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 [Cha09] 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.

2.3. Fundamental cycle localization

Our method of tori decomposition is based on computing fundamental cycles. [EW05] proposes a greedy algorithm to construct, in $O(n \cdot log(n))$ time, the shortest set of fundamental cycles which share a common point. Dey et al. [DLSCS08] study two special kinds of fundamental cycles *tunnels* and *handles*, by constructing curve skeletons for both interior and exterior spaces bounded by the original manifold mesh. The tunnels and handles are extremely useful for a lot of topology related problems. Our algorithm in this paper also requires finding tunnels and handles. The computation of the curve skeleton in [DLSCS08] requires tetrahedralizing the spaces, which is a challenging task in itself when the input mesh is large in size. So Dey et al. [DFW13] make use of the Reeb graphs, which can be computed efficiently, to avoid the tetrahedralization. By perturbing the cycles inside and outside, [DFW13] uses the winding number between the perturbed cycles to decide if a specific cycle is a tunnel or handle cycle. In this paper, we use the same method for classifying the fundamental cycles computed using our algorithm into tunnels and handles. Another set of works attempt to solve the problem in a surface's embedded graph and its dual graph. [Epp03] introduces tree-cotree decomposition, which efficiently finds $2g$ fundamental cycles. Diaz-Gutierrez et al. [DGEG09] 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. In this paper, we improve this method to guarantee that the computed fundamental cycles are independent.

3. Overview

An orientable surface is homeomorphic to a sphere where g disjoint disks are removed, and a torus with one boundary is attached to each of the g boundary circles. g is called the **genus** of the surface. The tori decomposition we propose in this paper is analogous to a reverse of this operation where we compute $g-1$ closed curves to cut off $g-1$ tori, and the base model, instead of a sphere, is also a torus. However, since our tori decomposition is sensitive to geometry, not all of the $g-1$ boundary curves may be on the base model, but may be distributed on other torus components as well.

Definitions: A **path** on surface M is a continuous map p: $[0,1] \rightarrow M$ with its two endpoints as $p(0)$ and $p(1)$. In a discrete setting, a **cycle** is a closed path without repeated vertices, except the endpoints. A cycle is **separating** if cutting the surface along the cycle gives rise to two connected components. Otherwise, it is **non-separating**. A cycle is **contractible** if it can be continuously contracted to a point, otherwise it is **non-contractible**. A contractible cycle is also a separating cycle. Simple, non-contractible, and non-separating cycles are called **fundamental cycles**. Simple, non-contractible, and separating cycles are called **splitting cycles**. Separating cycles divide the topology of the surface as well as the underlying graph. The goal of this paper is to decompose surface shape into topologically non-trivial components, so the splitting cycles are of special interest in our context. Fig. 3 shows examples of contractible, separating, and splitting 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 [DLSCS08]. 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 loops bound surfaces in I whereas tunnel loops bound them in O. For a formal treatment of the subject, readers are referred to [Hat02]. We use tunnel cycles to compute splitting cycles for tori decomposition.

There is a large body of works dedicated to finding short cycles with various topological properties on undirected surface graphs. For example, we can efficiently compute the shortest non-contractible and non-separating cycles and the shortest contractible cycles in polynomial time [EHP04] [CEN09] [EFN12]. In contrast, finding the shortest splitting cycle on a surface is known to be NP-hard [Cha09]. In this paper, we seek an approximate shortest splitting cycle.

3.1. 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 [CG87] decomposes a compact oriented 3-manifold that results from dividing it into

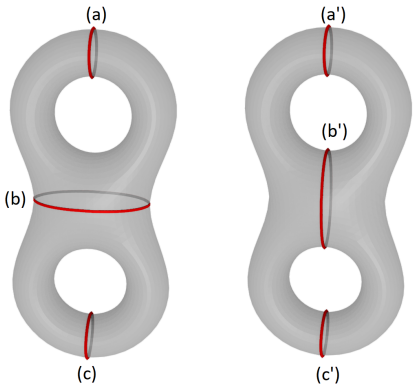

Figure 4: *Cut along the red cycles, both left and right figures result in valid pants decomposition where the surface is split into two pants components, each with three boundary cycles. The left decomposition can be converted to tori decomposition by repairing the cut (a) and (c), but for the decomposition on the right no such conversion exists.*

two handlebodies, but the result cannot be easily extended to 2-manifold surfaces. A definition of tori decomposition similar to ours is given in [Cha09], 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 paper, we remove this requirement.

