Robust Geometric Predicates for Bivariate Computational Topology

Petar Hristov* Ingrid Hotz† Talha Bin Masood‡

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

ABSTRACT

We present theory and practice for robust implementations of bivariate Jacobi set and Reeb space algorithms. Robustness is a fundamental topic in computational geometry that deals with the issues of numerical errors and degenerate cases in algorithm implementations. Computational topology already uses some robustness techniques for the development of scalar field algorithms, such as those for computing critical points, merge trees, contour trees, Reeb graphs, Morse-Smale complexes, and persistent homology. In most cases, robustness can be ensured with floating-point arithmetic, and degenerate cases can be resolved with a standard symbolic perturbation technique called Simulation of Simplicity. However, this becomes much more complex for topological data structures of multifields, such as Jacobi sets and Reeb spaces. The geometric predicates used in their computation require exact arithmetic and a more involved treatment of degenerate cases to ensure correctness. Neither of these challenges has been fully addressed in the literature so far. In this paper, we describe how exact arithmetic and symbolic perturbation schemes can be used to enable robust implementations of bivariate Jacobi set and Reeb space algorithms. In the process, we develop a method for automatically evaluating predicates that can be expressed as large symbolic polynomials, which are difficult to factor appropriately by hand, as is typically done in the computational geometry literature. We provide implementations of all proposed approaches and evaluate their efficiency.

Index Terms: Multivariate data, Reeb space, Jacobi set, Robustness, Simulation of Simplicity

1 INTRODUCTION

Topological data structures like Reeb graphs [4] and contour trees [6] are indispensable tools for identifying features in the analysis and visualization of scalar field data. When more than one scalar field is defined over a common domain, which is often the case in practice, these data structures need to be generalized. The set of critical points, where topological change occurs, is generalized to the Jacobi set [11]. Regions in data with uniform topological connectivity are captured by a generalization of the Reeb graph and contour tree, called the Reeb space [15]. For bivariate data, the Reeb space is a collection of two-dimensional sheets, attached to each other via edges and vertices (see Fig. 1).

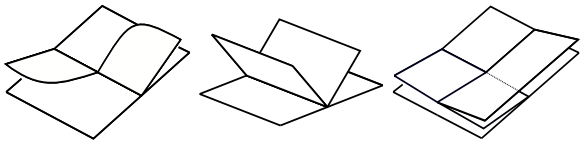
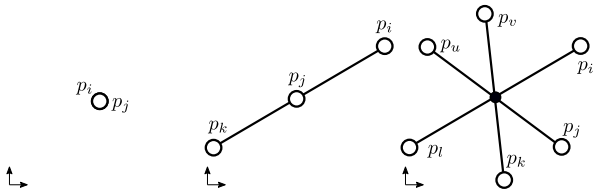


Figure 1: Examples of different Reeb spaces.

*e-mail: petar.hristov@liu.se
†e-mail: ingrid.hotz@liu.se
‡e-mail: talha.bin.masood@liu.se

One of the main practical challenges in computing Jacobi sets and Reeb spaces is that all current algorithms [38, 7, 15, 21] assume that the input data is generic (i.e., mathematically well behaved). For bivariate piecewise-linear (PL) data, typically this means that no two vertices map to the same point in the range, the images of no two edges are collinear, and the images of no three edges are concurrent (see Fig. 2). While the assumption of genericity greatly simplifies algorithm design, it is not realistic because degenerate cases do occur in real-world data and become more frequent as the size of the data increases. If not handled correctly, even a small number of degenerate cases can change the combinatorial properties of a topological data structure, resulting in incorrect analysis and feature identification.



(a) Duplicate points. (b) Collinear segments. (c) Concurrent segments.

Figure 2: Examples of the degenerate cases for generic bivariate PL maps. Each point and segment in the plane is the image of a vertex and edge of the input mesh.

In practice, genericity is often enforced by perturbing degenerate data values into general position. The standard perturbation scheme used for this purpose is called *Simulation of Simplicity* (SoS) [16], which symbolically adds infinitesimal numbers ε_i to the input values. These ε-terms are never explicitly evaluated; they are only used to break ties when the original values are not distinct. Since data is sampled at finite resolution, adding infinitesimal values with SoS preserves all non-degenerate features, which are usually the ones of interest in applications. Spurious features introduced by resolving degenerate configurations can be removed with simplification methods [10, 26].

