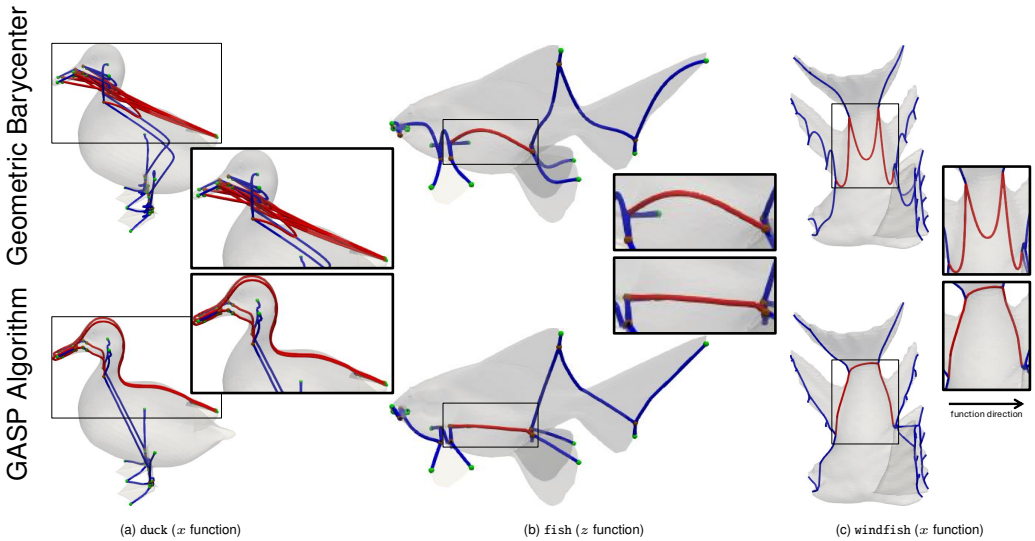
GASP: Gradient-Aware Shortest Path Algorithm for Boundary-Confined 2-Manifold Reeb Graph Visualization

Sefat Rahman Tushar M. Athawale, and Paul Rosen



(a) duck (x function) (b) fish (z function) (c) windfish (x function)

Fig. 1: Examples showing that the GASP algorithm (bottom) better accommodates the three desired properties for visualizing Reeb graphs, as compared to the geometric barycenter (GB) method, implemented in the Topology ToolKit (top). The complete Reeb graphs include both *red* and *blue* arcs, but *red* arcs highlight improvements over GB. (a) GASP constrains the path of the Reeb graph arcs such that they remain either inside or on the surface of the object, while many arcs generated by GB do not conform to the boundary of the model. (b) GASP chooses the shortest path between two critical points (while constrained by the boundary), while some GB arcs are unnecessarily long due to the desire for smoothness. (c) GASP maintains consistency with the gradient of the function, while in part due to smoothing, GB arcs travel nearly perpendicular to the direction of the function (left-to-right in this case).

Abstract—Reeb graphs are an important tool for abstracting and representing the topological structure of a function defined on a manifold. We have identified three properties for faithfully representing Reeb graphs in a visualization. Namely, they should be constrained to the boundary, compact, and aligned with the function gradient. Existing algorithms for drawing Reeb graphs are agnostic to or violate these properties. In this paper, we introduce an algorithm to generate Reeb graph visualizations, called *GASP*, that is cognizant of these properties, thereby producing visualizations that are more representative of the underlying data. To demonstrate the improvements, the resulting Reeb graphs are evaluated both qualitatively and quantitatively against the geometric barycenter algorithm, using its implementation available in the Topology ToolKit (TTK), a widely adopted tool for calculating and visualizing Reeb graphs.

Index Terms—Reeb graph visualization, geometric barycenter algorithm, Topology ToolKit

1 Introduction

The Reeb graph is a powerful tool for understanding the shape and structure of objects and data. It provides a compact structural abstraction of a function defined on a manifold (i.e., a skeleton of an object) [33]. A wide range of applications have utilized Reeb graphs [1, 25, 57], from analyzing 3D printing models [32] to assisting in disease studies, such as Alzheimer's [49]. However, faithfully visualizing the Reeb graph structure is of the utmost importance for interpretation [3].

Since the Reeb graph is drawn based on a scalar function defined over a manifold, it should closely reflect those objects. To ensure this, we identified three essential properties and one optional property that a Reeb graph visualization should possess. First, because the scalar field is defined on the manifold, the arcs of the Reeb graph should remain confined within the manifold. Any arc extending beyond it would imply the Reeb graph exists outside the manifold (see Fig. 1a top). Second, the Reeb graph arcs should be as short as possible to minimize visual clutter and to facilitate easier interpretation (see Fig. 1b top). Third, the arc orientations should closely align with the direction of the associated scalar field gradient. A mismatch between the gradient direction and the Reeb graph arcs can be confusing and make it harder to understand the scalar field (see Fig. 1c top). Fourth, smoother arcs provide a clearer presentation of the paths, making them easier to track, but this property is optional, as it should not be at the cost of the other properties.

Several algorithms for drawing Reeb graphs and contour trees (loop-free Reeb graphs) have been proposed (see Sec. 2.4). However, none explicitly identify or address all of these desired properties. Prior work on Reeb graph visualization has either focused on planar drawings or included only brief mentions within the broader context of Reeb graph construction [18, 46]. To the best of our knowledge, this paper is the first to focus exclusively on visualizing Reeb graphs in 3D embeddings.

Motivated by the goal of producing Reeb graph visualizations that adhere to the three essential properties, we developed an algorithm, called

- *Sefat Rahman and Paul Rosen are with the University of Utah. E-mail: { sefat.rahman | paul.rosen }@utah.edu.*
- *Tushar M. Athawale is with Oak Ridge National Laboratory. E-mail: athawaletm@ornl.gov.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

GASP, which produces Reeb graphs that are boundary-constrained and Gradient-Aware with Short Paths. GASP works by first decomposing the model into topological cylinders, each corresponding to a Reeb graph arc. These arcs are then drawn as the shortest paths between critical points over a graph constructed from successive (iso)contours. GASP arcs generally align with the first property (keeping arcs within the model) by leveraging the contours to constrain arc placement within the boundary (see Fig. 1a bottom). GASP addresses the second property (minimizing arc length) by computing the shortest paths between critical points (see Fig. 1b bottom). GASP satisfies the third property (following the gradient) by progressing from one contour to the next, ensuring that arcs align with the direction of the function (see Fig. 1c bottom). Finally, although GASP arcs are generally smooth, it does not directly address the optional property of smoothness.

Given its widespread use, we compare the GASP algorithm to the geometric barycenter [46] algorithm, as implemented within Topological ToolKit (TTK) [55], for visualizing the Reeb graph. For each of the three essential properties, confining arcs within the model (see Fig. 1a), shorter arc length (see Fig. 1b), and alignment with the functional gradient (see Fig. 1c), GASP outperforms the geometric barycenter algorithm at only a small cost to the optional property of smoothness.

In summary, the key contributions of this paper are:

- The identification of three essential and one optional property for faithfully visualizing Reeb graphs.
- A Reeb graph visualization algorithm, *GASP*, that is boundary-constrained and Gradient-Aware with Short Paths.
- An open-source implementation of GASP, and a set of new quantitative benchmarks for evaluating Reeb graph visualization quality that uses 30 triangle meshes with height and geodesic functions.

2 Background on Reeb Graphs

We briefly present the technical definitions of isocontours and critical points that are building blocks of a Reeb graph construction. We then describe Reeb graph construction, computation, and visualization.

2.1 Isocontours and Critical Points

Isocontours are a foundational tool in visualization. Isocontours represent a locus of points in a 2-manifold that attain a fixed function value, known as an isovalue. We restrict our discussion to scalar functions sampled on a 2-dimensional manifold, as other types of functions and manifolds are beyond the scope of this paper. Mathematically, if $f : \mathcal{M} \to \mathbb{R}$ is a scalar function defined on an 2-dimensional manifold, $\mathcal{M}$, then the isocontour $\mathcal{L}$ for isovalue k can be represented as $\mathcal{L}(k) \equiv \{P : P \in \mathcal{M} \wedge f(P) = k\}$, where P denotes domain positions with function value k. The change in data values over the domain can be understood by visualizing the evolution of isocontours for isovalues spaced at regular intervals. Fig. 2a visualizes isocontours (in *orange*) of a height function, f, for various isovalues, f_i, mapped on a torus. A single isovalue may produce a single or multiple connected components. As observed in Fig. 2a, the isocontours immediately beyond split and merge saddles comprise two connected components.

Critical points, C, are special points of a field that have a close relation with the evolution of isocontours. Technically, critical points denote domain positions, P, where the field gradient vanishes. Let $f : \mathcal{M} \to \mathbb{R}$ be a Morse function; ∇f denotes its gradient. A point $C \in \mathcal{M}$ is considered *critical* if $\nabla f(C) = 0$; otherwise it is *regular*. A critical point is categorized into three types: local minimum, local maximum, or saddle. In particular, if $f(C)$ is smaller than all of its neighbors, then it is a local minimum. Similarly, if $f(C)$ is greater than all of its neighbors, then it is a local maximum. If $f(C)$ is smaller than one neighbor and greater than the next neighbor in alternating fashion, with the neighbors visited sequentially in a clockwise/counterclockwise manner, it is a saddle. Critical points represent the important domain positions because they result in the birth, splitting, merging, and death of connected components in isocontours. Fig. 2a visualizes the critical points as black spheres. As observed, isocontours (depicted in *orange*) appear, split, merge, and disappear at the local minimum, split saddle, merge saddle, and local maximum, respectively, as the isovalue increases. The evolution of isocontours with respect to critical points is captured by Reeb graphs, as described in the next section.

