Singular Arrange and Traverse Algorithm for Computing Reeb Spaces of Bivariate PL Maps

Petar Hristov[1]
Scientific Visualization Group, Department of Science and Technology (ITN), Linköping University, Sweden

Ingrid Hotz
Scientific Visualization Group, Department of Science and Technology (ITN), Linköping University, Sweden

Talha Bin Masood
Scientific Visualization Group, Department of Science and Technology (ITN), Linköping University, Sweden

Abstract

We present an exact and efficient algorithm for computing the Reeb space of a bivariate PL map. The Reeb space is a topological structure that generalizes the Reeb graph to the setting of multiple scalar-valued functions defined over a shared domain, a situation that frequently arises in practical applications. While the Reeb graph has become a standard tool in computer graphics, shape analysis, and scientific visualization, the Reeb space is still in the early stages of adoption. Although several algorithms for computing the Reeb space have been proposed, none offer an implementation that is both exact and efficient, which has substantially limited its practical use. To address this gap, we introduce *singular arrange and traverse*, a new algorithm built upon the *arrange and traverse* framework [26]. Our method exploits the fact that, in the bivariate case, only singular edges contribute to the structure of Reeb space, allowing us to ignore many regular edges [40]. This observation results in substantial efficiency gains on datasets where most edges are regular, which is common in many numerical simulations of physical systems. We provide an implementation of our method and benchmark it against the original *arrange and traverse* algorithm, showing performance gains of up to four orders of magnitude on real-world datasets.

2012 ACM Subject Classification Theory of computation → Computational geometry; Mathematics of computing → Geometric topology

Keywords and phrases Computational topology, Reeb graph, Reeb space, Multivariate data, Multi-field, Geometric arrangement

Digital Object Identifier 10.4230/LIPIcs.SoCG.2026.57

Related Version *arXiv Version*: `https://arxiv.org/abs/2602.21087`

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.18759995`

Funding *Ingrid Hotz*: The author acknowledges support from the Swedish e-Science Research Center (SeRC), the ELLIIT environment for strategic research in Sweden, and the Swedish Research Council (VR) grant 2019-05487.
Talha Bin Masood: This work was primarily supported by the WASP-DDLS grant from Wallenberg Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The author acknowledges Swedish Research Council (VR) grant 2023-04806 and the Swedish e-Science Research Center (SeRC) for additional funding support.

Acknowledgements We thank Dr. Nanna H. List (KTH Royal Institute of Technology, Stockholm) for providing the MVK dataset used for evaluation in this paper. The Ethane-diol dataset is taken from the TTK dataset repository (`https://github.com/topology-tool-kit/ttk-data`) with acknowledgments to Roberto Alvarez Boto. The enzo dataset is taken from the *The Enzo Collaboration* with acknowledgments to Michael Norman (University of California, San Diego, USA).

[1] Corresponding author

1 Introduction

Visualizing and analyzing the structure of scalar-valued functions is fundamental across many disciplines, including physics [38], chemistry [3, 21], atmosphere science [29, 42], biology [32, 34], computer graphics [4, 22], and machine learning [12, 31]. Topological tools such as persistent homology, the Morse-Smale complex, and the Reeb graph are state-of-the-art methods for extracting meaningful features from scalar fields [23]. Their widespread adoption is largely due to the fact that they can be computed *robustly* and *efficiently*.

The multivariate setting, where multiple scalar functions are defined over a shared domain, is considerably more complex. In this context, the interrelated structure between the functions cannot be captured adequately by scalar field techniques. To address this limitation, the Reeb graph can be generalized to the Reeb space [18]. The Reeb space is a quotient space that contracts the components of all preimages of points, also called *fibers*. The Reeb space of a bivariate PL map over a compact PL manifold can be represented as a two-dimensional cell complex [18, 27], where the 2-cells, referred to as sheets, represent features in data.

Four algorithms have been proposed for computing the Reeb space of a PL map. The first algorithm [18] segments the domain into regions where fibers have uniform connectivity and then glues those regions appropriately to form the sheets of the Reeb space. The Jacobi fiber surfaces algorithm [40] significantly improves practical performance in the bivariate case by computing these regions only for the singular (or Jacobi) edges in the domain, where fibers can change their connectivity. The third algorithm uses a hierarchical decomposition-based approach [10] to compute the Reeb space of a bivariate PL map from a sequence of nested Reeb graphs. Finally, the arrange and traverse algorithm [26] computes combinatorial descriptions of fibers called preimage graphs and determines their topological correspondence to obtain regions of uniform fiber connectivity that match the sheets of the Reeb space.

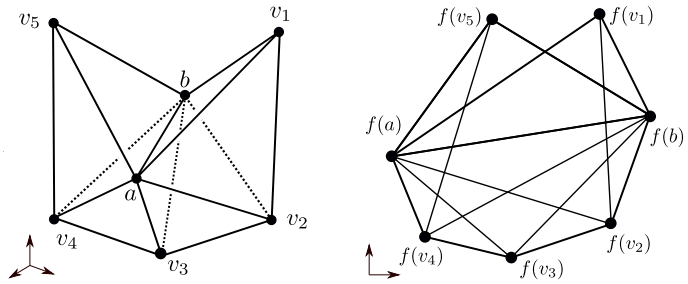
While all four Reeb space algorithms have close to quadratic algorithmic complexity in the bivariate case, their implementations suffer from either efficiency or robustness issues. The arrange and traverse algorithm [26] provides a robust implementation [24] in the bivariate case with CGAL's exact implementation of arrangements of 2D line segments [19]. However, due to its slow running time, it cannot handle real-life datasets without significant downsampling. The Jacobi fiber surfaces algorithm [40] offers better performance, but, as has been noted [10], its implementation is not exact without additional care in handling Jacobi fiber surface intersections. This paper aims to combine the strengths of these two approaches to provide a Reeb space algorithm with an implementation that is both efficient and robust.

We propose the *singular arrange and traverse* algorithm for computing the Reeb space of a generic bivariate PL map $f : |K| \to \mathbb{R}^2$, defined over a triangulation of a 3-manifold K. This is a specialization of the arrange and traverse algorithm that does not require the arrangement of the images of all edges of K in the range, only the singular ones, where the connectivity of the fibers can change. Using red-blue line segment intersection on the images of regular and singular edges, our algorithm computes the Reeb space from a subset of all preimage graphs, which we call *essential*. The worst-case running time of our algorithm is $O(N_s N_t \log N_t)$, where N_s is the number of singular edges and N_t is the number of triangles. This is a significant improvement because datasets with a low proportion of singular edges are common in practice. We provide an exact implementation and benchmark it against the arrange and traverse algorithm, the only other exact implementation known to the authors, achieving up to four orders of magnitude improvement in running time and memory footprint.

This paper is structured as follows: in Section 2, we introduce the background relevant to Reeb spaces of PL maps; in Section 3, we present the singular arrange and traverse algorithm; in Section 4, we prove its correctness and worst-case complexity; in Section 5 we describe our implementation, which we benchmark in Section 6. We conclude in Section 7.

2 Preliminaries

Let K be a triangulation of a 3-manifold and let $f : |K| \to \mathbb{R}^2$ be a piecewise linear (PL) map [35, 17]; see Figure 1. We will refer to f as a *bivariate* PL map because it can be decomposed into two scalar PL maps $f = (f_1, f_2)$, where $f_1, f_2 : |K| \to \mathbb{R}$. We assume that f is *generic*, in the sense that the images of no three vertices are collinear and the images of no three edges intersect at a single point in their interiors [25]. We refer to the images of edges under f in the range as *segments* and to the image of vertices under f as *vertex points*. For an edge e with endpoints a and b, we will also refer to e as ab.



