An Efficient Computation of Handle and Tunnel Loops via Reeb Graphs*

Tamal K. Dey† The Ohio State University, U.S.A
Fengtao Fan‡ The Ohio State University, U.S.A
Yusu Wang§ The Ohio State University, U.S.A

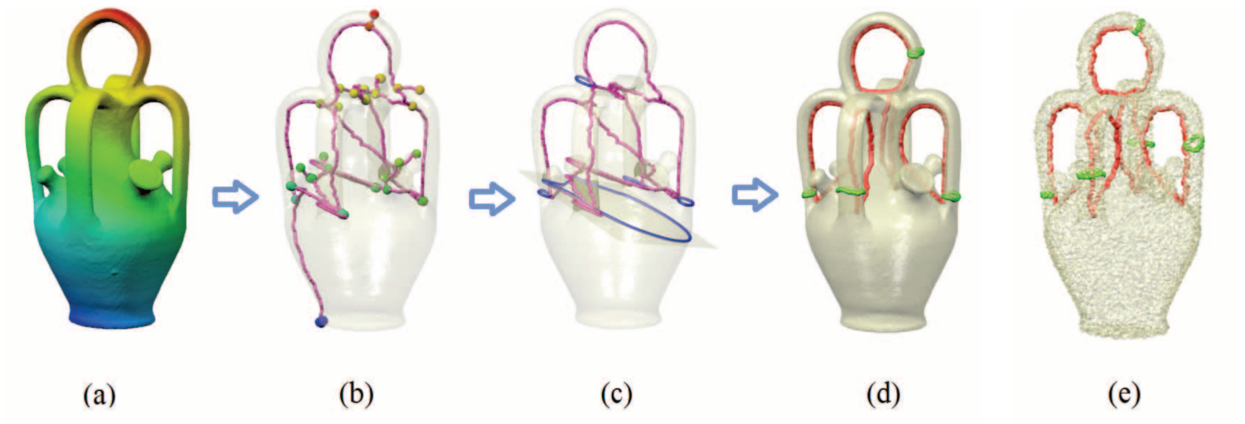


Figure 1: *(a) – (d) shows the pipeline of our algorithm: (a) The height function on the input surface. (b) Reeb graph w.r.t. the height function. (c) Initial handle and tunnel loops. (d) Final handle / tunnel loops after geometric optimization. (e) The output is stable under noise.*

Abstract

A special family of non-trivial loops on a surface called handle and tunnel loops associates closely to geometric features of "handles" and "tunnels" respectively in a 3D model. The identification of these handle and tunnel loops can benefit a broad range of applications from topology simplification / repair, and surface parameterization, to feature and shape recognition. Many of the existing efficient algorithms for computing non-trivial loops cannot be used to compute these special type of loops. The two algorithms known for computing handle and tunnel loops provably have a serious drawback that they both require a tessellation of the interior and exterior spaces bounded by the surface. Computing such a tessellation of three dimensional space around the surface is a non-trivial task and can be quite expensive. Furthermore, such a tessellation may need to refine the surface mesh, thus causing the undesirable side-effect of outputting the loops on an altered surface mesh.

In this paper, we present an efficient algorithm to compute a basis for handle and tunnel loops without requiring any 3D tessellation. This saves time considerably for large meshes making the algorithm scalable while computing the loops on the original input mesh and not on some refined version of it. We use the concept of the Reeb graph which together with several key theoretical insights on linking number provide an initial set of loops that provably constitute a handle and a tunnel basis. We further develop a novel strategy to tighten these handle and tunnel basis loops to make them geometrically relevant. We demonstrate the efficiency and effectiveness of our algorithm as well as show its robustness against noise, and other anomalies in the input.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

*Omitted proofs can be found in a supplementary material available from authors' web-pages

†e-mail:tamaldey@cse.ohio-state.edu

‡e-mail:fanf@cse.ohio-state.edu

§e-mail:yusu@cse.ohio-state.edu

Keywords: geometric processing, surface features, handle and tunnel loops, Reeb graph

Links: DL PDF

1 Introduction

The computation of topologically non-trivial loops on a surface embedded in three dimensions appears as a sub-step in many geometry processing applications. These applications generally benefit greatly if the non-trivial loops encode geometric features such as 'handles' and 'tunnels' of the 3D model. The non-trivial loops that identify such features are of great interest in topology repair [Bischoff and Kobbelt 2005; Zhou et al. 2007]. If these features appear as spurious, it is desirable to eliminate them so that they do no interfere with further processing. In surface parameterization [Ben-Chen et al. 2008; Gu et al. 2002; Sheffer and Hart 2002], the input surface mesh needs to be cut along non-trivial loops into a flat disk. This again benefits from the detection of loops that are small around 'handles' and 'tunnels" because the side-effect of boundary caused by cutting along such small features remains small. Feature recognition and shape correspondence [Biasotti et al. 2008; van Kaick et al. 2010] are key problems in various applications which clearly benefit from localizing the features to loops that are associated with 'handles' and 'tunnels' [Dey et al. 2008].

ACM Reference Format
Dey, T., Fan, F., Wang, Y. 2013. An Efficient Computation of Handle and Tunnel Loops via Reeb Graphs. ACM Trans. Graph. 32, 4, Article 32 (July 2013), 9 pages. DOI = 10.1145/2461912.2462017 http://doi.acm.org/10.1145/2461912.2462017.

The non-triviality of loops can be mathematically characterized by drawing upon the concepts of homology and homotopy [Munkres 1996]. Since these concepts alone do not take into account the surface embedding, they need not capture the geometric features such as handles and tunnels. To address this issue, the authors in [Dey et al. 2007] introduced the definitions of handle and tunnel loops that add the embedding into the definition of the non-triviality. In particular, a handle (tunnel) loop on a closed surface $\mathrm{M} \subset \mathbb{R}^3$ is a loop that is not a boundary of any subset of M but is so for a subset of the interior (exterior respectively) of M. Note that this definition renders many non-trivial loops to be neither a handle nor a tunnel such as the loop γ_3 in Figure 2. Consequently, many of the efficient algorithms for computing non-trivial loops on surfaces cannot be applied to compute handle and tunnel loops.

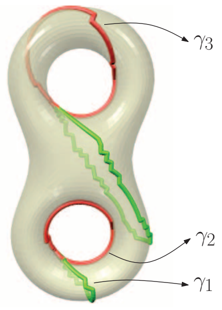


Figure 2: *γ_1 is a handle loop and γ_2 a tunnel loop. γ_3 is neither.*

It is clear that any algorithm for computing handle and tunnel loops has to take into account the embedding of the surface in $\mathbb{R}^3$. The authors in [Dey et al. 2007] achieved this by incorporating the curve skeleton of the interior and exterior spaces bounded by the surface into the algorithm. The computation of these curve skeletons requires a decomposition of the respective spaces by a triangulation or a similar subdivision and also restricts the class of surfaces on which the algorithm can be applied. Specifically, the algorithm could not be applied to the knotted surfaces such as the Knotty Cup in Figure 6. This restriction was removed by the algorithm in [Dey et al. 2008], but the need for triangulating the interior and exterior still persisted. Given a surface mesh in $\mathbb{R}^3$, it is a non-trivial task to generate the tetrahedral mesh of its interior and exterior. Although different algorithms have been discovered recently [Alliez et al. 2005; Cheng et al. 2012], a generic software that can take both smooth and non-smooth surfaces in a time efficient manner is still missing. The only provably correct algorithm that produces tetrahedral meshes out of a given surface mesh that is not necessarily smooth uses a Delaunay refinement technique [Cheng et al. 2012]. It becomes expensive in terms of computational time and memory for large meshes. Moreover, the tetrahedral meshing also changes the input surface mesh which is known to be unavoidable in general [Cheng et al. 2012]; see Figure 3 for an example. This makes the algorithm of [Dey et al. 2008] unusable in many cases. Here, we present a new algorithm for computing handle and tunnel loops on a surface mesh without computing any additional 3D structure such as a tetrahedral mesh. As a result, we eliminate the large preprocessing time of the previous algorithms [Dey et al. 2007; Dey et al. 2008] and also the undesirable side-effects of changing the input surface mesh.