In the scalar case, SoS can be implemented elegantly by breaking ties between equal data values using their unique memory indices. This is enough to resolve all degenerate cases in most scalar field computational topology algorithms. Since floating-point comparisons are sufficiently robust in practice, this pre-processing step is seamlessly integrated into most computational topology libraries [39, 29]. In fact, SoS is so ubiquitous that it is often cited [38, 7, 36, 21] as the standard method to ensure genericity in multivariate topological algorithms, such as those for Jacobi sets and Reeb spaces, without providing implementation details. However, extending SoS to multivariate data is nontrivial.

Symbolic perturbation techniques, including SoS, face several challenges even in two or three dimensions. First, even simple predicates like orientation tests can be numerically unstable with floating-point arithmetic, necessitating exact arithmetic – typically using arbitrary-precision rationals [19]. Second, predicates involving derived values (e.g., segment intersections) yield complex polynomials in the symbolic ε-variables, often with hundreds of terms.

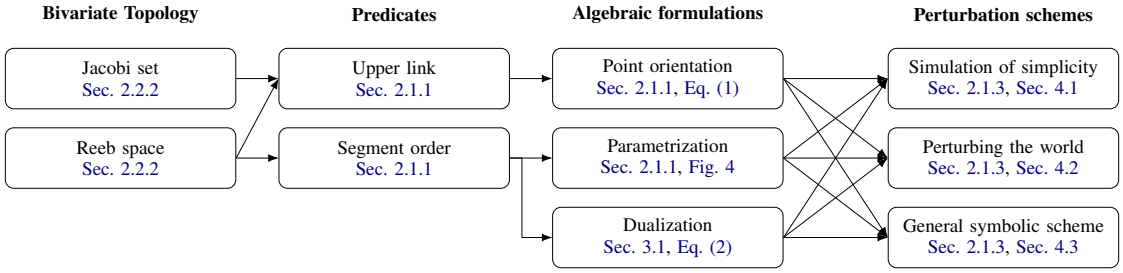


Figure 3: Overview: This work primarily investigates two bivariate topological structures – Jacobi sets and Reeb spaces. The computation of these structures relies on two key predicates, for which we present various algebraic formulations. Finally, we examine three distinct perturbation schemes aimed at ensuring robust implementation of these predicates.

These are notoriously difficult to handle manually, limiting the practicality of many symbolic perturbation schemes [42, 17].

In this work, we address the challenge of ensuring genericity in multivariate topological algorithms. We develop both the theoretical framework and practical tools required for the robust computation of geometric predicates used in bivariate Jacobi set and Reeb space algorithms. Our approach is compatible with two foundational symbolic perturbation techniques: SoS [16] and general symbolic perturbation [41, 42]. This enables us to transform any degenerate bivariate PL map f into a generic map f_ε, thereby ensuring the robustness of the existing Jacobi set and Reeb space algorithms against both numerical inaccuracies and degeneracies. For predicates with large symbolic expressions that are impractical to resolve manually, we introduce automated evaluation via symbolic math tools such as Mathematica or SymPy. These expressions are organized into evaluation tables [16], from which efficient C++ code can be generated and integrated into existing systems. All predicates are implemented and evaluated.

We review background from computational geometry and topology in Sec. 2. The predicates for computing Jacobi sets and Reeb spaces are outlined in Sec. 3, and their perturbation via symbolic schemes is described in Sec. 4. Genericity of perturbed maps is proved in Sec. 5. Section 6 introduces our approach to evaluating large symbolic expressions. Implementation and performance are covered in Secs. 7 and 8, followed by discussion and conclusions in Secs. 9 and 10. Fig. 3 provides a different perspective on the organization of this paper, highlighting the key steps required for the robust computation of bivariate topological structures.

2 Background

Before starting our discussion, we will introduce our main definitions. A *triangulation of a* $\mathsf{d}-$*manifold* is a d-dimensional simplicial complex M whose underlying geometric space is homeomorphic to a d-manifold (see Sec. A). A PL function $f : |M| \to \mathbb{R}$ whose range is the one-dimensional Euclidean space is called *scalar*. A collection of scalar PL functions defined over a common domain is called a *multifield* [18]. Equivalently, a multifield can be viewed as a multivariate PL map $f = (f_1, \ldots, f_\mathsf{m}) : M \to \mathbb{R}^\mathsf{m}$, where each f_i is called a *component function*. In this paper, we will primarily focus on multifields with two components, which are known as *bivariate fields*.