(a) Tetrahedral mesh. **(b)** Image in the range.

Figure 1 Example tetrahedral mesh (a) and its image in the range under a bivariate PL map (b).

The *upper link* of an edge ab is a sub-complex of the link of ab induced by the vertices v in the link, whose vertex points $f(v)$ have a positive orientation with respect to $f(a)$ and $f(b)$ [27]. The *lower link* is defined analogously by the vertex points with negative orientation. In general we will assume that the endpoints of an edge or a segment are ordered lexicographically in order to establish a canonical orientation. For example, the upper link of ab in Figure 1 is $\{v_1, v_5\}$ and the lower link is $\{v_2, v_3, v_4, v_2v_3, v_3v_4\}$.

An edge is *regular* if both its lower and upper links have one simply connected component. Otherwise, that edge is *singular*, or Jacobi [16]. A vertex is *singular* if it is incident to at least one singular edge and *regular* otherwise. We call the images of singular edges and vertices in the range *singular segments and vertex points* respectively. The collection of singular edges and vertices is a one-dimensional subcomplex of K, which we call the *singular set* S_f.

A *fiber* is the preimage of a point $f^{-1}((x_1, x_2)) \subset |K|$ for $(x_1, x_2) \in \mathbb{R}^2$, which we call a *fiber point* (see Figure 2). A simplex intersected by a fiber is called *active* with respect to that fiber. We refer to the connected components of fibers as *fiber components*. Since fiber components can change their connectivity only at the singular set [27, 40], those that intersect S_f are called *singular*, and those that do not are called *regular*. Regular fiber components are polylines, while singular fiber components are either an isolated point or a number of polylines glued together at exactly one or two points [24]. Fibers with only regular components are *regular*, otherwise, they are *singular*.

Singular edges, analogous to critical points in the univariate case, can be categorized in the following way: *definite* edges create or destroy fiber components (analogous to minima and maxima) and *indefinite* edges merge or split fiber components (analogous to saddles) [27, 26]. Indefinite edges that split or merge exactly two components are referred to as *simple*. The singular set is called *simple* [18, 10] when it is a collection of polylines and all indefinite edges are simple. Non-simple singular edges can be unfolded into multiple simple ones (like monkey saddles), but that requires the addition of new simplices to the input mesh [16].

The *Reeb space* R_f of a PL map f is a topological structure that contracts each fiber component to a single point. It is the quotient space of $|K|$ with respect to the equivalence relation $\sim$, where $x \sim y$ if and only if x and y belong to the same fiber component. The

Reeb space is a part of the Stein factorization $f = \pi \circ q$, where $q : |K| \to R_f$ is the quotient map with respect to $\sim$ and π is the natural map such that the diagram commutes [30]. For computational purposes, the Reeb space of a bivariate PL map can be represented as a two-dimensional cell complex whose 2-cells, referred to as *sheets*, are glued together along the image of the singular set under q [9]. For an example see Figure 2.

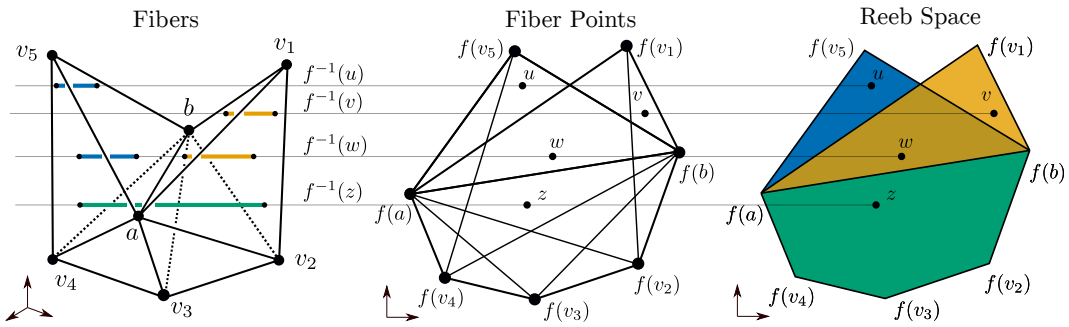


Figure 2 Examples of the change of connectivity of four fibers at the indefinite edge ab and the definite edges av_1 and bv_5. The Reeb space of f has three sheets (blue, yellow and green) – one for each fiber component. To visualize the Reeb space we map its sheets to the range and overlay them.

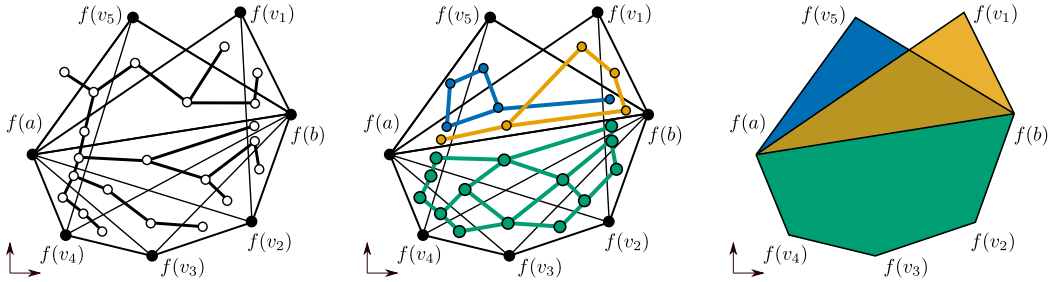
2.1 Arrange and Traverse Algorithm

The Reeb space of a PL map can be computed with four different approaches, which we outlined in Section 1. We will review the *arrange and traverse* algorithm [26] in detail because it forms the foundation for our new approach. The *arrange and traverse* algorithm computes the Reeb space by combinatorially representing fibers and tracking how their connectivity changes as they intersect the edges of K. Fibers are represented combinatorially with a preimage graph, similar to Parsa's optimal Reeb graph algorithm [33]. The *preimage graph* of a fiber $f^{-1}((x_1, x_2)) \subset |K|$ for a point $(x_1, x_2) \in \mathbb{R}^2$, which is not in the image of an edge or vertex, is a graph whose vertices are the active triangles with respect to that fiber and whose edges connect triangles which are both the faces of the same tetrahedron.

In the first stage of this algorithm, the edges of K are mapped to the range via f, and their geometric arrangement A is computed (see Section 2.2). Within each face F of A, all fibers have the same active triangles, so they can all be represented by a single *preimage graph*. In the second stage, the faces of A are traversed in order to compute their preimage graphs. The difference in the connectivity of the preimage graphs of two faces F_1 and F_2, which are incident via a segment $f(ab)$, where ab is an edge of K, is locally determined by the upper and lower link of ab. Assume that we cross from F_1 to F_2 via $f(ab)$ by crossing from the half-plane with negative orientation with respect to $f(a)$ and $f(b)$ to the half-plane with positive orientation. Then all triangles abv, where v is in the lower link of ab, are removed from the preimage graph of F_1, and all triangles abv', where v' is in the upper link of ab, are added to the preimage graph of F_1 to form the preimage graph of F_2. For the opposite direction of travel, all triangles abv are added and all triangles abv' are removed.

Once all preimage graphs are computed, the *correspondence graph* H is constructed. The vertices of the correspondence graph are the fiber components of the preimage graphs of all faces. Edges connect vertices representing fiber components when there is a correspondence between them – that is, when a fiber component passes from one face to another without any change in connectivity. Each sheet in the Reeb space is a collection of faces such that the connected components of the preimage graphs of those faces are in the same connected component in the correspondence graph. The adjacency of the sheets in the Reeb space is determined by the adjacency of the faces that comprise them.