Our contribution. In this paper, we present a simple algorithm to identify a basis for the family of tunnel loops and for the family of handle loops efficiently. Our algorithm leverages the concept of the Reeb graph. The main advantage of our approach is that, the handle and tunnel loops are computed directly from input surface mesh. It does not require any additional structure for either the interior or the exterior of the input surface. Given that the Reeb graph can be computed efficiently [Pascucci et al. 2007; Harvey et al. 2010; Parsa 2012], our algorithm is orders of magnitude faster than the existing algorithms for computing a basis for handle/tunnel loops. After computing a basis for handle and tunnel loops, we tighten them by using shortest path trees that have been used to compute loops in a shortest homology basis [Erickson and Whittlesey 2005; Dey et al. 2011]. We present various examples to demonstrate that our handle/tunnel computation algorithm is efficient; and our geometric optimization strategy is effective in producing short handle/tunnel loops. We also demonstrate that our method is robust against noise and is impervious to sparsity, non-smoothness, and non-uniformity that may be present in the input meshes.

Other previous work. A number of theoretically justified algorithms for computing optimal non-trivial loops on surfaces have been proposed in recent years; see e.g. [Erickson and Whittlesey 2005; Kutz 2006; Colin de Verdière and Erickson 2006]. These algorithms, however, do not guarantee that the loops are handle or tunnel. In the context of topological simplifications, several heuristics have been proposed to identify small handles [El-Sana and Varshney 1997; Guskov and Wood 2001]. Wood et al. [Wood et al. 2004] use Reeb graphs like ours to identify small handles and remove them for topology repair. These algorithms do not have theoretical guarantees and may not compute the handle and tunnel loops as we aim for.

The use of a graph for identifying non-trivial loops on a surface has been proposed earlier. This includes the use of the Reeb graphs in [Shattuck and Leahy 2001] and some graphs based on the medial axis in [Zhou et al. 2007]. None of these algorithms guarantee that the computed loops are handles and tunnels.

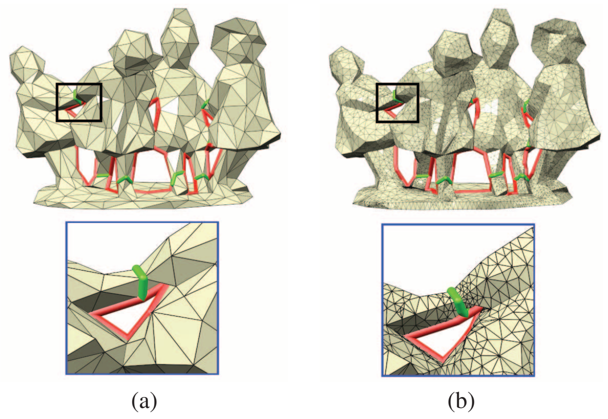


Figure 3: *The output of (a) our algorithm and (b) the algorithm of [Dey et al. 2008] for an input mesh with 449 vertices. Note that due to the tetrahedral meshing, the algorithm of [Dey et al. 2008] changes the input surface mesh and significantly increases its complexity to 7943 vertices. Our algorithm obtained handle and tunnel loops of good quality from the original sparse mesh.*

2 Background

2.1 Handle and tunnel loops

Given a topological space X, let $\mathsf{H}_p(\mathrm{X})$ denote the p-dimensional homology group of X under coefficient ring $\mathbb{Z}_2$; $\mathsf{H}_p(\mathrm{X})$ is a vector space since $\mathbb{Z}_2$ is a field. This paper is concerned only with the 1-dimensional homology group $\mathsf{H}_1(\mathrm{X})$. See [Munkres 1996] for detailed introduction on homology groups.

A *loop* on X is a map $\gamma : \mathbb{S}^1 \to \mathrm{X}$, and we say γ is *simple* if $\gamma(x) \neq \gamma(y)$ for any $x \neq y \in \mathbb{S}^1$. A loop γ is *trivial in* X if γ alone bounds a subset of X; that is, it is the boundary of a subset (a collection of surface patches in case X is a surface) of X. A 1-cycle is a loop or a union of them. Two 1-cycles γ_1 and γ_2 are *homologous*,

```
Input  : A closed triangular mesh $\mathrm{M} \subset \mathbb{R}^3$ with genus $g$;
Output: Handle and tunnel loops of $\mathrm{M}$;
Step 1 : Compute the Reeb graph $\mathrm{Rb_M} = \mathrm{Rb_M}(h)$ for a height function $h$ defined over $\mathrm{M}$;
Step 2 : Compute $g$ independent cycles in $\mathrm{Rb_M}$ by a maximum spanning tree algorithm (described later);
Step 3 : Map the $g$ cycles of $\mathrm{Rb_M}$ back to the surface $\mathrm{M}$. Denoting these cycles as $\gamma_i$ $(i = 1, .., g)$, compute the dual level set loop $\overline{\gamma}_i$
         for every $\gamma_i$ ;
Step 4 : Perturb every $\gamma_i$ and $\overline{\gamma}_i$ inside or outside appropriately to obtain $\alpha_i$ and $\overline{\alpha}_i$ respectively. The loops in $\{\alpha_i\}_{i=1}^g \cup \{\overline{\alpha}_i\}_{i=1}^g$ split
         into two groups, one forming a cycle basis for $\mathsf{H}_1(\mathbb{I})$ and the other for $\mathsf{H}_1(\mathbb{O})$.
Step 5 : Compute the $2g \times 2g$ matrix $L$ whose entries are the linking number of all possible pairs of curves, one from $\{\gamma_i\} \cup \{\overline{\gamma}_i\}$ and
         the other from $\{\alpha_j\} \cup \{\overline{\alpha}_j\}$. Compute a handle basis $\{h_i\}_{i=1}^g$ and a tunnel basis $\{t_i\}_{i=1}^g$ as a linear combinations of $\gamma_i$ and $\overline{\gamma}_i$
         by using $L^{-1}$;
Step 6 : Shorten $h_i$ and $t_i$ to get geometrically relevant loops (Section 3.2);
```

Algorithm 1: Algorithm overview.

if $\gamma_1 + \gamma_2$ is trivial. The homology class $[\gamma]$ corresponds to the family of 1-cycles that are homologous to γ. If γ is trivial, then the homology class $[\gamma]$ is the zero element of $\mathsf{H}_1(\mathrm{X})$. A set of loops are *independent* if their homology classes are independent. A set of loops $\{\gamma_1, \ldots, \gamma_k\}$ is a *cycle basis* for $\mathsf{H}_1(\mathrm{X})$ if these loops are independent and $[\gamma_1], \ldots, [\gamma_k]$ form a basis for the vector space $\mathsf{H}_1(\mathrm{X})$.

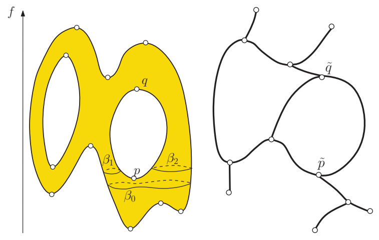
In this paper, the input we consider is a connected closed surface M embedded in $\mathbb{R}^3$. Such a surface M is necessarily orientable, and partitions $\mathbb{R}^3$ into two regions $\mathbb{I}$ and $\mathbb{O}$ such that $\mathbb{I} \cap \mathbb{O} = \mathrm{M}$ and $\mathbb{I} \cup \mathbb{O} = \mathbb{R}^3$ (see e.g, [Guillemin and Pollack 2010]). The unbounded region $\mathbb{O}$ is called *the exterior of* M, while the bounded one $\mathbb{I}$ is *the interior of* M. Note that $\mathbb{I}$ (resp. $\mathbb{O}$) is a 3-manifold with boundary $\partial\mathbb{I} = \mathrm{M}$ (resp. $\partial\mathbb{O} = \mathrm{M}$). The interior of $\mathbb{I}$ (resp. $\mathbb{O}$) is denoted by $\mathring{\mathbb{I}}$ (resp. $\mathring{\mathbb{O}}$).