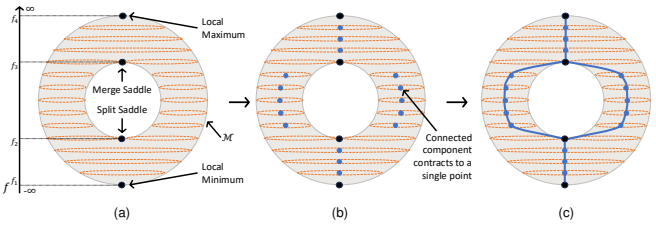


Fig. 2: Reeb graph construction. (a) Isocontours are visualized in *orange*, and the *black* spheres denote the positions of critical points. (b) Connected components are contracted to a single point depicted as *blue* dots. (c) Critical points and the point representation of connected components provide the Reeb graph as a structural abstraction.

2.2 The Reeb Graph

The Reeb graph is a structural abstraction that provides insight into the topological skeleton of scalar field data. Formally, the Reeb graph tracks the connected components of isocontours in the domain as f is swept from $-\infty \to +\infty$ [20]. The Reeb graph of a Morse function, f, defined on a simply connected manifold, $\mathcal{M}$, is loop-free and is referred to as a contour tree [10]. As shown in Fig. 2b, each connected component of isocontour (*orange* ellipses) can be contracted to a single point (depicted as *blue* dots). Connecting the points representing connected components forms arcs or edges (E) of a Reeb graph, as portrayed in Fig. 2c. In summary, the arcs (E) of a Reeb graph indicate connected components in the domain with the same topology, and critical points (C) in the Reeb graph represent nodes that indicate topological changes. Reeb graphs, therefore, provide a compact and abstract visual representation of the data topology.

2.3 Reeb Graph Computation

Considering the high utility of Reeb graphs in the analysis of complex data, several prior works focused on methods for efficient computation and visualization of Reeb graphs. Shinagawa and Kunii [50] proposed $O(n^2)$ algorithm for Reeb graph construction, where n represents the number of vertices in a 2-manifold triangular mesh. Since then, multiple new algorithms were proposed that increased the efficiency of Reeb graph constructions [13, 16, 17, 47, 56]. Harvey and Wang [29] achieved an optimal expected running time of $O(m \log m)$ through a randomized dynamic-connectivity strategy, and Parsa [44] derandomized this approach to obtain the same bound in the worst case. Gueunet et al. [26] presented a task-based parallel algorithm for Reeb Graph calculation and visualization, leveraging merge and split trees constructed from scalar field data. It uses Fibonacci heaps to efficiently handle dynamic operations like merging and splitting components. TTK implements this approach for Reeb graph computation. While this work targets parallel performance, its sequential core follows Parsa's framework, which itself builds on the randomized dynamic-connectivity formulation introduced by Harvey and Wang.

2.4 Reeb Graph Visualization

Several approaches have been used to visualize Reeb graphs, contour trees, and general graphs. Strodthoff and Jüttler [53] devised a layered-Reeb graph construction algorithm for efficient computation and visualization of Reeb graphs for solids with boundary embedded in 3D. Shinagawa et al. [51] devised a 2D layout for the representation of Reeb graphs. Pascucci et al. [45] devised "toporrery" as a way to draw contour trees in a radial manner that avoids the self-intersection of edges in planar methods. Heine et al. [30] leveraged advances in graph drawing techniques [4, 24] for drawing contour trees in 2D layouts. 3D medial axis algorithms [14] deal with a related problem, in that they produce a skeleton of a manifold. However, they do not have the constraints introduced by the scalar function defined on the manifold.

Similar to our approach, the embedded layout of Reeb graphs, as described by Doraiswamy and Natarajan [18], ensures that arcs lie within their respective monotone cylinders in the input domain. This approach leverages the LS-graph to trace paths confined to the interior of the corresponding cylinders, facilitating intuitive visualization within the original dataset's spatial context. The proposed technique addresses

the first desired property of a Reeb graph identified by us, but no experimental results were presented to support the claim. Further, the other two properties, i.e., generating shorter arcs and alignment with the gradient of the function, are not clearly addressed.

Geometric Barycenter Algorithm The Topology ToolKit (TTK) [55] is a frequently used tool for visualizing Reeb graphs, which uses the geometrical barycenter algorithm [46]. TTK's method for visualizing Reeb graphs involves three steps [46, 54]. First, Reeb graph edges connect critical points directly via straight lines (see Fig. 4a). Arcs are then produced using an arc sampling parameter that evenly divides the straight edge into sub-components (see Fig. 4b). Then, each sample is placed at the geometrical barycenter of the isocontours, and, finally, the arcs are smoothed geometrically to improve the aesthetics of the representation (see Fig. 4c). TTK produces smooth, aesthetically pleasing Reeb graph arcs; however, in many cases, it fails to produce Reeb graph arcs with the three essential properties we identified. First, the process used by TTK often results in arcs that extend beyond the model's boundary (see the red arcs in Fig. 1a top). Further, the sampling and smoothing processes often lead to unnecessarily long paths, which cause visual clutter (see the red arcs in Fig. 1b top). Finally, the process of smoothing can introduce significant deviations from the function gradient direction (see the red arcs in Fig. 1c top).

To the best of our knowledge, no prior work has thoroughly addressed the challenge of drawing Reeb graphs in 3D, focusing on the three essential properties we have identified.

3 Literature Review

In this section, we briefly discuss prior work on Reeb graphs for data analysis and topology-based visualizations.

3.1 Reeb Graphs for Data Analysis

Reeb graphs play a pivotal role in data analysis. Reeb graphs and contour trees (as well as other topology-based techniques) are often used as a way to compare and quantitatively measure the similarity of features across complex scalar fields, as surveyed in [59]. Various metrics, including interleaving [38], edit [52], and stable [8] distance have been proposed to quantitatively measure the distance between merge trees that are foundational to construction of Reeb graphs and contour trees. Recently, functional distortion metric [5] and intrinsic interleaving distances between merge trees [35] were used to compare Reeb graphs quantitatively. Chen et al. [11] analyzed time-dependent multi-fluid data by studying Reeb graphs of fluid density distributions. Weber et al. [58] proposed feature tracking using Reeb graphs in time-varying data for combustion applications. Reeb graphs are also a fundamental tool in shape analysis in computer graphics and other domains [6, 7, 43]. Natali et al. [40] proposed utilizing Reeb graphs for extracting the topological skeleton of point cloud data. Given the widespread use of Reeb graphs in data analysis, accurate visualization methods for Reeb graphs are a critical component of that analysis.

3.2 Topological Visualizations

Topological visualization is a powerful tool used to concisely convey the scale and position of important data features to facilitate analysis of complex scientific data [9, 41, 42, 48]. While isocontour [36], critical point [39], and Morse-Smale complex [21] techniques explicitly convey position of important data features, persistence diagram and persistence curve techniques [12] explicitly convey scale of data features. Reeb graphs [20] and contour trees [10] provide topological abstractions that convey feature positions through critical points and scale through arc length in a planar representation. The discrete graph-based representation of these topological visualization types, however, can make their perception and data analysis tasks difficult. Topological simplification [18, 22, 27] is, therefore, a standard method used to reduce the level of detail or noise in data to make topological visualizations more interpretable. Recent work by Athawale et al. [3] and previous work on vector field topology [23, 34] showed a limited capability of users to perceive topological features conveyed by various methods. Their study concluded a need to improve existing visualization designs to make topological visualizations more perceptible. In this paper, we propose a novel algorithm, *GASP*, to enhance the faithfulness of Reeb graph visualization compared to the geometric barycenter algorithm.

4 GASP (Gradient-Aware Shortest Path) Reeb Graphs

We target generating Reeb graphs that are better representations of the model and function used to generate them. To do this, we begin with a high-level conceptual description. The input to our approach is a 2-manifold embedded in 3D space, $\mathcal{M}$, a Morse function, f, and the Reeb graph consisting of a set of critical points, C, and Reeb graph edges, E, which connect pairs of critical points (i.e., $E_i = (C_j, C_k)$). Our process, shown in Fig. 3, contains three main steps.

4.1 Decomposition

To begin, our approach decomposes the model, $\mathcal{M}$, into a subset of objects, one per Reeb graph edge, $E_i = (C_j, C_k)$ (see Fig. 3b). Without a loss of generality, we assume $f(C_j) < f(C_k)$. The decomposition is done by slicing the model at $f(C_j) + \epsilon$ and $f(C_k) - \epsilon$, and selecting the portion of the model associated with the Reeb graph edge. Here, ϵ is assumed to be a very small number. One basic property of Reeb graphs is that each portion of the model associated with a Reeb graph edge, E_i, will be a *topological cylinder*, $\mathbb{C}_i$ [28].

4.2 Reeb Graph Arc in a Topological Cylinder

Next, GASP focuses on drawing a Reeb graph arc within a single topological cylinder, $\mathbb{C}_i$, associated with Reeb graph edge, $E_i = (C_j, C_k)$.

4.2.1 Contours

The first step of generating the Reeb graph for $\mathbb{C}_i$ involves calculating a series of isocontours on the cylinder. The goal is to create contours with similar spacing across the entire model. The number of isocontours is selected by $n = \lceil (f(C_k) - f(C_j))/S + 1 \rceil$, where S is a user-set spacing parameter. Those n contours are evenly spaced from $f(C_j) + \epsilon$ to $f(C_k) - \epsilon$. Fig. 3c shows an example.

4.2.2 Candidate Points

The next step involves generating a series of candidate points, p, using the contours from the prior step, which will be used to identify the Reeb graph path. We present two variations. The first variation called the *boundary approach* (see Fig. 3d), selects a series of candidate points along the contour boundary itself. As will be noted in the implementation, we directly use the points produced by isocontouring. The second variation called the *interior approach* (see Fig. 3e), generates a series of candidate points in a grid-like pattern inside the contour and some user-set distance from the boundary (denoted by the *orange* stripes).

