Loop surgery for volumetric meshes: Reeb graphs reduced to contour trees

Julien Tierny, Attila Gyulassy, Eddie Simon, Valerio Pascucci

Loop Surgery for Volumetric Meshes: Reeb Graphs Reduced to Contour Trees

Julien Tierny, Attila Gyulassy, Eddie Simon, and Valerio Pascucci, *Member, IEEE*

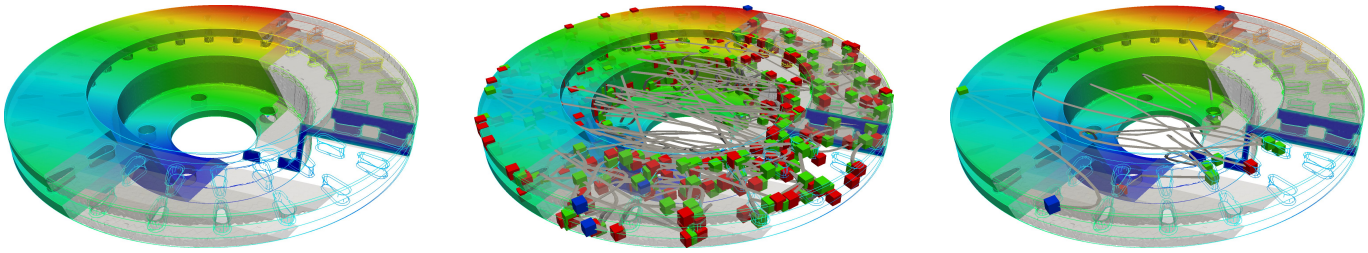


Fig. 1. The Reeb graph of a pressure stress function on the volumetric mesh of a brake disk is shown at several scales of hypervolume-based simplification. At the finest resolution of this data-set (3.5 million tetrahedra), our approach computes the Reeb graph in 7.8 seconds while the fastest previous techniques [19, 12] do not produce a result.

Abstract—This paper introduces an efficient algorithm for computing the Reeb graph of a scalar function f defined on a volumetric mesh M in $\mathbb{R}^3$. We introduce a procedure called *loop surgery* that transforms M into a mesh M' by a sequence of cuts and guarantees the Reeb graph of $f(M')$ to be loop free. Therefore, loop surgery reduces Reeb graph computation to the simpler problem of computing a contour tree, for which well-known algorithms exist that are theoretically efficient ($O(n \log n)$) and fast in practice. Inverse cuts reconstruct the loops removed at the beginning.

The time complexity of our algorithm is that of a contour tree computation plus a loop surgery overhead, which depends on the number of handles of the mesh. Our systematic experiments confirm that for real-life volumetric data, this overhead is comparable to the computation of the contour tree, demonstrating virtually linear scalability on meshes ranging from 70 thousand to 3.5 million tetrahedra. Performance numbers show that our algorithm, although restricted to volumetric data, has an average speedup factor of 6,500 over the previous fastest techniques, handling larger and more complex data-sets.

We demonstrate the versatility of our approach by extending fast topologically clean isosurface extraction to non-simply connected domains. We apply this technique in the context of pressure analysis for mechanical design. In this case, our technique produces results in matter of seconds even for the largest models. For the same models, previous Reeb graph techniques do not produce a result.

Index Terms—Reeb graph, scalar field topology, isosurfaces, topological simplification.

1 Introduction

As scientific data becomes larger and more complex, sophisticated techniques are required for its effective analysis and visualization. Topology-based methods are particularly useful in this context due to their ability to capture features directly, and interact with them at multiple resolutions. Reeb graphs are an efficient solution that encodes the behavior of level sets of scalar functions defined on manifold meshes of arbitrary topology, and therefore their efficient computation is an important challenge. In this paper, we present a new algorithm that computes Reeb graphs in an efficient manner, to enable in-practice use of the Reeb graph for various visualization and analysis tasks.

Topology-based techniques are becoming more common to solve complex visualization and data analysis challenges. In particular, the contour tree [10] is now commonly recognized as an efficient solution to capture the structure of scalar fields. Thanks to simple, robust and fast algorithms [23, 7], it has a wide spectrum of applications, including seed-set computation for fast isosurface extraction [24], topologically clean isosurface extraction [8], feature-extraction [4], feature-driven visualization metaphors [26], and automated transfer function design [27]. Algorithms for computing the contour tree [23, 7] require the scalar-valued data to be defined on a simply connected domain (a very restrictive topological constraint), but many scientific experiments and computer-aided design simulations do not meet this requirement. Then, the more general notion of Reeb graph [21] is needed. Only a few algorithms exist for computing Reeb graphs on volumetric meshes, and they are computationally expensive and therefore not practically applicable for use in interactive applications.

This paper presents an algorithm to compute Reeb graphs on volumetric meshes in $\mathbb{R}^3$ (in particular tetrahedral meshes) that runs in practice with the same efficiency as a contour-tree algorithm, enabling the practical generalization of contour tree based visualization techniques to meshes of arbitrary topology. Our approach is based on the key concept of *loop surgery*, inspired from surgery theory [25]. In particular, we transform the input domain by a sequence of symbolic cuts such that the Reeb graph of the input scalar field defined on the transformed domain is guaranteed to be loop free, and hence computable with efficient contour tree algorithms. Then, some *inverse symbolic cuts* are performed in the topology domain to convert the computed contour tree into the Reeb graph of the original scalar field. We show that these *symbolic cuts* can be computed in an efficient manner, with reasonable computation overhead with respect to contour tree computation. Extensive experiments show that for volumetric meshes our approach is orders of magnitude faster than state-of-the-art techniques, while maintaining a smaller memory footprint, as shown by the largest experiments. To demonstrate the potential of our approach, we picked a contour tree based visualization technique and extended it to volumetric meshes of arbitrary topology while still maintaining its interactive appeal. In particular, we implement an algorithm for fast topologically-clean isosurface extraction, and apply it in the context of pressure analysis for mechanical design.

- *Julien Tierny, Attila Gyulassy and Valerio Pascucci are with the Scientific Computing and Imaging Institute, University of Utah, E-mails: {jtierny, jediati, pascucci}@sci.utah.edu.*
- *Eddie Simon is with Dassault Systèmes, E-mail: eddie.simon@3ds.com.*

Manuscript received 31 March 2009; accepted 27 July 2009; posted online 11 October 2009; mailed on 5 October 2009.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org .

(a) $M, f(M)$ (b) $\mathscr{R}(f_\partial)$ (c) Cutting surfaces (d) $M', f'(M')$ (e) $\mathscr{R}(f')$

Fig. 2. Overview of Reeb graph computation based on loop surgery: The input defines a scalar function f (color gradient) on M (a). The Reeb graph of the function restricted to the boundary $\mathscr{R}(f_\partial)$ (b) is used to identify *loop saddles* (blue squares). *Cutting surfaces* (c) of each loop saddle are used to transform the domain (d) to M'. The Reeb graph $\mathscr{R}(f')$ is loop free (e). *Inverse cuts* are applied to circled critical point pairs to obtain the Reeb graph $\mathscr{R}(f)$ of the input function.

1.1 Related work

A preliminary contour tree algorithm was introduced by de Berg and van Kreveld [10]. Extending the previous work by Tarasov and Vyalyi [23], Carr et al. [7] presented a very elegant, simple, robust and fast algorithm for contour tree computation on simply connected simplicial complexes of arbitrary dimension, hereafter called the *join-split* algorithm. This approach implicitly exploits the fact that, over simply connected domains, the topological evolution of the connected components of the sub- and sur-level sets of the input field captures all the information about that of the connected components of level sets. Consequently, level-set connectivity tracking is efficiently achieved by only keeping track of the connectivity of the sub- and sur- level sets with a computationally inexpensive Union-Find data structure. The $O(n\log n + N\alpha(N))$ complexity of the join-split algorithm can be proven to be optimal for computing contour trees. The relationship between the topology of the sub- and sur- level sets and that of level sets no longer holds on domains of arbitrary topology. This remark motivates research for efficient Reeb graph computation algorithms.