Following [Dey et al. 2007; Dey et al. 2008], we define the handle and tunnel loops as follows. A loop γ is a *handle loop* if it is trivial in $\mathbb{I}$, but non-trivial in $\mathbb{O}$. Similarly, it is a *tunnel loop* if it is trivial in $\mathbb{O}$ but non-trivial in $\mathbb{I}$. See Figure 2 for examples. It turns out that the space of all loops on M can be split into the spaces of handle and tunnel loops each of which has dimension equal to the genus of the surface. Because of this a non-trivial loop in M cannot be trivial both in $\mathbb{I}$ and $\mathbb{O}$. It follows that a non-trivial loop in M is a handle (tunnel) if it is trivial in $\mathbb{I}$ (respectively $\mathbb{O}$).

Theorem 2.1 (Theorem 1 of [Dey et al. 2007]) *For any connected closed surface $\mathrm{M} \subset \mathbb{S}^3$ of genus g, $\mathsf{H}_1(\mathrm{M})$ is the direct sum of $\mathsf{H}_1(\mathbb{I})$ and $\mathsf{H}_1(\mathbb{O})$ each of which is g dimensional. Moreover, the space of handle loops generates $\mathsf{H}_1(\mathbb{O})$, while the space of tunnel loops generates $\mathsf{H}_1(\mathbb{I})$.*

2.2 Reeb graph

Our handle/tunnel computation algorithm relies on the concept of Reeb graph for a function $f : \mathrm{X} \to \mathbb{R}$ defined on X.

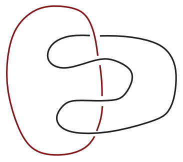


Intuitively, the Reeb graph $\mathrm{Rb_X}(f)$ is obtained by continuously collapsing each contour in the level set into a single point. See the right figure where we plot the Reeb graph such that the height of each point is the value of f on points from its preimage in X. Formally, there is a continuous surjection $\Phi : \mathrm{X} \to \mathrm{Rb_X}(f)$ such that $\Phi(x) = \Phi(y)$ if and only if x and y belong to the same connected component (a *contour*) of a level set $f^{-1}(\alpha)$ for some $\alpha \in \mathbb{R}$. Specifically, imagine we sweep X in increasing value of f. As we pass through a minimum (resp. a maximum) of f in X, a new connected component is created (resp. a contour disappears) in the level set, giving rise to a degree-1 node in $\mathrm{Rb_X}(f)$. As we pass through a *splitting saddle* of f in X (e.g, p in the right figure), a connected component splits into two (e.g, the contour β_0 splits into β_1 and β_2). This gives rise to an *up-fork saddle* in the Reeb graph $\mathrm{Rb_X}$ ($\tilde{p}$ in the figure), which is a degree-3 node in the Reeb graph with up-degree 2. Symmetrically, a *merging saddle* of f in X will give rise to a *down-fork saddle* in the Reeb graph with down-degree 2 (see q and $\tilde{q}$ in the right figure). For a Morse function $f : \mathrm{X} \to \mathbb{R}$, these are all the *non-regular points* of a Reeb graph (empty dots in the figure). All other nodes in the Reeb graph are *regular*, with both up-degree and down-degree being 1.

The Reeb graph is simple, meaningful and can be computed efficiently [Pascucci et al. 2007; Tierny et al. 2009; Doraiswamy and Natarajan 2009; Harvey et al. 2010; Parsa 2012]. It has found a large number of practical applications in computer graphics and visualization; see e.g. the survey [Biasotti et al. 2008] and references therein.

For an orientable compact surface M, it turns out that the Reeb graph $\mathrm{Rb_M}(f)$ encodes valuable information about M. For example, if the surface M has genus g, then there are g independent loops in the Reeb graph $\mathrm{Rb_M}(f)$ for any Morse function $f : \mathrm{M} \to \mathbb{R}$ defined on M [Cole-McLaughlin et al. 2004]. One of the key observations of the current paper is that we can compute a special set of loops using the Reeb graph w.r.t. a specific function (the height function), that can help us untangle the interplay of the interior $\mathbb{I}$ and the exterior $\mathbb{O}$ on the loops of the surface M, and eventually to separate handle and tunnel loops.

2.3 Linking numbers

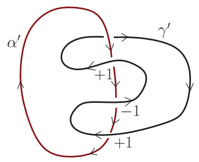


Our algorithm also uses the concept of linking number that helps us to decide whether a loop is a handle or tunnel (or neither). The linking number between two disjoint loops embedded in $\mathbb{R}^3$ gives the information on how one winds around the other. Since we work with $\mathbb{Z}_2$, this number is 1 if one winds around the other odd number of times and zero otherwise. The figure on right shows two loops with a linking number 1. Formally we define:

Definition 2.2 *([Rolfsen 1976]) Let γ and α be two disjoint loops in $\mathbb{R}^3$. Let $[\tau]$ be a generator of $\mathsf{H}_1(\mathbb{R}^3 - \gamma)(\cong \mathbb{Z}_2)$. If $[\alpha] = b[\tau]$, the linking number $\mathrm{Lk}(\gamma, \alpha)$ is defined to be $b \in \{0, 1\}$.*

From this definition, one immediately gets the following which will be useful in checking for handles and tunnels later.

1. $\mathrm{Lk}(\gamma, \alpha) = 0$ if and only if $[\alpha]$ is trivial in $\mathsf{H}_1(\mathbb{R}^3 - \gamma)$;
2. If $[\alpha_1] = [\alpha_2]$ for $\alpha_1, \alpha_2 \in \mathbb{R}^3 - \gamma$, then $\mathrm{Lk}(\gamma, \alpha_1) = \mathrm{Lk}(\gamma, \alpha_2)$.



The linking number can be computed using the following simple procedure (which in fact offers an alternative definition for the linking number) [Rolfsen 1976][1]: Assign an arbitrary orientation to γ and to α. Take any regular projection of $\gamma \cup \alpha$ onto a plane P such that γ' and α', the projection of γ and α in P respectively, intersects transversely. Consider only the set of *positive crossings* C of γ' and α' at which γ crosses *above* α. For each crossing x, assign $sign(x) = 1$ if it is a counter-clockwise turn from the orientation of γ' to that of α'; otherwise, $sign(x) = -1$. See the right figure for an illustration where the top crossing is not a positive crossing. The linking number $\mathrm{Lk}(\gamma, \alpha)$ is the sum, modulo 2, of these $sign(\cdot)$s over all positive crossings of γ' and α'; that is, $\mathrm{Lk}(\gamma, \alpha) = [\ \sum_{x \in C} sign(x)\ ]\ mod\ 2$.

3 Algorithm

Our algorithm contains two parts. The first part is an algorithm which, given a surface mesh M, computes a basis for the handle loops as well as a basis for the tunnel loops. The second part takes these initial basis loops as input, performs further geometric optimization to tighten them and return new basis loops that are also geometrically relevant. Below we describe each part in detail. The overall algorithm is shown in Algorithm 1.

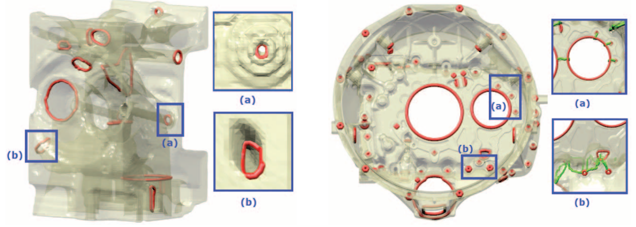


Figure 4: ENGINE *model (left) and* GEARBOX *model (right). Only the tunnel loops are shown to reduce the clutter in display.*

3.1 Handle/Tunnel computation

We elaborate on each step of Algorithm1 justifying them by appropriate theory. We remark that while the correctness of our algorithm requires non-trivial theoretical analysis, the algorithm itself (as described in Algorithm1) is easy to implement.

The input is a triangular mesh M in $\mathbb{R}^3$ whose underlying space $|\mathrm{M}|$ is a closed surface. Suppose that M has genus g and contains n vertices. For the Reeb graph, we use the height function $h : |\mathrm{M}| \to \mathbb{R}$ defined on M where $h(x)$ is the height of x in an arbitrary but fixed direction. We assume that h is a Morse function, which roughly means that no two vertices share the same function value, and no critical point is degenerate. This does not cause loss of generality as the set of directions such that h is not Morse is of measure zero among all possible directions. For this height function h, the level set $h^{-1}(\alpha)$ consists of a set of contours (simple loops) for any generic α, except when α is a critical value where h has a minimum, maximum, or a saddle. We call the height function $h : \mathbb{R}^3 \to \mathbb{R}$ defined on $\mathbb{R}^3 (\supset \mathrm{M})$ the *ambient space height function*.

