Computing Geometry-aware Handle and Tunnel Loops in 3D Models

Tamal K. Dey* Ohio State University
Kuiyu Li† Ohio State University
Jian Sun‡ Stanford University
David Cohen-Steiner§ Inria, Sophia-Antipolis

Abstract

Many applications such as topology repair, model editing, surface parameterization, and feature recognition benefit from computing loops on surfaces that wrap around their 'handles' and 'tunnels'. Computing such loops while optimizing their geometric lengths is difficult. On the other hand, computing such loops without considering geometry is easy but may not be very useful. In this paper we strike a balance by computing topologically correct loops that are also geometrically relevant. Our algorithm is a novel application of the concepts from topological persistence introduced recently in computational topology. The usability of the computed loops is demonstrated with some examples in feature identification and topology simplification.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, Surface, Solid, and Object Representations; I.3.4 [Computer Graphics]: Graphics Utilities—Graphics Editors

Keywords: Surface loop, topology, persistent homology, shape analysis, topology repair, feature identification

1 Introduction

Many applications such as topology repair of 3D models [Bischoff and Kobbelt 2005; Zhou et al. 2007], surface parameterization [Ben-Chen et al. 2008; Gu et al. 2002; Sheffer and Hart 2002], and feature recognition [Biasotti et al. 2008; Dey et al. 2007] benefit from automatic detection of loops on surfaces that are associated with features such as 'handles' and 'tunnels'. A 3D model created from a point cloud data often contains spurious topology such as tiny handles and tunnels [Levoy et al. 2000]. If appropriate loops around these features can be computed, they can be eliminated to clean up the model. In surface parameterization, many algorithms need to cut a surface into a disk. These cutting loops should be small to lessen the effect of discontinuity in parameterization across the boundary. Again, if these loops are chosen around small parts of the handles and tunnels, the length of the boundary in the flattened disk remains small. Feature recognition such as identifying handles and tunnels in a 3D model can use a representative loop for each such feature.

It is known that a closed surface of genus g can be cut into a disk by cutting it along $2g$ loops. If one is only concerned with the genus reduction in topology simplification, these $2g$ loops can be computed easily without being aware of the geometry. However, as we argued above, most applications cannot be geometry-oblivious and should choose these loops to be small and respect the embedding of the shape. For example, to remove a handle in a 3D shape as in Figure 1, we need to identify a small loop around the handle which spans a 'surface' in the interior of the shape. Similarly, to close a small tunnel as in Figure 7, we need to identify a small loop around the tunnel which spans a 'surface' in the exterior of the shape.

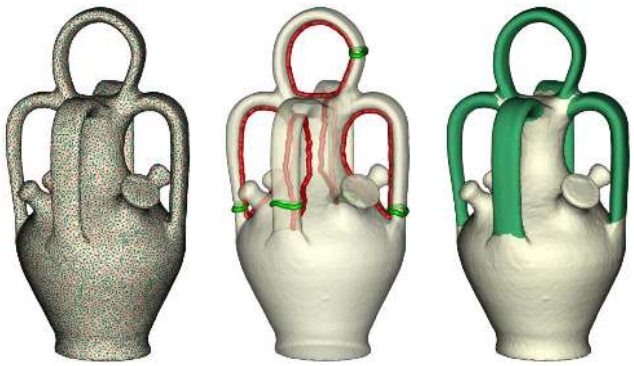


Figure 1: *Computing handle and tunnel loops on Botijo. From left to right: Input model, handle (green) and tunnel (red) loops, handle features (green) identified on Botijo.*

The requirement that the loops span a 'surface' in the complement space of the input surface disallows many loops which would otherwise be eligible for simplifying the topology of the surface alone. Since 3D models not only embody the bounding surface but also the space surrounded by it, the cutting loops also should take into account the embedding of the surface. To reach this goal, the authors in [Dey et al. 2007] defined a class of loops on a surface $\mathbb{M} \subset \mathbb{R}^3$, called handle and tunnel loops in terms of a homology group. A loop is a *handle* if it spans a surface, say D, in the *bounded* space bordered by $\mathbb{M}$ but does not do so in $\mathbb{M}$ itself. If one cuts $\mathbb{M}$ along such a loop and fills the boundaries with two copies of D, one eliminates a handle. Similarly, a loop is a *tunnel* if it spans a surface in the *unbounded* space bordered by $\mathbb{M}$ and does not do so in $\mathbb{M}$. Its removal eliminates a tunnel. The middle picture in Figure 1 shows such handle and tunnel loops for the model *Botijo*.

Contribution. In this paper we present an algorithm to compute handle and tunnel loops based on persistent homology pioneered by Edelsbrunner, Letscher, and Zomorodian [Edelsbrunner et al. 2002]. The algorithm computes a set of well defined g handle and g tunnel loops for a closed surface of genus g in $\mathbb{R}^3$. Compared to the earlier methods, our algorithm has the advantage that it provides a mathematical guarantee on detecting handle and tunnel loops and does not require computing any extra structure such as Reeb graph, medial axis, or curve skeletons. Further, it does not impose any restriction on the class of input 3D models as in some of the earlier works.

Our use of persistent homology enables us to compute the g handle and g tunnel loops which are topologically correct and geometrically small. A geodesic measure associated with the loops can be used to simplify features based on their sizes. The method is simple and is general enough to be applied to any surface mesh that conforms to an associated volume mesh. Specifically, the input $\mathbb{M}$ should be a subcomplex of a complex $\mathbb{K}$ that tessellates the convex hull of $\mathbb{M}$. The actual input surface may satisfy this condition directly or may need preprocessing to satisfy it.

*e-mail: tamaldey@cse.ohio-state.edu
†e-mail: liku@cse.ohio-state.edu
‡e-mail: sunjian@stanford.edu
§e-mail: david.cohen-steiner@sophia.inria.fr

If the input surface $\mathbb{M}$ is a sub-complex of the Delaunay triangulation of its vertex set, the complex $\mathbb{K}$ representing the volume mesh is immediately obtained from the Delaunay triangulation. For example, many Delaunay based surface reconstruction algorithms [Dey 2007] and meshing algorithms [Alliez et al. 2005; Cheng et al. 2007; Oudot et al. 2005] produce such surface meshes. If the input is an iso-surface extracted from a scalar grid data, a volume mesh conforming to the surface can be easily generated by computing intersections between cubic faces and the iso-surface. The most demanding case arises when the input surface is neither Delaunay nor an iso-surface. In this case one may remesh the surface so that it becomes a Delaunay mesh using any of the existing algorithms [Alliez et al. 2005; Cheng et al. 2007; Oudot et al. 2005]. In our experiments we use the Delaunay mesh method of [Cheng et al. 2007] since it can handle non-smooth surfaces with arbitrary small input angles. Alternatively, one may use the scan-conversion algorithm as in [Bischoff and Kobbelt 2005; Zhou et al. 2007] to create a volume data out of a surface mesh and apply our algorithm as in the iso-surface case.