Approximation algorithms exist for computing Reeb graphs based on image domain regular partitioning [5, 16, 28], but they can lead to inaccurate results due to dependence on sampling rates or thresholds. The first combinatorial algorithm for Reeb graph computation on PL 2-manifolds was introduced by Shinagawa and Kunii [22], with an $O(n_t^2)$ time complexity where n_t stands for the number of triangles in the mesh. Cole-McLaughlin et al. [9] introduced an optimal algorithm for PL 2-manifolds, with $O(n_e \log(n_e))$ time complexity (where n_e is the number of edges) by taking advantage of the mono-dimensionality of level sets in the special case of PL 2-manifolds by maintaining them dynamically in a sorted representation. Since it can be computed efficiently for PL 2-manifolds, the Reeb graph has been a very popular shape abstraction for several computer graphics tasks, such as surface parameterization [29], shape retrieval [16], character animation [3], and others. We defer the reader to [6] for a comprehensive survey on the applications of Reeb graphs in shape modeling.

Many properties of two-dimensional domains exploited by the above algorithms do not hold for three-dimensional domains, making the problem more challenging. There are only a few techniques for the computation of Reeb graphs in higher dimensions and thus usable for volumetric meshes. Pascucci et al. [19], presented a streaming approach taking as an input streamed meshes of arbitrary dimension. Due to its intrinsic ability to handle non-manifold meshes, the algorithm computes Reeb graphs by considering the restriction of the input field to the 2-skeleton of the mesh, whose Reeb graph is proved to equal that of the original field. However, isosurfaces have to be maintained, and since no natural order is possible for two-dimensional level sets (unlike the case of triangular meshes [9]), the algorithm exhibits a quadratic worse case time complexity. Doraiswamy and Natarajan [13] proposed an extension of Cole-McLaughlin et al.'s approach [9] to three-dimensions by dynamically maintaining isosurfaces with a complex data structure based on a spanning tree, resulting in a time complexity of $O(n_t \log^2 n_t)$, where n_t is the number of triangles in the mesh. They showed that an implementation with a lighter data structure performed better in practice despite a complexity of $O(ng\log^2 n)$, where g is the maximum genus of an isosurface. The same authors also introduced recently another algorithm [12] with improved practical performance, which extends a previous approach [20] from surface to tetrahedral meshes. In [12], the authors adapt image domain regular partitioning techniques by partitioning the domain precisely along critical level sets and deducing the Reeb graph from the domain partition. With practical fields studied in scientific visualization or computer-aided design simulations, the number of critical points is often linear with the number of vertices in the mesh, and the size of the partition boundaries can also be proportional to the number of tetrahedra, resulting rapidly in a quadratic run-time behavior in practice.

1.2 Contributions

This paper makes the following new contributions:

1. We introduce a new procedure called *loop surgery* to reduce the problem of computing a Reeb graph to that of a contour tree. We believe this is an important result, since the join-split algorithm [7] for computing contour trees is well known to have not only optimal theoretical complexity, but also simple and practical implementation.
2. We describe a practical algorithm for computing Reeb graphs with complexity $O(n\log n + gN)$. For practical examples, g, which is equal to the number of handles of the domain, is a small constant, and systematic experiments show a speedup over previous algorithms by several orders of magnitude on average.
3. We provide a proof showing necessary and sufficient conditions for a loop free Reeb graph to be computed correctly by the join-split contour tree algorithm.
4. We extend topologically-clean isosurface extraction to volumetric meshes of arbitrary topology, and apply it in the context of mechanical design, where it enables a direct visualization of the major trends of physical simulations.

2 Preliminary Results

In this section, we briefly describe the formal setting of our approach and present some preliminary results on the topology of Reeb graphs on tetrahedral meshes. We defer the reader to [17] and [15] for reference books on Morse theory and homology.

2.1 Background and definitions

Consider a real-valued function $f : M \to \mathbb{R}$ defined on a manifold M. One fundamental way to study the function f is to extract its level sets. For a given scalar w the level set $L(w)$ is defined as the inverse image of w onto M through f, $L(w) = f^{-1}(w)$. We call each connected component of the level set $L(w)$ a *contour*. A *sub-level set* is defined as the inverse image of the open interval $(-\infty, w)$ onto M through f, $L^-(w) = \{x \in M | f(x) < w\}$. Symmetrically, a *sur-level set* is the inverse image of the open interval (w, ∞) onto M through f, $L^+(w) = \{x \in M | f(x) > w\}$.

One aspect that is well understood in Morse theory [17] is the evolution of the homology classes of the contours of f while w changes continuously in $\mathbb{R}$. The points at which the topology of a contour changes are called *critical points* and the corresponding function values are called critical values. If all of the critical points of f are non-degenerate and have distinct values, then f is a *Morse function*.

A *retraction* is defined [15] as a continuous map such that its image is a subset of the domain and the restriction of the map to the image is the identity. We define a *contour retraction* of a manifold M under a function f to be a continuous map that retracts each contour (connected component of level set) to a single point. Notice that, by continuity, adjacent contours are retracted to adjacent points; distinct contours are retracted to distinct points. This gives the definition:

The *Reeb graph* $\mathscr{R}(f)$ is the *contour retract* of M under f.

The Reeb graph is represented as a graph consisting of *nodes* and *arcs*. Branching in $\mathscr{R}(f)$ occurs only at critical values of f. Figure 2(b) shows a simple scalar function with the associated Reeb graph. A Reeb graph is called a *contour tree* when it has no loops. For example, this is true when M is simply connected.

Note that f can be decomposed into $f = \psi \circ \phi$, where $\phi : M \to \mathscr{R}(f)$ is a contour retraction that maps points in the same contour of f to the same point of $\mathscr{R}(f)$, and $\psi : \mathscr{R}(f) \to \mathbb{R}$ is a continuous function that maps points of $\mathscr{R}(f)$ to a scalar value in $\mathbb{R}$.

Consider the case where the domain of $f : M \to \mathbb{R}$ is a simplicial complex with boundary, with n vertices and N simplices. Within each simplex of M, the function f is the linear interpolation of its values at the vertices, and we say that f is a piecewise-linear (PL) function. The following operations on a simplicial complex M are used to identify critical points. The *star* of a simplex u is the set of simplices that contain u as a face: $St(u) = \{\sigma \in M | u \leq \sigma\}$, where $u \leq \sigma$ denotes that u is a face of σ. The *link* of the simplex u is the set of simplices in the closure of the star of u that are not also in its star, *i.e.*, $Lk(u) = \overline{St}(u) - St(u)$, where $\overline{St}$ denotes the closure of the star. In PL functions, critical points can only occur at vertices. The *lower link* of u is the subset of the link containing only simplices with all their vertices lower in function value than u: $Lk^-(u) = \{\sigma \in Lk(u) | \, v \leq \sigma \to f(v) < f(u)\}$. Symmetrically, the *upper link* is $Lk^+(u) = \{\sigma \in Lk(u) | \, v \leq \sigma \to f(u) < f(v)\}$. Similarly, the lower star of u is $St^-(u) = \{\sigma \in St(u) | \, v \leq \sigma \to f(v) < f(u)\}$. A vertex u in M is *regular* if and only if both $Lk^-(u)$ and $Lk^+(u)$ are simply connected, otherwise u is a critical point of f (notice that both must be simply connected since M has a boundary).