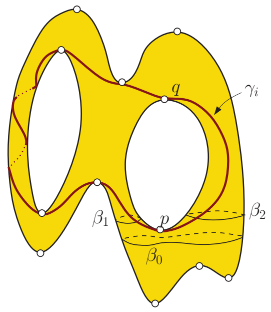
Step 1. The Reeb graph of a function on a surface mesh M can be computed efficiently by a number of algorithms in $O(n \log n)$ time (e.g, [Cole-McLaughlin et al. 2004; Harvey et al. 2010; Parsa 2012]). We compute the Reeb graph Rb_M for the height function h by the algorithm of [Harvey et al. 2010].

Step 2. Given the graph Rb_M, we compute a specific cycle basis of it. It is known that a cycle basis of Rb_M contains g cycles if M is of genus g [Cole-McLaughlin et al. 2004]. We compute a cycle basis of Rb_M with some special properties that help later to compute a basis for $\mathsf{H}_1(\mathbb{I})$ and $\mathsf{H}_1(\mathbb{O})$, and to claim that a certain matrix L obtained from linking numbers is invertible.

For each arc e in Rb_M, we associate a weight $w(e)$ where $w(e)$ is the height value of the lower end-point of e. Next, we compute a maximum-weight spanning tree T of Rb_M based on these edge weights. Each edge e of Rb_M not in T induces *a canonical cycle* c_e in Rb_M, which is the unique cycle formed by e and some tree edges of T. It is known that for any spanning tree T, the set of cycles $\{c_e\}_{e \in \mathrm{Rb}_\mathrm{M} \setminus T}$ form a cycle basis of Rb_M. Since T is a maximum weight spanning tree with edge weights as specified, we have the following property:

- The lower endpoint $\tilde{p}$ of e is the lowest point of the loop c_e, and $\tilde{p}$ is an up-fork saddle of Rb_M.

Step 3. We map back the cycles $\{c_e\}$ detected in the previous step to the surface M. That is, for each Reeb graph loop c_e, we find a pre-image loop in M that is mapped to c_e under the surjection $\Phi : \mathrm{M} \to \mathrm{Rb}_\mathrm{M}$. The mapping can be done by finding a pre-image path for each Reeb arc, and assembling such paths to obtain a loop for each γ_i. Let $\gamma_1, \gamma_2, \ldots, \gamma_g$ denote the resulting pre-image loops in M (for Reeb graph loops $\{c_e\}$) sorted in increasing order of their lowest points (vertices). Since h is a Morse function, the lowest points of these loops are all distinct and have different function values. Furthermore, each lowest point is necessarily a splitting saddle in M.



For every loop γ_i, we compute a dual loop $\overline{\gamma_i}$ in the level set; see the figure on right for an illustration. Take the lowest vertex p of γ_i and consider the level set L_p^+ at height $h(p) + \epsilon$ just above the vertex p for a sufficiently small ϵ. Since p is a splitting saddle for the height function h, there are exactly two contours (simple loops β_1 and β_2) in the level set L_p^+ that intersects γ_i. Either of these two contours can be taken as the dual loop $\overline{\gamma}_i$: For specificity, we take the leftmost one (i.e, the one whose smallest x-coordinate is smaller, which is β_1 in the figure). See also Figure 1 (c) for the set of γ_is (red) and their duals (blue). The loops $\{\gamma_i\}_{i=1}^g$ and $\{\overline{\gamma}_i\}_{i=1}^g$ have the following properties:

L0. $\{[\gamma_i]\}_{i=1}^g \cup \{[\overline{\gamma}_i]\}_{i=1}^g$ is a basis of $\mathsf{H}_1(\mathrm{M})$ (proof in Section 1 of the supplement [2]).

L1. γ_i and $\overline{\gamma}_i$ have exactly one intersection point; and γ_i and $\overline{\gamma}_j$ have at most one intersection point.

[1]The definition and computation of the linking number use coefficient ring $\mathbb{Z}$ in [Rolfsen 1976]. They can be easily extended to $\mathbb{Z}_2$ coefficients as we use.

[2]available from authors' web-pages

L2. All cycles in $\{\overline{\gamma}_i\}$ are disjoint and each of them lies in a different level set. Thus, they are all unlinked.

L3. $\overline{\gamma}_i$ is disjoint and unlinked with γ_j for $j > i$. This is because the lowest point of γ_j is above the level set containing $\overline{\gamma}_i$.

Step 4. In this step we push every loop γ_i and $\overline{\gamma}_j$ either toward inside $\mathbb{I}$ or toward outside $\mathbb{O}$ based on certain simple rules. These perturbed loops constitute the bases of $\mathsf{H}_1(\mathbb{I})$ and $\mathsf{H}_1(\mathbb{O})$. First, we

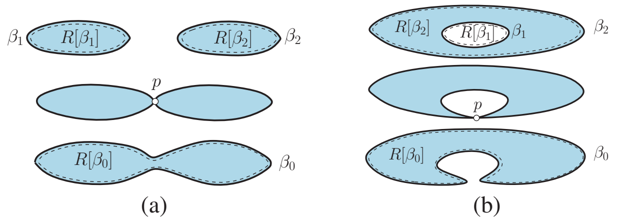


Figure 5: *Dashed loops are pushed version β_i' of β_i. (a) $R[\beta_1]$ and $R[\beta_2]$ are disjoint. (b) $R[\beta_1] \subset R[\beta_2]$.*

make a distinction among the loops γ_i and $\overline{\gamma}_i$ according to the following four configurations. Recall that p is the lowest point of γ_i and is a splitting saddle point on M with respect to the height function h. Let β_0 be the loop, in the level set L_p^- just below $h(p)$, which splits above p. Let β_1 and β_2 be the two new loops in the level set L_p^+ just above $h(p)$. Denote by $R[\beta]$ the closed region bounded by a simple loop β within the corresponding level set for the ambient space height function $h : \mathbb{R}^3 \to \mathbb{R}$ (which is a plane). Now, for each β_i, we push it to the interior of $R[\beta_i]$ by an infinitesimally small offset and obtain another loop β_i', for $i = 0, 1, 2$. See Figure 5. There are four configurations[3] for p:

1. β_1 and β_2 doesn't contain each other (Figure 5 (a)):
   (a) $\beta_0' \subset \mathbb{I}$, which implies that $\beta_i' \subset \mathbb{I}$ for $i = 1, 2$;
   (b) $\beta_0' \subset \mathbb{O}$, which implies that $\beta_i' \subset \mathbb{O}$ for $i = 1, 2$;
2. One is contained in the region bounded by the other, say, $R[\beta_1] \subset R[\beta_2]$ (Figure 5 (b)):
   (a) $\beta_0' \subset \mathbb{I}$, which implies that $\beta_1' \subset \mathbb{O}$ and $\beta_2' \subset \mathbb{I}$;
   (b) $\beta_0' \subset \mathbb{O}$, which implies that $\beta_1' \subset \mathbb{I}$ and $\beta_2' \subset \mathbb{O}$.