Because of its simplicity, our algorithm applies to a large class of input. It runs reasonably fast in practice. In particular, it scales well with large data sets for the iso-surface case. We show examples where the computed loops are used to simplify topology or to identify features such as 'handles' and 'tunnels' in a 3D model.

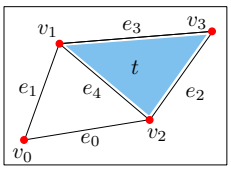
Previous work. Various algorithms for computing different types of non-trivial loops on surfaces have been proposed in recent years, e.g., see [Erickson and Whittlesey 2005; Chen and Freedman 2008; Ni et al. 2004; Éric Colin de Verdière and Lazarus 2005; Zomorodian and Carlsson 2007] and the references therein. The algorithms of [Erickson and Whittlesey 2005; Chen and Freedman 2008; Éric Colin de Verdière and Lazarus 2005] compute loops on surfaces which are optimal with some geometric measures. These works do not guarantee handle and tunnel loops as we define. Furthermore, results on practical validity of these algorithms are not yet available.

Several approaches of computing non-trivial loops on surfaces have been proposed in the context of topology simplification. In [El-Sana and Varshney 1997], El-Sana and Varshney use the concept of α-hulls to identify small tunnels and concavities that are not accessible by a user-specified ball rolling on the surface. Guskov and Wood [Guskov and Wood 2001] propose a surface growing strategy to remove small handles contained completely in a mesh neighborhood of a given size. Nooruddin and Turk [Nooruddin and Turk 2003] apply some morphological operations such as opening and closing on the volume meshes to remove small handles. Wood et al. [Wood et al. 2004] use cycles in the Reeb graph to compute handles and perform a disk-filling procedure to remove handles. These works do not provide any mathematical guarantees of producing handle and tunnel loops as we aim for. Some of them may also generate new handles and tunnels [Nooruddin and Turk 2003; Wood et al. 2004] as a side effect of removing others.

Another set of algorithms uses some kind of graph structures built from the input model to compute the handles and tunnels. Shattuck and Leahy [Shattuck and Leahy 2001] build a graph from a Reeb graph and remove handles by breaking the cycles in this graph. Although a robust algorithm for computing Reeb graphs is available [Pascucci et al. 2007], choosing an appropriate sweeping direction to build the Reeb graph remains a challenge. Zhou, Ju, and Hu [Zhou et al. 2007] propose a repair technique based on detecting loops in the medial axis. This method, because of its use of the medial axis, cannot be applied to a class of models whose medial axis form closed surfaces. Moreover, the construction of medial axis with reasonable accuracy is a non-trivial task. Dey, Li, and Sun [Dey et al. 2007] propose an algorithm which requires computing a contraction of the medial axis for producing curve skeletons [Dey and Sun 2006]. It also suffers from the problem of computing these structures robustly.

2 Cycle, boundary and homology

In this section, we give a very brief introduction to homology through an intuitive example since handle and tunnel loops are defined via homology. For a formal treatment on homology, readers are referred to [Hatcher 2002].



In a simplicial complex $\mathbb{K}$, a *p-chain* c is a formal sum of p-simplices in $\mathbb{K}$, i.e., $c = \sum_i a_i \sigma_i$ where σ_i is a p-simplex in $\mathbb{K}$ and a_i is an integer modulo 2. Consider the simplicial complex $\mathbb{K}$ in the figure on right. Here v_0 is a 0-chain and $(e_0 + e_3)$ is a 1-chain. Two p-chains are added component-wise, i.e., if $c' = \sum_i a'_i \sigma_i$ then $c + c' = \sum (a_i + a'_i)\sigma_i$. Notice that, because of modulo 2 addition, we have $c + c = 0$. The set of p-chains together with the addition operation form the *group of p-chains* denoted as C_p. The boundary of a p-simplex is the sum of its $(p-1)$-dimensional faces. Specifically, for the simplex spanned by the vertices v_0 up to v_p,

$$\partial_p \sigma = \partial_p [v_0, v_1, \cdots, v_p] = \sum_{i=0}^{p} [v_0, \cdots, \hat{v_i}, \cdots v_p]$$

where the hat indicates that v_i is dropped. For example, $\partial t = (e_2 + e_3 + e_4)$. By linear extension, a p-chain $c = \sum_i a_i \sigma_i$ has boundary $\partial_p c = \sum_i a_i \partial_p \sigma_i$. For example, $\partial(e_0 + e_1 + e_4) = (v_0 + v_2) + (v_0 + v_1) + (v_1 + v_2) = 0$. Thus, we relate chain groups using the *boundary map* $\partial_p : C_p \to C_{p-1}$. The index of the boundary operator ∂_p can be dropped for convenience since it is implied by the dimension of the chain.

A p-chain c is a *p-cycle* if $\partial c = 0$. For example, $(e_0 + e_1 + e_4)$ is a 1-cycle. A p-chain c is a *p-boundary* if there exists a $(p+1)$-chain d with $c = \partial d$. For example $(e_2 + e_3 + e_4)$ is a 1-boundary. One can verify that a p-boundary has to be a p-cycle. The set of all p-cycles, denoted by Z_p, is the kernel of the pth boundary map, i.e., $Z_p =$ Ker ∂_p. The set of all p-boundaries, denoted by B_p, is the image of the $(p+1)$th-boundary map, i.e., $B_p =$ Img ∂_{p+1}. Thus, $B_p \subset Z_p$. The homology groups are defined as quotients, $H_p = Z_p / B_p$ for each p. One can think each element in H_p as an equivalent class. Two p-cycles c_1 and c_2 are in the same equivalent class if the difference between them is a p-boundary, i.e., $c_1 - c_2 \in B_p$. For example $(e_0 + e_1 + e_4)$ and $(e_0 + e_1 + e_2 + e_3)$ are equivalent. It is customary to use $[c]$ to denote the equivalent class represented by a p-cycle c. A p-cycle c is *trivial* in H_p if $[c]$ represents the zero element in H_p. Any p-boundary is trivial in H_p. For example, $[e_2 + e_3 + e_4]$ is trivial in $H_1(\mathbb{K})$. In our setting[1], H_p simply is a vector space generated by a basis. The number of elements in a basis of H_p is its *dimension/rank*, which is also known as pth Betti number β_p. In our example, $H_1(\mathbb{K})$ is only one dimensional and $[e_0 + e_1 + e_4]$ is a basis.

In this paper, we are interested in the first homology group H_1. We give a 1-cycle a more intuitive name, *loop*. Now we give a formal definition of handle and tunnel loops for a surface in $\mathbb{R}^3$.

3 Handle/Tunnel loops

We are interested in computing a representation of handles and tunnels in a shape. Toward this goal we use the definition of handle

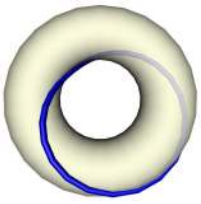
[1] The assumption that a_i's are integers modulo 2

and tunnel loops introduced in [Dey et al. 2007] and present some explanations why such definition is useful.