4.2.3 Path Graph and Path Extraction

The next step of the process extracts a path graph, which is used to find the arc path. This is done by forming a graph between the candidate points found in the prior step. Two types of edges are created. First, between two adjacent contours, path graph edges are formed between all pairs of candidate points in neighboring levels. Second, the critical points are connected to all candidate points in the adjacent contour. All edges are weighted by the Euclidean distance between their endpoints. The resulting graph will look similar to Fig. 3f. The final step of the process extracts the shortest path between the two critical points using Dijkstra's algorithm, as denoted by the *blue* path in Fig. 3f.

4.3 Final Assembly

The final Reeb graph is assembled by combining the individual arcs formed per topological cylinder, sharing common points only at the critical points (see Fig. 3g).

Summary This approach has several desirable properties. First, decomposing the mesh into individual cylinders simplifies the problem significantly. Second, using the isocontours to derive the path of the arc results in an arc that is generally inside or near the surface of the manifold, with only minor deviations possible in non-convex regions. Third, selecting the shortest path along adjacent isocontours results in shorter arcs that also travel in the same direction as the function. The second and third properties, in particular, directly address the three Reeb graph properties we highlighted in Sec. 1.

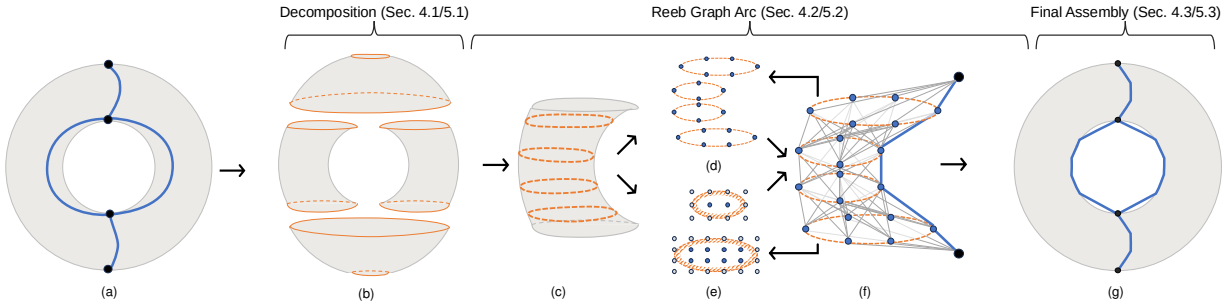


Fig. 3: High-level algorithm description. (a) A Reeb graph is calculated. Although it is shown on the mesh, the calculation only provides a graph object with critical points as nodes and arcs as edges. (b) Our approach first decomposes the mesh into a set of topological cylinders, one per edge. (c) A series of isocontours are calculated for each topological cylinder, and candidate points for the arc path are calculated by (d) the *boundary approach* using the contours themselves or (e) the *interior approach* by finding points inside the contours. (f) Once the candidate points are found, graph edges are formed between the critical points and the adjacent contours and between adjacent contours, and the output arc is calculated as the shortest path between the critical points. (g) Finally, the arcs from all topological cylinders are assembled into a final Reeb graph.

5 Implementation

In this study, we utilized triangular meshes as our input, $\mathcal{M}$. For simplicity of the description, we show a principal direction as the height function, f, on the torus and modified torus models found in Fig. 5-9. However, as we will describe, our algorithm is independent of this constraint, and our evaluation uses height and geodesic functions. To produce the Reeb graph, we utilized Recon [19].

5.1 Decomposition of the Triangle Mesh

Decomposition is performed on a Reeb graph edge-by-edge basis.

5.1.1 Splitting Between Critical Points

Assume we have a Reeb graph edge, E_i, with critical points C_j and C_k, with function values, $f(C_j)$ and $f(C_k)$, where $f(C_j) < f(C_k)$. We will apply two cut operations on the mesh at $f(C_j) + \epsilon$, referred to as ***source cut***, and at $f(C_k) - \epsilon$, referred to as ***destination cut***.

Initially, to speed the process, a rough cut is performed to remove triangles outside the processing area. To do this, triangles are preprocessed into bins by their function value. Bins that would not contain any triangles between the critical points are excluded from the cutting process. Fig. 5b shows one such rough cut. Here, the models were divided into 12 bins[1]. The bins in *cyan* and *blue* are selected for processing, while the *white* and *gray* bins are ignored.

The source and destination cut operations are then performed per triangle. For a source cut, triangles fall into one of 3 cases: (1) completely above the cut value, which is retained as is (see Fig. 6a/6b type 1); (2) completely below the cut value, which is excluded from further computation (see Fig. 6a/6b type 2); and (3) crossing the cut value, in which case triangles are cut along that value. The portion above the cut line is retained in the form of a smaller triangle (see Fig. 6a/6b type 3a) or a quadrilateral, which is subdivided into 2 triangles (see Fig. 6a/6b type 3b). A similar process is repeated for the destination cut, shown in Fig. 6c/6d. Fig. 5c and Fig. 5d show example source and destination cuts on models, respectively.

[1]By default and in experiments, we use 20 bins, but the value is user-settable.

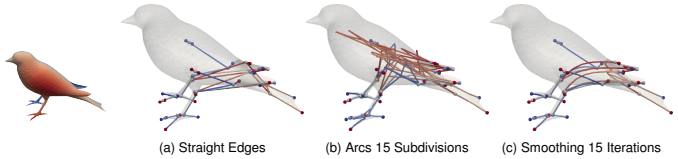


Fig. 4: Illustration of TTK Reeb graph drawing for `bird` (x function). (a) TTK starts by drawing straight edges between the critical points. (b) Next, TTK applies the arc sampling to subdivide the edges into arcs. (c) Finally, a geometric smoother is applied to improve the arc aesthetics.

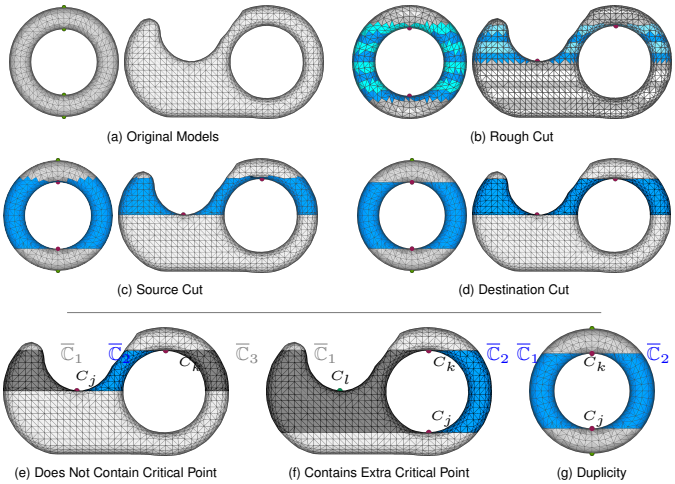


Fig. 5: (a) Torus (left) and Modified Torus (right) are (b) binned by function value and have a rough cut in *cyan* and *blue* performed between the critical points in *red*. Then, the triangles in *blue* have the (c) source cut applied, (d) followed by the destination cut. (e-g) Examples of different candidate connected component validation criteria after a cut, where valid cylinders are *blue*, and invalid ones are *gray*. (e) After a cut, a valid component will connect the two critical points (c_j and c_k), e.g., $\overline{\mathbb{C}}_2$, whereas $\overline{\mathbb{C}}_1$ and $\overline{\mathbb{C}}_3$ only connect to one critical point, c_j and c_k, respectively. (f) After a cut, a valid connected component will not contain any other critical points, e.g., $\overline{\mathbb{C}}_2$, where $\overline{\mathbb{C}}_1$ contains the extra critical point c_l. (g) Special care must be taken for Reeb graph edges with duplicity as multiple valid topological cylinders will be identified (e.g., both $\overline{\mathbb{C}}_1$ and $\overline{\mathbb{C}}_2$) and must be matched to those edges.

5.1.2 Isolating the Topological Cylinder for a Reeb Graph Edge

After the cut operations, one or more components will be extracted. To isolate the cylinder associated with the E_i, from the extracted component(s), the triangles are separated into candidate cylinders by calculating connected components. Candidate cylinders are discarded if: (1) there is no path between C_j and C_k using the candidate cylinder triangles (see Fig. 5e), or (2) there is another critical point contained within the set of triangles from the candidate cylinder (see Fig. 5f). If a candidate cylinder passes both tests, it is associated with the Reeb graph edge connecting the critical points. Special consideration needs to be made to simultaneously process Reeb graph edges that have duplicity (i.e., edges with the same critical points), such as in the torus (e.g., Fig. 5g). This way, each of the multiple valid cylinders can be assigned to a unique Reeb graph edge.

5.2 Reeb Graph Arc in a Topological Cylinder Mesh

Once a topological cylinder is isolated, the Reeb graph arc is formed.

5.2.1 Calculating Contours

We use the marching triangles technique [31] to generate the contours on the object. The calculated number and spacing of contours follows the description in Sec. 4.2.

5.2.2 Calculating Candidate Points

Next, the candidate points, p, for the Reeb graph arc path are calculated using one of two strategies, following Fig. 7.

Boundary Approach After the contours are generated (see Fig. 7a), the vertices coming from marching triangles are used as the vertices in the path graph (see Fig. 7d).

Interior Approach If the contours are approximately planar, the interior approach can identify candidate points inside of them (see Fig. 7a). It does this by first projecting the contour onto a 2D plane using principal component analysis (see Fig. 7b). Then, a regular grid of candidate points is formed within the bounds of the contour. Those points are tested to see if they are inside the contour (point inside polygon test), and their distance from the contour is larger than a user-set buffer, B. Those points are projected back to 3D space (see Fig. 7c) and used as the candidate points for the path graph (see Fig. 7g). The interior approach is more likely to keep Reeb graph arcs inside the object than the boundary approach. When the contour is planar or nearly planar, it can select suitable interior points. However, for non-planar contours, it may place some points outside the surface.