Which configuration γ_i is in can be checked locally by inspecting only the one-ring neighborhood $\mathrm{NN}(p)$ of the saddle point p in $O(|\mathrm{NN}(p)|)$ time. We prove (Section 2 of the supplement):

Theorem 3.1 *The loops γ_i and $\overline{\gamma}_i$ are non-trivial in $\mathsf{H}_1(\mathbb{I})$ and $\mathsf{H}_1(\mathbb{O})$ respectively when the lowest point of γ_i admits the configuration (1.a) or (2.b). They are non-trivial in $\mathsf{H}_1(\mathbb{O})$ and $\mathsf{H}_1(\mathbb{I})$ respectively when the configuration is (1.b) or (2.a).*

According to the above theorem, we can divide the loops $\{\gamma_i\}_{i=1}^g$ into two groups: $\{\gamma_{\pi_j}\}_{j=1}^n$ and $\{\gamma_{\pi_j}\}_{j=n+1}^g$ such that $[\gamma_{\pi_j}]$, $1 \le j \le n$ is nontrivial in $\mathsf{H}_1(\mathbb{I})$; and $[\gamma_{\pi_j}]$, $n+1 \le j \le g$ is nontrivial in $\mathsf{H}_1(\mathbb{O})$. By theorem 3.1, we have that the homology classes of $\{\overline{\gamma}_{\pi_j}\}_{j=n+1}^g$ are nontrivial in $\mathsf{H}_1(\mathbb{I})$, and the homology classes of $\{\overline{\gamma}_{\pi_j}\}_{j=1}^n$ are nontrivial in $\mathsf{H}_1(\mathbb{O})$. These give rise to two family of loops: $\Gamma_1 := \{\gamma_{\pi_j}\}_{j=1}^n \cup \{\overline{\gamma}_{\pi_j}\}_{j=n+1}^g$, which are non-trivial in $\mathsf{H}_1(\mathbb{I})$; and $\Gamma_2 := \{\gamma_{\pi_j}\}_{j=n+1}^g \cup \{\overline{\gamma}_{\pi_j}\}_{j=1}^n$, which are non-trivial in $\mathsf{H}_1(\mathbb{O})$. Furthermore, it turns out (from the proof of Theorem 3.2) that loops in Γ_1 (resp. in Γ_2) are independent in $\mathbb{I}$ (resp. in $\mathbb{O}$). Since the dimensions of $\mathsf{H}_1(\mathbb{I})$ and of $\mathsf{H}_1(\mathbb{O})$ are both g (Theorem 2.1), it follows that Γ_1 and Γ_2 form a cycle basis for $\mathsf{H}_1(\mathbb{I})$ and for $\mathsf{H}_1(\mathbb{O})$, respectively.

Note that Theorem 3.1 does not state that a loop in Γ_1 is necessarily trivial in $\mathsf{H}_1(\mathbb{O})$ (see e.g the loop γ_3 in Figure 2, which could belong to Γ_1). Hence, loops in Γ_1, while forming a basis for $\mathsf{H}_1(\mathbb{I})$, *may not* yet form a basis for the tunnel loops. Similarly, loops in Γ_2 are not necessarily handles. To compute handle / tunnel loops, we will need (in Step 5) to compute a linking number matrix whose entries are the linking numbers between basis elements of $\mathsf{H}_1(\mathbb{I}) \oplus \mathsf{H}_1(\mathbb{O})$ and the loops $\{\gamma_i\} \cup \{\overline{\gamma}_i\}$. The definition of the linking number requires that the two participating curves be disjoint (while loops in $\{\gamma_i\} \cup \{\overline{\gamma}_i\}$ may intersect).

To this end, we perturb the loops γ_is and $\overline{\gamma}_i$s to get a new set of basis cycles for $\mathsf{H}_1(\mathbb{I})$ and $\mathsf{H}_1(\mathbb{O})$ which are disjoint from the loops with which we compute the linking number.

Specifically, we push each loop in Γ_1 (which is non-trivial in $\mathsf{H}_1(\mathbb{I})$) toward the interior $\mathring{\mathbb{I}}$ of $\mathbb{I}$. Symmetrically, we push each loop in Γ_2 (which is non-trivial in $\mathsf{H}_1(\mathbb{O})$) toward the interior $\mathring{\mathbb{O}}$ of the exterior $\mathbb{O}$. The loops in Γ_1 (resp. in Γ_2) remain homologous to their perturbed version in $\mathbb{I}$ (resp. $\mathbb{O}$). The loop $\overline{\gamma}_i$ is only perturbed in its level set. Let the perturbed loops of $\{\gamma_i\}_{i=1}^g$ and $\{\overline{\gamma}_i\}_{i=1}^g$ be $\{\alpha_i\}_{i=1}^g$ and $\{\overline{\alpha}_i\}_{i=1}^g$, respectively. Let A_1 and A_2 denote the set of perturbed loops for Γ_1 and Γ_2, respectively. Note that all loops in A_1 are now contained in $\mathring{\mathbb{I}}$, while all loops in A_2 are in $\mathring{\mathbb{O}}$. Hence, they are disjoint from loops in $\{\gamma_i\} \cup \{\overline{\gamma}_i\}$ which are contained in M.

Theorem 3.2 *(i) $A_1 = \{\alpha_{\pi_j}\}_{j=1}^n \cup \{\overline{\alpha}_{\pi_j}\}_{j=n+1}^g$ is a cycle basis of $\mathsf{H}_1(\mathbb{I})$. $A_2 = \{\alpha_{\pi_j}\}_{j=n+1}^g \cup \{\overline{\alpha}_{\pi_j}\}_{j=1}^n$ is a cycle basis of $\mathsf{H}_1(\mathbb{O})$.*

(ii) Let D denote the $g \times g$ matrix of linking numbers where $D[i][j]$ is the linking number of the i-th loop from A_1 with the j-th loop from A_2. Then the matrix D is non-singular.

Not every cycle basis for $\mathsf{H}_1(\mathbb{I})$ and $\mathsf{H}_1(\mathbb{O})$ satisfies claim (ii) above. This special property, which will be important to derive our main result, holds for A_1 and A_2 due to Property (L1), (L2) and (L3) of γ_is and $\overline{\gamma}_i$s stated earlier in (**Step 3**). See Section 3 of the supplement for the proof of this theorem.

Later in (Step 5), we would need to compute the linking number between a loop from $\{\gamma_i\} \cup \{\overline{\gamma}_i\}$ and a loop from $\{\alpha_i\} \cup \{\overline{\alpha}_i\}$, say $\mathrm{Lk}(\gamma_i, \alpha_j)$. We remark that in our implementation, we *do not* physically perturb the entire curve γ_j or $\overline{\gamma}_j$ to get α_j or $\overline{\alpha}_j$. Rather, we only perturb each loop *locally* around its intersections with the loop that we compute linking number with. See Section 1 of the supplement for the implementation details.

Step 5. The identification of the handle and tunnel loops relies on the following key lemma, whose proof (in Section 4 of the supplement) uses Theorem 3.2 (i) and (ii), as well as the observations below Definition 2.2.

Lemma 3.3 *If a loop ℓ has zero linking number with every loop in A_1, but has a non-zero linking number with at least one loop in A_2, then ℓ is a tunnel loop. Symmetrically, if ℓ has zero linking number with every loop in A_2, but has a non-zero linking number with at least one loop in A_1, then ℓ is a handle loop.*

To compute loops satisfying the conditions in the above claim, we compute a matrix L of linking numbers to be defined shortly. First, consider the loops $\gamma_1, \gamma_2, \ldots, \gamma_g$ on M enumerated in increasing order of the height values of their lowest points. The

[3]We note that each region $R[\beta_i]$ may not be completely contained in $\mathbb{I}$ nor $\mathbb{O}$, since there could be other contours in the level set, not drawn in Figure 5, that are contained within these regions. , we use β_i' to help distinguish different configurations.

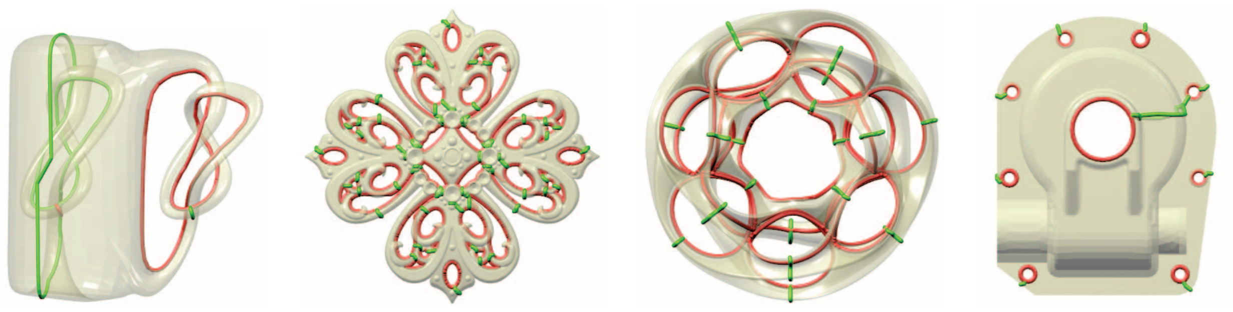


Figure 6: *Various examples. From left to right:* KNOTTY-CUP, FILIGREE, HEPTOROID *and* CASTING.