For an example of stages of the arrange and traverse algorithm refer to Figure 3. In the bivariate case, the only geometric computations performed by this algorithm are in 2D – computing the arrangement and orientation tests for upper and lower links. The rest of the computation is entirely combinatorial. This is a significant advantage over other Reeb space algorithms, as it allows the use of only 2D geometric predicates [25]. An exact open-source implementation using CGAL can be found on GitHub [24]. Finally, this algorithm can be extended to higher dimensional domains K^d and ranges $\mathbb{R}^k$ such that $d > k$.



(a) Arrangement traversal. **(b)** Correspondence graph. **(c)** Reeb space.

Figure 3 The key stages of the arrange and traverse algorithm [26] for Figure 1.

2.2 Segment Intersections and Arrangements

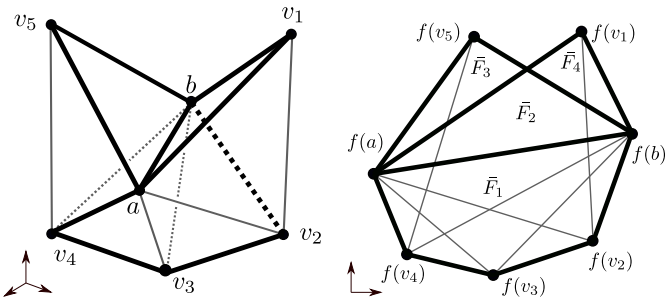
The intersections of n line segments in the plane can be computed in time $O(n \log n + k)$, where k is the number of intersections [11, 2]. A subdivision of the plane by segments into vertices, edges, and faces is called an *arrangement* and can be computed in $O(n \log n + k \log n)$ time [14, 20] and represented with a *doubly connected edge list (DCEL)*. The special case of reporting the intersections between two sets of segments, typically labeled *red* and *blue*, can be solved in $O(n \log n + t)$ time, where t is the number of intersections between the two sets.

3 Singular Arrange and Traverse Algorithm

The *key idea* of the algorithm we propose is that *not all faces and preimage graphs are needed* to compute the Reeb space in the arrange and traverse algorithm (see Section 2.1). The only faces where the connectivity of fibers can change are the ones that are incident to at least one singular segment or singular vertex point. We call these faces *essential* because they outline a part of the boundary of at least one sheet of the Reeb space. More precisely, they outline the images of sheets in the plane under π. We refer to all other faces as *nonessential* because they do not contribute to the shape of the sheets. Our goal is to compute the Reeb space by *using only the essential faces* and their preimage graphs.

3.1 Definitions

Let K be a triangulation of a 3-manifold and let $f : |K| \to \mathbb{R}^2$ be a generic PL map. Let S_f be the singular set of f and let R_f be the Reeb space of f (see Section 2). We *do not* assume that the singular set is simple. For clarity of exposition, we will assume that $f(S_f)$ is connected – we will deal with the case where it is not in Section 3.3. We will refer to preimage graphs as *fiber graphs*, since fibers are defined as preimages of points under f. Let $A = \text{Arrangement}(f(K^{(1)}))$, where $K^{(1)}$ is the 1-skeleton of K, be the arrangement of all segments, and let $\bar{A} = \text{Arrangement}(f(S_f))$ be the arrangement of all singular segments. We will refer to A and $\bar{A}$ as the *full* and *singular* arrangement respectively (see Figure 4).



(a) Singular edges (in bold). **(b)** Singular segments (in bold).

Figure 4 The singular set (a) of Figure 1 and its image in the range (b). The arrangement $\bar{A}$ of the singular segments has four bounded faces, each subdivided by the faces of the full arrangement.

Consider a face of the singular arrangement $\bar{F} \in \bar{A}$. The fibers that correspond to points in the interior of $\bar{F}$ are homeomorphic to one another because they are all regular – no change in connectivity happens to any of them. Since $\bar{A}$ is an arrangement of a subset of the segments used in A, $\bar{F}$ is subdivided into a set of faces in A such that $\bar{F} = \cup_i F_i$, where $F_i \in A$. We describe the connectivity of fibers inside $\bar{F}$ by grouping the fiber graphs of all faces that subdivide $\bar{F}$ into a fiber graph class.

▶ **Definition 1** (Fiber Graph Class). *Let $\bar{F}$ be a face of the singular arrangement $\bar{A}$ and let $\{F_i\}_i$ be the set of faces of the full arrangement that are in $\bar{F}$. The fiber graph class $\bar{G}_{\bar{F}}$ of $\bar{F}$ is the set of fiber graphs $\bar{G}_{\bar{F}} = \{G_{F_i}\}_i$, where G_{F_i} is the fiber graph of F_i.*

Fiber graph classes allow us to *bundle* together fibers with the same topology to examine their connectivity in bulk. For an example see Figure 5. In order to examine that connectivity, we define the notion of a component of a fiber graph class.

▶ **Definition 2** (Fiber Graph Class Component). *Let $\bar{G}_{\bar{F}} = \{G_{F_i}\}_i$ be the fiber graph class of a face $\bar{F}$ of $\bar{A}$. A component $\bar{C}$ of a fiber graph class $\bar{G}_{\bar{F}}$ is a set $\{C_i\}_i$, where each C_i is a connected component of G_{F_i}, such that any two connected components C_i and C_j of $\bar{C}$ correspond to each other (see correspondence in Section 2.1).*

The connected components of fiber graphs within a single fiber graph class component are in the same connected component of the correspondence graph H. In order to determine how the components of fiber graph classes change their topology across the faces of the singular arrangement we define their correspondence.

▶ **Definition 3** (Fiber Graph Class Component Correspondence). *Two components $\bar{C}$ and $\bar{C}'$ of two fiber graph classes $\bar{G}_{\bar{F}}$ and $\bar{G}_{\bar{F}'}$ correspond to each other if there exist two connected components $C \in \bar{C}$ and $C' \in \bar{C}'$ that correspond to each other.*

We can describe the connectivity of fiber graph class components across all faces of the singular arrangement with the singular correspondence graph.

▶ **Definition 4** (Singular Correspondence Graph). *The singular correspondence graph $\bar{H}$ is a graph whose vertices are the set of all fiber graph class components and whose edges connect corresponding ones (see Definition 3).*

The sheets of the Reeb space have a one-to-one matching with the connected components of the singular correspondence graph. The geometry of a sheet of the Reeb space, or its image in the plane via π, is then the union of all faces $\cup_i \bar{F}_i$ whose fiber graph classes have components that span the connected component of $\bar{H}$ that matches that sheet.

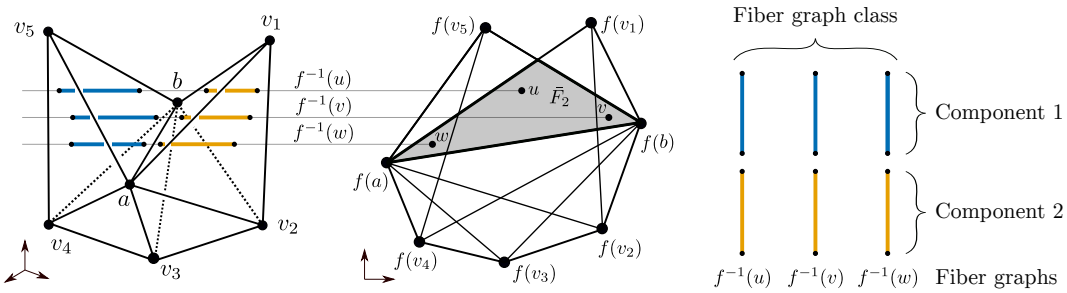


Figure 5 The fiber graph class of the face $\bar{F}_2$ is the collection of the fiber graphs of faces of the full arrangement that are in $\bar{F}_2$. All yellow (and respectively blue) fiber components correspond to one another; they change continuously from one to another as a fiber point moves around $\bar{F}_2$. All corresponding fiber graph components form a component of the fiber graph class of $\bar{F}_2$.