2.2 Loops in Reeb graphs on PL 3-manifolds in $\mathbb{R}^3$

Consider a scalar function f defined on a PL 3-manifold mesh M in $\mathbb{R}^3$. The key idea of *loop surgery* is to define a sequence of operations that transform M to M' with $f' : M' \to \mathbb{R}$ valued by f such that $\mathscr{R}(f')$ becomes loop-free, and then efficiently computable with contour tree algorithms. Therefore, we carefully characterize the loops (independent cycles) of $\mathscr{R}(f)$ prior to introducing loop surgery.

Let ∂M be the boundary of M. Since M is compact and embeddable in $\mathbb{R}^3$, ∂M is necessarily non-empty, orientable and closed [11] but possibly disconnected.

Let f_∂ be the restriction of f to ∂M. We will first assume that both f and f_∂ are PL Morse functions (degenerate cases will be discussed later). Let $\mathscr{R}(f_\partial)$ be the Reeb graph of f_∂. Then, we have the following relation [9]:

$$\#loops(\mathscr{R}(f_\partial)) = g \tag{1}$$

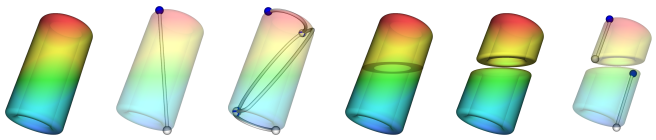
where g is the sum of the genera of the boundary components of M. The key property that will allow us to implement loop surgery in an efficient manner is the fact than the topology of M is closely related to that of ∂M. In particular, the number of handles of M is the first Betti number, $\beta_1(M)$, which is given by the following relation [11]:

$$\beta_1(M) = g \tag{2}$$

where $\beta_1(M)$ is the number of independent 1-cycles in M, formally the rank of the first homology group of M. In simpler words, this relation expresses the fact that each handle of the volume M corresponds to a tunnel of its boundary surface.

As discussed in [9] in any dimension the construction of the Reeb graph can lead to the removal of 1-cycles, but not to the creation of new ones. Therefore, the number of loops of $\mathscr{R}(f)$ cannot be greater than the first Betti number of M :

$$\#loops(\mathscr{R}(f)) \leq \beta_1(M) \tag{3}$$



(a) $f(M)$ (b) $\mathscr{R}(f)$ (c) $\mathscr{R}(f_\partial)$ (d) $\mathscr{S}_i$ (e) M' (f) $\mathscr{R}(f')$

Fig. 3. Example where the domain (a) is not simply connected and $\mathscr{R}(f)$ is loop free (b). The Reeb graph of the boundary, $\mathscr{R}(f_\partial)$, has a loop (c), therefore loop surgery is performed (d), (e). Even after loop surgery, each component of M' is not necessarily simply connected, but the Reeb graph of each component is loop free (f).

In conclusion, the number of loops of $\mathscr{R}(f)$ cannot be greater than the number of loops of $\mathscr{R}(f_\partial)$:

$$\#loops(\mathscr{R}(f)) \leq \beta_1(M) = g = \#loops(\mathscr{R}(f_\partial)) \tag{4}$$

A direct extension of the equation 4 is the following lemma:

Lemma 1. *The existence of loops in $\mathscr{R}(f)$ implies the existence of corresponding tunnels in both M and ∂M, and thus of corresponding loops in $\mathscr{R}(f_\partial)$. The inverse is not necessarily true.*

A result of this lemma is that one can deduce information about the loops of $\mathscr{R}(f)$ by just studying $\mathscr{R}(f_\partial)$.

2.3 Loop surgery

When f_∂ is PL Morse, there exists a unique pair of saddle points for each loop, consisting of an "opening" split saddle and a "closing" join saddle. The existence of such pairs is guaranteed by extended persistence [2]. We uniquely associate each loop with the closing saddle of this pair, and call that saddle a *loop saddle*. Moreover, by lemma 1, each loop in $\mathscr{R}(f)$ can also be associated with the same loop saddle as the corresponding loop in $\mathscr{R}(f_\partial)$. Notice that some loop saddles of $\mathscr{R}(f_\partial)$ may not be associated with any loop of $\mathscr{R}(f)$.

Loop surgery consists of transforming the domain M such that $\mathscr{R}(f)$ becomes loop-free. In other words, loop surgery breaks the loops of $\mathscr{R}(f)$ and reflects that transformation on M. Since we have an injection from the loops of $\mathscr{R}(f)$ to the set of loop saddles, it is sufficient to reason only with the loop saddles to achieve these transformations. In particular, for each loop saddle s_i with value $f(s_i)$, we define its *cutting surface* $\mathscr{S}_i$ as a contour of f (a connected component of isosurface inside the volume) at value $f(s_i) - \varepsilon$ such that $f(s_i) - \varepsilon$ is a regular value of f, there is no critical value in $[f(s_i) - \varepsilon, f(s_i))$ and $\mathscr{S}_i$ intersects one of the connected components of the lower star $St^-(s_i)$ in the volume. The symbolic cuts transforming M into M' consist of cutting M along each defined cutting surface, as illustrated in figures 2(c) and 2(d).

On the topology domain, cutting along a cutting surface at a regular value is equivalent to cutting an arc of $\mathscr{R}(f)$ and creating a new pair of critical nodes (a minimum and a maximum, as illustrated in figure 2(e)). Since we have an injection from the set of loops of $\mathscr{R}(f)$ to the set of cutting surfaces, the Reeb graph of the function $f' : M' \to \mathbb{R}$, f' being the function valued by f after symbolic cuts, is guaranteed to be loop free: all the possible loops have indeed been broken, as shown in figure 2(e).

Once the loop-free Reeb graph $\mathscr{R}(f')$ is computed, *inverse cuts* can be applied in a straightforward manner by removing pairs of minimum and maximum nodes generated by the same cutting surface, and gluing together the corresponding arcs.

2.4 Loop free Reeb graph computation

The algorithm presented by Carr et al. [7] computes the contour tree by tracking the joining of sub- and sur-level set components. Traditionally, simply-connectedness of M has been used as a condition to ensure correctness of this algorithm. However, a Reeb graph can be loop free even when the domain is not simply connected, as shown in figure 3(b). This is especially important, since our loop surgery procedure does not guarantee that the domain is divided into simply connected regions. Therefore, we prove necessary and sufficient conditions for this contour tree algorithm to work. The following lemma shows when a saddle creates a loop in the Reeb graph.

Lemma 2. *Let $f : \mathbb{M} \to \mathbb{R}$ be a Morse function and n be a degree-3 node of $\mathscr{R}(f)$ corresponding to a join saddle s (see figure 4). If the contours joined by the saddle are on the boundary of the same sub-level set component, then there exists a loop in $\mathscr{R}(f)$ for which n is the highest saddle.*

Proof. Refer to figure 4 in the following. Given n, s, f, and $\mathscr{R}(f)$, let ε be a small number such that there is no critical value in the range $[f(s)-\varepsilon, f(s))$. The existence of such an epsilon is guaranteed because the critical values of a Morse function are distinct. Let $w = f(s)-\varepsilon$, and a and b be the points on the two downward arcs from n such that the value of their corresponding contours c_a and c_b is w, *i.e.*, $f(c_a) = f(c_b) = w$. By construction, n is a join saddle, therefore there exists a path p^+ in M connecting a point in c_a with a point in c_b such that all its interior is in the sur-level set $L^+(w)$. The map $\phi : M \to \mathscr{R}(f)$, mapping points in the same contour in M to the same point in $\mathscr{R}(f)$, is a continuous surjection (by definition of a contour retraction), therefore a connected component p^+ in M is mapped to a connected component $\phi(p^+)$ in $\mathscr{R}(f)$, such that a and b are connected in $\mathscr{R}(f)$ by a path $q^+ \subseteq \phi(p^+)$ whose interior is strictly above w. Since by hypothesis c_a and c_b are on the boundary of the same sub-level set component, there exists also a path p^- in M connecting a point in c_a with a point in c_b such that all its interior is in the sub-level set $L^-(w)$. Therefore a and b are also connected in $\mathscr{R}(f)$ by a path $q^- \subseteq \phi(p^-)$ whose interior is strictly below w. The two paths $q^+ + q^-$ form a loop. □

By applying lemma 2, we prove necessary and sufficient conditions for computing correct loop free Reeb graphs using tracking of sub- and sur-level sets.