3.2. Relationship between pants decomposition and torus decomposition

A **pants decomposition** partitions a manifold surface into components, each of which has genus 0 and three boundaries [ZL14] [LGQ09]. 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, this is not always true. Fig. 4 shows two different pants decompositions of a double-torus, out of which only one of them can be converted into tori decomposition. In fact, any topologically non-trivial shape with genus > 1 has multiple pants decompositions, but very few of them can be converted into a tori decomposition. Conversely, every tori decomposition can be further segmented into pants using the A-move and S-move introduced in [HLS00]. 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. Iterative localization of fundamental cycles

We compute the desired splitting cycles for tori decomposition as cycles that separate tunnel cycles. In other words, our approach computes geometry aware independent fundamental cycles, classifies them as handles and tunnels, and uses the tunnel cycles to find separating cycles between them for geometry aware tori decomposition.

Cycles with desired topological properties can be computed using Morse theory. But such methods rely on persistence filtration, which is computation-intensive and inefficient when the input model is large in size. A Reeb graph based method [DFW13] is more efficient than the other methods, as the Reeb graph captures the topological features of the surface mesh while greatly reduces complexity. However, the quality of its results depends on which Morse function is used to construct the Reeb graph. Compared to the aforementioned methods, the tree-cotree decomposition based method is efficient as it does not need to compute persistent homology, nor extra structures such as Reeb graph, curve skeleton or medial axis.

4.1. Tree-cotree algorithm

The tree-cotree algorithm for computing fundamental cycles on a mesh is based on computing spanning trees in two graphs. The **primal graph** takes the vertices in the original mesh as nodes in the graph and the mesh edges as graph edges. 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. 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. We first calculate a spanning tree in the primal graph, which we call a **tree**, and remove the dual graph edges that correspond to the mesh edges in the tree. On the resulting subgraph of the dual graph, we compute a spanning tree, which we call a **cotree**. A tree-cotree decomposition of a mesh partitions the set E of edges of a mesh into three sets $(T, C, E \setminus (T \cup C))$, where T is the set of edges in the *tree* and C is the set of mesh edges corresponding to the edges of *cotree* [Epp03]. The set $E \setminus (T \cup C)$ contains exactly $2g$ edges, and introducing these edges into *tree* creates $2g$ cycles, which are independent fundamental cycles of the mesh.

The above algorithm can be used to design fundamental cycles with certain properties by assigning weights to the mesh and dual graph edges based on geometric features and computing weighted spanning trees. [DGEG09] assigns mesh edge weights based on their alignment towards the minimum curvature direction, so that the fundamental cycles go along the minimum curvature direction. In a typical case, fundamental cycles will occur as g pairs of intersecting cycles. If g of them go in minimum curvature direction, the other g curves would prefer to go in the maximum curvature direction, but are forced to remain close to the minimum curvature direction because of the edge weight function. [DGEG09] proposes to run the algorithm twice with two different weighting functions, one preferring cycles along the minimum curvature direction, and the other preferring the maximum curvature direction. In each of the runs, the algorithm is guaranteed to compute $2g$ independent fundamental cycles. However, [DGEG09] picks g cycles from the first run and g from the second run, and thus combines them to get $2g$ cycles. However, the chosen cycles across different runs are not guaranteed to be independent. To resolve this issue, we propose a robust and theoretically correct method to compute $2g$ indepen-