5.2.3 Constructing the Path Graph and Path Extraction

Given the candidate points from the prior step, the path graph is formed next. Candidate points serve as path graph vertices. Candidate points from adjacent contours have path graph edges added between them with weights set by their Euclidean distance. In addition, the contours adjacent to the critical points have path graph edges added between them and the critical points with weights set by their Euclidean distance.

After formulating the path graph, Dijkstra's algorithm is used to find the shortest path between the critical points, which represents the Reeb graph arc, as shown in *blue* in Fig. 7d and Fig. 7g for the boundary and interior versions, respectively.

5.2.4 Iterative Refinement

To improve the quality and speed of the operation, the process of calculating candidate points (see Sec. 5.2.2) and constructing the path graph and extracting the path (see Sec. 5.2.3) is done over multiple iterations at increasing resolutions. For both variations, the first iteration begins with a sparse selection of up to 40 candidate points per contour (see Fig. 7d and Fig. 7g). Once an initial path is identified, a new set with a similar number of candidate points is selected at a finer granularity in the neighborhood of the initial path (see Fig. 7e and Fig. 7h). Our implementation repeats this step one additional time (i.e., 2 refinement steps total, see Fig. 7f and Fig. 7i). The benefit of this approach is that it limits the complexity of the path graph while providing a high fidelity in the output position.

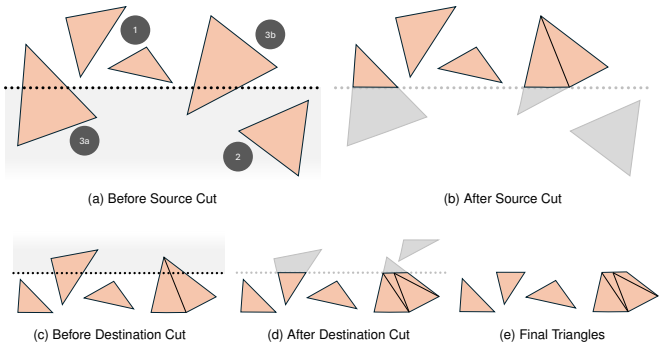


Fig. 6: Examples of triangles (a/c) before and (b/d) after (a/b) source and (c/d) destination cuts. The 4 types of source cuts include: (1) completely above, (2) completely below; (3a) partially above keeping a triangle, and (3b) partially above keeping a quadrilateral. *Beige* triangles are retained, while *gray* ones are discarded.

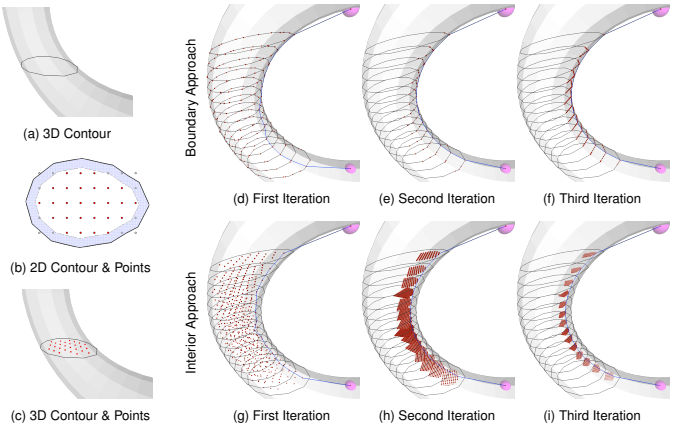


Fig. 7: *Left*: The process for the interior method (a) starts with a contour in 3D. (b) Principal component analysis is used to project the contour onto a plane, where a regular grid of points is set up. The points are checked to see if they are inside of the contour and no closer than a user-set distance from the contour (region in *blue*). (c) Those points that pass both conditions (in *red*) are projected back to 3D space. *Right*: Example of the boundary approach (top) and interior method (bottom) candidate points in *red* and the calculated path in *blue*. The process is iterative, (d,g) starting with a coarse resolution, and refining it two times (e,h followed by f,i). The output path of the final step (f,i) is the Reeb graph arc used for the critical point pair.

5.2.5 Thin Features

For some topological cylinders, the functional distance between critical points is less than the contour spacing parameter (i.e., $f(C_k) - f(C_j) < S$). Under the process described above, these arcs would directly connect critical points, and the visualization would, therefore, be of low quality. Instead, we introduce a modified process for these *thin features*, irrespective of whether the boundary or interior approach is selected.

Decomposition The decomposition process is identical in all regards, except the cuts occur at $f(C_j)$ and $f(C_k)$, in other words excluding the ϵ offset (see Fig. 8a).

Path Graph and Path Extraction The path graph for thin features uses the edges of the extracted triangle mesh. The shortest path between critical points is found using Dijkstra's algorithm (see Fig. 8a).

Iterative Refinement To produce a smoother arc, triangles that are adjacent to the initial path are subdivided and the path is re-computed (see Fig. 8b). This step is performed up to five times or when the refinement no longer changes the path (see Fig. 8c).

Discussion Compared with drawing Reeb graph arcs from contours, the thin feature ensures that the arcs stay on the surface. However, applying this approach to the entire model introduces some drawbacks. First, the arcs that are generated are *not guaranteed* to align with the function gradients. Additionally, because it operates directly on mesh triangles, the smoothness of the arcs depends on the input mesh resolution. Therefore, we employ the thin feature only when it is necessary.

5.3 Reeb Graph Assembly

For each pair of connected critical points, we generated the arcs of the Reeb graph as described above. Combining all of the arcs, the complete Reeb graph is realized. Fig. 9 shows examples of the torus and modified torus with both the boundary and interior approach.

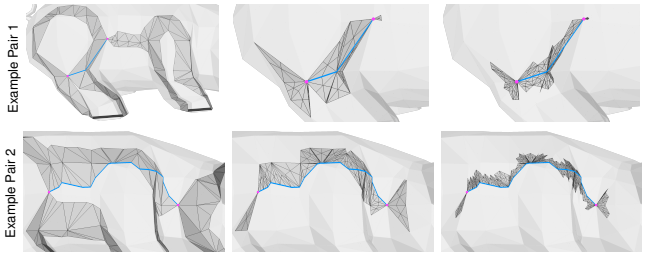


Fig. 8: Two examples of of thin features. (a) Initially, a cut is performed between the two critical points in *pink* and the shortest path, in *blue*, along the triangle edges is found. (b) Triangles adjacent to the initial path are subdivided and the path is recomputed. (c) The process is repeated until the path no longer changes or a maximum of five iterations.

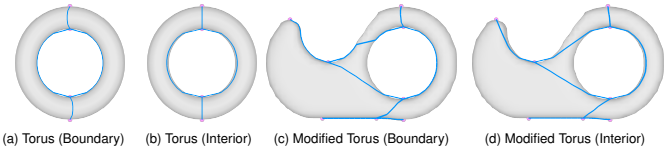


Fig. 9: Final Reeb graph visualizations for the (a,b) torus and (c,d) modified torus models with the (a,c) boundary and (b,d) interior approaches.

6 Evaluation

We perform a mixed-method evaluation of our approach and compare it to Geometric Barycenter (GB) algorithm, as implemented in TTK. We primarily frame our evaluation around the three essential properties of a Reeb graph that impact the faithfulness of its representation (see Sec. 6.1-Sec. 6.3) and one optional property (see Sec. 6.4). Through this detailed comparative analysis, we highlighted the improvements offered by GASP. We also demonstrate the impact of the user-set parameters of our approach have on the quality of the output Reeb graphs. Finally, we discuss the computational complexity of our algorithm (see Sec. 6.5).

Data and Visualizations We have utilized 30 triangle meshes from [3]. All models were normalized such that their largest dimension range was $[-1, 1]$. For each model, we generated Reeb graphs for height functions in all three principal directions (i.e., x, y, and z) and geodesic distance functions that emanate from the most extreme point in each principal direction (i.e., top, $bottom$, $left$, $right$, $back$, and $front$). While the Reeb graph arc paths are calculated using our approach, all visualizations are rendered using *ParaView* [2]. Only a subset of the datasets are discussed in the paper, but the analysis of all datasets, source code, and a video of our approach compared to GB (TTK) can be found in the supplemental materials.

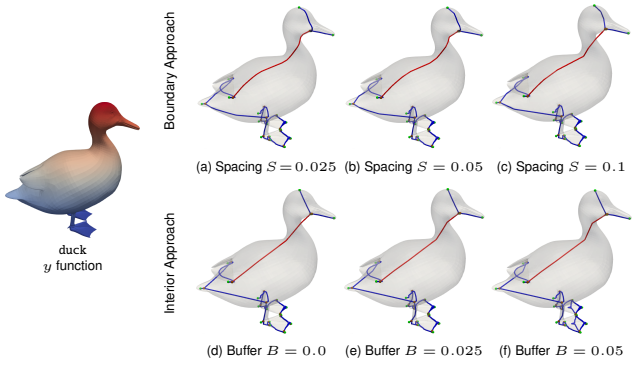


Fig. 10: Illustration of the effects of varying the (a-c) *contour spacing* for the boundary approach and the (d-f) *buffer size* for the interior approach with fixed contour spacing ($S = 0.1$). Complete Reeb graphs include *red* and *blue* arcs; *red* arcs highlight comparisons.

GASP Parameters As outlined in Sec. 4.2.2, our approach features two main variations: *boundary* and *interior*. Both approaches incorporate a *contour spacing* parameter, S, while the interior approach has an additional *buffer size* parameter, B. The smaller the contour spacing, the finer the detail in the output. Throughout the evaluation, we utilize 3 levels of contour spacing with the largest being 0.1 (see Fig. 10c), representing 5% of the $[-1, 1]$ domain, and 0.05 (see Fig. 10b) and 0.025 (see Fig. 10a) representing $2\times$ and $4\times$ the resolution, respectively. The buffer spacing determines the minimum distance an arc should be from the contour in the interior method. Again, 3 levels are utilized from 0 (see Fig. 10d), which allows arcs to touch the contour, to 0.025 (see Fig. 10e), to 0.05 (see Fig. 10f) representing 2.5% of the domain. Parameters for testing were selected such that at the lowest level of detail, the output quality would be good but noticeably different from the highest level of detail. We selected the highest level of detail such that additional detail would not be noticeable.