Lemma 3. *Let $f : \mathbb{M} \to \mathbb{R}$ be a Morse function. Its Reeb graph $\mathscr{R}(f)$ is loop free if and only if every degree-3 node in $\mathscr{R}(f)$ corresponds to a saddle of f where distinct components of sub- or sur-level sets join.*

Proof. First we prove that if a Reeb graph $\mathscr{R}(f)$ is loop free, every degree-3 node n corresponds to a saddle s of f that joins distinct components of sub- or sur-level sets. Assume that there exists a degree-3 node that joins contours that are on the boundary of the same sub- or sur-level set component. By lemma 2, there must be a loop in $\mathscr{R}(f)$. This contradicts the hypothesis statement that $\mathscr{R}(f)$ is loop free.

Next, we prove that if every degree-3 node corresponding to a saddle in $\mathscr{R}(f)$ joins distinct sub- or sur-level set components, then $\mathscr{R}(f)$ is loop free. Assume for contradiction that $\mathscr{R}(f)$ has a loop, s is the highest join saddle in the loop, and n the corresponding degree-3 node. Pick ε, w, a, b, c_a, and c_b as before. The paths q^+ and q^- exist in $\mathscr{R}(f)$ connecting a and b in a loop. Since ϕ is a contour retraction, any connected component of $\mathscr{R}(f)$ is a connected subset of M. Therefore $\phi^{-1}(q^-)$ is a connected component in M, connecting c_a to c_b in $L^-(w)$. Therefore, c_a and c_b are on the boundary of the same connected component of $L^-(w)$, which contradicts the hypothesis. The same argument holds for split saddles. □

Lemma 3 shows that we can use the contour tree algorithm presented by Carr et al. [7] to compute loop free Reeb graphs, by tracking the connectivity evolution of sub- and sur- level sets. Performing loop surgery guarantees that $\mathscr{R}(f')$ is loop free, and then computable with the contour tree algorithm.

3 Algorithm

Our algorithm for Reeb graph computation is summarized by the following pseudocode:

Algorithm 1. Reeb Graph Overview

```
INPUT: f, M
(0) M' = M, GL = {}
(1) if genus_diagnostic(∂M) > 0 then
(2)     S∂ = find_loop_saddles(M, ∂M, f)
(3)     for each s in S∂
(4)         M' = cut(M', cutting_surface(s))
(5)         GL = GL ∪ node_pairs(s)
(6) RG = contour_tree(M')
(7) for each p in GL
(8)     RG = glue(RG, p)
(9) return RG
```

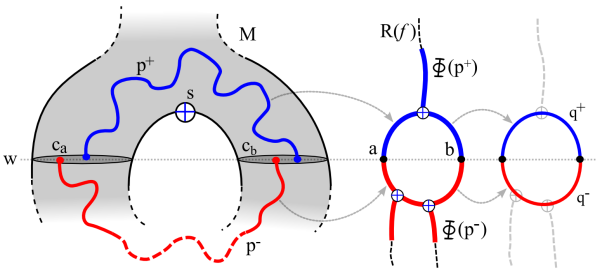


Fig. 4. Reference figure for lemma 2. Paths forming a loop in M map to connected components forming a loop in $\mathscr{R}(f)$.

The input is a PL function `f`, and a simplicial mesh `M`. In lines (1-5), the domain is symbolically cut to ensure that its Reeb graph is loop free. If a diagnostic (line 1) shows that the domain has no handles (section 3.1.1), then no loop surgery needs to be done. Otherwise, loop saddles (section 3.1.2) are detected (line 2), and the domain is symbolically cut (line 4) along cutting surfaces (section 3.1.3). We keep track of the extra pairs of nodes created in the Reeb graph by each cut (line 5). Since the Reeb graph is guaranteed to be loop free, we compute it (line 6) using a modified version of the join-split algorithm (section 3.2). Finally, the loop free Reeb graph is transformed into the correct Reeb graph of the input function by inverse cuts (lines 7-8) (section 3.3).

3.1 Loop surgery

The purpose of *loop surgery* is to symbolically cut the domain such that $\mathscr{R}(f')$ is guaranteed to be loop free. This procedure corresponds to lines 1-5 of algorithm 1.

3.1.1 Genus diagnostic

We first check if any loop surgery is needed by checking the presence of tunnels on the boundary ∂M. This implements the `genus_diagnostic` function in line 1 of algorithm 1. In particular, we use the Euler formula on each connected component of ∂M:

$$\chi = 2 - 2g = n_v^\partial - n_e^\partial + n_f^\partial \tag{5}$$

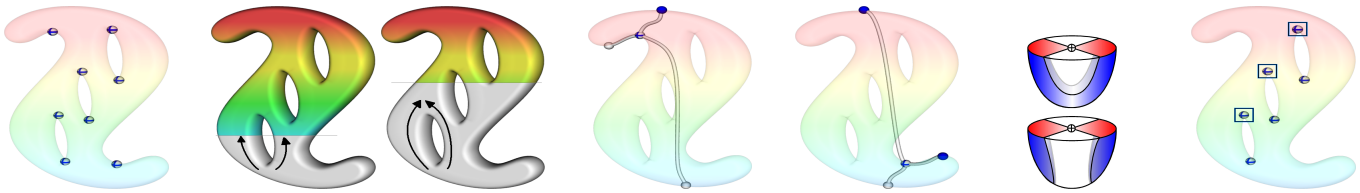
where n_v^∂, n_e^∂ and n_f^∂ stand for the numbers of vertices, edges and triangles of the considered component of ∂M. The sum of the genera g of the connected components of ∂M then gives the number of tunnels in ∂M and hence the number of cuts needed to ensure that $\mathscr{R}(f')$ is loop free. Loop surgery is needed only if the sum of the genera is non-zero.

3.1.2 Loop saddles

If the domain is cut at every loop saddle (defined in section 2.3) then $\mathscr{R}(f')$ is loop free. In this section, we implement `find_loop_saddles` in line 2 of algorithm 1. One technique for finding loop saddles is computing the Reeb graph of the boundary (using an existing technique such as [19]), and then identifying the loop saddles using extended persistence [2]. However, the benefits of using a simpler technique for finding a small superset of loop saddles outweigh the cost of performing additional cuts.

By lemma 1, we know that loop saddles must exist as a subset of the saddles on the boundary, therefore, we carefully select these from the set of all saddles of f_∂ in a three-step process: *(i)* all saddles of f_∂ are identified; *(ii)* we apply lemma 3 and remove from these the ones that join distinct sub- or sur-level set components; and *(iii)*, we further remove those that do not join the same sub-level set component in the volume. The rules we employ resolve degenerate saddles implicitly. These steps are explained in detail below.

(i) Saddles of f_∂ occur at vertices of ∂M. A vertex $x \in \partial M$ is a saddle if and only if the number of connected components of its lower link is greater than one. This number can be computed by a simple link traversal technique [9]. The set of all the saddles on the boundary is denoted S_∂.



(a) $\partial M, f_\partial(\partial M)$ (b) Sublevel sets of f_∂ (c) Join tree of f_∂ (d) Split tree of f_∂ (e) $Lk^-(s)$ (f) Loop saddles

Fig. 5. Identifying loop saddles: f_∂ and its saddle points (a). Loops are "absorbed" by sub-level sets of f_∂ (b). Join tree (c) and split tree (d) of f_∂ identify non-loop saddles. Configurations of the lower link (e) in the volume distinguish between opening and closing of loops. Loop saddles (blue squares) are returned (f).