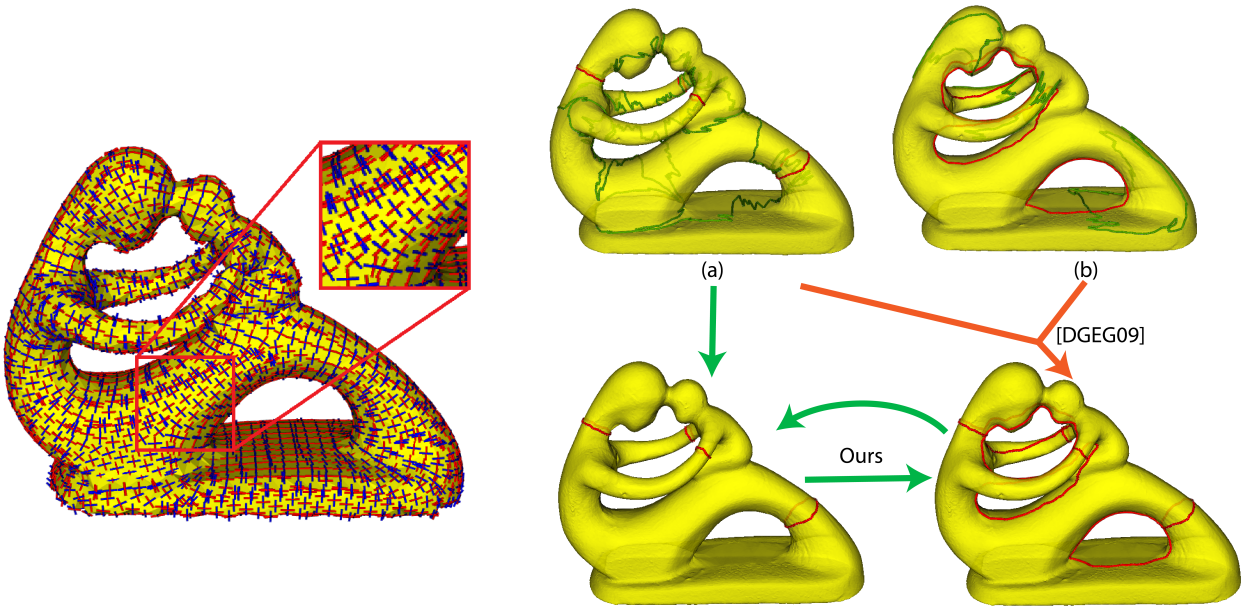

Figure 5: *Major steps for finding handle and tunnel loops. Left: Principal curvature directions. Red lines denote minimum curvature direction and the blue lines denote the maximum curvature direction. Right:(a) Tree-cotree algorithm results with maximum curvature direction based edge weights, with cycles of good quality highlighted in red color (b) Tree-cotree algorithm results with minimum curvature direction based edge weights. While [DGEG09] 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.*

dent fundamental cycles over multiple runs with different weighting functions.

4.2. Iterative method for multi-objective fundamental cycle computation

In this paper, we follow [DGEG09] to determine the edge weights using principal curvature directions. As shown in Fig. 5, 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 tunnels and handles respectively well.

In this paper, 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 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 [CJG18]. 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. 5 shows the major steps of our iterative method. Unlike [DGEG09], our algorithm never generates redundant cycles, so the cycles that are computed are guaranteed to be an independent set of cycles. The algorithm progressively finds better cycles and terminates if all the $2g$ found cycles are good according to [CJG18]'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. 14, 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 [DFW13].

```
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 ∈ c for both
              edge_weights_min and edge_weights_max;
        end
    end
end
```

Algorithm 1: Iterative method for localizing the g tunnel and g handle cycles.