By reframing the computation of the Reeb space in terms of fiber graph classes, their components, and the singular correspondence graph $\bar{H}$, we can optimize running time and memory efficiency in several ways. First, we can completely skip the computation of all nonessential fiber graphs because the correspondence between two fiber graph class components can only be established with components of essential fiber graphs. Second, we can save a significant amount of memory, which is the primary bottleneck of the arrange and traverse algorithm, by representing every fiber graph class component as a single vertex in $\bar{H}$. Finally, since all essential faces are incident to singular segments, we do not have to compute the full arrangement A, only the singular one $\bar{A}$. Then we can identify the essential faces by finding the intersections between $\bar{A}$ and the regular segments.

3.2 Algorithm Description

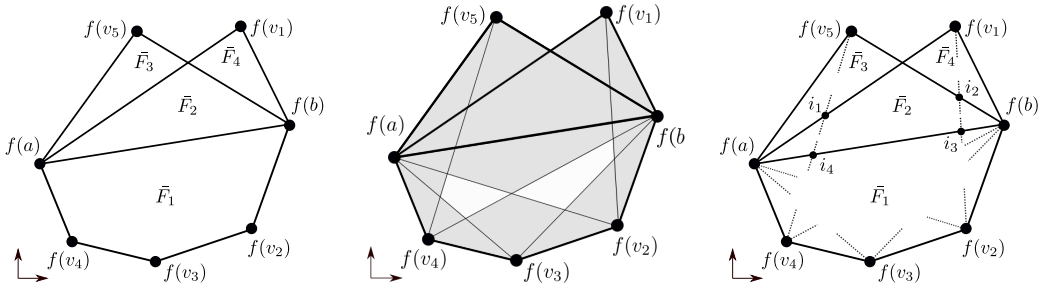
We describe the *singular arrange and traverse algorithm* in three stages. First, we compute all essential faces in the *geometric preprocessing stage*. Then, we compute their fiber graphs and the singular correspondence graph in the *topological traversal stage*. Finally, we compute the geometry of the sheets and their adjacency in the *geometric postprocessing stage*.

Stage I. Geometric Preprocessing

In the first stage, we compute all essential faces *implicitly* without using the full arrangement A. To do so, we compute the singular set S_f, then map it to the range to compute the singular arrangement $\bar{A}$, and finally we compute the intersection of $\bar{A}$ and the regular segments. We store all regular segments that intersect the boundary $\partial\bar{F}$ of each face $\bar{F}$ of $\bar{A}$ in a *circular counterclockwise ordered list* $Q_{\bar{F}}$.

When two regular segments intersect the interiors of two different half-edges or vertices of $\bar{F}$ we use the counterclockwise order defined on the boundary of $\bar{F}$ by the DCEL data structure of $\bar{A}$. When two regular segments intersect the interior of the same half-edge of $\bar{F}$, we compare the barycentric coordinates of the points of intersection. When two regular segments share a vertex of $\bar{F}$ as a common endpoint, we use their radial ordering. We use our running example to illustrate this in Figure 6, where $Q_{\bar{F}_2} = \{f(v_1v_2), f(v_1v_2), f(v_4v_5), f(v_4v_5)\}$.

Each essential face $F \subseteq \bar{F}$ is implicitly represented with the part of its boundary that intersects $\partial\bar{F}$. This can be identified by pairs of consecutive regular segments in $Q_{\bar{F}}$. Note also that an essential face may be represented multiple times when $\partial F \cap \partial\bar{F}$ has multiple connected components. Such is the case for the essential face whose boundary vertices include i_1, i_2, i_3 and i_4 in the middle of face $\bar{F}_2$ in Figure 6c.



(a) Singular arrangement. **(b)** Essential faces (shaded). **(c)** Regular segment intersections.

Figure 6 Our algorithm computes the arrangement $\bar{A}$ of all singular segments (a), then idenfities all essential faces (b) by computing the intersections between $\bar{A}$ and all regular segments (c).

Stage II. Topological Traversal

In the second stage, we traverse the faces of the singular arrangement along their shared segments using breadth-first search (BFS) to compute the essential fiber graphs and the singular correspondence graph $\bar{H}$. Starting with the unbounded face outside $\bar{A}$, whose fiber graph is empty, we visit one of its incident bounded faces $\bar{F}$ via a segment $f(ab)$, where ab is an edge in K. The edge ab is definite (see Section 2) because $f(ab)$ is on the boundary of $\bar{A}$, so we add the triangles abv, where v is in the upper (or lower, depending on the direction of travel) link of ab, to the first essential fiber graph in the fiber graph class of $\bar{F}$.

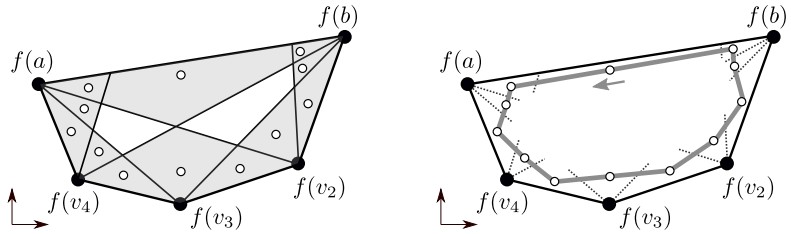
Next, we compute all other essential fiber graphs of $\bar{F}$ with a loop operation. A *loop operation* implicitly loops around the inner boundary of $\bar{F}$ to visit all essential faces in $\bar{F}$ using the order $Q_{\bar{F}}$ we have established in the geometric preprocessing stage. During the loop of $\bar{F}$, we cross from one essential face to the next via the regular segments listed in $Q_{\bar{F}}$. We add and remove the appropriate triangles related to the upper and lower links of the edges that map to those segments (see Section 2.1).

After we have computed the essential faces of $\bar{F}$, we visit its neighboring faces. We compute the new fiber graphs by adding and removing triangles according to the upper and lower links of the singular edges that map to shared segments on the boundaries of incident singular faces. We continue to loop each face as we visit it in the BFS traversal until we have computed the fiber graphs of all essential faces. For an example of the loop operation, see Figure 7, and for an example of the overall traversal, see Figure 8.

Finally, we construct the singular correspondence graph $\bar{H}$. Let $\bar{F}$ and $\bar{F}'$ be two singular faces which are incident via the segment $f(ab)$, and let F and F' be any two essential faces in $\bar{F}$ and $\bar{F}'$ respectively, which are also incident via $f(ab)$. We use the upper and lower links of ab to determine which components in the fiber graphs of F and F' are not affected by the transition from F to F' via $f(ab)$ and establish correspondence between them [26]. This allows us to establish correspondence between the components of the fiber graph classes of $\bar{F}$ and $\bar{F}'$. We do this for all pairs of incident singular faces.