Let $\mathbb{M}$ be a closed (compact and without boundary) surface in $\mathbb{R}^3$. For simplicity, we assume $\mathbb{M}$ is connected. In case $\mathbb{M}$ is not connected, our result can be applied component-wise. The genus g of $\mathbb{M}$ is the maximum number of disjoint simple loops whose removal does not disconnect $\mathbb{M}$. $\mathbb{M}$ separates $\mathbb{R}^3$ into two parts: *inside* denoted $\mathbb{I}$ and *outside* denoted $\mathbb{O}$ both including $\mathbb{M}$.

Definition 1 *A loop in $\mathbb{M}$ is a* tunnel loop *if it is trivial in $H_1(\mathbb{O})$ and non trivial in $H_1(\mathbb{M})$.*

Definition 2 *A loop in $\mathbb{M}$ is a* handle loop *if it is trivial in $H_1(\mathbb{I})$ and non trivial in $H_1(\mathbb{M})$.*



By definition, a tunnel loop spans the union of a set of triangles in $\mathbb{O}$ since, being trivial in $\mathbb{O}$, it has to bound a 2-chain. Similar statement holds for handle loops. These properties commensurate with the requirement that the handle loops should bound 'surfaces' in $\mathbb{I}$ whereas tunnel loops should bound them in $\mathbb{O}$. The set of tunnel loops are disjoint from the set of handle loops, i.e., no loop on $\mathbb{M}$ can be both handle and tunnel. This fact follows from Theorem 1. By definition, a tunnel loop or a handle loop must be non trivial in $H_1(\mathbb{M})$. However, a non trivial loop on $\mathbb{M}$ may neither be handle nor tunnel. For example, the loop shown on the torus is neither a handle nor a tunnel since it is non trivial in both $H_1(\mathbb{I})$ and $H_1(\mathbb{O})$. Such loop is not desirable in some applications such as topological simplification since after cutting the surface $\mathbb{M}$ along such loop one can not fill the boundary with a disk without intersecting $\mathbb{M}$.

The following theorem elucidates some structural properties about the spaces generated by handle and tunnel loops. It can be derived from Mayer-Vietoris sequence given by $\mathbb{M} = \mathbb{I} \cap \mathbb{O}$; see [Dey et al. 2007] for a line of proof.

Theorem 1 *For any connected closed surface $\mathbb{M} \subset \mathbb{R}^3$ of genus g, $H_1(\mathbb{M})$ is the direct sum of the spaces generated by handle and tunnel loops each of which is g-dimensional. Moreover, handle loops generate $H_1(\mathbb{O})$ and tunnel loops generate $H_1(\mathbb{I})$.*

4 Positive/Negative simplices

Our algorithm is based on a pairing of simplices which is a key concept in computing persistent homology [Edelsbrunner et al. 2002]. A *filtration* of a simplicial complex $\mathbb{K}$ is a nested sequence of complexes,

$$\emptyset = \mathbb{K}_{-1} \subset \mathbb{K}_0 \subset \mathbb{K}_1 \subset \cdots \subset \mathbb{K}_n = \mathbb{K}.$$

The inclusion map $f : \mathbb{K}_{i-1} \hookrightarrow \mathbb{K}_i$ defined by $f(x) = x$ induces a map between the homology groups: $f_* : H_p(\mathbb{K}_{i-1}) \to H_p(\mathbb{K}_i)$. The nested sequence of complexes corresponds to a sequence of homology groups connected by the induced maps,

$$0 = H_p(\mathbb{K}_{-1}) \to H_p(\mathbb{K}_0) \to \cdots \to H_p(\mathbb{K}_n) = H_p(\mathbb{K}).$$

Persistent homology basically studies how the homology groups change over the filtration. In this paper, we only explain the pairing concept in persistent homology relevant to our algorithm. For a formal treatment of persistent homology, readers are referred to [Edelsbrunner et al. 2002; Zomorodian and Carlsson 2005].

We may assume $\mathbb{K}_i - \mathbb{K}_{i-1} = \sigma_i$, i.e., adding a single simplex each time in the filtration. Only two possible changes can happen when a single simplex σ_i of dimension p is added to $\mathbb{K}_{i-1}$. One is the creation of a non-boundary p-cycle when σ_i is called a *positive p-simplex*. The other change is the killing of an existing $(p-1)$-cycle when σ_i is called a *negative p-simplex*.

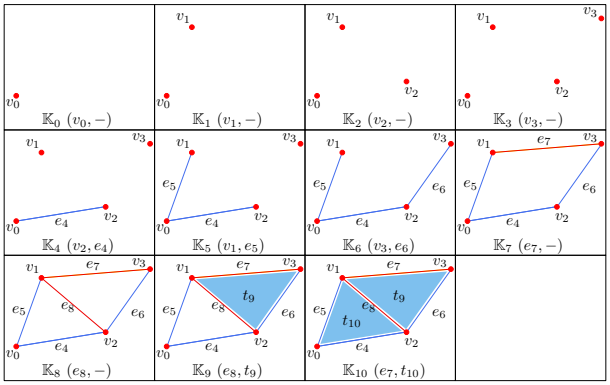


Figure 2: *Red simplices are positive and blue ones are negative. The simplices are indexed to coincide with their order in the filtration. $(\cdot,\cdot)$ in each subcomplex shows the pairing between the positive and the negative. The second component missing in the parenthesis shows the introducing of a positive simplex.*

We elaborate the above two changes through an example depicted in Figure 2. When one moves from $\mathbb{K}_6$ to $\mathbb{K}_7$, a non-boundary loop which is a 1-cycle $(e_4 + e_5 + e_6 + e_7)$ is created after adding edge e_7. Strictly speaking, a positive p-simplex may create more than one p-cycle. Only one of them is independent and the others are its linear combinations with the existing ones in $\mathbb{K}_{i-1}$. From $\mathbb{K}_7$ to $\mathbb{K}_8$, the introduction of edge e_8 creates two non-boundary loops $(e_4 + e_5 + e_8)$ and $(e_6 + e_7 + e_8)$. But any one of them is the linear combination of the other one with the existing loop $(e_4 + e_5 + e_6 + e_7)$. Notice that there is no canonical way to choose an independent one. However, the creation of a loop is reflected in the increase of the dimension of H_1. In other words, in general, the Betti number β_p increases by 1 for a positive simplex. For a negative simplex, we get the opposite effect. In this case β_{p-1} decreases by 1 signifying a death of a cycle. However, unlike positive simplices, the killed cycle is determined uniquely up to homology, which is the equivalent class carried by the boundary of σ_i. For example, in Figure 2, the loop $(e_6 + e_7 + e_8)$ gets killed by triangle t_9 when we go from $\mathbb{K}_8$ to $\mathbb{K}_9$.