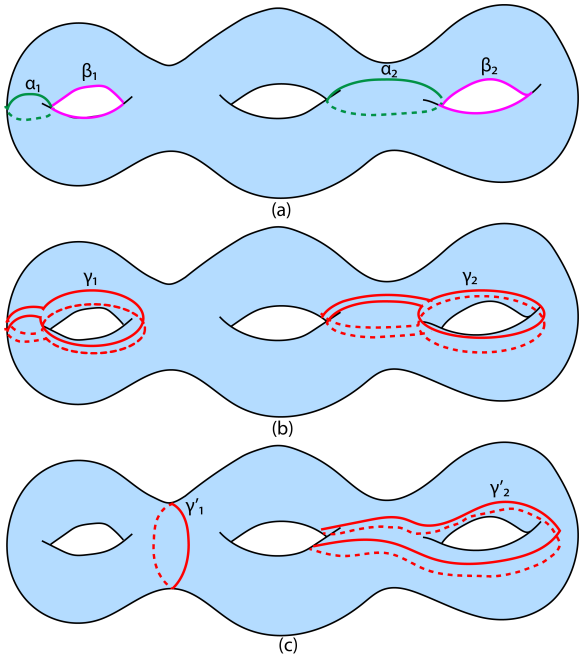

Figure 6: *A well-known technique to compose a splitting cycle from the fundamental cycles: (a) given a pair of intersecting fundamental cycles α and β, (b) the cycle $\gamma = \alpha \cdot \beta \cdot \bar{\alpha} \cdot \bar{\beta}$ is guaranteed to be a splitting cycle [Cha09]. Note that the resulting 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.*

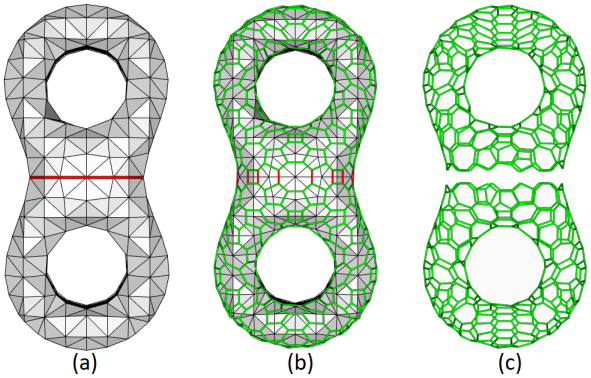

Figure 7: *A splitting cycle and its dual cut. (a) a surface and a splitting cycle which splits it into two genus-1 pieces (b) the dual graph with the edges dual to the splitting cycle colored in red (c) if the edges dual to the splitting cycle is removed from the dual graph, the graph is split into two parts. In other words, the splitting cycle is dual to a cut in the dual graph.*

5. 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, 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 [Cha09], where $\cdot$ denotes concatenation and $\bar{\alpha}$ denotes the reversed path of α. However, as shown in Fig. 6(b), the result of this operation is always unnecessarily long. Fig. 6(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 [Cha09] 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.

5.1. 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 [GF08]. 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

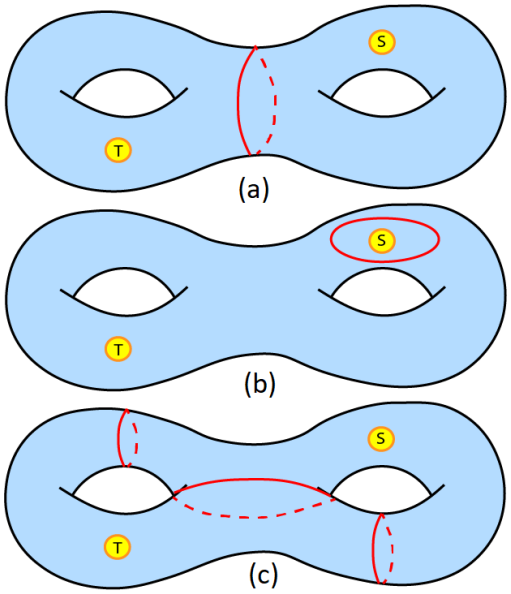

Figure 8: *Three types of graph-cut results on embedded graphs: (a) splitting, for which the cut is a splitting cycle, (b) contractible, for which the cut is a contractible cycle, and (c) decomposable, for which the cut is composed of multiple non-separating and non-contractible cycles. Note that unlike the case for a planar graph, a cut on surface-embedded graph may be composed of multiple cycles.*

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. [KT03] constrain cuts to lie within a "fuzzy" area to avoid the problem of making trivial cuts that encompass geometrically small size regions. In this paper, 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. 7. 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. 8, a cut in the dual graph may be dual to cases such as Fig. 8(b) and Fig. 8(c). 8(b) is not a splitting cycle as it is trivial, 8(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 are splitting cycles.