TTK Parameters As briefly outlined in Sec. 2.4, TTK's approach to drawing Reeb graphs utilizing GB subdivides a straight edge into multiple arc segments based on a user-defined *arc sampling* parameter. It then applies *geometric smoothing* over several iterations, controlled by another user-defined parameter. Fig. 11 illustrates the effect of varying both TTK parameters. For the evaluation, we consider two versions of TTK, both with smoothing set to 15 iterations and arc sampling set to 5 (referred to as *TTK 5/15*), representing a less smooth variant (see Fig. 11a), and arc sampling 15 (referred to as *TTK 15/15*), representing a smoother, aesthetically more pleasing version (see Fig. 11c).

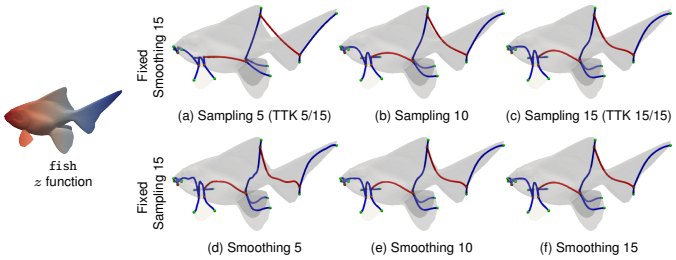


Fig. 11: Illustration of varying TTK parameters: (a-c) arc sampling and (d-f) arc smoothing. As arc sampling increases, arcs get longer. As smoothing increases, the output arcs take smoother, more aesthetically appealing routes. Complete Reeb graphs include *red* and *blue* arcs; *red* arcs highlight parameter variations.

6.1 Property 1: Arcs Confined to the Model

The first property for visualizing Reeb graphs is that arcs should ideally be contained within the model. For example, in Fig. 12c-12d, the GB Reeb graph generated by TTK has several arcs outside of the `horse` mesh. The position of these arcs outside of the mesh is important, as it may misrepresent the function, which travels on the surface of the mesh, and cause unnecessary visual clutter.

Comparison with GB GASP uses a strategies that try to constrain the Reeb graph to the mesh surface using the boundary approach (see Fig. 12a) or inside the mesh when using the interior approach (see Fig. 12b), while GB Reeb graphs extend beyond the boundary (see Fig. 12c-12d). We quantitatively evaluate the effectiveness of GASP and GB by measuring the ratio of arc $Length_{outside}/Length_{Total}$ (0 means the arc is entirely interior, while 1 indicates the arc is entirely outside the mesh). The ratio is calculated per arc, and we present the average values per Reeb graph. To measure how far an arc extends beyond the model, we calculate the sum of the areas of all arc segments outside the mesh surface. In particular, if an arc segment of length l whose endpoints are d_1 and d_2 units away from the boundary of the mesh, respectively, the area is $a = l \cdot \frac{d_1+d_2}{2}$. Lower area values indicate the extended arcs are closer to the mesh. The scatter plots in Fig. 13a (ratio of arcs outside the model) and Fig. 13d (area of arcs outside the model) show the improved performance of GASP compared to the two TTK variants of GB across different models for the same function (scatter plots for additional functions are provided in the supplement).

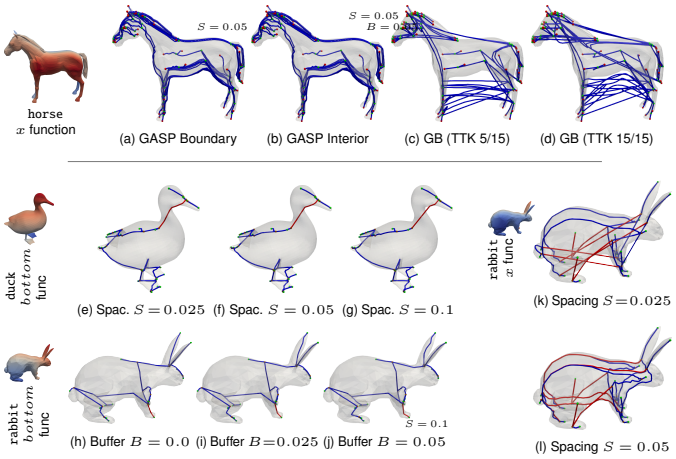


Fig. 12: Examples for Property 1, arc confined to the model. *Top*: Examples comparing (a-b) GASP arcs that remain confined to the model's boundary, while (c-d) some GB arcs go far beyond it. *Bottom*: (e-f) At lower contour spacing, the Reeb graph is more likely to remain inside the model, (g) while at larger contour spacing values, arcs are more likely to extend outside the model (e.g., near the neck). (h) When buffer size is 0, arcs around the front legs are partially outside the boundary (not easily visible in the figures). (i-j) When buffer size increases to 0.025 and 0.05, arcs move further inside the model. (k) One GASP limitation is that for critical points with small function distance and large Euclidean distance, long arcs can appear. (l) Interestingly and somewhat counterintuitively, as contour spacing increases (i.e., fewer contours are used), these long arcs disappear because topological cylinders switch from regular features to thin features. Complete Reeb graphs include *red* and *blue* arcs; *red* arcs highlight comparisons. Additional examples in supplement.

Furthermore, the GASP interior method is generally more effective than the boundary approach.

Effects of Contour Spacing and Buffer Size Contour Spacing: With GASP, the Reeb graph arc can go outside the model in two situations. It occurs most frequently when the Reeb graph arc covers non-convex portions of mesh, but it can also occur when the Reeb graph arc is traveling through thinner parts of the model (see Fig. 12g). Reducing contour spacing (i.e., increasing the number of contours per arc) can mitigate these issues (see Fig. 12e-12f). The quantitative results in Fig. 13b and Fig. 13e show that across a variety of models, larger contour spacing generally results in a slight increase in the arcs being outside of the model. The outliers will be discussed in the forthcoming limitations. Buffer Size: The buffer causes candidate points to be positioned deeper inside the mesh (see Sec. 5.2.2), causing the Reeb graph to shift inward. Consequently, a larger buffer size confines more of the Reeb graph within the model compared to a smaller buffer size. This effect is illustrated in Fig. 12i-12j. With a small buffer size (see Fig. 12h), some arcs extend outside the model. As the buffer size increases (see Fig. 12i-12j), the arcs shift inward. Fig. 13c and Fig. 13f further quantify this negative correlation, showing that a larger buffer size reduces the mean percentage of arcs extending outside the model.

Summary and Limitations Overall, GASP outperformed the TTK implementation of GB in confining Reeb graph arcs within the model. However, GASP can struggle when a large arc segment passes through a non-convex region. This can happen when a pair of critical points are close in function distance but far in Euclidean distance (see Fig. 12k). This problem can be mitigated by either decreasing the contour spacing, which adds more contours, or by increasing the contour spacing, which causes those features to switch from regular features to thin features (see Fig. 12l). It would, of course, be possible to heuristically force long edges to switch to thin features, but we chose not to make that a core feature of our implementation. These edges are also responsible for the few irregular patterns observed in Fig. 13b.

6.2 Property 2: Compact Arcs

To reduce visual clutter and make arcs easier to track, a Reeb graph should be as compact as possible by minimizing arc length.

Comparison with GB When TTK creates a Reeb graph of the model using GB, it transforms the straight lines into arcs by utilizing arc sampling and smoothing. The length of the arcs increases as the arc sampling parameter increases. Moreover, in some cases, adjusting the iteration parameter allows the arc to position itself within the model at the cost of increased length. Fig. 11 shows an example where increasing the sampling causes the arcs to respect the model boundary while also getting longer. In contrast, GASP employs the shortest path to ensure that the path taken is minimal while also respecting the boundary (up to the limits of contour spacing). An example is illustrated in Fig. 14a-14d. GB results in several unnecessarily long arcs (highlighted in *red*), with the less smooth version (see Fig. 14c) producing shorter arcs than the smoother version (see Fig. 14d). However, both variations of GASP, respect the boundary of the model while keeping their length minimal (see Fig. 14a-14b). To quantitatively measure how well a Reeb graph minimizes arc length, we measure the total length of each Reeb graph arc and compare it to the Euclidean distance between the associated critical points. The ratio $Length_{Total}/Distance_{CP}$ are calculated per arc, and smaller is generally better (1 is a straight edge). We report the average values per Reeb graph. Fig. 15a shows that for almost all cases, GASP arcs are significantly shorter than GB arcs and that the GASP interior approach arcs are shorter than boundary approach arcs.

Contour Spacing and Buffer Size Contour Spacing: In principle, increases in the contour spacing should result in a decrease in the length of arcs because of new shorter path opportunities, as can be observed in the `hand point prep` model in Fig. 14e-14g. However, in general, increases in contour spacing increase the number of thin features, which tend to be longer than their regular counterparts. Therefore, we often observed an overall increase, albeit small, in average arc length as contour spacing increased, which can be seen in Fig. 15b. Buffer Size: Increasing buffer size forces Reeb graph arcs to follow paths farther from the surface and critical points, resulting in longer arcs, as shown in Fig. 15c. This trend is observed in the Fig. 14h-14j, where arc

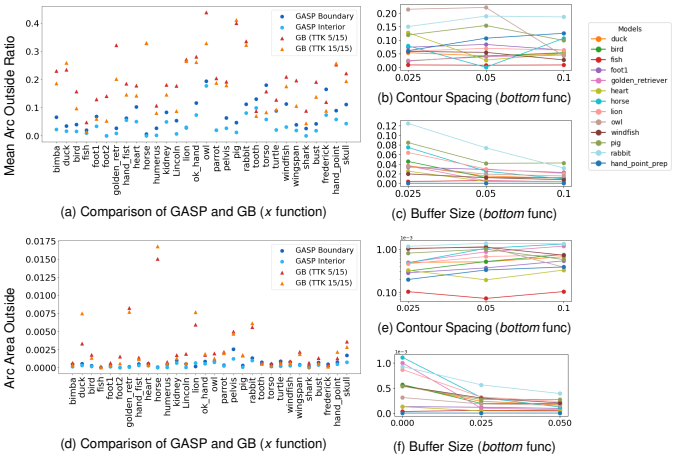


Fig. 13: Quantitative analysis for Property 1, arc confined to the model, using the (top) mean ratio and (bottom) total area of arcs outside the model (lower is better). (a/d) Comparison of GASP and GB across all models shows that for the majority, both configurations of GASP outperformed GB. (b/e) Charts illustrating the influence of the contour spacing on GASP for models used in the paper. Generally, as contour spacing increases, slightly more of the arcs end up outside of the model. The models with irregular patterns arise from variations in the number of thin and regular features as contour spacing changes. (c/f) Charts illustrating the influence of the buffer size on GASP for models in the paper. Generally, buffer size increases result in a gradual decrease in arcs outside of the model, suggesting that larger buffer sizes help constrain arcs within the model's surface. Additional examples in supplement.