Let $V = \{v_1, \ldots, v_{n_0}\}$ and $E = \{e_1, \ldots, e_{n_1}\}$ be the vertices and edges of M respectively. We denote the images of V and E in $\mathbb{R}^\mathsf{m}$ as $P = \{p_1, \ldots, p_{n_0}\}$ and $S = \{s_1, \ldots, s_{n_1}\}$, where $p_i = f(v_i)$ for $1 \leq i \leq n_0$ and $s_i = f(e_i)$ for $1 \leq i \leq n_1$. We will refer to the elements of P and also more generally of the range space as *points*, and to the elements of S as line segments, or simply *segments*. Each point has coordinates $p_i = (p_{i,1}, \ldots, p_{i,\mathsf{m}})$ We will refer to the edges of M and the segments in $\mathbb{R}^\mathsf{m}$ using their endpoints – if an edge e has endpoints v_i and v_j, we write $v_i v_j = e \in M$, and $s_{i,j} = f(e) \subset \mathbb{R}^\mathsf{m}$.

We proceed with background from computational geometry in Sec. 2.1 and computational topology in Sec. 2.2.

2.1 Computational Geometry

Computational geometry algorithms have two key operations: *predicates* and *constructions*. A geometric predicate is a function that determines a combinatorial relationship between geometric objects when an algorithm has to make a decision. These include, for example, orientation tests or intersection tests. Predicates are usually evaluated on the basis of the sign of a determinant or, more generally, the sign of a polynomial. Constructions are made on the basis of the outcomes of predicates. For example, adding new vertices at intersection points or new lines on the boundaries of convex hulls.

2.1.1 Arrangements of Line Segments

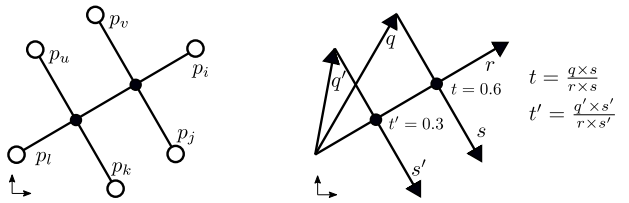
A fundamental problem in computational geometry is partitioning the Euclidean plane into regions, called faces, using a set of lines or line segments. The resulting partition is known as an *arrangement* and is typically computed in three stages: (1) finding segment intersections, often using a plane sweep algorithm [19]; (2) determining the relative order of these intersections; and (3) building a doubly connected edge list (DCEL) to represent the arrangement [8]. Computing an arrangement of line segments can be done in $O(n \log n + k \log n)$ time [19]. Arrangements can be computed for many different types of geometric objects such as triangles, hyperplanes, circles, and more general curves [19].

Required Predicates. Arrangement algorithms for line segments rely on three fundamental predicates: *lexicographic order of points*, *orientation of points* and *order of segments*. Lexicographic order compares two points by their first component and, if those are equal, then by their second component. It is used by plane sweep algorithms to maintain an event queue of intersection points and segment endpoints. Orientation tests determine which half-plane a point $p_k = (p_{k,1}, p_{k,2})$ is in with respect to the line defined by two other points $p_i = (p_{i,1}, p_{i,2})$ and $p_j = (p_{j,1}, p_{j,2})$. This can be evaluated by computing the determinant of the following matrix:

$$\triangle_p = \begin{pmatrix} p_{i,1} & p_{i,2} & 1 \\ p_{j,1} & p_{j,2} & 1 \\ p_{k,1} & p_{k,2} & 1 \end{pmatrix}. \tag{1}$$

Orientation tests are used to detect intersections between segments and to order segments around a vertex in the DCEL data structure. The third predicate, segment order, determines the relative order in which the three segments intersect each other. Although it can be derived from orientation tests, doing so requires computing intersection points, which is costly. A more efficient alternative is to parametrize the segments and compare the parameter values (see

Fig. 4). Rearranging the resulting expression to eliminate the denominators avoids division, improving both performance and robustness.



(a) Intersecting segments. (b) Segment vectors.

Figure 4: The relative order of intersection of three segments can be determined by comparing the parameters t and t' of the intersection points over a parametrization of the first segment. We compute t and t' using the scalar cross product $\times$ with r, s, s', r and r'.

2.1.2 Robustness

Computational geometry algorithms typically assume infinite numerical precision and inputs in general position. These assumptions simplify exposition, but are impractical in real implementations. Ensuring correctness despite numerical errors or degeneracies is known as *robustness*. Robustness is critical for predicates that determine the combinatorial structure of the output.