(ii) We remove saddles of S_∂ that do not open or close a loop. To use the result of lemma 2 in determining whether or not the saddle can be part of a loop, we identify the sub- and sur-level set associated with each connected component of the lower and upper link of a saddle of S_∂. The classification of sub- and sur- level set components is computed by constructing the join tree and the split tree in the join-split contour tree algorithm [7]. The join sweep that builds the join tree processes vertices of ∂M in order of increasing function value, maintaining sub-level sets via a Union-Find data structure. In the computed join tree of f_∂, the number of downward arcs of each node is equal to the number of distinct sub-level set components of f_∂ that are joined at the corresponding vertex. Similarly, a split sweep builds the split tree by processing vertices of ∂M from highest to lowest. The resulting split tree provides the information about the evolution of sur-level set components. When the number of downward arcs in the join tree equals the number of connected components of the lower link or the number of upward arcs in the split tree equals the number of connected components of the upper link, the saddle does not open or close a loop in $\mathscr{R}(f_\partial)$. These saddles are removed from S_∂. In practice, this step removes 87% of the saddles of S_∂ on average.

(iii) The set S_∂ now contains exactly those saddles that correspond to the opening or closing of a loop on the boundary. Furthermore, the sub- and sur-level set component associated with each connected component of the lower and upper link of every saddle is also known from step *(ii)*. Let s be a saddle of S_∂. If two or more connected components of $Lk^-_\partial(s)$ are part of the same sub-level set component, then by lemma 2, s forms a loop in $\mathscr{R}(f_\partial)$. However, if they also belong to the same contour of f in the volume, then s does not create a loop in $\mathscr{R}(f)$. Therefore, we keep only saddles s in S_∂ where distinct connected components in $Lk^-_\partial(s)$ are the boundary components of the same sub-level set component, and are also disconnected in the link of s in the volume, $Lk^-(s)$. All other saddles are removed from S_∂. In practice, this step removes 50% of the remaining saddles of S_∂.

3.1.3 Cutting surfaces

We symbolically cut M through a sequence of symbolic cuts, implementing lines (3-5) of the algorithm. According to lemma 3, to ensure that $\mathscr{R}(f)$ be loop free, every saddle must join distinct sub-level set components. A saddle s in S_∂ does not have this property, therefore we perform symbolic cuts on M such that each connected component of the lower link of s has a unique sub-level set component.

A symbolic cut is an isosurface traversal that updates pointers in the tetrahera that are crossed. A cutting surface $\mathscr{S}$ is a simple data structure with a unique identifier that is the record of the symbolic cut. Let s be a saddle of S_∂ with value $f(s)$, and let C_i and C_j be connected components in $Lk^-(s)$. We perform a cut with value $f(s)-\varepsilon$ for every pair C_i and C_j that are on the boundary of the same sub-level set component. We start the traversal at a tetrahedron in the star of s that has a vertex in C_i, formally, $\sigma \in St(s) | \overline{\sigma} \cap C_i \neq \emptyset$. The sub-level set information necessary to compare C_i and C_j is found in step *(ii)* of loop saddle identification (section 3.1.2). We skip components of $Lk^-(s)$ that do not touch the boundary. Each symbolic cut produces a new sub-level set component, which is recorded in the cutting surface data structure. For example, if n components of $Lk^-(s)$ initially are on the boundary of the same sub-level set component, $n-1$ symbolic cuts will be performed.

To keep track of the symbolic cuts in the rest of the algorithm, each tetrahedron crossed by any cutting surface stores a pointer for each of its vertices to the highest cutting surface passing below and the lowest cutting surface passing above the vertex. Additionally, each vertex is marked with a *top* flag if it lies above a cutting surface crossing the tetrahedron, and also a *bottom* flag if it lies below a cutting surface crossing the tetrahedron. Finally, each saddle that generates symbolic cuts stores pointers to the corresponding cutting surfaces.

3.2 Loop free Reeb graph computation

By lemma 3, a loop free Reeb graph can be computed using a contour tree algorithm. Since we cut M only symbolically, we use a modified version of the join-split algorithm [7] that behaves "as if" M were actually transformed into M'. Building the loop free Reeb graph using the modified join-split algorithm implements line (8) in algorithm 1.

The following pseudo-code for computing the join tree simulates the cuts and uses a Union-Find data structure (using path compression and union by rank) implemented to return the highest element of a set.

Algorithm 2. Modified Join Tree

```
INPUT: f, M
(0) P = sort_vertices(M), UF={}, JT={}
(1) for i in [0,...,|P|-1], v = P[i]
(2)   a = JT.add_node(v)
(3)   UF.make_set(a)
(4)   if v ∈ S∂ then
(5)     for each 𝒮 in cutting_surfaces(v)
(6)       b = JT.add_node(𝒮)
(7)       UF.make_set(b)
(8)   for each tet σ∈St(v) with σ̄∩Lk⁻(v)≠∅
(9)     if has_cut(σ) then
(10)      𝒮= highest_cut_below(σ, v)
(11)      c = UF.find(𝒮)
(12)      if c ≠ a then
(13)        UF.union(c, a)
(14)        JT.add_edge(c, a)
(15)      for each vertex u in σ̄ ∩ Lk⁻(v)
(16)        if f(u) > f(𝒮) then
(17)          d = UF.find(u)
(18)          if d ≠ a then
(19)            UF.union(d, a)
(20)            JT.add_edge(d, a)
(21)    else
(22)      for each vertex u in σ̄ ∩ Lk⁻(v)
(23)        c = UF.find(u)
(24)        if c ≠ a then
(25)          UF.union(c, a)
(26)          JT.add_edge(c, a)
(27) return JT
```

Vertices are processed in order of increasing function value (lines 0,1), and we add the vertex v to the join tree and to a new set in the Union-Find (lines 2,3). If v is a saddle that generated a symbolic cut (line 4), we simulate M being cut. We also simulate the existence of a new sub-level set and a new minimum by adding each cutting surface $\mathscr{S}$ to the join tree and the Union-Find (lines 6,7). The tetrahedra in the star of v that have a vertex in the lower link of v are iterated upon (line 8). For each such tetrahedron σ, if there is a cut $\mathscr{S}$ crossing σ (line 9), we pick the highest $\mathscr{S}$ that is below v (line 10). This is a constant time operation due to having stored pointers in each tetrahedron in the cutting surface computation (section 3.1.3). We perform a

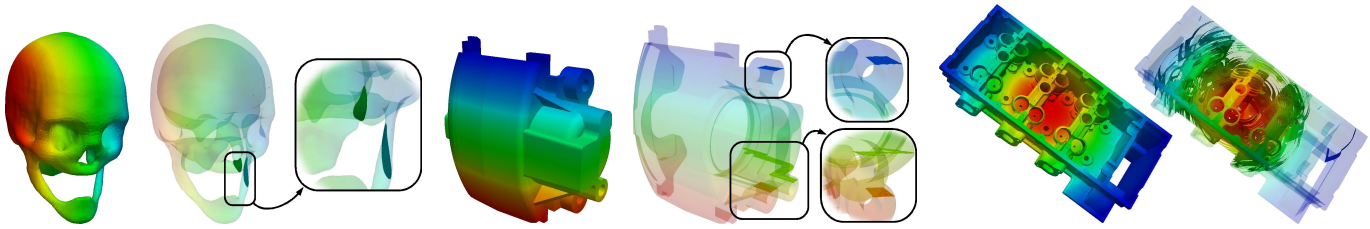


Fig. 6. Examples of *cutting surfaces* of 3D scalar fields defined on non-simply connected domains: skull (2 handles), brake caliper (3 handles), cylinder head (82 handles).