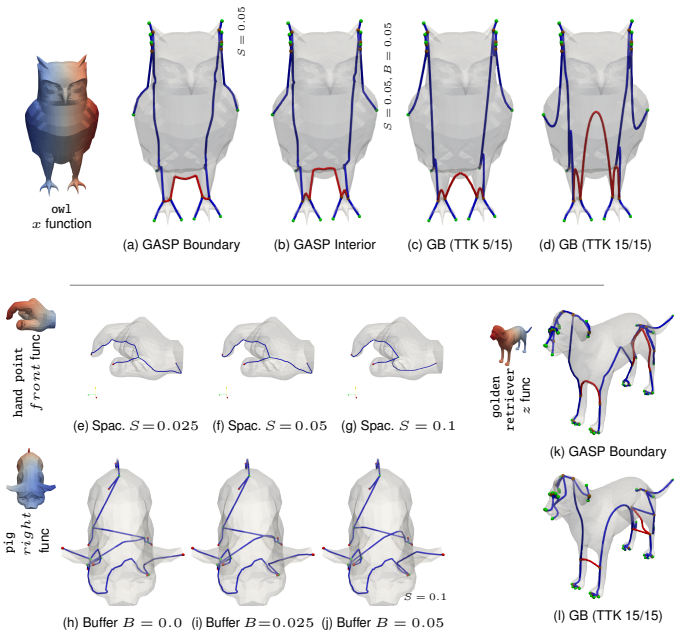


Fig. 14: Examples for Property 2, compact arc. *Top*: Examples comparing GASP and GB show that (c-d) GB can produce Reeb graph arcs that are unnecessarily longer, while GASP arcs cover a minimal distance. *Bottom*: (e-g) Compared to smaller contour spacing, e.g., (e) 0.025, (f) 0.05, larger contour spacing, e.g., (g) 0.1, causes arcs to be more subdivided, resulting in slightly shorter arc lengths. (h-j) Increases in buffer size from (h) 0.0 to (i) 0.025 to (j) 0.05 increases arc length by pushing arcs toward the interior of the model. (k-l) One limitation of (k) GASP is that it can produce long arcs in non-convex areas of high curvature compared to (g) GB. However, the GB arcs are only shorter because they go outside of the mesh, highlighting an important trade-off. Complete Reeb graphs include *red* and *blue* arcs; *red* arcs highlight comparisons. For additional models and functions, see supplement.

length slightly increases with buffer size. However, the change in the visualization is subtle and difficult to observe in the image.

Summary and Limitations Our evaluation shows that GASP generally produces shorter arcs than GB, and increases in both contour spacing and buffer size generally result in small increases in arc length. One important note about these results is that GASP does particularly well on nearby critical points because it connects the critical points

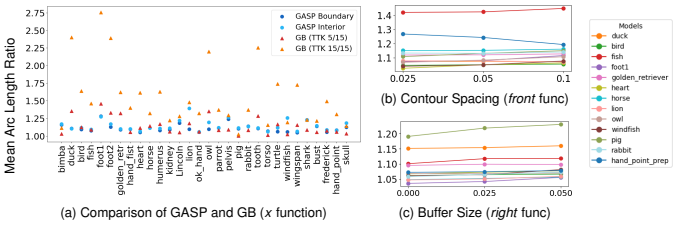


Fig. 15: Quantitative analysis for Property 2, compact arc, using the mean ratio of the arc length to the distance between critical points (lower is better). (a) Comparison of GASP and GB across all models shows that for the majority of configurations, GASP outperformed GB. (b) Chart illustrating the influence of contour spacing on GASP for the subset of models used in the paper. Increases in spacing result in increases in arc length, which is mostly attributed to an increase in thin features, which tend to be longer than their regular counterparts. (c) Chart illustrates the influence of buffer size on GASP for the same subset of models. A slight increase in arc length is observed. As buffer size increases, arcs are pushed further inside of the model, causing their length to increase. Additional examples are in the supplement.

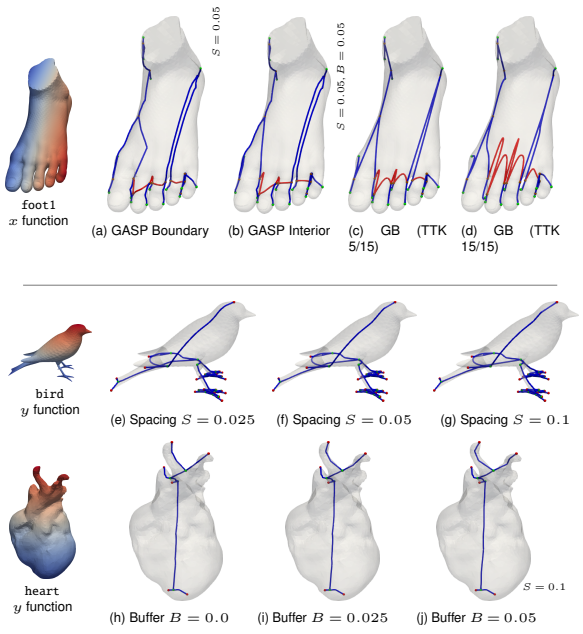


Fig. 16: Examples for Property 3, following function gradient. *Top*: Examples of GASP and GB show that the highlighted (c-d) GB arcs do not do a good job traveling in the direction of the function gradient, while the (a-b) GASP arcs do. *Bottom*: (e-g) Changes in GASP contour spacing have little impact on the alignment of the arcs with the gradient. (h-j) Similarly, changes in buffer size have little impact on the alignment of the arcs with the gradient. Complete Reeb graphs include *red* and *blue* arcs; *red* arcs highlight comparisons. For additional examples, see supplement.

more directly. However, for some longer arcs, particularly in non-convex areas of high curvature, pose challenges to GASP that may result in producing longer arcs because the model's boundary constrains it. Fig. 14l shows an example where the GB arcs in *red* are shorter than the GASP arcs in Fig. 14k.

6.3 Property 3: Following the Function Gradient

Reeb graph arcs represent the monotonic change of the function value between critical points. Therefore, the arc should ideally follow the gradient of the function. However, in practice, it is not always possible to draw a Reeb graph that completely follows the function's gradient direction because of the complex shape of the model and data.

Comparison with GB GASP creates arcs that move contour to contour, meaning that its Reeb graphs move with the gradient, such as in Fig. 16a-16b. On the other hand, the sampling and smoothing of GB (TTK) cause its Reeb graph arcs (highlighted in *red*) to move up and down (Fig. 16c-16d), nearly orthogonal to the function gradient (from left to right). To measure how well the direction of Reeb graph arcs reflect the flow direction of the function, we measure the total length in the principal direction of the function (i.e., x, y, or z) and compare it to the total length in non-principal directions (i.e., $y + z$, $x + z$, and $x + y$, respectively)[2]. Values are calculated per arc using the ratio of $(Distance_{NP_1} + Distance_{NP_2})/Length_{PD}$, where PD is the principal direction and NP_1 and NP_2 are the non-principal directions (lower is better). The results in Fig. 17a show that for most models, GASP produced arcs better aligned with the gradient than GB.

Contour Spacing and Buffer Size Contour Spacing: Our observation is that at lower contour spacing, arcs tend to be less smooth, while higher contour spacing produces longer, smoother arcs. However, in both cases, the arcs travel from contour to contour in the gradient direction, making their differences appear minimal (see Fig. 16e-16g). This small effect is further confirmed by the negligible slope across

[2]For ease of calculating the principal and non-principal direction, we only perform this analysis on height functions.

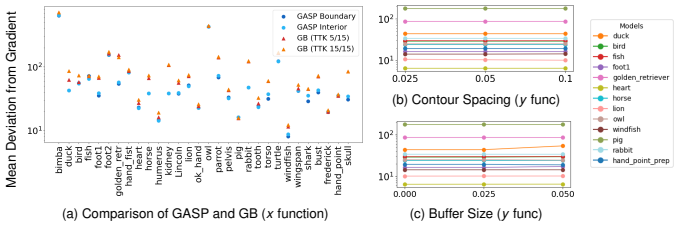


Fig. 17: Quantitative analysis for Property 3, following function gradient, using mean ratio of arc length in non-principal directions to the principal direction (log scale, lower is better). (a) Comparison of GASP and GB across all models shows that for the majority of configurations, GASP approaches yield better alignment with the gradient compared to both GB variations. (b) Charts illustrating the effect of contour spacing on GASP for the subset of models used in the paper. The near 0 slope indicates a negligible impact on the alignment arcs with the gradient. (c) Charts illustrating the effect of buffer size on GASP for the same subset of models. The near 0 slope indicates a negligible impact on the alignment arcs with the gradient. Additional examples in supplement.

all models in Fig. 17b. Buffer Size: Similarly, buffer size appears to have little to no effect on the Reeb graph's alignment with the function gradient (see Fig. 16h-16j). Fig. 17c further confirms this, showing minimal slope across all models, indicating that buffer size has a negligible influence on gradient alignment.