Numerical Stability. It is well known that floating point arithmetic suffers from precision errors, particularly when comparing expressions that are close to zero, as is the case with geometric predicates. We encourage the reader to consult the following examples of how orientation tests can fail due to such inaccuracies [25, Section 3]. When predicates fail, they can compromise the accuracy of algorithms, potentially leading to errors in the global combinatorial structure of the computed solution.

Robustness in the presence of numerical errors is a broad topic. Here, we briefly highlight some key concepts and refer the reader to a comprehensive overview in [19] for further details. A central approach in this area is the *exact geometric computation (EGC)* paradigm, which is central to many modern implementations of computational geometry algorithms, such as those found in the open-source CGAL library [37]. EGC avoids numerical errors arising from floating-point arithmetic by using exact arithmetic with rational number types and arbitrary-precision integers. To avoid the performance bottleneck of exact arithmetic, typically *filters* are employed to first compute predicates approximately with floating-point numbers. Exact arithmetic is then only used when the result of the floating-point computation is outside a controlled error bound.

Degenerate input configurations. A remaining challenge for robustness is handling degenerate input configurations. Degeneracies are algorithm-dependent. Some algorithms may fail on collinear points, while others may not. The most direct and often preferred approach in practical implementations is to explicitly handle degenerate cases as usual parts of the input [37]. However, this is not always feasible, especially when a complete classification of all possible degeneracies and their effects on the algorithm's output is unavailable. As an alternative, there are various techniques that slightly perturb the input into general position. The simplest form of input perturbation is *random numeric perturbation*, where small random values are added to the input data. Although this approach can resolve many degenerate cases, it is not guaranteed to eliminate all of them due to finite precision and its random nature. In contrast, *controlled perturbation* uses adaptive precision to exactly compute a non-degenerate input configuration [27]. Finally, several *symbolic perturbation* schemes exist in which the actual perturbation is never explicitly calculated. Instead, the output of a predicate is decided as if the input was perturbed by an infinitesimally small amount. The goal is to ensure that the perturbed objects are in general position, to keep all non-degenerate properties of the original data, and to lend to an efficient implementation [16].

2.1.3 Symbolic Perturbation

One of the first symbolic perturbation schemes is called Simulation of Simplicity (SoS) [16]. Given an input with a number of geometric objects, each specified by a number of *coordinates* $p_i = (p_{i,1}, \ldots, p_{i,\mathsf{m}})$, the ε-*expansion* of p_i is defined as $p_i(\varepsilon) = (p_{i,1} + \varepsilon_{i,1}, \ldots, p_{i,\mathsf{m}} + \varepsilon_{i,\mathsf{m}})$, where each $\varepsilon_{i,j}$ is an infinitesimal number called an ε-*variable*, for $j \in \{1, \ldots, \mathsf{m}\}$. The product of multiple ε-variables $\varepsilon_{i_1,j_1}\varepsilon_{i_2,j_2}\cdots\varepsilon_{i_n,j_n}$ is called an ε-*product* if all ε_{i_k,j_k} are unique for $k \in \{1, \ldots, n\}$. The specific choice of ε-variables is not important, as long as they satisfy properties that establish a lexicographic order among ε-variables and their products, which reflects differences in their orders of magnitude.

In the SoS framework, many predicates expressed as determinants are guaranteed to be nonzero. For example, for an orientation test, this ensures that no three points are treated as collinear. To determine the sign of a perturbed predicate, SoS successively evaluates *subdeterminants* from the original determinant. These subdeterminants are lexicographically ordered according to their associated ε-products and organized into a *table of relevant determinants*. Once a constant non-zero subdeterminant is found, the evaluation stops, as smaller ε-products cannot affect the overall sign. The number of rows in the table is referred to as the maximum *depth*.

A more general approach that allows predicates to be expressed as rational polynomials is given by Yap's *general symbolic scheme* [41, 42], which avoids explicitly specifying an ε-perturbation. It gives a general method for evaluating the perturbed sign $s \in \{-1, 1\}$ of a rational polynomial $f(x_1, \ldots, x_n)$ with input $q \in \mathbb{Q}^n$ even when $f(q) = 0$. This is achieved by establishing a total *admissible* ordering on the partial derivatives of f, and then differentiating f with respect to the ordered partial derivatives until a nonzero value is found. Two examples of total admissible orderings are *lexicographic ordering* and *total ordering*, where the total order is first compared and then the lexicographic ordering is used to break ties. The partial derivatives can be computed symbolically with computer algebra packages [35].