5.2. 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 [EFN12]:
Case 1. The boundary is a splitting cycle, as shown in Fig. 8(a);
Case 2. The boundary is a contractible cycle, as shown in Fig. 8(b);
Case 3. The boundary is composed of non-contractible cycles, as shown in Fig. 8(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 [EFN12]. 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 2. 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. 9(1), we add cycle c into S. When P_s has genus larger than 1, as shown in the bottom of Fig. 9(1), we add c into T. When P_s has genus 1, P_s is a torus which is desired and we

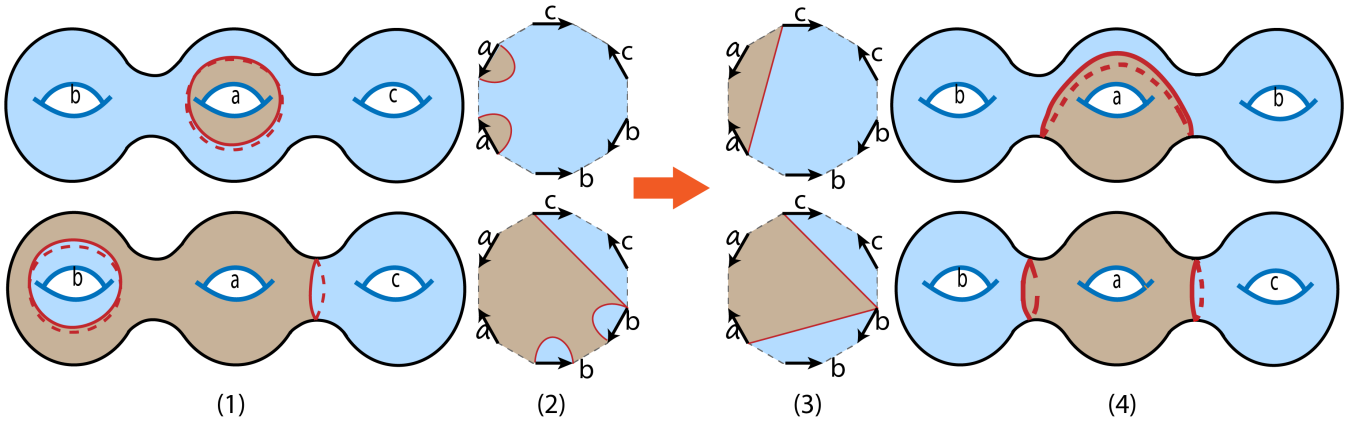

Figure 9: *Assume that the tunnel cycle a is the source and the tunnel cycles b and c are targets. (1) shows two undesirable results where some of the cut cycles are not splitting cycles, (4) shows two desirable results where the cut cycles are all splitting cycles. (2) and (3) show the fundamental polygonal (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).*

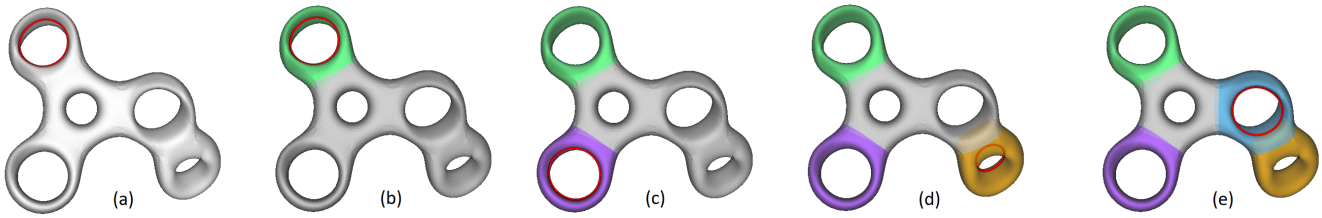

Figure 10: *Steps to finalize each region. Each time a tunnel cycle is chosen as source, and the other tunnel cycles as target, the min-cut found in each step would split the surface further.*

do nothing to the cycles. As shown in Fig. 9(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 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. 14(c), can be avoided by dynamic re-triangulation during run time.