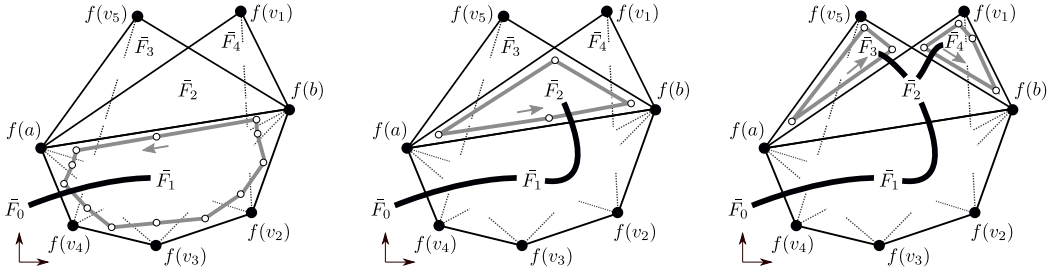
Stage III. Geometric Postprocessing

Once we have computed the singular correspondence graph, we can compute the geometry and adjacency of the sheets of the Reeb space R_f. By the geometry of a sheet of the Reeb space, we mean its image into the plane via $\pi : R_f \to \mathbb{R}^2$. Let L be a sheet that matches a connected component of $\bar{H}$. Then, the geometry of L is exactly the union of faces in the singular arrangement whose fiber graph class components span that component of



(a) Essential fiber graphs of $\bar{F}_1$. **(b)** Looping all essential fiber graphs.

Figure 7 Computing all essential fiber graphs of $\bar{F}_1$ from Figure 4 with a loop operation.



(a) Looping $\bar{F}_1$. **(b)** Looping $\bar{F}_2$. **(c)** Looping $\bar{F}_3$ & $\bar{F}_4$.

Figure 8 Traversal of the singular arrangement and looping each face.

$\bar{H}$. Sheets are adjacent when they have incident faces in the singular arrangement. The singular correspondence graph and the Reeb space of our running example are given in Figure 9. Notice how much smaller the singular correspondence graph is compared to the correspondence graph in Figure 3b.

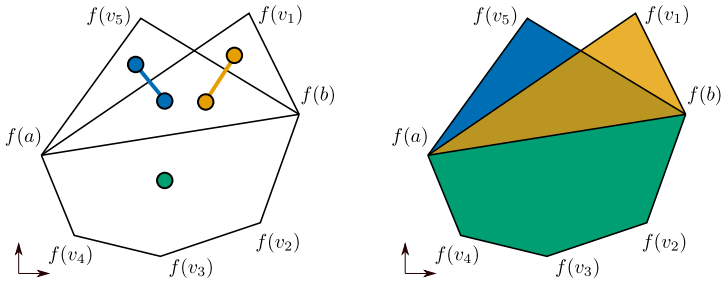
3.3 Nested Faces

We now address the case where the image of the singular set is *not* connected. In this case, either the unbounded face of the singular arrangement has more than one hole, or some bounded face has at least one hole, or both. If the unbounded face of the arrangement has multiple holes, we can traverse them independently. If a bounded face $\bar{F}_1$ has a nested face $\bar{F}_2$ (a hole), we find the shortest path in the 1-skeleton of K that connects a vertex that maps to $\partial\bar{F}_1$ to a vertex that maps to $\partial\bar{F}_2$. We mark the regular edges on this path as *pseudo-singular* and compute the arrangement of all singular and pseudo-singular segments, which is now guaranteed to be connected. After this, our algorithm proceeds as usual.

Note that this may introduce additional faces in the singular arrangement, such as $\bar{F}_3$ in Figure 10, whose nonessential fiber graphs will be computed by our algorithm. We will demonstrate that this is not an issue in practice and that only a negligible number of pseudo-singular edges are added for real-life datasets we evaluate in Section 6.

3.4 Higher Dimensional Domains

Finally, we describe an extension of our algorithm for computing the Reeb space of a PL map $f : |K| \to \mathbb{R}^2$, where K is a triangulation of a d-manifold with $d > 3$. Due to the skeleton lemma [18], the Reeb space of $f : |K| \to \mathbb{R}^2$ is homeomorphic to the Reeb space of $f' : K^{(3)} \to \mathbb{R}^2$, which is the restriction of f to the 3-skeleton of K. In order to compute the Reeb space of f', we compute all singular edges of K [16] and discard all simplices



(a) Correspondence graph. **(b)** Reeb space.

Figure 9 Singular correspondence graph and Reeb space of the toy example from Figure 1.

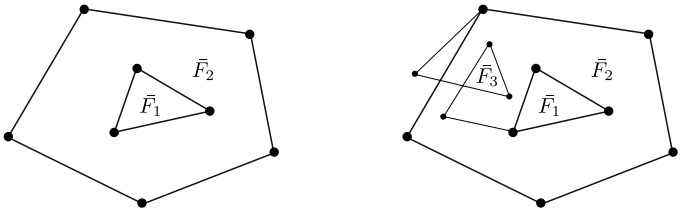


Figure 10 Connecting nested faces by adding pseudo-singular segments.

of dimension bigger than three from K. Then, we map the singular edges to the plane, compute their arrangement, and traverse it to compute the singular correspondence graph. The only difference is that we need a new subroutine to track the connectivity of fiber graphs. This is because the regular fibers of f' are not PL curves but the 1-skeleton of triangulated $(d-2)$-manifolds. This requires more involved data structures for dynamic graph connectivity tracking, similar to how Reeb graph algorithms handle the scalar case [15, 33].

4 Proofs

In this section we prove the correctness and worst-case running time of our algorithm using our definitions from the start of Section 3.

4.1 Correctness

First, we prove the correctness of each stage in our algorithm (see Section 3.2).

▶ **Lemma 5** (Essential Faces). *The singular arrange and traverse algorithm correctly identifies all essential faces.*

Proof. Let F be an essential face of the full arrangement A, and let $\bar{F}$ be the face of the singular arrangement $\bar{A}$ that contains F. The case when $F = \bar{F}$ is trivial because then F is already correctly represented in $\bar{A}$. Suppose that $F \subset \bar{F}$. Let B be a connected component of $\partial F \cap \partial \bar{F}$. Then B is a path from i_1 to i_2, where i_1 and i_2 are the intersection points of two regular segments s_1 and s_2 with $\partial \bar{F}$.

First, we examine the case where $B = i_1 = i_2$. In the DCEL of the full arrangement A, the half-edges of the boundary cycle of F that originate from s_1 and s_2 must be consecutive. This means that no other regular segment comes between s_1 and s_2 in the counterclockwise order of segments around i_1 in A. Otherwise, that other segment would be on the boundary of F. Therefore, the segments s_1 and s_2 are present as a consecutive pair in the circular list

$Q_{\bar{F}}$ of regular segments of $\bar{F}$. If on the other hand $i_1 \neq i_2$, then B is subdivided into at least one half-edge in the DCEL of A. No other regular segment intersects the interior of B, because then the essential face F would have been subdivided differently in A. Similarly to the previous case, other regular segments whose endpoint is either i_1 or i_2 are either before s_1 or after s_2 in the counterclockwise order of segments around those vertex points. Therefore, s_1 and s_2 are consecutive in $Q_{\bar{F}}$.