For example, Tab. 1 shows a direct comparison between SoS and the general symbolic scheme for an orientation test of two points on the projective line in homogeneous coordinates. Each row in the table for SoS lists subdeterminants that multiply ε-products in decreasing order of significance [16, Eq. 6, Sec. 4]. For the general scheme, we list the expressions that correspond to totally ordered partial derivatives.

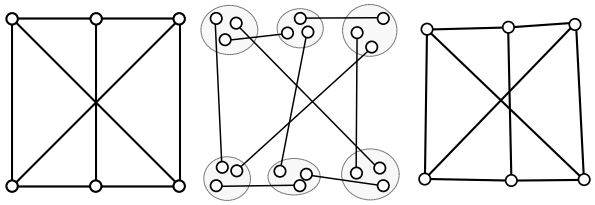
	SoS [16]		General scheme [42]	
Row	Subdeterminant	ε-product	Expression	Derivative
0	$\begin{vmatrix} p_{i,1} & p_{i,2} \\ p_{j,1} & p_{j,2} \end{vmatrix}$	$\emptyset$	$\begin{vmatrix} p_{i,1} & p_{i,2} \\ p_{j,1} & p_{j,2} \end{vmatrix}$	$\emptyset$
1	$p_{j,1}$	$\varepsilon_{i,2}$	$p_{j,2}$	$\partial p_{i,1}$
2	$-p_{j,2}$	$\varepsilon_{i,1}$	$-p_{j,1}$	$\partial p_{i,2}$
3	$p_{i,1}$	$\varepsilon_{i,2}$	$-p_{i,2}$	$\partial p_{j,1}$
4	1	$\varepsilon_{j,2}\varepsilon_{i,1}$	$p_{i,1}$	$\partial p_{j,2}$
5			1	$\partial p_{i,1} p_{j,2}$

Table 1: Evaluation table comparisons between SoS and Yap's general symbolic scheme for the scalar cross product of two vectors.

The subsequent work [17] improves efficiency by reducing the algebraic degree of perturbed expressions using a *linear perturbation* $p_{i,j}(\varepsilon) = p_{i,j} + i^j\varepsilon$. This scheme is less general than SoS but has been successfully applied to orientation tests and in-sphere tests. More recent work [23] has focused on combining the efficiency of linear perturbations [17] with the general symbolic

scheme [42] or on geometric symbolic perturbation schemes with qualitative symbolic perturbation [9]. Finally, we note that all symbolic perturbation schemes assume exact computation [9, 35].

Symbolic Perturbation of Lines and Line Segments. Symbolic perturbation can be extended to line segments by representing each segment as a point in $\mathbb{R}^4$, formed by concatenating its endpoints. This resolves common degeneracies such as segments with endpoints or intersection points that share the x- or y-coordinates, collinear segments, concurrent segments, and segments that share endpoints. Alternatively, perturbing only the endpoints of segments addresses the first three types of degeneracies, while preserving the incidence relationships between segments that share endpoints [2]. While the paper does not provide an explicit perturbation scheme for segments, it suggests that it would be similar to the one used for 2D Delaunay triangulations: $p_i(\varepsilon) = (p_{i,1} + \varepsilon p_{i,2}, p_{i,2} + \varepsilon^2 p_{i,1} + \varepsilon^3(p_{i,1}^2 + p_{i,2}^2))$. The technique is referred to as *perturbing the world* and applies a single global ε-variable. Fig. 5 shows the two alternatives for symbolic perturbation of line segments.



(a) Input segments. (b) Segment perturbation. (c) Endpoint perturbation.

Figure 5: Two perturbations of line segments that remove different types of degenerate cases. Perturbing the segments (b) as four-dimensional points resolves all degenerate cases, but segments that used to share endpoints no longer do. Perturbing the endpoints of segments (c) keeps the overlapping endpoint incidence relationship between segments, while removing all other degenerate cases.

2.2 Computational Topology

In this section, we review key results from computational topology for both scalar fields and multi-fields, with an emphasis on Jacobi set and Reeb space algorithms. We give special attention to generic piecewise-linear (PL) maps and their various definitions in the literature. For a general introduction to PL and computational topology, we refer the reader to established textbooks [13, 31].