By the theory of persistent homology [Edelsbrunner et al. 2002], a negative p-simplex σ is always paired with a unique positive $(p-1)$-simplex σ' where σ kills a $(p-1)$-cycle created by σ'. The pairing of positive simplices with negative simplices is the key concept in persistent homology. Each pair tells the birth time of a non-boundary cycle as well as its death time in the filtration. The life time of a non-boundary cycle measures its persistence.

Pairing algorithm. The goal of the persistence algorithm is to compute all the pairings between positive and negative simplices. Of course, one needs to determine if a simplex is positive or negative. It turns out that this goal can be achieved by applying the following simple procedure to each simplex in the order of filtration. We say a simplex σ is *younger* than a simplex σ' if σ appears later than σ' in the filtration.

Let us again consider the example in Figure 2 and see how the algorithm PAIR works. From $\mathbb{K}_6$ to $\mathbb{K}_7$, e_7 is added. Its boundary is $c = (v_1 + v_3)$. The vertex v_3 is the youngest positive vertex in c but it is paired with e_6 in $\mathbb{K}_6$. Thus, c is updated to $(v_2 + v_3 + v_3 + v_1) = (v_2 + v_1)$. The vertex v_2 becomes the youngest

Algorithm 1 PAIR(σ)

```
1: c = ∂_p σ
2: d is the youngest positive (p−1)-simplex in c.
3: while d is paired and c is not empty do
4:    Let c' be the cycle killed by the simplex paired with d
5:    c = c' + c  \*this addition may cancel simplices*\
6:    Update d to be the youngest positive (p−1)-simplex in c
7: end while
8: if c is not empty then
9:    σ is a negative p-simplex and paired with d
10: else
11:   σ is a positive p-simplex
12: end if
```

positive one but it is paired with e_4. So, c is updated to $(v_0 + v_1)$. The vertex v_1 becomes the youngest positive one but it is paired with e_5. So, c is updated to empty. Hence e_7 is a positive edge. Now we examine the addition of the triangle t_{10} from $\mathbb{K}_9$ to $\mathbb{K}_{10}$. The boundary of t_{10} is $c = (e_4 + e_5 + e_8)$. The youngest positive edge e_8 is paired with t_9. Thus, c is updated by adding the cycle killed by t_9 to $(e_4 + e_5 + e_6 + e_7)$. Since e_7 is the youngest positive edge that is not yet paired, t_{10} finds e_7 as its paired positive edge. Observe that, we finally obtain a loop that is killed by adding the negative triangle. For example, we obtain the loop $(e_4 + e_5 + e_6 + e_7)$ by adding t_{10}. The following fact about such loop is used later to prove the correctness of our algorithm.

Fact 1 *The paired positive edge in a killed loop is the youngest among all edges in the loop.*

We are interested in the killed loops. The handle and tunnel loops are actually chosen from them. In fact, we get a series of loops in the procedure for finding the paired positive edge for a negative triangle, as we can see from the while-loop in PAIR. These loops are homologous, i.e., they belong to the same homology class. Although they are topologically equivalent, we need to choose one among them that is also good geometrically.

The persistence can also be explained nicely in terms of matrix operations. The interested readers are referred to [Zomorodian and Carlsson 2005; Cohen-Steiner et al. 2006].

5 Computing handle and tunnel loops

In this section we describe an algorithm for computing a set of generating handle and tunnel loops on arbitrary closed surfaces. We first describe an algorithm that computes topologically correct handle and tunnel loops. We then refine the algorithm by incorporating geometry. As a result, we obtain a set of handle and tunnel loops that are geometrically meaningful.

In order to apply the algorithm for persistence, we assume that the input surface $\mathbb{M}$ is presented with a simplicial complex $\mathbb{K}$ which tessellates the convex hull of $\mathbb{M}$ and $\mathbb{M}$ is a subcomplex of $\mathbb{K}$. This means that we have the explicit simplicial representations for both inside space $\mathbb{I}$ and outside space $\mathbb{O}$. Actually, we only need the 2-skeleton of the complex $\mathbb{K}$ for the algorithm. As we have discussed in the introduction, one may have such a simplicial complex $\mathbb{K}$ for free in case $\mathbb{M}$ is a Delaunay mesh. If $\mathbb{M}$ is an iso-surface associated with some scalar volume data, we obtain the desired 2-skeleton (triangles only) of $\mathbb{K}$ by intersecting the cubic faces with $\mathbb{M}$ and producing a 2D triangulation of the polygons and squares trivially. For all other surface mesh $\mathbb{M}$, one may obtain $\mathbb{K}$ by using existing Delaunay meshing algorithms [Alliez et al. 2005; Cheng et al. 2007; Oudot et al. 2005]. In our experiments, we use DELPSC algorithm/software of [Cheng et al. 2007].

5.1 Topological algorithm

Assume $\mathbb{K}$ is given. We only need to consider the 2-skeleton (points, edges and triangles) of $\mathbb{K}$ since we are only interested in obtaining 1-cycles/loops. We build the filtration of the 2-skeleton of $\mathbb{K}$ in the following three steps.

Step 1: The simplices on the surface $\mathbb{M}$ are added into the filtration in any arbitrary order. Since $H_1(\mathbb{M})$ is of rank $2g$, the algorithm PAIR generates $2g$ number of unpaired positive edges. Figure 3 shows these unpaired positive edges for the model *Mother*.

Step 2: The simplices up to dimension 2 in $\mathbb{I}$ are added into the filtration. Since $H_1(\mathbb{I})$ is of rank g, half of $2g$ positive edges generated in Step 1 get paired with the negative triangles in $\mathbb{I}$. Each pair corresponds to a killed loop. Thus we obtain g loops, denoted $\{h_i\}_{i=1}^{g}$. Figure 3 shows these g loops as well as the paired negative triangles for the model *Mother*. Theorem 2 says that these g loops are a set of generating handle loops.

Step 3: The simplices up to dimension 2 in $\mathbb{O}$ are added into the filtration. Since $H_1(\mathbb{K})$ is trivial, the remaining half of positive edges generated in Step 1 get paired with the negative triangles in $\mathbb{O}$. As before we obtain another g loops, denoted $\{t_i\}_{i=1}^{g}$. Figure 3 shows these g loops as well as the paired negative triangles for the model *Mother*. Theorem 2 says that these g loops are a set of generating tunnel loops.

Observe that loops killed by negative triangles in $\mathbb{I}$ and $\mathbb{O}$ have all their edges lying in $\mathbb{M}$ because all edges of $\mathbb{M}$ including the positive ones are added before any other edge in $\mathbb{K}$ and Fact 1 applies. Observe also that one can skip adding simplices from $\mathbb{I}$ or $\mathbb{O}$ once half of the unpaired edges are paired.