Conversely, we can use a similar argument to show that each consecutive pair of regular segments in $Q_{\bar{F}}$ can be uniquely identified with an essential face. ◀

▶ **Lemma 6** (Essential Fiber graphs). *The singular arrange and traverse algorithm correctly computes all essential fiber graphs.*

Proof. Since all essential faces are identified correctly by Lemma 5, our algorithm can traverse them, analogously to the arrange and traverse algorithm, but in a different order. To see why this does not affect the correctness, consider a path P in the range that only intersects regular or singular segments, from the unbounded outside face to an essential face. Any such path gives rise to a sequence of additions and removals of triangles, based on the upper and lower links of edges whose segments P intersects. This sequence can be used to build the fiber graph of the essential face, which is the same as the fiber graph computed by our algorithm. Additional pseudo-singular segments in $\bar{A}$ (see Section 3.3) do not affect the correctness because our algorithm still identifies such a path P correctly. ◀

▶ **Lemma 7** (Singular Correspondence Graph). *The connected components of the singular correspondence graph correctly represent the sheets of the Reeb space.*

Proof. The singular correspondence graph $\bar{H}$ can be obtained from the correspondence graph H by contracting all vertices that are associated with the components of fiber graphs within the same fiber graph class. A contraction of a graph does not affect its connectivity. Each vertex of $\bar{H}$ simply represents a larger area in the range of fiber points whose fibers have the same topology. The singular correspondence graph can be computed correctly by establishing correspondence only between the essential fiber graphs because nonessential fiber graphs do not introduce correspondences with new fiber graph classes. They only correspond directly to other components within their own class, and it is precisely these correspondences that are lost in the contraction of H to H'. The correctness of the correspondence procedure between fiber graphs is proven in the original arrange and traverse algorithm [26]. ◀

▶ **Corollary 8** (Reeb space). *The singular arrange and traverse algorithm correctly computes the Reeb space of a generic PL map $f : |K| \to \mathbb{R}^2$, where K is a triangulation of a 3-manifold.*

Proof. By Lemma 5 and Lemma 6, the essential faces and essential fiber graphs are computed correctly. By Lemma 7, the singular correspondence graph represents the sheets of the Reeb space and is computed correctly. Therefore, the singular arrange and traverse algorithm correctly computes the Reeb space of f. ◀

4.2 Complexity

Let N_e and N_t be the number of edges and triangles in K respectively. Let N_s be the number of singular edges. Let k_s be the number of intersections between all singular segments, and let k_r be the number of intersections between all regular and singular segments. First, we compute the arrangement of all singular segments in time $O((N_s + k_s) \log N_s)$. Next, we compute the number of intersections between all regular and singular segments and

their relative order in time $O((N_e + k_r) \log N_e)$. We use a red-blue segment intersection algorithm (see Section 2.2) and sort the intersection points along each singular segment using their barycentric coordinates. This concludes the geometric part of our computation. What remains is the complexity of computing the essential fiber graphs and the singular correspondence graph, which we demonstrate with the following lemma.

▶ **Lemma 9.** *Computing the essential fiber graphs and the singular correspondence graph has worst-case running time* $O(N_s N_t \log N_t)$.

Proof. Let $k(e)$ be the number of times an edge $e \in K$ is visited by the algorithm. A visit can mean either crossing a singular segment in the traversal from one face to another in the singular arrangement, or crossing a regular segment in the looping of a single face. When we visit an edge with degree $\deg(e)$, where we define the degree of an edge as the number of its incident triangles, we add and remove at most $\deg(e)$ triangles from the relevant fiber graphs, and perform at most $\deg(e)$ connectivity queries. Each of those operations takes at most $O(\log N_t)$ time if we represent each fiber graph with a balanced search tree [13].

The overall worst-case running time can then be obtained by summing the costs of all visits across all edges $O\big(\sum_{e \in E} k(e) \deg(e) \log(N_t)\big)$, where E is the set of edges in K. Since we do not consider any intersections between regular segments, each segment is intersected, and hence visited, at most N_s times. Using $k(e) \leq N_s$ we obtain $O\big(N_s \log(N_t) \sum_{e \in E} \deg(e)\big)$. Using a straightforward counting argument, we can show that $\sum_{e \in E} \deg(e) = 3N_t$. This leads to a worst-case running time of $O(N_s N_t \log N_t)$. ◀

Computing the connected components of $\bar{H}$ can be done in linear time in its size with a graph search. The geometry of the sheets of the Reeb space can be obtained by combining all faces of the singular arrangement that belong to each sheet. Thus, *the overall complexity of our algorithm is* $O(N_s N_t \log N_t)$. Note that, practical performance depends primarily on k_s and k_r, which determine how many times an edge is actually visited.

When $O(N_s) = O(N_e)$ our algorithm has the same running time as arrange and traverse. However, our algorithm is significantly more efficient for practical data sets where the number of singular edges is orders of magnitude less than the number of regular edges, as we will demonstrate in our evaluation in Section 6.

5 Implementation

We provide an open-source implementation of the singular arrange and traverse algorithm in C++. The input is a tetrahedral mesh with two scalar values defined at each vertex, represented in the `.vtu` file format with VTK [36]. For the geometric preprocessing operations we used CGAL 6.0.2 [39] with the exact predicates and exact constructions kernel [5]. We compute the red-blue intersections with CGAL's `box_intersection_d` [28], which has quadratic complexity. While, this is not optimal, it has an efficient implementation.

We track changes in the connectivity of fiber graphs with a graph search over the affected [26] fiber components. While this operation is not logarithmic in the size of the fiber, in practice most of the connectivity computations are done within the loop operation for a face of the singular arrangement, so all triangles can be directly added or removed from the same fiber graph component without affecting the connectivity.

Another important practical optimization, similar to arrange and traverse, is to only keep the fiber graphs at the wave front of the BFS search in memory since those are the only ones needed to establish correspondence to create the singular correspondence graph [26]. While

degenerate cases, which occur when the input PL map is not generic (see Section 2), can be handled robustly with simulation of simplicity [25], random numerical perturbation with double floating-point precision was sufficient for our tests.

6 Evaluation

In this section, we perform a practical evaluation of the efficiency of our algorithm in terms of running time and memory. We benchmark our algorithm against the arrange and traverse algorithm [26], since that is the only other open-source exact Reeb space implementation known to the authors. Our tests were performed on a PC with two Intel Xeon Gold 6130 processors with 32 cores clocked at 2.1GHz and 384GB memory with Rocky Linux 9.6. We measured execution time using the C++ `chrono` library and estimated memory usage via the process's resident set size by reading `/proc/self/status`.

We benchmark our algorithm on three numerical simulation datasets – MVK [8, 37], ethane-diol [7, 41] and enzo [6, 7]. MVK comes from a quantum chemistry simulation studying photo-induced changes in electron transition orbitals. Ethane-diol is a molecular dynamics simulation of a molecule's electron density field and its reduced gradient. Lastly, the origin of Enzo is cosmological simulation of the universe's expansion; here, the bivariate field represents matter and dark matter concentration. Each dataset is given on a three-dimensional regular grid and then tetrahedralized using Paraview's [1] default filter.

We summarize our results in Table 1. Columns labeled $\times$ show the speedup of our algorithm over arrange and traverse. In the time columns, this indicates how many times faster it is; in the memory columns, how much less memory it uses. The timings for the geometric preprocessing stage are given in the *Prep.* column and the timings for the topological traversal stage are given in the R_S column. The timings for the geometric postprocessing stage are omitted, as the computation time is negligible compared to the first two stages and depends on how the Reeb space will be used in downstream tasks. Each entry is rounded to the nearest whole number. We downsampled each dataset multiple times to obtain a wide range of data sizes and proportion of the number singular edges, ranging from 0.20% for MVK to 3.73% for ethane-diol and finally 43.78% for enzo. The enzo datasets and ethane-diol with 3,150K tetrahedra required the use of perturbation with strength 0.0001.