5.3. 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. 10 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 [Cha09], 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 $O(g \cdot n^2 \cdot log(n))$.

```
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
        Subdivide the mesh
    end
end
```

Algorithm 2: Finding a cut composed of only splitting cycles that separates tunnel t_i from the other tunnels.

5.4. 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. 11, 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 [CWW13] 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* [LLS*05]. 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

$$w_e = l_e + \alpha(dist_i + dist_j) + \beta(\kappa(v_i) + \kappa(v_j)) \quad (1)$$

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,

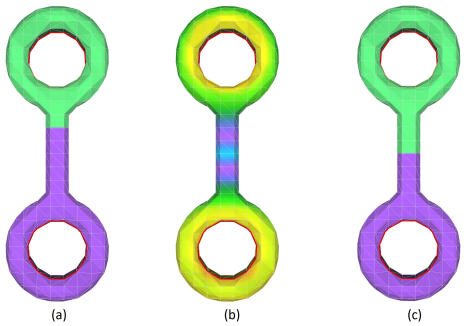

Figure 11: *The role of distances to tunnels in determining the optimal splitting cycle. (a) optimal splitting cycle found using edge lengths as edge weights. Note that all the cross sections of the central cylinder have the same diameter, so any one of them may be picked by the algorithm as optimal splitting cycle. (b) distance map formed by using tunnels as the source. (c) optimal splitting cycle found using distance aware edge weights.*

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.

6. Experimental 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 [BK04]'s implementation, which is not optimal in theoretical time complexity, but faster than most of the other implementations in practice. Fig. 12 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. 13 shows our result for mesh with boundary, as the two key steps tree-cotree decomposition and min-cut both work with meshes with boundary, no special precautions are necessary to handle them. In Fig. 13(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. 14 shows our results from different triangulations. Fig. 14(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 [YXG*13] can also be applied to post-process the boundaries. Fig. 14(c) shows that even when the triangulation is extremely sparse, the decomposition result of our ap-

Figure 12: *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.

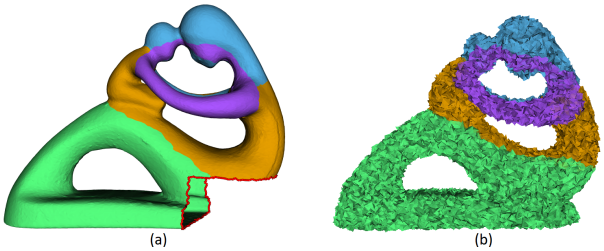

Figure 13: *Imperfect input: (a) mesh with boundaries (boundary labelled in red) (b) mesh with vertex perturbation noise.*

proach is still a valid tori decomposition, and the topological correctness of our approach is not affected by the triangulation.

Performance. We test our algorithm on various mesh models. Table 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 [CWW13] to prefactor a pair of sparse linear systems. The timing reported here does not include this pre-computation.

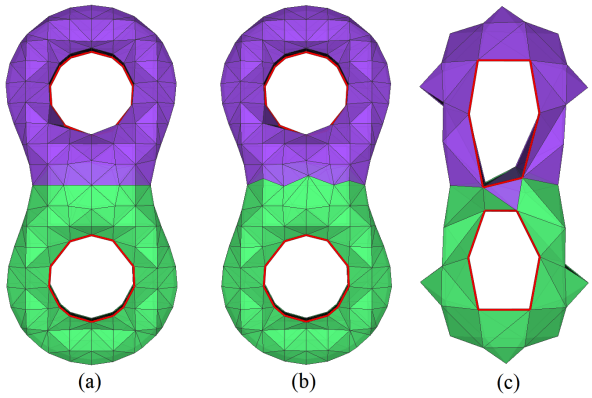

Figure 14: *Tori decomposition results from different triangulations. The smoothness of the splitting cycles is affected by the triangulation (a)(b), but even when the triangulation is extremely sparse (c), the components found by our method are still topological tori.*