Properties (L0)- (L3) of these loops induce similar properties for their perturbed versions $\alpha_1, \alpha_2, \ldots, \alpha_g$ and their dual counterparts $\overline{\alpha_1}, \overline{\alpha_2}, \ldots, \overline{\alpha_g}$. In particular, we have:

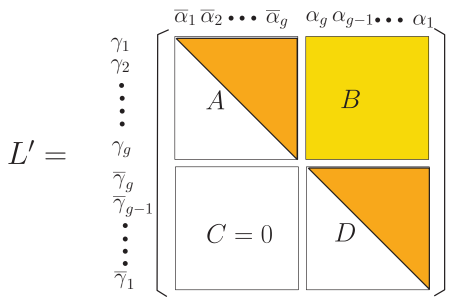
P0. Loops in $\{\gamma_i\}_{i=1}^g \cup \{\overline{\gamma}_i\}_{i=1}^g$ are disjoint from loops in $\{\alpha_i\}_{i=1}^g \cup \{\overline{\alpha}_i\}_{i=1}^g$.

P1. For $i = 1, \ldots, g$, $\mathrm{Lk}(\gamma_i, \overline{\alpha}_i) = \mathrm{Lk}(\overline{\gamma}_i, \alpha_i) = 1$.

P2. Every $\overline{\alpha}_i$ is in a different level set and hence all $\overline{\alpha}_i$s are mutually unlinked. They also do not link with any loop in $\{\overline{\gamma}_j\}$.

P3. If $i > j$, then $\overline{\alpha}_j$ is unlinked with α_i because the lowest point of α_i is above the level set that contains $\overline{\alpha}_j$.

To describe the entries of the linking number matrix L, write $\gamma_{g+i} = \overline{\gamma}_i$ and $\alpha_{g+i} = \overline{\alpha}_i$. Then $L = \{l_{ij}\}$ where $l_{ij} = \mathrm{Lk}(\gamma_i, \alpha_j)$, for $i, j \in [1, 2g]$. It turns out that L is nonsingular and hence invertible. To see why this is the case, consider the matrix L' shown below, which is obtained by permuting some columns and rows of L. Hence, L is non-singular *if and only if* L' is non-singular.


$L' =$
$\overline{\alpha}_1\ \overline{\alpha}_2 \cdots \overline{\alpha}_g$
$\alpha_g\ \alpha_{g-1} \cdots \alpha_1$
γ_1
γ_2
γ_g
$\overline{\gamma}_g$
$\overline{\gamma}_{g-1}$
$\overline{\gamma}_1$
A
B
$C = 0$
D

The matrix L' consists of four submatrices as shown above, where sub-matrices A and D are both upper-triangular by Property (P3) and submatrix $C = 0$ by Property (P2). Thus, L' is upper-triangular. Furthermore, all diagonal elements of A and D (and thus of L') are non-zero due to Property (P1). Hence, L' is non-singular, implying that L is also non-singular.

Since L is nonsingular, one can take its inverse $L^{-1} = \{l'_{ij}\}$ in the field $\mathbb{Z}_2$. We take $2g$ new loops on M as linear combinations of $\{\gamma_i\}$ according to L^{-1}, that is, the set of loops given by $\{\beta_i := \sum_{j=1}^{2g} l'_{ij}\gamma_j\}_{i=1}^{2g}$. In particular, consider the set of $\{\beta_k\}$ where each k is a column index in L' that corresponds to a loop in A_1. This set forms a basis for the tunnel loops, and we denote them by $\{t_i\}_{i=1}^g$. Symmetrically, the set of β_is whose indices correspond to those in A_2 forms a basis for the handle loops, and we denote these loops by $\{h_i\}_{i=1}^g$. To see that loops $\{h_i\}$ and $\{t_i\}$ are indeed handle and tunnel loops, note that the new linking number matrix formed by these loops with α_is is $L^{-1}L = I$. Hence, by Lemma 3.3, each h_i is a handle and each t_i is a tunnel, for $i = 1, \ldots, g$.

Remark. A shortest handle or tunnel loop may contain multiple connected components. In our algorithm, while each loop γ_i (and $\overline{\gamma}_j$) is connected, each h_i (or t_j) may contain multiple components (loops) due to the matrix inversion in (Step 5). However, such occurrences seem rare and we have not observed it in our experiments.

3.2 Tightening the loops

We have now obtained a set of g independent handle loops $\{h_i\}_{i=1}^g$ and a set of g independent tunnel loops $\{t_i\}_{i=1}^g$. However, they may not have desirable geometry; specifically, they may be unnecessarily long and winding. See Figure 1 (c). In this step, we wish to tighten these loops so that they "hug" the handle and tunnels more closely as in Figure 1 (d).

Ideally, we wish to obtain the shortest basis for handle loops (resp. for tunnel loops) whose total length is the smallest among all basis for handles (resp. tunnels). A polynomial time greedy algorithm to compute the shortest cycle basis for $\mathsf{H}_1(\mathrm{M})$ exists [Erickson and Whittlesey 2005]. However, loops in a shortest cycle basis may be neither handles nor tunnels, and it is not clear how to solve this optimization problem in polynomial time when only constrained to the set of handle (or tunnel) loops. Instead, we develop an iterative approach that is both simple and effective in practice. We describe it for the handles, and the case for tunnel loops is similar.

It is observed in [Dey et al. 2011] that any loop ℓ in the shortest cycle basis for $\mathsf{H}_1(\mathrm{M})$ is necessarily *canonical* in the sense that it consists of an edge $e = (u, v)$ concatenated with two shortest paths $\pi(u, w)$ and $\pi(v, w)$ from u and v to another vertex w in the mesh M. Using ideas from [Erickson and Whittlesey 2005], the following approach is proposed in [Dey et al. 2011] to compute a shortest cycle basis for $\mathsf{H}_1(\mathrm{M})$ (in fact, the algorithm works for any simplicial complex): (a) first enumerate all canonical loops on M, ordering them by their lengths, and (b) then find the first $2g$ independent loops greedily.

We modify the above approach in two ways: First, in step (b), instead of finding the first $2g$ independent loops, we find the first g independent *handle loops* from the set of canonical loops. To test whether a canonical loop ℓ is a handle loop or not, we check whether $[\ell]$ can be written as a linear combination of handle loops from the basis $\{h_i\}_{i=1}^g$ that we already computed. This checking is done by the procedure described in [Busaryev et al. 2012] using the so-called *annotation* for edges in M.

Second, to improve efficiency, in step (a), instead of using all canonical loops (which involves computing n shortest path trees each rooted at a vertex of M), we use an iterative framework. At each iteration in step (a), we use only a subset C of canonical loops. Specifically, instead of computing a shortest path tree at each of the n vertices of M, we compute them only at $O(g)$ number of *base points* at each iteration. In the first iteration, the set of base points is taken as the set of lowest points $\{p_1, \ldots, p_g\}$ of all loops in $\{\gamma_i\}$. For subsequent k-th iterations, $k > 1$, assume we have already computed a basis for handle loops $\mathcal{B}^{(k-1)}$ in the previous iteration. We now pick a constant number of points (the constant is set to be 2 in our implementation) randomly from each handle loop in the current basis $\mathcal{B}^{(k-1)}$ as base points. Once these $O(g)$ base points are selected, we build a shortest path tree T_i rooted at each base point p_i. Now consider T_i: any edge $e \notin T_i$ forms a unique canonical loop by concatenating e with the two shortest paths from each of its endpoint to the root p_i. The collection of such canonical loops for all T_is constitute C. We also insert loops in h_i to C to guarantee that a basis for handles can always be identified from C. Finally, we sort loops in C by their lengths and feed them to step (b).