Our *primary result* is that the singular arrange and traverse algorithm is up to four orders of magnitude faster for MVK at 189K tetrahedra. For this dataset, arrange and traverse requires over 13 hours of computation time and about 100 GB of memory. Singular arrange and traverse not only takes less than 3 seconds and uses less than 700 MB of memory, but is also able to compute the Reeb space at full resolution (1,540K tetrahedra) in less than 10 seconds and 6 GB of memory. This speedup is due to the small number of singular edges, which our algorithm can successfully exploit.

Although the two other datasets have a larger singular set, we still see a significant speedup – at least two orders of magnitude for ethane-diol and at least one for enzo. Note that even though the number of singular edges in enzo is about half the total number of edges, we still see a significantly higher speedup – not just two-fold but up to 23x. This is due to the fact that the singular arrange and traverse algorithm not only allows us to avoid computing nonessential faces, but also allows us to group the computation of regular fiber graphs and significantly reduce the size of the correspondence graph.

In the case of MVK and ethane-diol, we are able to compute the Reeb space of their original non-downsampled resolution. This is something that would have been extremely difficult with arrange and traverse. The enzo dataset, however, has 83 million tetrahedra in its non-downsampled resolution and a high proportion of singular edges, which makes it difficult for our algorithm to handle.

Table 1 Performance metrics for real-life datasets. We report the number of input tetrahedra N_T, the percentage of singular edges N_s relative to all edges N_e, computation times, and memory usage for both preprocessing and Reeb space computations, including scaling factors (×) relative to arrange and traverse. Runs with over a day of compute were terminated and marked with "–".

	N_T	N_s/N_e	Time				Memory			
			Prep.(s)	×	R_S(s)	×	Prep.(Mb)	×	R_S(Mb)	×
MVK	22K	2.93%	2	14	<1	2K	<1	5K	<1	1K
	53K	1.70%	3	28	<1	6K	<2	12K	<1	4K
	189K	0.78%	8	72	2	25K	<2	64K	<1	18K
	1,540K	0.20%	41	–	6	–	<2	–	<1	–
Ethane-diol	25K	12.09%	85	0.9	30	129	204	58	237	23
	46K	9.33%	153	1.0	50	204	311	84	433	26
	113K	6.50%	402	1.4	123	426	600	139	810	46
	384K	3.73%	1,843	–	556	–	1,610	–	2,059	–
	3,150K	1.44%	19,259	–	5,238	–	7,438	–	7,389	–
Enzo	1K	49.88%	8	0.2	15	8	107	3	124	6
	4K	48.53%	189	0.1	323	13	869	4	988	17
	9K	44.62%	2,535	0.1	4,012	23	3,543	4	4,743	30
	14K	43.78%	7,127	–	11,294	–	7,882	–	11,140	–

We verified the correctness of our algorithm by comparing the sheets it outputs against arrange and traverse. For all datasets for which we could compute arrange and traverse, the outputs of both algorithms were identical. Finally, we note that the amount of additional computation needed to connect the nested faces was negligible. In the MVK datasets our algorithm added one to two additional pseudosingular edges to connect the arrangement and avoid nested faces. In the ethane-diol datasets that number was from 20 to 94.

The geometric preprocessing stage of our algorithm is significantly faster for MVK and ethane-diol and uses substantially less memory. Our memory usage improvement holds for enzo as well, but it takes up to 10 times longer to compute, even though arrange and traverse computes the arrangement of all segments. This may be addressed by implementing a more efficient algorithm for red-blue segment intersection.

7 Conclusion

In this paper, we presented the singular arrange and traverse algorithm for the exact and efficient computation of Reeb spaces of bivariate PL maps. This algorithm takes advantage of the fact that datasets with a small number of singular edges, relative to the total number of edges, have smaller Reeb spaces, which allows us to avoid nonessential computation from previous Reeb space algorithms such as arrange and traverse [26].

We presented a reformulation of the computation in terms of fiber graph classes and the singular correspondence graph, enabling the collective treatment of many related fiber graphs and thereby improving efficiency. We further provided an exact implementation of this approach and benchmarked it against the original arrange and traverse algorithm, demonstrating speedups ranging from one to four orders of magnitude across multiple real-world datasets derived from numerical simulations.

Future work will focus on optimising the running time through parallelisation as well as simplification for datasets with large singular sets. Other open problems include non-heuristic methods for dealing with nested faces and extensions to higher dimensional ranges.

References

1 James Ahrens, Berk Geveci, and Charles Law. Paraview: An end-user tool for large-data visualization. *Visualization Handbook*, pages 717–731, 2005. doi:10.1016/B978-012387582-2/50038-1.

2 Ivan J. Balaban. An optimal algorithm for finding segments intersections. In *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, SCG '95, page 211, New York, NY, USA, 1995. Association for Computing Machinery. doi:10.1145/220279.220302.

3 Harsh Bhatia, Attila G. Gyulassy, Vincenzo Lordi, John E. Pask, Valerio Pascucci, and Peer-Timo Bremer. TopoMS: Comprehensive topological exploration for molecular and condensed-matter systems. *Journal of Computational Chemistry*, 39(16):936–952, 2018. doi:10.1002/jcc.25181.

4 S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theoretical Computer Science*, 392(1-3):5–22, February 2008. doi:10.1016/j.tcs.2007.10.018.

5 Hervé Brönnimann, Andreas Fabri, Geert-Jan Giezeman, Susan Hert, Michael Hoffmann, Lutz Kettner, Sylvain Pion, and Stefan Schirra. 2D and 3D linear geometry kernel. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0 edition, 2024. URL: https://doc.cgal.org/6.0/Manual/packages.html#PkgKernel23.

6 Greg L. Bryan, Michael L. Norman, Brian W. O'Shea, Tom Abel, John H. Wise, Matthew J. Turk, Daniel R. Reynolds, David C. Collins, Peng Wang, Samuel W. Skillman, Britton Smith, Robert P. Harkness, James Bordner, Ji-hoon Kim, Michael Kuhlen, Hao Xu, Nathan Goldbaum, Cameron Hummels, Alexei G. Kritsuk, Elizabeth Tasker, Stephen Skory, Christine M. Simpson, Oliver Hahn, Jeffrey S. Oishi, Geoffrey C. So, Fen Zhao, Renyue Cen, Yuan Li, and The Enzo Collaboration. ENZO: An Adaptive Mesh Refinement Code For Astrophysics. *The Astrophysical Journal Supplement Series*, 211(2):19, March 2014. doi:10.1088/0067-0049/211/2/19.

7 Hamish Carr, Zhao Geng, Julien Tierny, Amit Chattopadhyay, and Aaron Knoll. Fiber surfaces: Generalizing isosurfaces to bivariate data. *Computer Graphics Forum*, 34(3):241–250, 2015. doi:10.1111/cgf.12636.

8 Pratip Chakraborty, Rafael C. Couto, and Nanna H. List. Deciphering Methylation Effects on S2($\pi\pi^*$) Internal Conversion in the Simplest Linear α,β-Unsaturated Carbonyl. *The Journal of Physical Chemistry A*, 127(25):5360–5373, 2023. doi:10.1021/acs.jpca.3c02582.

9 Amit Chattopadhyay, Hamish Carr, David Duke, and Zhao Geng. Extracting jacobi structures in reeb spaces. In N. Elmqvist, M. Hlawitschka, and J. Kennedy, editors, *EuroVis - Short Papers*. The Eurographics Association, 2014. doi:10.2312/eurovisshort.20141156.

10 Amit Chattopadhyay, Yashwanth Ramamurthi, and Osamu Saeki. An Algorithm for Fast and Correct Computation of Reeb Spaces for PL Bivariate Fields. doi:10.48550/arXiv.2403.06564.