Theorem 2 *The loops $\{h_i\}_{i=1}^{g}$ and $\{t_i\}_{i=1}^{g}$ are handle and tunnel loops forming a basis for $H_1(\mathbb{O})$ and $H_1(\mathbb{I})$ respectively.*

Proof 1 *As we observed, by Fact 1, each h_i and t_i lie on the surface $\mathbb{M}$. By construction, h_is (resp. t_is) form a basis of the kernel of the map from $H_1(\mathbb{M})$ to $H_1(\mathbb{I})$ (resp. $H_1(\mathbb{O})$) induced by inclusion. That is, they respectively form bases of the spaces generated by handle loops and by tunnel loops. The conclusion follows from theorem 1.*

5.2 Bringing geometry

Although the handle and tunnel loops as computed in the previous section are guaranteed to be topologically correct, they may not be very good geometrically as seen in Figure 3. Our goal is to compute handle and tunnel loops of small size. One way to achieve this goal would be to tighten the computed loops. Some methods to tighten loops within the same homotopy type by utilizing universal cover is known, see [Yin et al. 2007; Éric Colin de Verdière and Erickson 2006] for example. However, there is no known algorithm to tighten the loops up to homology. We suspect that this problem is very hard. Thus, instead of computing the universal cover or employing an optimization step which would add an extra level of complication to the algorithm, we exploit a flexibility built into the algorithm for persistence. First, we choose a specific loop among all the loops encountered during the search for a loop killed by a negative triangle. Second, we control the order of the simplices in $\mathbb{I}$ and $\mathbb{O}$ added into the filtration so that the negative triangles are located at the places where the shape is of small size. For this, we need the notion of *geodesic size*.

Geodesic Size. First, we assign each edge in $\mathbb{K}$ a *geodesic size* as follows. Consider the case that $\mathbb{K}$ has all vertices on the surface $\mathbb{M}$ as shown in Figure 4. This case occurs, for example, when $\mathbb{K}$ is a Delaunay triangulation of a set of vertices on $\mathbb{M}$. For any edge e with endpoints a and b, we define the geodesic size $g(e)$ as the

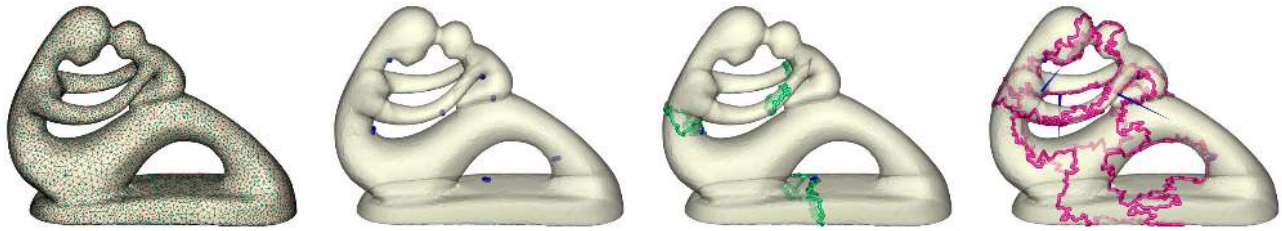


Figure 3: *Handle and tunnel loops computed with the topological algorithm on Mother model (g = 4). From left to right: Input model, 2g unpaired positive edges on $\mathbb{M}$, handle loops are generated after g unpaired edges get paired with g negative triangles in $\mathbb{I}$, tunnel loops are generated after the rest of g unpaired edges get paired with g negative triangles in $\mathbb{O}$.*

minimal geodesic distance between a and b over the edges of the surface $\mathbb{M}$ (using Dijkstra's shortest path). For a general $\mathbb{K}$ where the endpoints a and b may not be on the surface $\mathbb{M}$ (Figure 5), we find the closest points on $\mathbb{M}$ for two endpoints, denoted a' and b' respectively, and define $g(e)$ as the minimal geodesic distance between a' and b'. Now we assign each triangle the maximal geodesic size of its edges, namely $g(t) = max_{e \in t} g(e)$ for a triangle t.

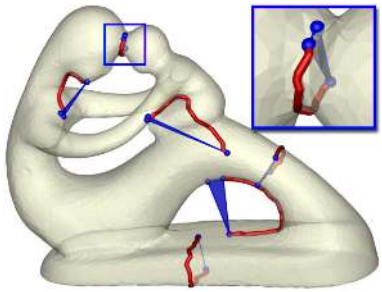


Figure 4: *Six triangles (blue) in the Delaunay triangulation of Mother with the geodesic curves (red) determining their geodesic sizes.*

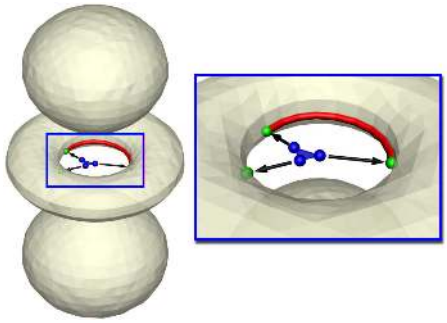


Figure 5: *Computation of $g(t)$ for a triangle t in $\mathbb{O}$ of the scalar volume data Atom.*

We make two refinements to the topological algorithm to incorporate the geometry.

Refinement 1: In the pairing algorithm, we obtain a series of homologous loops while searching for the paired positive edge for a negative triangle σ (the while-loop in PAIR). This is a series of expanding loops since each loop is obtained by adding a boundary loop into the previous one. Among those consider the ones that lie completely on the surface $\mathbb{M}$. We know that this set cannot be empty as the loop killed by σ lies completely in $\mathbb{M}$. Let $\ell_1, \ell_2, ..., \ell_k$ be this set of loops sorted in the sequence they are found during the search. This means that the youngest edge in ℓ_i appears later in the filtration than the youngest edge in ℓ_{i+1}. In other words, ℓ_1 is the first loop obtained during expansion which has all edges on $\mathbb{M}$. We write $\ell(\sigma) = \ell_1$ and refine Step 2 and 3 by outputting h_i or t_i as $\ell(\sigma)$. The reason for such a choice over the one containing the positive edge is that we want the loop to lie as close as possible to the negative triangle. The location of the negative triangle indicates small size of the shape by Refinement 2 whereas we cannot make any such assumption about the positive edge. Figure 6 (top row) shows how the loops in Figure 3 are tightened by Refinement 1.