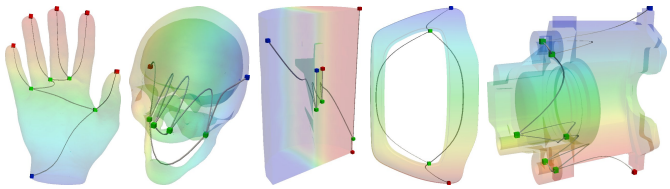


Fig. 7. Examples of Reeb graphs computed from height functions defined on the experiment data-sets. Persistence-based simplification and arc smooth embedding can optionally be computed in a post-process.

find and then merge the sets, also adding an arc to the join tree (lines 12-14). This simulates having cut M by $\mathscr{S}$, since it disconnects part of the lower link, and instead connects v to an "artificial" minimum, which is the node returned by the Union-Find. Next, the vertices in the lower link of v that were not disconnected by any cutting surface are processed (lines 15-20). If no tetrahedra in the lower star of v are crossed by a cutting surface (*i.e.*, v did not get marked as *top* in section 3.1.3 (line 21)) then the lower link of v can be processed (lines 22-26) with no changes to the algorithm in [7]. The join tree of f' is returned. The split tree is computed symmetrically, and merging the two trees occurs exactly as in the join-split algorithm. This computes the loop free Reeb graph $\mathscr{R}(f')$.

3.3 Inverse cuts

Transforming the loop-free Reeb graph $\mathscr{R}(f')$ into $\mathscr{R}(f)$ requires gluing minimum-maximum pairs of each cutting surface. This implements lines 7-8 of algorithm 1. For each cutting surface, pointers in the data structure identify the minimum it generated in the join tree and the maximum it generated in the split tree. The two nodes are found in $\mathscr{R}(f')$, and are glued together by concatenating the up-arc of the minimum, and the down-arc of the maximum. This inverts the changes made to $\mathscr{R}(f')$ by loop surgery.

4 Results

We implemented Reeb graph computation based on loop surgery in standard C under GNU/Linux. All the experiments presented below were run on a standard desktop computer with a 64-bit 2.83 GHz CPU and 8 GB of memory. Data-sets are courtesy of the AIM@SHAPE shape repository [1] and collaborating mechanical design experts.

In our experiments, we compare running times with recent Reeb graph computation techniques for tetrahedral meshes [19, 12]. We used the original implementations of these approaches, kindly provided by their respective authors. Furthermore, we compared the output of our approach to that presented in [19] using the exact same simulation of simplicity [14], and found the two algorithms output identical Reeb graphs for all the available data-sets.

4.1 Time complexity

Let n and N be the number of vertices and simplices of M. Let n_∂, N_∂ and $N_\mathscr{S}$ be the numbers of vertices of ∂M, the number of simplices of ∂M, and the number of simplices of M crossed by cutting surfaces. We present a complexity analysis for each step in our algorithm:

Loop saddle extraction: Saddle identification by link traversal requires $O(n_\partial)$ steps. Join tree and split tree computation both require $O(n_\partial log(n_\partial) + N_\partial \alpha(N_\partial))$ where $\alpha()$ is an exponentially decreasing function (inverse of the Ackermann function). Loop saddle distinction is performed in $O(g)$ steps, where g is the genus of ∂M.

Symbolic cuts computation: $O(g \times N_\mathscr{S})$ steps: there are at most $O(g)$ cuts, and each may cross at most $O(N_\mathscr{S})$ tetrahedra.

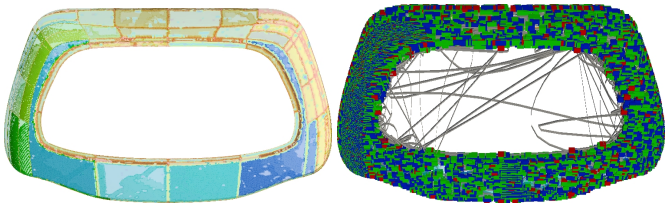


Fig. 8. Trunk data-set: due to the thinness of the mesh, the pressure simulation failed, resulting in a noisy field (left), where 20% of the vertices of the mesh generate branching in the Reeb graph (right).

Loop-free Reeb graph computation: the contour tree algorithm variant requires $O(nlog(n) + N\alpha(N))$ steps [7].

Inverse cuts: $O(g)$ steps: there is an explicit list of cutting surfaces, and gluing the min-max pairs of each takes constant time.

Overall bound: $O(nlog(n) + N\alpha(N) + g \times N_\mathscr{S})$.

The worst case scenario is reached when both g and $N_\mathscr{S}$ (loop surgery process) are linear with the size of the mesh, in which case our algorithm has a quadratic complexity. However, with real-life data, g is a small constant, resulting in virtually linear scalability in practice.

4.2 Performance Comparison

We compare the running times of our approach with those of the two fastest previous techniques, presented in [19] and [12]. Notice however that these techniques provide a more general solution since they handle meshes of arbitrary dimension. Our approach, instead, has been specifically designed for volumetric meshes embedded in $\mathbb{R}^3$, given their importance in visualization applications.

The scalar data used in our experiments represents a variety of physical phenomena: air turbulence, pressure, liquid oxygen diffusion, rock density, etc. Table 1 reports the running times of the three methods on these data. Our approach achieves significant improvement in terms of running time for each data-set, including those with the highest number of handles, resulting in an average speedup factor of 6,500.

The approaches presented in [19] and [12] exhibit a quadratic behavior on real-life volumetric data. The streaming approach [19] becomes very memory intensive because it has to maintain the boundaries of incomplete level sets, which are represented efficiently only for surface meshes. The memory footprint of the output sensitive approach [12] is dependent on the number of critical points and the size of their contours, which in real-life data-sets can be prohibitively large. In our loop surgery, instead, it is difficult to reach with real-life data the worst case scenario of the theoretical quadratic complexity.

4.3 Loop surgery overhead

We analyze the internal behavior of our approach and compute the running time overhead due to loop surgery. Table 2 reports the average running times of the individual steps of our approach. The running times agree with our complexity analysis, as the loop surgery overhead increases when the number of handles increases. The pressure field on the trunk data-set (fig. 8) is a special case: it is an extremely noisy field that illustrates that we are performing more symbolic cuts than necessary, due to ambiguities in resolving degenerate critical points. Overall, we observe that the loop surgery overhead is proportional to the cost of computing the contour tree with real-life data.

4.4 Asymptotic stress tests

Finally, we provide a stress test to show the performance of the algorithm in a worst-case scenario. We generate meshes by tetrahedralizing a rectilinear grid on from which rows and columns have been

Table 1. Comparison of the Reeb graph computation times. In the table *#H* indicates the number of handles in a mesh, *LS*, *SA*, and *OS* indicate the loop surgery, the streaming [19] and the output sensitive [12] approaches respectively. On average, the loop surgery processes 428k tets per second, providing a speedup factor of 6,510. The symbol *me* indicates that the process finished because of a memory exception.

Data	Tets	#H	LS	SA [19]		OS [12]	
			Time (s.)	Time (s.)	Speedup	Time (s.)	Speedup
Langley Fighter	70,125	0	0.35	43.70	**126**	650.10	**1,879**
Cylinder Head (low res.)	116,274	82	0.66	10.20	**15**	257.19	**390**
Hood (low res.)	120,501	31	0.34	45.77	**135**	52.60	**158**
Trunk (low res.)	143,366	1	1.87	27.97	**15**	406.67	**218**
Liquid Oxygen Post	616,050	1	0.69	435.20	**634**	15.54	**23**
Brake Caliper (med res.)	1,155,317	3	2.24	*me: 167,382.52*	-	180,637.74	**80,570**
Buckminster Fullerene	1,250,235	0	2.51	9,887.00	**3,942**	781.00	**311**
Plasma	1,310,720	0	2.20	11,983.21	**5,442**	*me: 406.83*	-
S. Fernando Earthquake	2,067,739	0	4.07	15,949.10	**3,921**	*me: 327.33*	-
Brake Disk (high res.)	3,554,828	54	7.80	*me: 56,929.80*	-	*me: 762.14*	-

Table 2. Running times of the different stages of the algorithm for real-life fields on non-simply connected domains. Loop surgery overhead represents 43% of the computation in average.