We iterate steps (a) and (b) for a few rounds till the lengths of the output handle loops converge or we reach a maximum number of iterations. The algorithm is summarized in Algorithm 2. In all our experiments, only a small number of iterations, usually less than 100, are needed for the total lengths of the output handle loops to converge.

```
Input  : A set of initial handles $\{h_i\}_{i=1}^g$ and tunnels $\{t_i\}_{i=1}^g$;
Output: A set of new basis $\{h_i^*\}_{i=1}^g$ for handle loops;

Set current basis of handle loops $\mathcal{B}^{(0)} = \{h_i^{(0)} := h_i\}_{i=1}^g$;
Set $k = 0$ ;
repeat
    (a).  Compute a set of canonical loops $C$ from the shortest
    path trees for $O(g)$ base points, and sort them by length ;
    (b).  Compute the first $g$ independent handle loops $\mathcal{B}^{(k+1)}$
    from $C$ using annotations [Busaryev et al. 2012] ;
until (total length of $\mathcal{B}^{(k)}$ converge) or ($k \geq \mathrm{MaxItNum}$);
Output the current basis $\mathcal{B}^{(k+1)}$ for handle loops;
```

Algorithm 2: Algorithm for tightening the handle loops.

3.3 Time complexity analysis

Given a triangulation of n vertices modeling a compact orientable surface of genus g, (Step 1) takes $O(n \log n)$ time to compute the Reeb graph, and (Step 2) takes $O(n \log n + gn)$ time to construct the maximum spanning tree and report the set of g basis cycles in the Reeb graph. (Step 3) takes time linear in the total complexity of all the level sets passing through the saddle points of h, which in the worst case can be $O(n^2)$, but in practice is much smaller. (Step 4) and (Step 5) takes $O(g^2n^2)$ time to compute the linking number matrix L, $O(g^3)$ time to compute the inverse matrix L^{-1}, and $O(g^2n)$ time to construct the new handle and tunnel basis $\{h_i\}_{i=1}^g$ and $\{t_i\}_{i=1}^g$. Overall, the first part of the algorithm takes $O(g^2n^2)$ time. We observe in practice that this part is very efficient and takes time linear in n, instead of quadratic in n. Indeed, as we see in Table 1, for all our tested examples, the first part ran much faster than the second part of the algorithm.

For the second part of the algorithm, first observe that, since M is a surface mesh, it needs only $O(gn)$ time to compute the annotation for edges in M [Erickson and Nayyeri 2011]. These annotations associate each edge e with a size g vector $\mathsf{a}(e)$ [Busaryev et al. 2012]. Each iteration of Algorithm 2 takes $O(gn \log n)$ time to compute $O(g)$ number of shortest path trees. Each shortest path tree produces at most n canonical loops, which can be computed and annotated in $O(gn)$ time (see [Busaryev et al. 2012]). Altogether, we have $O(gn)$ number of canonical loops in C whose annotations take $O(g^2n)$ time altogether. Once these annotations are computed, checking for independence takes $O(g^2)$ time for each loop [Busaryev et al. 2012] and $O(g^3n)$ time for all loops in C. Finally, assume that our algorithm performs k number of iterations. The overall time complexity for Algorithm 2 is thus $O(gnk \log n + g^3nk)$. Note that, k is only a small constant for all of our tested examples.

In contrast, we note that the algorithm of [Dey et al. 2008] takes $O(N^3)$ time incurred by the persistence algorithm, where N is the size of the tetrahedral mesh after tessellating the space around the input surface mesh (which has only $O(n)$ complexity). In the worst case, N could be $\Omega(n^2)$.

4 Experiments

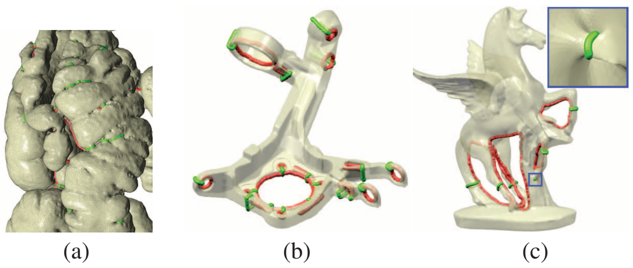


Figure 7: *More examples. (a)* COLON, *(b)* FUSEE, *(c)* PEGASUS.

We have run our algorithms on a variety of models, and some examples are shown in Figure 4, 6 and 7, which include smooth surfaces, isosurface, knotted surface, and CAD models (containing sharp features). It also handles surfaces with high genus well, see Figure 4 and Figure 7(a) for a portion of the COLON model (genus 160). The sizes of these models can be found in Table 1.

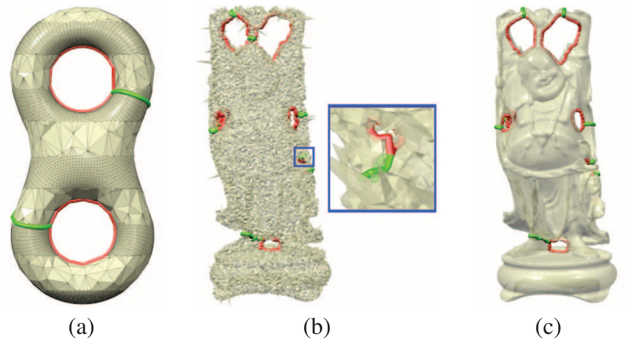


Figure 8: *(a) Non-uniform and (b) noisy meshes. (c) Handle/tunnels from a clean* BUDDHA *model.*

Comparisons. In Table 1, we show the timing of main components of our algorithm, as well as that for the previous state-of-the-art handle/tunnel loop identification algorithm of [Dey et al. 2008]. For the algorithm of [Dey et al. 2008], we distinguish between their preprocessing time to construct tetrahedral mesh (using the DelPSC algorithm [Cheng et al. 2012]) and their handle/tunnel loop computation algorithm after the tetrahedral mesh is computed.

Note that our handle/tunnel basis computation (Column "Reeb graph" + Column "Step 2–5") is several orders of magnitude faster than the previous approach. This is achieved without the need for

Model detail			Our Algorithm Timing (sec)				[Dey et al. 2008] (sec)	
Name	Size (#Ver, #Tri),	Genus	Reeb Graph	Step 2–5	Tightening	Total	Pre-process	Loop compl
KNOTTY-CUP	$(5.4K, 10.8K)$	2	0.03	0.03	1.11	1.17	12.1	3.01
CASTING	$(20K, 40.8K)$	9	0.23	0.08	1.7	2.01	99.8	13.7
BOTIJO	$(33.7K, 67.4K)$	5	0.52	0.2	2.08	2.8	166.1	40.1
BUDDHA	$(54K, 108K)$	9	0.78	0.12	4.96	5.86	697.3	Fail
FUSEE	$(121K, 243K)$	18	2.9	0.37	40.19	43.46	1713.5	559.7
GEARBOX	$(238K, 477K)$	78	4.64	53.95	340.64	399.23	N/A	
HEPTOROID	$(287K, 573K)$	22	4.04	3.06	118.27	125.37	8797.1	2980.0
COLON	$(427K, 854K)$	160	6.38	39.47	2790.41	2836.26	N/A	
FILIGREE	$(514K, 1.03M)$	65	79.97	25.17	559.12	664.26	N/A	

Table 1: *Timing results in seconds. "Fail" means that the algorithm crashes on the input, while "N/A" means that the algorithm has not finished after running for 10 hours.*

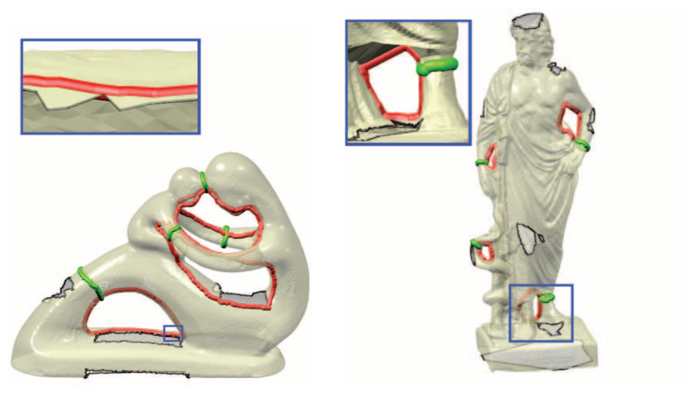


Figure 9: *Meshes with boundary.*

constructing any volumetric mesh, which itself can be non-trivial, and could also require the input surface mesh change in order to obtain volume meshes that are compatible to it.