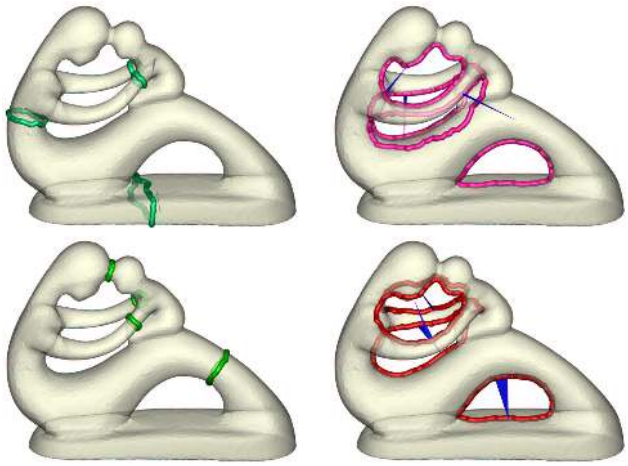


Figure 6: *Top row: Handle and tunnel loops computed after Refinement 1. Bottom row: Handle and tunnel loops computed after Refinements 1 and 2.*

Refinement 2: In this refinement we add the triangles in $\mathbb{I}$ or $\mathbb{O}$ in the increasing order of their geodesic sizes into the filtration. The intuition of this refinement is based on the following observation. Let $D \subset \mathbb{K}$ form a 2-chain in $\mathbb{I}$ or $\mathbb{O}$ with boundary in $\mathbb{M}$. Abusing notations, we call D a *cross section* for $\mathbb{M}$. A positive edge survived after Step 1 gets paired only when a cross section of $\mathbb{M}$ gets filled. Adding the triangles in the order of increasing geodesic size forces the cross sections with small size to get filled first, which makes the negative triangles to be at places admitting small cross sections. Figure 6 shows how the the loops move to smaller regions due to Refinement 2. The algorithm HANTUN represents the entire procedure with refinements.

We implemented our algorithm HANTUN in C++, and ran a series of tests on both Delaunay meshes and scalar volume data. All experiments were done on a Dell PC with 2.0GHz Intel Xeon CPU and 2GB RAM. The number of detected handle and tunnel loops in all models agrees with the genus of the model which we veri-

Algorithm 2 HANTUN($\mathbb{K}$)

Require: $\mathbb{M}$ is a subcomplex of $\mathbb{K}$
Ensure: $\{h_i\}$ and $\{t_i\}$ are a set of generating handles and tunnels of small size respectively.
1: Compute the geodesic size for edges and triangles in $\mathbb{I}$ and $\mathbb{O}$
2: For each simplex σ on $\mathbb{M}$, Pair(σ)
3: Let E be the set of the unpaired positive edges.
4: In the order of increasing geodesic size, Pair(σ) for each simplex $\sigma \in \mathbb{I}$
5: **if** σ is negative and its paired positive edge is in E **then**
6: Output the loop $\ell(\sigma)$ as a handle loop h_i.
7: **end if**
8: In the order of increasing geodesic size, Pair(σ) for each simplex $\sigma \in \mathbb{O}$
9: **if** σ is negative and its paired positive edge is in E **then**
10: Output the loop $\ell(\sigma)$ as a tunnel loop t_i.
11: **end if**

fied with Euler's characteristic. The times are summarized in Table 1. Observe that our method runs faster than the method in [Dey et al. 2007] which also computes handle and tunnel loops but for a smaller class of inputs. In most cases geodesic size computation dominates the computation time. The times for surface meshes do not scale as good as in the volume data.

Data	Size	Ptime	Geod	Loop	DLS
Knotty cup	5.4k, 31.9k	6.9	3.2	0.8	fail
Mother	19.5k, 117k	30.3	13.9	4.1	46.5
Molecule	19.9k, 115k	35.8	5.5	0.9	29.8
Botijo	30k, 176k	77.1	20.6	2.3	75.0
Casting	31.9k, 169k	92.7	25.2	14.0	84.91
Buddha	109k, 614k	1377.1	168.7	16.2	543.27
Hip	173k, 996k	2400.0	648.1	265.5	fail
Children	199k, 1168k	12.5	1350.8	2267.3	fail
Gearbox	478k, 2582k	23759.9	2058.6	33769.1	fail
Colon	854k, 4966k	30572.2	12103.9	37093.4	fail
Atom	3.3k, 41.8k	0.5	0.3	0.2	fail
Aneurism	23k, 5210k	3.1	9.1	73.0	fail
Engine 1	157k, 2766k	3.2	520.0	207.1	fail
Engine 2	629k, 20040k	11.3	7812.6	3392.6	fail

Table 1: *Times (seconds) broken into two parts, one for geodesic size computations (Geod), and the other for loop computations (Loop). (n,m) in Size column indicates n surface triangles in $\mathbb{M}$ and m volume triangles in $\mathbb{K} \setminus \mathbb{M}$. Ptime column shows the preprocessing time for producing volume meshes from the surface. For point cloud data* Children*, this is simply the time for a Delaunay based surface reconstruction (*COCONE *software). DLS column shows the time for the method in [Dey et al. 2007] denoted as DLS algorithm.*

6 Feature identification and simplification

In this section, we apply the handle and tunnel loops for computing handle and tunnel features on 3D models. We also show how these features can help in modifying the topology of the 3D models.

To identify the handle or tunnel features, we sweep a handle or tunnel loop on both sides over the surface. We run Dijkstra's shortest path algorithm with multiple sources, namely treating all vertices on a handle or tunnel loop as sources. At any moment of the sweep, the modified Dijkstra's algorithm maintains a wave front of shortest distance to the initial loop. The propagation continues until the ratio between the length of the wave front and that of the initial loop exceeds a user-defined threshold. We take the region swept by the two wave fronts on each side as the feature represented by this loop. Figures 1 and 7 show the handle and tunnel features respectively computed with this method.

Figure 7: *From left to right: handle and tunnel loops computed on Casting, tunnel features are identified, small tunnel features are filled.*

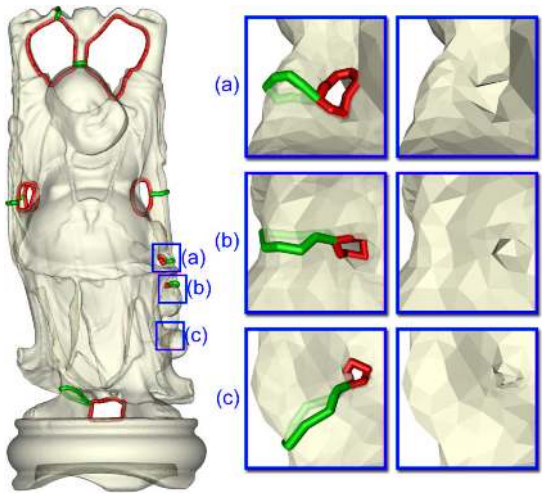


Figure 8: *Three small tunnels in Buddha are filled.*

Once the handle and tunnel features are computed, one can perform simple topology simplifications locally by either cutting a handle feature or filling a tunnel one. This can be easily done with the help of the volume representation of the shape. For example, in the case of a Delaunay surface, all Delaunay tetrahedra tessellate the volume either inside or outside the surface. To fill a tunnel feature, we first detect all the outside tetrahedra that have four vertices on this tunnel feature, then transfer them from outside volume to inside. As a result, a tunnel is filled. Similar operation for handle features eliminates a handle. For the case of iso-surface in a volume data, topology simplification can be achieved in a similar way.

Figure 8 shows the handle and tunnel loops on a *Buddha* model computed with our algorithm. Three small tunnels considered as noises are filled up.