11 Bernard Chazelle and Herbert Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM*, 39(1):1–54, January 1992. doi:10.1145/147508.147511.

12 James R. Clough, Nicholas Byrne, Ilkay Oksuz, Veronika A. Zimmer, Julia A. Schnabel, and Andrew P. King. A topological loss function for deep-learning based image segmentation using persistent homology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):8766–8778, 2022. doi:10.1109/TPAMI.2020.3013679.

13 Kree Cole-McLaughlin, Herbert Edelsbrunner, John Harer, Vijay Natarajan, and Valerio Pascucci. Loops in reeb graphs of 2-manifolds. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, pages 344–350, San Diego California USA, June 2003. ACM. doi:10.1145/777792.777844.

14 Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-77974-2.

15 Harish Doraiswamy and Vijay Natarajan. Efficient algorithms for computing Reeb graphs. *Computational Geometry*, 42(6-7):606–616, August 2009. doi:10.1016/j.comgeo.2008.12.003.

16 Herbert Edelsbrunner and John Harer. *Jacobi sets of multiple Morse functions*, volume 312, pages 37–57. Springer, 2004. doi:10.1017/CBO9781139106962.003.

17 Herbert Edelsbrunner and John Harer. *Computational Topology*. American Mathematical Society, Providence, Rhode Island, December 2009. doi:10.1090/mbk/069.

18 Herbert Edelsbrunner, John Harer, and Amit K. Patel. Reeb Spaces of Piecewise Linear Mappings. In *Proceedings of the Twenty-fourth Annual Symposium on Computational Geometry*, SCG '08, pages 242–250. ACM, 2008. doi:10.1145/1377676.1377720.

19 Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra, and Sven Schönherr. On the design of CGAL a computational geometry algorithms library. *Software: Practice and Experience*, 30(11):1167–1202, September 2000. doi:10.1002/1097-024X(200009)30:11<1167::AID-SPE337>3.0.CO;2-B.

20 Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors. *Handbook of Discrete and Computational Geometry*. Discrete Mathematics and Its Applications. CRC Press, third edition edition, 2018.

21 David Günther, Roberto A. Boto, Juila Contreras-Garcia, Jean-Philip Piquemal, and Julien Tierny. Characterizing molecular interactions in chemical systems. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2476–2485, 2014. doi:10.1109/TVCG.2014.2346403.

22 M. Hachani, A. Ouled Zaid, and W. Puech. Segmentation of 3D dynamic meshes based on Reeb graph approach. In *2014 22nd European Signal Processing Conference (EUSIPCO)*, pages 2175–2179, 2014.

23 C. Heine, H. Leitte, M. Hlawitschka, F. Iuricich, L. De Floriani, G. Scheuermann, H. Hagen, and C. Garth. A Survey of Topology-based Methods in Visualization. *Computer Graphics Forum*, 35(3):643–667, 2016. doi:10.1111/cgf.12933.

24 Petar Hristov. Arrange_and_Traverse_Algorithm, 2025. GitHub repository. URL: https://github.com/peter-hristov/arrange-and-traverse-algorithm.

25 Petar Hristov, Ingrid Hotz, and Talha Bin Masood. Robust geometric predicates for bivariate computational topology. In *2025 IEEE Workshop on Topological Data Analysis and Visualization (TopoInVis)*, pages 63–73, 2025. doi:10.1109/TopoInVis68599.2025.00011.

26 Petar Hristov, Daisuke Sakurai, Hamish Carr, Ingrid Hotz, and Talha Bin Masood. Arrange and traverse algorithm for computation of reeb spaces of piecewise linear maps. *Computer Graphics Forum*, 44(5):e70206, 2025. doi:10.1111/cgf.70206.

27 Petar Georgiev Hristov. *Hypersweeps, Convective Clouds and Reeb Spaces*. PhD thesis, University of Leeds, June 2022. URL: https://etheses.whiterose.ac.uk/31965/.

28 Lutz Kettner, Andreas Meyer, and Afra Zomorodian. Intersecting sequences of dD iso-oriented boxes. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0 edition, 2024. URL: https://doc.cgal.org/6.0/Manual/packages.html#PkgBoxIntersectionD.

29 Alexander Kuhn, Wito Engelke, Markus Flatken, Hans-Christian Hege, and Ingrid Hotz. Topology-Based Analysis for Multimodal Atmospheric Data of Volcano Eruptions. In Hamish Carr, Christoph Garth, and Tino Weinkauf, editors, *Topological Methods in Data Analysis and Visualization IV*, pages 35–50, Cham, 2017. Springer International Publishing.

30 Walter Motta, Jr Porto, Paulo, and Osamu Saeki. Stable Maps of 3-Manifolds Into the Plane and their Quotient Spaces. *Proceedings of the London Mathematical Society*, s3-71(1):158–174, July 1995. doi:10.1112/plms/s3-71.1.158.

31 Gregory Naitzat, Andrey Zhitnikov, and Lek-Heng Lim. Topology of Deep Neural Networks. *Journal of Machine Learning Research*, 21(184):1–40, 2020. URL: https://jmlr.org/papers/v21/20-345.html.

32 Monica Nicolau, Arnold J. Levine, and Gunnar Carlsson. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *Proceedings of the National Academy of Sciences*, 108(17):7265–7270, April 2011. doi:10.1073/pnas.1102826108.

33 Salman Parsa. A deterministic $O(m \log m)$ time algorithm for the reeb graph. In *Proceedings of the Twenty-Eighth Annual Symposium on Computational Geometry*, SoCG '12, pages 269–276. Association for Computing Machinery, 2012. doi:10.1145/2261250.2261289.

34 Michael W. Reimann, Max Nolte, Martina Scolamiero, Katharine Turner, Rodrigo Perin, Giuseppe Chindemi, Paweł Dłotko, Ran Levi, Kathryn Hess, and Henry Markram. Cliques of Neurons Bound into Cavities Provide a Missing Link between Structure and Function. *Frontiers in Computational Neuroscience*, 11:48, June 2017. doi:10.3389/fncom.2017.00048.

35 Colin Patrick Rourke. *Introduction to Piecewise-Linear Topology*. Springer Study Edition Ser. Springer Berlin / Heidelberg, Berlin, Heidelberg, 1982.

36 Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit (4th ed.)*. Kitware, 2006.

37 Mohit Sharma, Talha Bin Masood, Nanna Holmgaard List, Ingrid Hotz, and Vijay Natarajan. Continuous scatterplot and image moments for time-varying bivariate field analysis of electronic structure evolution. *IEEE Transactions on Visualization and Computer Graphics*, 31(10):7229–7242, 2025. doi:10.1109/TVCG.2025.3543619.

38 T. Sousbie. The persistent cosmic web and its filamentary structure – I. Theory and implementation. *Monthly Notices of the Royal Astronomical Society*, 414(1):350–383, June 2011. doi:10.1111/j.1365-2966.2011.18394.x.

39 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0 edition, 2024. URL: https://doc.cgal.org/6.0/Manual/packages.html.

40 J. Tierny and H. Carr. Jacobi Fiber Surfaces for Bivariate Reeb Space Computation. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):960–969, January 2017. doi:10.1109/TVCG.2016.2599017.

41 Julien Tierny, Guillaume Favelier, Joshua A. Levine, Charles Gueunet, and Michael Michaux. The topology toolkit. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):832–842, 2018. doi:10.1109/TVCG.2017.2743938.

42 Xavier Tricoche and Christoph Garth. Topological Methods for Visualizing Vortical Flows. In Torsten Möller, Bernd Hamann, and Robert D. Russell, editors, *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, pages 89–107. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi:10.1007/B106657_5.