The step which tightens the loops typically takes more time, especially when the genus g of the input surface is large. However, the speed of our loop-tightening step still compares more favorably with the previous approach even when we ignore its pre-processing time, which is typically order of magnitude larger than its loop-computation step.

We remark that since the preprocessing of [Dey et al. 2008] may increase the size of the input mesh, for fair comparison of the computational time, we choose parameters of DelPSC algorithm so that the output surface mesh has a size at most that of our input. Furthermore, our algorithm is very robust w.r.t. sparse meshes. Indeed, the DANCING CHILDREN model in Figure 3 (a) shows very little degradation of loop quality even though the mesh has only 459 vertices (the algorithm runs in $0.07sec$). We have performed our algorithm for denser versions of this model with $180K$ and $724K$ vertices, respectively. The running times are $24.6sec$ and $179.1sec$, respectively. The loops produced remain stable as the mesh becomes coarser.

Imperfect input. To demonstrate that our algorithm is robust with respect to diverse variations and potential defects of input models, we consider the following scenarios: (i) surface with non-uniform meshing, (ii) surface with noise, and (iii) surface with small holes. Figure 8 (a) shows a case of non-uniform sampling. The handles / tunnels produced by a noisy BOTIJO model is already given in Figure 1. Another example is shown in Figure 8 (b), where we also give a few vertices large perturbation to have some "outliers". We note that computing tetrahedral mesh is even harder for noisy surfaces.

Finally, for surfaces with boundary, the handle / tunnel loops are not well-defined. Furthermore, the boundary loops can be knotted disallowing sealing them with disks. Currently, we consider only small disk-sealable holes and follow a simple heuristic. We seal each hole by putting an extra vertex v at the centroid of the vertices on its boundary and connecting v to the edges on boundary. We put large weights for these extra edges so that optimized handle and tunnel loops on the sealed surface do not contain these extra edges. This approach works for simple holes, as shown in Figure 9. More sophisticated approach is necessary to handle more complex holes, which we leave for future research.

5 Limitation and Discussion

In this work, we presented an algorithm that can compute a basis for handle and tunnel loops on an input surface mesh. The main advantage of this new algorithm is that, unlike the previous approaches, we do not need any expensive tetrahedral mesh of a 3D space which may also change the input surface as a result. The method is robust against noise and can sustain anomalies such as non-uniform sampling, non-smoothness, and small simple holes.

There are two main limitations worth investigating further. The first one is how to handle boundaries in input (orientable) surfaces. Handles / tunnels are not well-defined for surfaces with boundaries. Indeed, one can seal the holes in an orientable surface in different ways changing the interior/exterior of the resulting surface and thereby changing the classification of a loop being handle or tunnel or neither. Furthermore, sealing holes while preserving the embedding of the resulting closed surface is not a trivial task itself. It would be interesting to explore whether it is possible to obtain a set of handles / tunnels for a specific valid sealing sequence, but without explicitly performing the sealing operation.

The second limitation of the method is that, though we compute a handle/tunnel basis with a theoretical guarantee, we cannot produce a shortest basis with such a guarantee: our algorithm currently uses a heuristic to tighten the initial set of handle/tunnel loops. Although the algorithms for producing a shortest basis for H_1 homology (i.e, for non-null homologous loops) exist [Erickson and Whittlesey 2005; Dey et al. 2011], the same approaches do not directly lead to shortest system of handle/tunnel loops. Specifically, computing shortest handle/tunnel loops requires tightening a set of cycles within a subgroup of H_1 homology group. This gives it a flavor similar to the problem of computing an optimal cycle within a given homology class (under $\mathbb{Z}_2$ coefficients), which is a NP-hard problem in general (see [Chambers et al. 2009; Cabello et al. 2011]). Furthermore, the polynomial-time algorithm of computing shortest basis for H_1 homology relies on the fact that each cycle in this basis is "tight", meaning that it contains a shortest path between every pairs of points in it. This characterization unfortunately does not hold for the shortest cycle within a fixed homology class, nor for shortest handle / tunnel loops. Hence, it is possible that it is NP-hard to compute a shortest basis for handle/tunnel loops. It would be interesting to settle this question formally, that is, either to prove that the problem is indeed NP-hard and possibly come up with an even better heuristic, or design an optimal algorithm if that is not the case.

Acknowledgments

The authors thank reviewers for insightful comments and AIM@SHAPE for making models available. This work is partially supported by NSF grants CCF 1064416 and 1116258.

References

ALLIEZ, P., COHEN-STEINER, D., YVINEC, M., AND DESBRUN, M. 2005. Variational 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. *Theor. Comput. Sci. 392*, 1-3, 5–22.

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

BUSARYEV, O., CABELLO, S., CHEN, C., DEY, T. K., AND WANG, Y. 2012. Annotating simplicies with a homology basis and its applications. In *SWAT*, 189–200.

CABELLO, S., COLIN DE VERDIÈRE, É., AND LAZARUS, F. 2011. Finding cycles with topological properties in embedded graphs. *SIAM J. Discret. Math. 25*, 4, 1600–1614.

CHAMBERS, E. W., ERICKSON, J., AND NAYYERI, A. 2009. Minimum cuts and shortest homologous cycles. In *Proc. ACM Sympos. Comput. Geom.*, 377–385.

CHENG, S.-W., DEY, T. K., AND SHEWCHUK, J. R. 2012. *Delaunay Mesh Generation*. CRC Press, Boca Raton, Florida.

COLE-MCLAUGHLIN, K., EDELSBRUNNER, H., HARER, J., NATARAJAN, V., AND PASCUCCI, V. 2004. Loops in reeb graphs of 2-manifolds. *Discrete Comput. Geom. 32*, 231–244.

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

DEY, T. K., LI, K., AND SUN, J. 2007. On computing handle and tunnel loops. In *Proceedings of the 2007 International Conference on Cyberworlds*, CW '07, 357–366.

DEY, T. K., LI, K., SUN, J., AND COHEN-STEINER, D. 2008. Computing geometry-aware handle and tunnel loops in 3d models. In *ACM SIGGRAPH'08*, 45:1–45:9.

DEY, T. K., SUN, J., AND WANG, Y. 2011. Approximating cycles in a shortest basis of the first homology group from point data. *Inverse Problems 27*.

DORAISWAMY, H., AND NATARAJAN, V. 2009. Efficient algorithms for computing Reeb graphs. *Computational Geometry: Theory and Applications 42*, 606–616.

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

ERICKSON, J., AND NAYYERI, A. 2011. Minimum cuts and shortest non-separating cycles via homology covers. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, 1166–1176.

ERICKSON, J., AND WHITTLESEY, K. 2005. Greedy optimal homotopy and homology generators. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1038–1046.

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

GUILLEMIN, V., AND POLLACK, A. 2010. *Differential Topology*. American Mathematical Society.

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

HARVEY, W., WANG, Y., AND WENGER, R. 2010. A randomized O(m logm) time algorithm for computing reeb graph of arbitrary simplicial complexes. In *Proc. 26th. Annu. Sympos. Comput. Geom.*, 267–276.

KUTZ, M. 2006. Computing shortest non-trivial cycles on orientable surfaces of bounded genus in almost linear time. In *Proc. 22nd Annu. Sympos. Comput. Geom.*, 430–438.

MUNKRES, J. R. 1996. *Elements of Algebraic Topology*. Westview Press.

PARSA, S. 2012. A deterministic $O(mlogm)$ time algorithm for the Reeb graph. In *ACM Sympos. Comput. Geom. (SoCG)*, 269–276.

PASCUCCI, V., SCORZELLI, G., BREMER, P.-T., AND MASCARENHAS, A. 2007. Robust on-line computation of Reeb graphs: simplicity and speed. *ACM Trans. Graph. 26*, 3, 58.

ROLFSEN, D. 1976. *Knots and Links*. Publish or Perish.

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.

TIERNY, J., GYULASSY, A., SIMON, E., AND PASCUCCI, V. 2009. Loop surgery for volumetric meshes: Reeb graphs reduced to contour trees. *IEEE Trans. Vis. Comput. Graph. 15*, 6, 1177–1184.

VAN KAICK, O., ZHANG, H., HAMARNEH, G., AND COHEN-OR, D. 2010. A survey on shape correspondence. In *Proc. of Eurographics State-of-the-art Report*, 1–22.

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

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.