7 More results

Figure 9 shows only the tunnels computed on the scalar volume data *Engine* of size 256×256×256. All 20 tunnels of this model of genus 20 have been detected. The small tunnel in the second closeup view is actually lying *inside* which, if not detected, is difficult to see with usual visualization tools.

Figure 10 (left) shows the handle and tunnel loops for another scalar

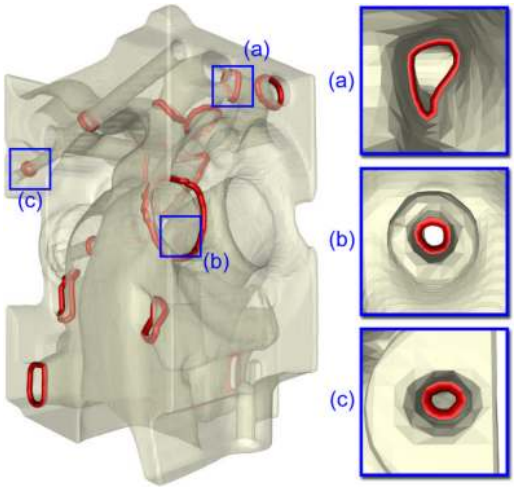


Figure 9: *Closeup views of small tunnels in Engine.*

volume data *Aneurism*. Notice that, although there are many tiny isolated components on this iso-surface, our algorithm has no problem in capturing nice handle and tunnel loops. Figure 10 (right) shows that our algorithm can compute handle and tunnels loops on knotted models which is not possible with the algorithm of [Dey et al. 2007].

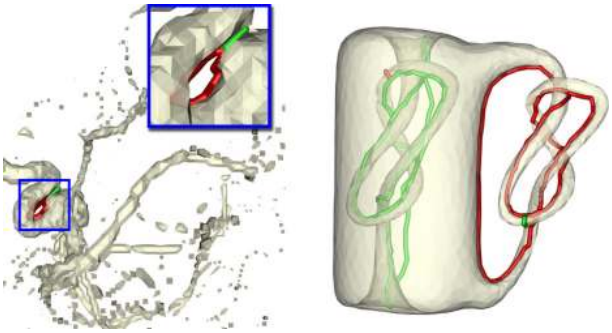


Figure 10: *Aneurism and Knotty Cup.*

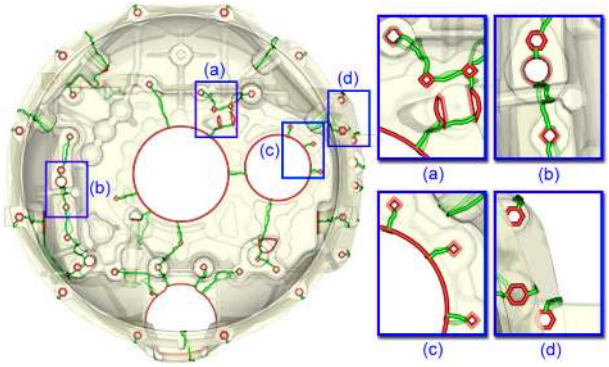


Figure 11: *Gearbox: genus 78 surface meshed with (478K,2582K) Delaunay triangles.*

Figure 11 shows handle and tunnel loops computed by HANTUN in a CAD model of genus 78. We remeshed the original surface with DELPSC to produce about 478K surface triangles and 2.5 million

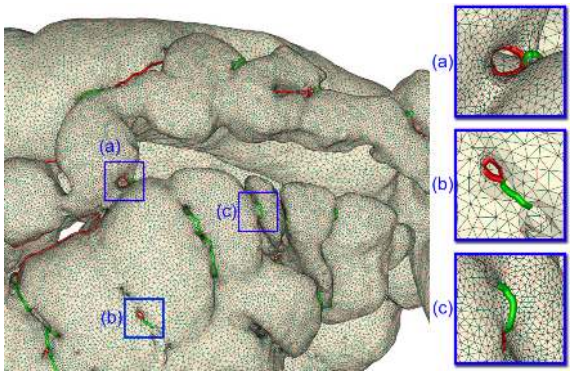


Figure 12: *Colon: genus 160 surface meshed with (854K,4966K) Delaunay triangles. Three closeup views show the small handle and tunnel loops detected.*

volume triangles on which HANTUN was run. Figure 12 shows another large model of genus 160 with about 854K surface and 4.9 million volume triangles. The high genus of this surface is a result of nearby pieces of the surface getting joined as an artifact of the data acquisition.

Our algorithm HANTUN is stable against the granularity of the surface mesh $\mathbb{M}$ and the volume mesh $\mathbb{K}$. Figure 13 exemplifies this point. Handle and tunnel loops shown in the highlighted boxes of the *Hip* model remain mostly stable at different densities of the surface mesh. Similar stability is observed for the *Molecule* under different volume meshing.

8 Conclusions

We have designed a persistence based algorithm to compute well defined handle and tunnel loops for a 3D model. The method derives its simplicity and generality from the algorithm for persistence. The algorithm runs fast and scales well with volume data sets (12 minutes for volume data sets with 2.8 million triangles). With Delaunay surface meshes, the algorithm takes more time as evident from data. One shortcoming of the method is that it requires to convert a surface mesh into a volume mesh which, in some cases, may incur not so negligible pre-processing overhead.

We incorporate geometry by a geodesic measure. Although intuitively it is clear why such a measure gives good loops, it would be nice to have a formal proof for this fact along the line of [Erickson and Whittlesey 2005].

Our algorithm works on any valid 3D model bounded by a closed surface. If the surface has boundaries, the algorithm as described does not work. Since there is no space bounded by such a surface, definitions of handle and tunnel loops become invalid. Also, one cannot partition the simplices as being part of *inside* or *outside* to run the persistence based algorithm. However, the algorithm can be easily modified to handle such surfaces and even simplicial 2-complexes as follows. Assume $\mathbb{M}$, a 2-complex, is a subcomplex of $\mathbb{K}$. After running the pairing algorithm on $\mathbb{M}$, simply add simplices from $\mathbb{K}$ without distinguishing inside and outside. The result of this minor modification is shown in Figure 14. Of course handle and tunnel loops are not differentiated though they are detected even in the presence of boundaries and non-manifolds.