Summary By focusing on contours for arc generation, GASP arcs align more with the gradient than GB, better representing the direction of the function of the scalar field. Further, contour spacings and buffer size show a negligible impact on the quality of GASP Reeb graphs.

6.4 Property 4: Optional Arc Smoothness

The arc smoothness of the Reeb graph is an aesthetic quality that can provide the auxiliary benefit of a clearer representation [15, 37]. However, ideally, improvements in smoothness should be achieved without sacrificing the three faithfulness properties. Our primary goal was to improve the faithfulness of the Reeb graph visualization, a choice which necessitates a compromise in this aesthetic aspect. This trade-off is observed as lower *smoothness* in some GASP Reeb graphs. To quantify this, we calculated smoothness, M, as the mean of the tangents of a Reeb graph arc, with a corresponding value $\alpha = \frac{1}{1+M}$ in the range $[0, 1]$ (lower is smoother). Fig. 18a compares the smoothness of GASP and GB for the models. The GASP boundary approach achieves better smoothness than the interior approach, as interior arcs must deviate from the boundary and critical points, leading to more abrupt path changes (see Fig. 18b-18c), whereas GB produces smoother Reeb graphs for nearly all models (see Fig. 18d-18e).

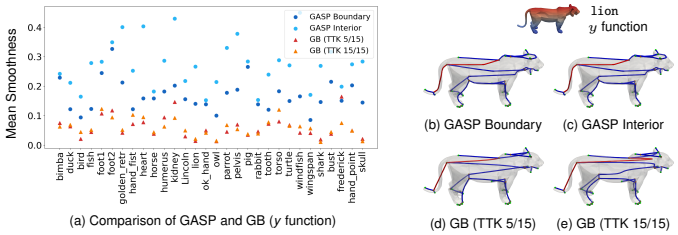


Fig. 18: Quantitative analysis and examples of smoothness for the models used in the experiment using the smoothness aesthetic measurement (lower is better). (a) Comparison of GASP and GB shows that GB has better aesthetics than GASP. Further, the GASP boundary method has better aesthetics than the interior method. For additional functions, see the supplement. (b-e) Examples of (b-c) GASP and (d-e) GB that show differences in the smoothness of arcs. Complete Reeb graphs include *red* and *blue* arcs; *red* arcs highlight comparisons.

6.5 Computational Performance

We briefly discuss the computational complexity of GASP and demonstrate the scaling effects.

Complexity Analysis We consider a mesh of T triangles and t triangles per cut. In the average case, each triangle appears in a small number of cylinders, so $T = t \times |\mathbb{C}|$ (i.e., the number of topological cylinders or the number of Reeb graph edges). Decomposing the model through cut operations iterates through all triangles, such that it takes on average $\Theta(T)$. However, in the worst-case situation, each cut operation can end up with all $t = T$ triangles, making the worst-case complexity $O(T \times |\mathbb{C}|)$. Identifying each topological cylinder uses a combination of disjoint sets, which are $O(t\alpha(t))$ (α is the inverse Ackermann function), and priority queues, which are $O(t \log(t))$, for a worst case complexity of approximately $O(T \times |\mathbb{C}| \log(t))$ or average case of $\Theta(T \log(t))$. By using binning approach from Sec. 5.1, isocontouring takes approximately $O(T)$, while building path graphs and taking the shortest path is $O(|E| \log(|V|))$ per arc ($|E|$ edges and $|V|$ vertices). Unfortunately, $|E|$ and $|V|$ are dependent on the contour spacing. All of this is to say that the average case for GASP is mostly dependent upon the number of triangles and topological cylinders ($T \times |\mathbb{C}|$), with some variation based on the data and parameters.

Computational Performance To experimentally confirm this analysis, we recorded the execution time of all models and function directions on a M1 Mac Mini for the boundary approach (see Fig. 19a) and interior approach (see Fig. 19b) across different contour spacing and buffer size. Reeb graph arcs were calculated using a thread pool of 8 simultaneous threads. The resulting performance was linear with respect to the total number of triangles in the mesh times the number of Reeb graph edges (goodness of fit, $R^2 \approx 0.8$ for all boundary configurations, and $R^2 \approx 0.7$ for all interior configurations). The performance was slightly lower as contour spacing was reduced (i.e., more contours) and as buffer size for the interior approach was reduced (i.e., more points inside the contours).

Comparison With GB Comparing the computational performance of GASP to TTK implementation of GB is an apple-to-oranges comparison, as GASP is implemented as a standalone in Python, and TTK utilizes C++ implementation of GB as part of Paraview. Nevertheless, for context, Fig. 19c compares the computational performance of GB with regression lines for several variations of GASP's boundary and interior approaches. GB's performance was, unsurprisingly, better than GASP. We also observed that GB's performance was largely parameter dependent, not data dependent.

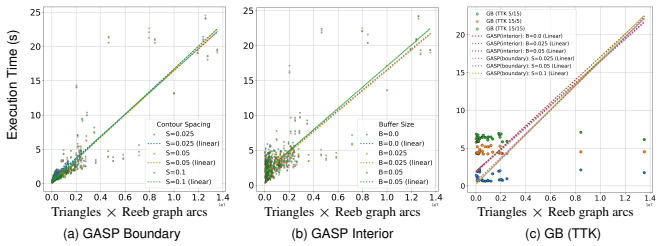


Fig. 19: Computational time of (a-b) GASP for all models and functions for the (a) boundary method (varied S) and (b) interior method (fixed S, varied B). GASP showed highly linear scaling (goodness of fit, $R^2 \approx 0.8$ for boundary and $R^2 \approx 0.7$ for interior) with respect to the number of triangles $\times$ the number of Reeb graph arcs. (c) GB (TTK, implemented in C++) performance for all models for the x function (other functions showed similar performance) along with the regression lines of GASP variations (implemented in Python). Unlike GASP, GB's performance is mostly related to parameter selection.

7 Conclusion and Future Work

Efficient calculation of the Reeb graphs has been a well-studied problem. However, the actual visualization of Reeb graphs is understudied. When abstracting and representing data, accurately representing it is of the utmost importance [3]. The geometric barycenter algorithm used

in the state-of-the-practice tool TTK, has some drawbacks, including producing arcs outside of the model, which are longer than necessary and do not optimally follow the function gradient. GASP demonstrates significant qualitative and quantitative improvements over the geometric barycenter algorithm by constraining arcs to the model's surface and utilizing a gradient-aware shortest paths.

Despite the improvement provided by GASP, the approach still presents some limitations. First, the interior method assumes that the contour is approximately planar. As the shape of the contour becomes less planar, it increases the likelihood that the interior approach will produce a low-quality Reeb graph arc. In addition, we have implemented GASP using Python. In the future, we will investigate the extension of our methods to 3D meshes, with challenges associated with Reeb graph representation arising from tetrahedra and isosurface evolution.

REFERENCES

[1] P. K. Agarwal, K. Fox, and A. Nath. Maintaining Reeb Graphs of Triangulated 2-Manifolds. In S. Lokam and R. Ramanujam, eds., *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, vol. 93 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 8:1–8:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2018. doi: 10.4230/LIPIcs.FSTTCS.2017.8 1

[2] J. Ahrens, B. Geveci, and C. Law. *ParaView: An End-User Tool for Large Data Visualization*, chap. 36, pp. 717–731. Elsevier, 2005. doi: 10.1016/B978-012387582-2/50038-1 6

[3] T. M. Athawale, B. Triana, T. Kotha, D. Pugmire, and P. Rosen. A comparative study of the perceptual sensitivity of topological visualizations to feature variations. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):1074–1084, 2024. doi: 10.1109/TVCG.2023.3326592 1, 3, 6, 9

[4] D. Auber. *Tulip — A Huge Graph Visualization Framework*, pp. 105–126. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. doi: 10.1007/978-3-642-18638-7_5 2

[5] U. Bauer, X. Ge, and Y. Wang. Measuring distance between Reeb graphs. In *Symposium on Computational Geometry (SoCG)*, pp. 464–473. ACM, 2014. doi: 10.1145/2582112.2582169 3

[6] S. Biasotti, B. Falcidieno, and M. Spagnuolo. Extended Reeb graphs for surface understanding and description. In G. Borgefors, I. Nyström, and G. S. di Baja, eds., *Discrete Geometry for Computer Imagery*, pp. 185–197. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. doi: 10.1007/3-540-44438-6_16 3

[7] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theoretical Computer Science*, 392(1):5–22, 2008. Computational Algebraic Geometry and Applications. doi: 10.1016/j.tcs.2007.10.018 3

[8] B. Bollen, P. Tennakoon, and J. A. Levine. Computing a stable distance on merge trees. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):1168–1177, 2023. doi: 10.1109/TVCG.2022.3209395 3

[9] P.-T. Bremer, G. Weber, V. Pascucci, M. Day, and J. Bell. Analyzing and tracking burning structures in lean premixed hydrogen flames. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):248–260, 2010. doi: 10.1109/TVCG.2009.69 3

[10] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, 2003. doi: 10.1016/S0925-7721(02)00093-7 2, 3

[11] F. Chen, H. Obermaier, H. Hagen, B. Hamann, J. Tierny, and V. Pascucci. Topology analysis of time-dependent multi-fluid data using the Reeb graph. *Computer Aided Geometric Design*, 30(6):557–566, 2013. Foundations of Topological Analysis. doi: 10.1016/j.cagd.2012.03.019 3

[12] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. In *Symposium on Computational Geometry (SoCG)*, pp. 263–271. ACM, 2005. doi: 10.1145/1064092.1064133 3

[13] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Loops in Reeb graphs of 2-manifolds. *Discrete and Computational Geometry*, 32(2):231–244, 2004. doi: 10.1145/777792.777844 2