Data	Tets	#H	Loop surgery (s. / %)		contour tree (s. / %)	
Cylinder Head (low res.)	116,274	82	0.45	**68.18**	0.21	**31.82**
Hood (low res.)	120,501	31	0.13	**38.24**	0.21	**61.76**
Trunk (low res.)	143,366	1	1.42	**75.94**	0.45	**24.06**
Liquid Oxygen Post	616,050	1	0.09	**13.43**	0.58	**86.57**
Brake Caliper (medium res.)	1,155,317	3	0.52	**23.11**	1.73	**76.89**
Brake Disk (high res.)	3,554,828	54	3.01	**38.89**	4.79	**61.41**

removed, allowing us to increase the number of tets and the number of handles independently. The function value of each vertex is its y coordinate. Figure 9 illustrates these meshes. Figure 10 shows the memory footprint and the running time as a function of the number of handles and the size of the mesh (in logarithmic scale). This experiment shows that for a constant number of handles, our algorithm scales linearly with the size of the mesh (slopes of one on the log-log scale). The memory footprint follows the same linear behavior as the execution time, due to non-optimized data structures for storing cutting surfaces.

4.5 Limitations

A limitation of the algorithm is that it requires the mesh to be manifold with boundary to ensure that the boundary of the mesh is a 2-manifold without boundary. Another is that it is often not possible to disambiguate the behavior of contours by inspecting only the link and sub-level sets of a saddle, therefore we perform extra cuts.

5 Application: Fast Topologically Clean Isosurface Extraction on Non-Simply Connected Domains

To demonstrate the potential and the versatility of our approach, we generalize fast topologically clean isosurface extraction to non-simply connected domains. In particular, we focus on the analysis of pressure fields in mechanical design (where the majority of meshes have handles), a case study where contour-tree based techniques could not previously apply. One experiment in the process of mechanical design involves the analysis of the resistance of mechanical pieces made of different materials to pressure stress. In such experiments, mechanical experts first consider simulation previews computed on low resolution meshes. This step illustrates the approximate behavior of the material. Its understanding is crucial to define correct parameters for the actual simulation at high resolution. However, as shown in figure 11, this preview can be noisy, making its interpretation difficult.

We overcome this problem with a fast topologically clean isosurface extraction system. First, the Reeb graph of the low-resolution pressure stress function is computed in a pre-process. Then, local geometric measures [8] (extended to tetrahedral meshes) are computed for each arc of the Reeb graph. Then, users may select thresholds for geometric measures to simplify the Reeb graph, as described in [19]. This filtering of the arcs in our examples took at most 0.03 seconds. Our approach maintains degree two nodes in the Reeb graph, representing regular vertices of the function. Consequently, the Reeb graph provides a seed vertex for each contour the user wants to display. We store the non-simplified arcs of the Reeb graph in a balanced interval tree. An isosurface extraction query consists of searching in this tree for a valid seed set. In our examples, this is performed in less than

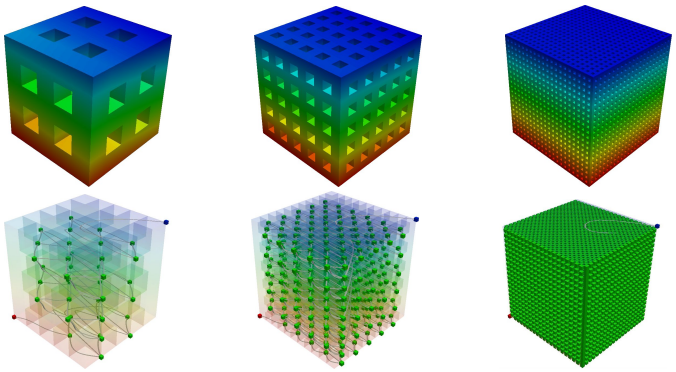


Fig. 9. Handle stress tests: examples of generated meshes with increasing number of handles and the Reeb graph of their height function.

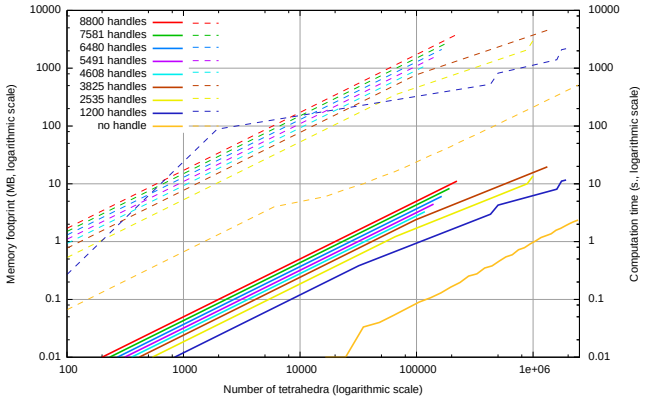


Fig. 10. Memory footprint (dashed) and running time (solid) when increasing the number of tets for a mesh with constant number of handles.

nanoseconds, starting standard isosurface traversal techniques at these seeds in interactive times.

Figure 11 illustrates this process on pressure stress functions computed on a brake disk and a cylinder head, where the user progressively increases a simplification threshold with the hyper-volume measure [8]. The Reeb graphs are simplified, and as a result, the small connected components (noise) of the queried isosurface are progressively removed and the most important features are highlighted. This enables a direct visualization of the major trends of the simulation. In the future, we would like to introduce metrics between the Reeb graphs at low and high mesh resolutions, in order to provide quality scores for the low-resolution previews, to accelerate the validation of the final simulation.

6 Conclusion

In this paper, we present a novel algorithm for fast Reeb graph computation on tetrahedral meshes in $\mathbb{R}^3$. By providing theoretical results on the topology of such Reeb graphs, we show their computation could be reduced to a contour tree computation through a technique called loop surgery. Experiments demonstrate in practice the scalability of the algorithm. Moreover, we show that our approach improves in term of running time, for the special case of volumetric meshes, the fastest previous techniques on real-life data by several orders of magnitude. We extend fast topologically clean isosurface extraction to non-simply connected domains in the context of mechanical design.

Reducing the computational requirements of Reeb graphs to that of contour trees enables the generalization of the contour-tree based visualization techniques to volumetric meshes of arbitrary topology and thus opens several avenues for future visualization research. Furthermore, as contour tree computation has been shown to be parallelizable [18], we plan to investigate a parallel version of the algorithm for large-scale data analysis. An extension of our approach to volumetric meshes not embeddable in $\mathbb{R}^3$ and of higher dimensions would address a larger class of problems. However, as it is no longer true that such meshes necessarily admit boundary, the loop surgery concept would need to be extended and generalized.