References

ALLIEZ, P., COHEN-STEINER, D., YVINEC, M., AND DESBRUN, M. 2005. Varia-

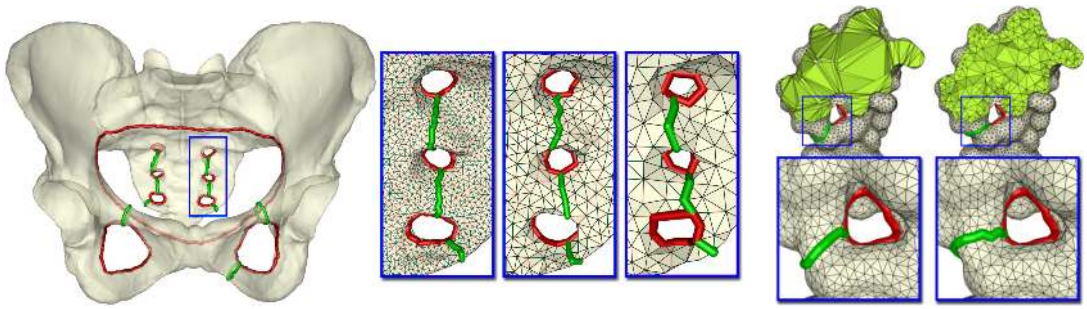


Figure 13: *Hip: Loops computed with different surface mesh densities. Molecule: Volume meshed differently.*

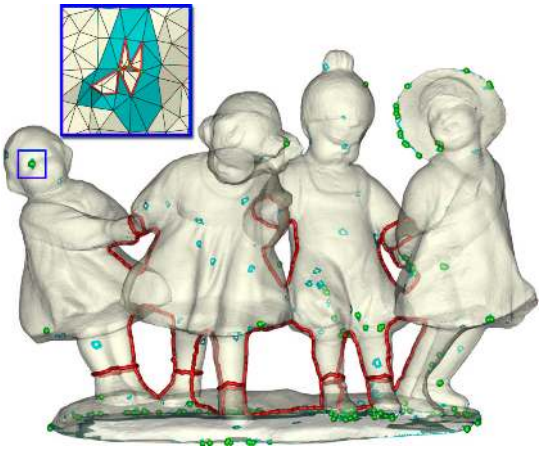


Figure 14: *Children: Delaunay surface mesh reconstructed from point cloud data has artifacts as holes and nonmanifolds.*

tional tetrahedral meshing. In *Proc. SIGGRAPH 2005*, 617–625.

BEN-CHEN, M., GOTSMAN, C., AND BUNIN, G. 2008. Conformal flattening by curvature prescription and metric scaling. *Computer Graphics Forum (Proc. Eurographics) 27*.

BIASOTTI, S., GIORGI, D., SPAGNUOLO, M., AND FALCIDIENO, B. 2008. Reeb graphs for shape analysis and applications. *Theoretical Comput. Sci. 392*, 5–22.

BISCHOFF, S., AND KOBBELT, L. 2005. Structure preserving CAD model repair. *Comput. Graphics Forum 24*, 527–536.

CHEN, C., AND FREEDMAN, D. 2008. Quantifying homology classes. In *Sympos. Theoretical Aspects Comput. Sci.*, 169–180.

CHENG, S.-W., DEY, T. K., AND LEVINE, J. A. 2007. A practical Delaunay meshing algorithm for a large class of domains. In *Proc. 16th Internat. Meshing Roundtable*, 477–494.

COHEN-STEINER, D., EDELSBRUNNER, H., AND MOROZOV, D. 2006. Vines and vineyards by updating persistence in linear time. In *Proc. 22nd Ann. Sympos. Comput. Geom.*, 119–126.

DEY, T. K., AND SUN, J. 2006. Defining and computing curve-skeletons with medial geodesic function. In *Proc. Sympos. Geom. Processing*, 143–152.

DEY, T. K., LI, K., AND SUN, J. 2007. On computing handle and tunnel loops. In *IEEE Proc. Internat. Conf. Cyberworlds*, 357–366.

DEY, T. K. 2007. *Curve and surface reconstruction: algorithms with mathematical analysis*. Cambridge University Press.

EDELSBRUNNER, H., LETSCHER, D., AND ZOMORODIAN, A. 2002. Topological persistence and simplification. *Discrete Comput. Geom. 28*, 511–533.

EL-SANA, J., AND VARSHNEY, A. 1997. Controlled simplification of genus for polygonal models. In *Proc. IEEE Visualization*, 403–412.

ÉRIC COLIN DE VERDIÈRE, AND ERICKSON, J. 2006. Tightening non-simple paths and cycles on surfaces. In *Proc. ACM-SIAM Sympos. Discrete Algorithms*, 192–201.

ÉRIC COLIN DE VERDIÈRE, AND LAZARUS, F. 2005. Optimal system of loops on an orientable surface. *Discrete Comput. Geom. 33*, 507–534.

ERICKSON, J., AND WHITTLESEY, K. 2005. Greedy optimal homotopy and homology generators. In *Proc. ACM-SIAM Sympos. Discrete Algorithms*, 1038–1046.

GU, X., GORTLER, S., AND HOPPE, H. 2002. Geometry images. In *Proc. SIGGRAPH 2002*, 355–361.

GUSKOV, I., AND WOOD, Z. 2001. Topological noise removal. In *Proc. Graphics Interface 2001*, 19–26.

HATCHER, A. 2002. *Algebraic Topology*. Cambridge University Press.

LEVOY, M., PULLI, K., CURLESS, B., RUSINKIEWICZ, S., KOLLER, D., PEREIRA, L., GINZTON, M., ANDERSON, S., DAVIS, J., GINSBERG, J., SHADE, J., AND FULK, D. 2000. The digital Michelangelo project: 3d scanning of large statues. In *Proc. SIGGRAPH 2000*, 131–144.

NI, X., GARLAND, M., AND HART, J. C. 2004. Fair morse functions for extracting the topological structure of a surface mesh. In *Proc. SIGGRAPH 2004*, 613–622.

NOORUDDIN, F. S., AND TURK, G. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE Trans. Vis. Comput. Graph. 9*, 191–205.

OUDOT, S., RINEAU, L., AND YVINEC, M. 2005. Meshing volumes bounded by smooth surfaces. In *Proc. 14th Internat. Meshing Roundtable*, 203–219.

PASCUCCI, V., SCORZELLI, G., BREMER, P.-T., AND MASCARENHAS, A. 2007. Robust on-line computation of Reeb graphs: simplicity and speed. *ACM Trans. Graphics. 58*, 1–9.

SHATTUCK, D. W., AND LEAHY, R. M. 2001. Automated graph-based analysis and correction of cortical volume topology. *IEEE Trans. Med. Imaging 20*, 1167–1177.

SHEFFER, A., AND HART, J. 2002. Seamster: inconspicuous low-distortion texture seam layout. In *Proc. IEEE Visualization*, 291–298.

WOOD, Z., HOPPE, H., DESBRUN, M., AND SCHRÖDER, P. 2004. Removing excess topology from isosurfaces. *ACM Trans. Graphics 23*, 511–533.

YIN, X., JIN, M., AND GU, X. 2007. Computing shortest cycles using universal covering space. *Visual Computer 23*, 999–1004.

ZHOU, Q.-Y., JU, T., AND HU, S.-M. 2007. Topology repair of solid models using skeletons. *IEEE Trans. Vis. Comput. Graph. 13*, 675–685.

ZOMORODIAN, A., AND CARLSSON, G. 2005. Computing persistent homology. *Discrete Comput. Geom. 33*, 249–274.

ZOMORODIAN, A., AND CARLSSON, G. 2007. Localized homology. In *Proc. Shape Modeling Internat.*, 189–198.