7. Summary

Model	#Tri	g	t_{pre}	t_{cycles}	t_{decomp}	t_{total}
Fig. 1	29,734	5	0.253	0.387	0.524	1.164
Fig. 12(a)	2,420	5	0.036	0.031	0.053	0.120
Fig. 12(b)	220,390	2	0.548	1.532	1.396	3.476
Fig. 12(c)	63,454	8	0.329	2.585	4.204	7.118
Fig. 12(d)	62,540	5	0.363	2.386	4.193	6.942
Fig. 12(e)	20,094	3	0.223	0.837	0.103	1.163
Fig. 12(f)	4,696	2	0.121	0.437	0.033	0.591
Fig. 12(g)	342,064	64	0.984	32.245	67.378	100.607
Fig. 13(a)	50,000	4	0.268	1.299	1.171	2.738
Fig. 13(b)	50,000	4	0.271	1.345	1.402	3.018

Table 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.*

In this paper, 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.

References

[AFS06] ATTENE M., FALCIDIENO B., SPAGNUOLO M.: Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer 22*, 3 (2006), 181–193. 2

[AZC*12] AU O. K.-C., ZHENG Y., CHEN M., XU P., TAI C.-L.: Mesh segmentation with concavity-aware fields. *IEEE Transactions on Visualization and Computer Graphics 18*, 7 (2012), 1125–1134. 2

[BK04] BOYKOV Y., KOLMOGOROV V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence 26*, 9 (2004), 1124–1137. 9

[CEN09] CHAMBERS E. W., ERICKSON J., NAYYERI A.: Minimum cuts and shortest homologous cycles. In *Proceedings of the twenty-fifth annual symposium on Computational geometry* (2009), ACM, pp. 377–385. 3

[CG87] CASSON A. J., GORDON C. M.: Reducing heegaard splittings. *Topology and its Applications 27*, 3 (1987), 275–283. 3

[Cha09] CHAMBERS E. W.: *Computing interesting topological features*. Tech. rep., 2009. 2, 3, 4, 6, 8

[CJG18] CHEN J., JESTER J., GOPI M.: Fast computation of tunnels in corneal collagen structure. In *Proceedings of Computer Graphics International 2018* (New York, NY, USA, 2018), CGI 2018, ACM, pp. 57–65. 5

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. In *ACM Transactions on Graphics (ToG)* (2004), vol. 23, ACM, pp. 905–914. 2

[CWW13] CRANE K., WEISCHEDEL C., WARDETZKY M.: Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics (TOG) 32*, 5 (2013), 152. 9, 10

[CZL*15] CHEN X., ZHOU B., LU F., WANG L., BI L., TAN P.: Garment modeling with a depth camera. *ACM Transactions on Graphics (TOG) 34*, 6 (2015), 203. 1

[DFW13] DEY T. K., FAN F., WANG Y.: An efficient computation of handle and tunnel loops via reeb graphs. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 32. 1, 3, 4, 5

[DGEG09] DIAZ-GUTIERREZ P., EPPSTEIN D., GOPI M.: Curvature aware fundamental cycles. In *Computer Graphics Forum* (2009), vol. 28, Wiley Online Library, pp. 2015–2024. 3, 4, 5

[DLSCS08] DEY T. K., LI K., SUN J., COHEN-STEINER D.: Computing geometry-aware handle and tunnel loops in 3d models. *ACM Transactions on Graphics (TOG) 27*, 3 (2008), 45. 1, 3

[EFN12] ERICKSON J., FOX K., NAYYERI A.: Global minimum cuts in surface embedded graphs. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms* (2012), Society for Industrial and Applied Mathematics, pp. 1309–1318. 3, 7

[EHP04] ERICKSON J., HAR-PELED S.: Optimally cutting a surface into a disk. *Discrete & Computational Geometry 31*, 1 (2004), 37–59. 1, 3

[Epp03] EPPSTEIN D.: Dynamic generators of topologically embedded graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms* (2003), Society for Industrial and Applied Mathematics, pp. 599–608. 3, 4