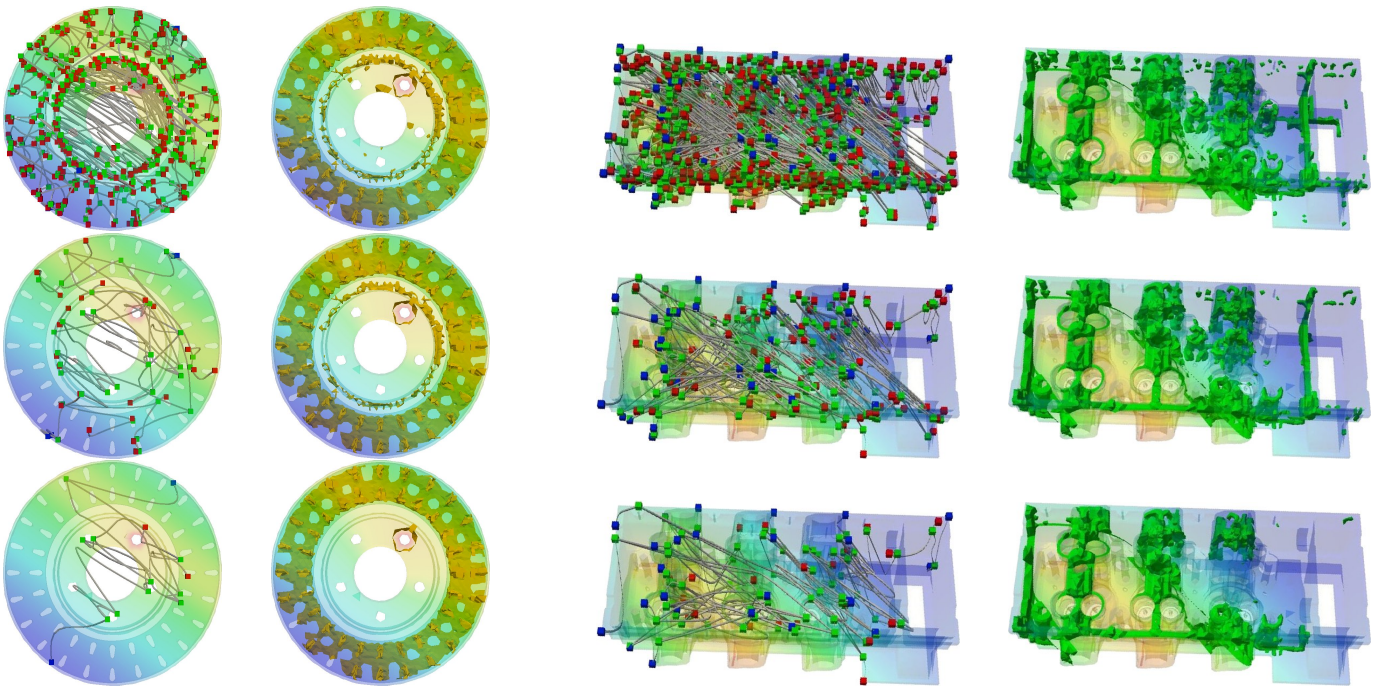


Fig. 11. Topologically clean isosurface extraction on a pressure stress simulation. The Reeb graph is progressively simplified (from top to bottom) with increasing hyper-volume scale. As a result, small components (noise) of the considered isosurface are progressively removed and the most important features are progressively highlighted. Brake disk, from top to bottom: 92, 8 and 2 connected components. Cylinder head, from top to bottom: 215, 58 and 9 connected components.

ACKNOWLEDGMENTS

Julien Tierny was supported by the Fulbright Program (U.S. Department of State) and by the Lavoisier Program (French Ministry of Foreign Affairs). Attila Gyulassy was supported in part by the National Science Foundation, under grant CCF-0702817. This work has also been performed under the auspices of the U.S. Department of Energy by the University of Utah under Contract DE-FC02-06ER25781. The authors would like to thank Vijay Natarajan for providing the implementation of the paper [12].

REFERENCES

[1] AIM@SHAPE Shape Repository. *http://shapes.aim-at-shape.net/*, 2006.

[2] P. K. Agarwal, H. Edelsbrunner, J. Harer, and Y. Wang. Extreme elevation on a 2-manifold. In *ACM Symp. on Computational Geometry*, pages 357–365, 2004.

[3] G. Aujay, F. Hétroy, F. Lazarus, and C. Depraz. Harmonic skeletons for realistic character animation. In *Eurographics Symp. on Computer Animation*, pages 151–160, 2007.

[4] C. L. Bajaj, V. Pascucci, and D. Schikore. The contour spectrum. In *IEEE Visualization*, pages 167–174, 1997.

[5] S. Biasotti, B. Falcidieno, and M. Spagnuolo. Extended Reeb graphs for surface understanding and description. In *Discrete Geometry for Computer Imagery*, pages 185–197, 2000.

[6] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theoretical Computer Science*, 392:5–22, 2008.

[7] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *ACM Symp. on Discrete Algorithms*, pages 918–926, 2000.

[8] H. Carr, J. Snoeyink, and M. V. de Panne. Simplifying flexible isosurfaces using local geometric measures. In *IEEE Visualization*, pages 497–504, 2004.

[9] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Loops in Reeb graphs of 2-manifolds. In *ACM Symp. on Computational Geometry*, pages 344–350, 2003.

[10] M. de Berg and M. J. van Kreveld. Trekking in the alps without freezing or getting tired. In *European Symp. on Algorithms*, pages 121–132, 1993.

[11] T. K. Dey and S. Guha. Computing homology groups of simplicial complexes in $\mathbb{R}^3$. *Journal of the ACM*, 45:266–287, 1998.

[12] H. Doraiswamy and V. Natarajan. Efficient output-sensitive construction of Reeb graphs. In *International Symp. on Algorithms and Computation*, pages 557–568, 2008.

[13] H. Doraiswamy and V. Natarajan. Efficient algorithms for computing Reeb graphs. *Computational Geometry: Theory and Applications*, 42:606–616, 2009.

[14] H. Edelsbrunner and E. P. Mucke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. on Graphics*, 9:66–104, 1990.

[15] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

[16] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. In *SIGGRAPH*, pages 203–212, 2001.

[17] J. Milnor. *Morse Theory*. Princeton University Press, 1963.

[18] V. Pascucci and K. Cole-McLaughlin. Parallel computation of the topology of level sets. *Algorithmica*, 38:249–268, 2003.

[19] V. Pascucci, G. Scorzelli, P. T. Bremer, and A. Mascarenhas. Robust online computation of Reeb graphs: simplicity and speed. *ACM Trans. on Graphics*, 26:58.1–58.9, 2007.

[20] G. Patanè, M. Spagnuolo, and B. Falcidieno. Reeb graph computation based on minimal contouring. In *IEEE Shape Modeling International*, pages 73–82, 2008.

[21] G. Reeb. Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique. *Comptes-rendus de l'Académie des Sciences*, 222:847–849, 1946.

[22] Y. Shinagawa, T. L. Kunii, and Y. L. Kergosien. Surface coding based on Morse theory. *IEEE Computer Graphics and Applications*, 11:66–78, 1991.

[23] S. Tarasov and M. Vyalyi. Construction of contour trees in 3D in O((n) log (n)) steps. In *ACM Symp. on Computational Geometry*, pages 68–75, 1998.

[24] M. van Kreveld, R. van Oostrum, L. Bajaj, C, V. Pascucci, and R. Shcikore, D. Contour trees and small seed sets for isosurface traversal. In *ACM Symp. on Computational Geometry*, pages 212–220, 1997.

[25] C. T. C. Wall. *Surgery on compact manifolds*. American Mathematical Society, 1970.

[26] G. H. Weber, P.-T. Bremer, and V. Pascucci. Topological landscapes: a terrain metaphor for scientific data. *IEEE Trans. on Visualization and Computer Graphics*, 13:1416–1423, 2007.

[27] G. H. Weber, S. E. Dillard, H. Carr, V. Pascucci, and B. Hamann. Topology-controlled volume rendering. *IEEE Trans. on Visualization and Computer Graphics*, 13:330–341, 2007.

[28] J. Wood, Z, H. Hoppe, M. Desbrun, and P. Schrder. Removing excess topology from isosurfaces. *ACM Trans. on Graphics*, 23:190–208, 2004.

[29] E. Zhang, K. Mischaikow, and G. Turk. Feature-based surface parametrization and texture mapping. *ACM Trans. on Graphics*, 24:1–27, 2005.