[14] N. D. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on visualization and computer graphics*, 13(3):530–548, 2024. 2

[15] K. Dev, M. Lau, and L. Liu. A perceptual aesthetics measure for 3d shapes. *arXiv preprint*, 2016. 9

[16] H. Doraiswamy and V. Natarajan. Efficient output-sensitive construction of Reeb graphs. In S.-H. Hong, H. Nagamochi, and T. Fukunaga, eds., *Algorithms and Computation*, pp. 556–567. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi: 10.1007/978-3-540-92182-0_50 2

[17] H. Doraiswamy and V. Natarajan. Efficient algorithms for computing Reeb graphs. *Computational Geometry*, 42(6):606–616, 2009. doi: 10.1016/j.comgeo.2008.12.003 2

[18] H. Doraiswamy and V. Natarajan. Output-sensitive construction of Reeb graphs. *IEEE Transactions on Visualization and Computer Graphics*, 18(1):146–159, 2012. doi: 10.1109/TVCG.2011.37 1, 2, 3

[19] H. Doraiswamy and V. Natarajan. Computing Reeb graphs as a union of contour trees. *IEEE Transactions on Visualization and Computer Graphics*, 19(2):249–262, 2013. doi: 10.1109/TVCG.2012.115 4

[20] H. Edelsbrunner and J. Harer. *Computational Topology - an Introduction.* American Mathematical Society, 2010. 2, 3

[21] H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical Morse—Smale complexes for piecewise linear 2-manifolds. *Discrete and Computational Geometry*, 30(1):87–107, 2003. doi: 10.1007/s00454-003-2926-5 3

[22] H. Edelsbrunner, D. Letscher, and A. J. Zomorodian. Topological persistence and simplification. *Discrete and Computational Geometry*, 28:511–533, 2002. doi: 10.1109/SFCS.2000.892133 3

[23] A. Forsberg, J. Chen, and D. Laidlaw. Comparing 3D vector field visualization methods: A user study. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1219–1226, 2009. doi: 10.1109/TVCG.2009.126 3

[24] E. Gansner, E. Koutsofios, S. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993. doi: 10.1109/32.221135 2

[25] I. Gelbukh. Reeb graphs of circle-valued functions: A survey and basic facts. *Topological Methods in Nonlinear Analysis*, 61(1):59–81, Mar. 2023. doi: 10.12775/TMNA.2022.023 1

[26] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny. Task-based augmented Reeb graphs with dynamic ST-trees. In *Eurographics Symposium on Parallel Graphics and Visualization*, 2019. 2

[27] A. Gyulassy and V. Natarajan. Topology-based simplification for feature extraction from 3D scalar fields. In *IEEE Visualization*, pp. 535–542, 2005. doi: 10.1109/VISUAL.2005.1532839 3

[28] M. Hajij and P. Rosen. An efficient data retrieval parallel Reeb graph algorithm. *Algorithms*, 13(10):258, 2020. doi: 10.3390/a13100258 3

[29] W. Harvey, Y. Wang, and R. Wenger. A randomized O(m log m) time algorithm for computing Reeb graphs of arbitrary simplicial complexes. In *Proceedings of the twenty-sixth annual symposium on Computational geometry*, pp. 267–276, 2010. 2

[30] C. Heine, D. Schneider, H. Carr, and G. Scheuermann. Drawing contour trees in the plane. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1599–1611, 2011. doi: 10.1109/TVCG.2010.270 2

[31] A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt. Marching triangles: range image fusion for complex object modelling. In *IEEE International Conference on Image Processing*, vol. 2, pp. 381–384, 1996. doi: 10.1109/ICIP.1996.560840 5

[32] J. Khatkar, L. Clemon, R. Fitch, and R. Mettu. A Reeb graph approach for faster 3D printing. In *IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 277–282, 2022. doi: 10.1109/CASE49997.2022.9926485 1

[33] N. Kitazawa. Notes on Reeb graphs of real algebraic functions which may not be planar. *arXiv preprint*, 2023. 1

[34] D. Laidlaw, R. Kirby, C. Jackson, J. Davidson, T. Miller, M. da Silva, W. Warren, and M. Tarr. Comparing 2D vector field visualization methods: a user study. *IEEE Transactions on Visualization and Computer Graphics*, 11(1):59–70, 2005. doi: 10.1109/TVCG.2005.4 3

[35] F. Lan, S. Parsa, and B. Wang. Labeled interleaving distance for Reeb graphs, 2023. doi: 10.48550/arXiv.2306.01186 3

[36] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Computer Graphics*, 21(4):163–169, August 1987. doi: 10.1145/37402.37422 3

[37] K. T. Miura and G. RU. Aesthetic curves and surfaces in computer aided geometric design. *International Journal of Automation Technology*, 8(3):304–316, 2014. doi: 10.20965/ijat.2014.p0304 9

[38] D. Morozov, K. Beketayev, and G. Weber. Interleaving distance between merge trees. In *TopoInVis: Topological Methods in Data Analysis and Visualization*, 2013. 3

[39] M. Morse. The critical points of a function of n variables. *National Academy of Sciences*, 16(11):777, 1930. doi: 10.1073/pnas.16.11.777 3

[40] M. Natali, S. Biasotti, G. Patanè, and B. Falcidieno. Graph-based representations of point clouds. *Graphical Models*, 73(5):151–164, 2011. doi: 10.1016/j.gmod.2011.03.002 3

[41] V. Natarajan, P. Koehl, Y. Wang, and B. Hamann. Visual analysis of biomolecular surfaces. In L. Linsen, H. Hagen, and B. Hamann, eds., *Visualization in Medicine and Life Sciences*, pp. 237–255. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi: 10.1007/978-3-540-72630-2_14 3

[42] F. Nauleau, F. Vivodtzev, T. Bridel-Bertomeu, H. Beaugendre, and J. Tierny. Topological analysis of ensembles of hydrodynamic turbulent flows – an experimental study. In *Symposium on Large Data Analysis and Visualization (LDAV)*, 2022. doi: 10.1109/LDAV57265.2022.9966403 3

[43] K. Ohmori and T. L. Kunii. Three dimensional sketch for a landscape using Morse theory and Reeb graphs. In *International Conference on Cyberworlds*, pp. 178–183, 2012. doi: 10.1109/CW.2012.32 3

[44] S. Parsa. A deterministic O(m log m) time algorithm for the Reeb graph. In *Proceedings of the twenty-eighth annual symposium on Computational geometry*, pp. 269–276, 2012. 2

[45] V. Pascucci, K. Cole-McLaughlin, and G. Scorzelli. *The TOPORRERY: computation and presentation of multi-resolution topology*, pp. 19–40. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi: 10.1007/b106657_2 2

[46] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas. Robust on-line computation of Reeb graphs: simplicity and speed. In *ACM SIGGRAPH*, pp. 58–es, 2007. doi: 10.1145/1275808.1276449 1, 2, 3

[47] G. Patane, M. Spagnuolo, and B. Falcidieno. A minimal contouring approach to the computation of the Reeb graph. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):583–595, 2009. doi: 10.1109/TVCG.2009.22 2

[48] P. Rosen, A. Seth, E. Mills, A. Ginsburg, J. Kamenetzky, J. Kern, C. R. Johnson, and B. Wang. Using contour trees in the analysis and visualization of radio astronomy data cubes. In I. Hotz, T. Bin Masood, F. Sadlo, and J. Tierny, eds., *Topological Methods in Data Analysis and Visualization VI – Theory, Applications, and Software*, pp. 87–108. Springer, Cham, 2021. doi: 10.1007/978-3-030-83500-2_6 3

[49] S. Shailja, S. T. Grafton, and B. S. Manjunath. A robust Reeb graph model of white matter fibers with application to alzheimer's disease progression*. *bioRxiv preprint*, 2022. doi: 10.1101/2022.03.11.482601 1

[50] Y. Shinagawa and T. Kunii. Constructing a Reeb graph automatically from cross sections. *IEEE Computer Graphics and Applications*, 11(6):44–51, 1991. doi: 10.1109/38.103393 2

[51] Y. Shinagawa, T. Kunii, and Y. Kergosien. Surface coding based on Morse theory. *IEEE Computer Graphics and Applications*, 11(5):66–78, 1991. doi: 10.1109/38.90568 2

[52] R. Sridharamurthy, T. B. Masood, A. Kamakshidasan, and V. Natarajan. Edit distance between merge trees. *IEEE Transactions on Visualization and Computer Graphics*, 26(3):1518–1531, 2020. doi: 10.1109/TVCG.2018.2873612 3

[53] B. Strodthoff and B. Jüttler. Layered Reeb graphs for three-dimensional manifolds in boundary representation. *Computers and Graphics*, 46:186–197, 2015. Shape Modeling International 2014. doi: 10.1016/j.cag.2014.09.026 2

[54] J. Tierny. Personal Communication, 2024. 3

[55] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux. The topology toolkit. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):832–842, 2018. doi: 10.1109/TVCG.2017.2743938 2, 3

[56] J. Tierny, A. Gyulassy, E. Simon, and V. Pascucci. Loop surgery for volumetric meshes: Reeb graphs reduced to contour trees. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1177–1184, 2009. doi: 10.1109/TVCG.2009.163 2

[57] S. Wang, W. Wang, and H. Zhao. Using foliation leaves to extract Reeb graphs on surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 29(4):2117–2131, 2023. doi: 10.1109/TVCG.2022.3141764 1

[58] G. Weber, P.-T. Bremer, M. Day, J. Bell, and V. Pascucci. *Feature Tracking Using Reeb Graphs*, pp. 241–253. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi: 10.1007/978-3-642-15014-2_20 3

[59] L. Yan, T. B. Masood, R. Sridharamurthy, F. Rasheed, V. Natarajan, I. Hotz, and B. Wang. Scalar field comparison with topological descriptors: Properties and applications for scientific visualization. *Computer Graphics Forum*, 40(3):599–633, 2021. doi: 10.1111/cgf.14331 3