[EW05] ERICKSON J., WHITTLESEY K.: Greedy optimal homotopy and homology generators. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms* (2005), Society for Industrial and Applied Mathematics, pp. 1038–1046. 3

[GF08] GOLOVINSKIY A., FUNKHOUSER T.: Randomized cuts for 3d mesh analysis. In *ACM transactions on graphics (TOG)* (2008), vol. 27, ACM, p. 145. 6

[Hat02] HATCHER A.: Algebraic topology. 2002. *Cambridge UP, Cambridge 606*, 9 (2002). 3

[HLS00] HATCHER A., LOCHAK P., SCHNEPS L.: On the teichmuller tower of mapping class groups. *Journal fur die Reine und Angewandte Mathematik* (2000), 1–24. 4

[KT03] KATZ S., TAL A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. In *ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), SIGGRAPH '03, ACM, pp. 954–961. 7

[LAPS17] LIVESU M., ATTENE M., PATANÉ G., SPAGNUOLO M.: Explicit cylindrical maps for general tubular shapes. *Computer-Aided Design 90* (2017), 27–36. 2

[LCHB12] LV J., CHEN X., HUANG J., BAO H.: Semi-supervised mesh segmentation and labeling. In *Computer Graphics Forum* (2012), vol. 31, Wiley Online Library, pp. 2241–2248. 2

[LGQ09] LI X., GU X., QIN H.: Surface mapping using consistent pants decomposition. *IEEE Transactions on Visualization and Computer Graphics 15*, 4 (2009), 558–571. 1, 2, 4

[LLS*05] LEE Y., LEE S., SHAMIR A., COHEN-OR D., SEIDEL H.-P.: Mesh scissoring with minima rule and part salience. *Computer Aided Geometric Design 22*, 5 (2005), 444–465. 2, 9

[LVS*13] LIVESU M., VINING N., SHEFFER A., GREGSON J., SCATENI R.: Polycut: monotone graph-cuts for polycube base-complex construction. *ACM Transactions on Graphics (TOG) 32*, 6 (2013), 171. 1

[RMG18] RODRIGUES R. S., MORGADO J. F., GOMES A. J.: Part-based mesh segmentation: A survey. In *Computer Graphics Forum* (2018), vol. 37, Wiley Online Library, pp. 235–274. 2

[Séq12] SÉQUIN C. H.: Topological tori as abstract art. *Journal of Mathematics and the Arts 6*, 4 (2012), 191–209. 1, 2

[Sha08] SHAMIR A.: A survey on mesh segmentation techniques. In *Computer graphics forum* (2008), vol. 27, Wiley Online Library, pp. 1539–1556. 2

[TVD07] TIERNY J., VANDEBORRE J.-P., DAOUDI M.: Topology driven 3d mesh hierarchical segmentation. In *null* (2007), IEEE, pp. 215–220. 2

[Yao18] YAO L.: Making a torus. http://fab.cba.mit.edu/classes/863.11/people/lining.yao/design%20Projects/project1.html, 2018. [Online; accessed 29-Jan-2018]. 2

[YL11] YU W., LI X.: Computing 3d shape guarding and star decomposition. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 2087–2096. 1

[YXG*13] YANG Y., XU W., GUO X., ZHOU K., GUO B.: Boundary-aware multidomain subspace deformation. *IEEE transactions on visualization and computer graphics 19*, 10 (2013), 1633–1645. 1, 9

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with poisson-based gradient field manipulation. *ACM Transactions on Graphics (TOG) 23*, 3 (2004), 644–651. 1

[ZL14] ZHANG K., LI X.: Searching geometry-aware pants decomposition in different isotopy classes. *Geometry, Imaging and Computing 1*, 3 (2014), 367–393. 2, 4

[ZYH*15] ZHOU Y., YIN K., HUANG H., ZHANG H., GONG M., COHEN-OR D.: Generalized cylinder decomposition. *ACM Trans. Graph. 34*, 6 (2015), 171–1. 1