2.2.1 Scalar Field Topology

Scalar field topology studies the structure of real-valued functions $f : |M| \to \mathbb{R}$ called scalar fields, where the input data M is typically represented by a regular grid or a triangulated manifold of dimension d (see Sec. A). The main topological structures used to extract and analyze features in scalar fields are merge trees [3], contour trees [6], Reeb graphs [4], Morse-Smale complexes [14], and persistent homology [12]. The success of these methods is largely due to efficient combinatorial algorithms for their computation, which require a robust handling of geometric predicates.

The core geometric predicate used by these algorithms is a comparison that determines the upper and lower link of a vertex. The upper and lower link partition the vertices of the link into those with higher function value (upper link) and lower function value (lower link). These predicates can be evaluated efficiently and robustly using floating-point arithmetic, requiring only a single comparison per link vertex. To handle degenerate cases, such as vertices with identical function values, SoS is used to break the ties [13].

Simulation of Simplicity [16] simplifies algorithm design by ensuring a scalar field is *generic*, that is, free of degeneracies. Symbolic perturbation, however, comes at a cost: it can introduce artificial features, particularly in flat regions. To remove these, a subsequent *simplification* step is applied, typically guided by a geometric measure such as persistence, volume, or hypervolume.

2.2.2 Multifield Topology

In many real-world applications, the key features of interest arise from complex interactions among the component scalar fields in multifield data. Since these relationships cannot be captured by the topological descriptors of individual fields or their pairwise combinations, growing attention has been devoted to extending topological data structures to the multivariate setting. Two topological data structures have been successfully generalized with efficient algorithms: Jacobi sets [36, 38], which extend the concept of critical points, and Reeb spaces [21], which generalize Reeb graphs and contour trees. However, research on the robustness and reliability of these algorithms remains in its early stages.

Jacobi Sets. The Jacobi set of a multifield $f = (f_1, \ldots, f_\mathsf{m}) : |M| \to \mathbb{R}^\mathsf{m}$, where M is a triangulation of a d-manifold, generalizes the notion of critical vertices in the scalar case to Jacobi simplices [11]. A simplex $\sigma \in M$ of dimension $\mathsf{d} - \mathsf{m}$ is part of the Jacobi set, depending on the topology of its upper and lower link, computed in terms of reduced Betti numbers.

The geometric predicate used to decide whether a vertex $v \in \mathrm{Link}(\sigma)$ is in the upper or lower link of σ uses a scalar function $h_\lambda : \mathrm{Link}(\sigma) \to \mathbb{R}$. The vertex v belongs to the upper link if $h_\lambda(v) > h_\lambda(\sigma)$, and to the lower link if $h_\lambda(v) < h_\lambda(\sigma)$. In the bivariate case for an edge $ab \in M$, the scalar function is defined as $h_\lambda(v) = \lambda f_1(v) + f_2(v)$, where λ is the negative slope of the line between the points $f(a)$ and $f(b)$, as shown in Fig. 6. In practice, this predicate can be computed with an orientation test [22] using the points $f(a), f(b)$ and $f(v)$ or using the range-based signed distance field with respect to the image of the line segment $f(ab)$ in the range [36, 38].

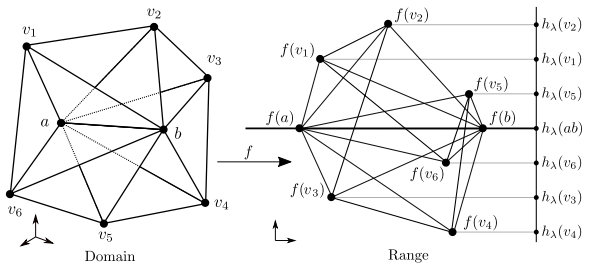


Figure 6: Geometric interpretation of the scalar function h_λ used to define Jacobi edges. Vertices mapped above the line defined by $f(ab)$ belong to the upper link of ab, and the ones that map below belong to the lower link.

Unlike in the scalar case, robustly handling degenerate situations where $h_\lambda(v) = h_\lambda(ab)$ is considerably more challenging. To the best of our knowledge, this issue has only been addressed for Jacobi sets of bivariate maps defined over triangulations of 2-manifolds [30]. Given a bivariate PL map $(f_1, f_2) : |M| \to \mathbb{R}^2$, the authors use a symbolic perturbation scheme similar to SoS. The ε-expansion is given as $f_1(v_i) + \varepsilon_i$ and $f_2(v_i) + \delta_i$, where v_i is the vertex of the triangulation with index i. The order of the symbolic variables is $\varepsilon_1 \gg \cdots \gg \varepsilon_n \gg \delta_1 \gg \cdots \gg \delta_n$, where n is the number of vertices. The definition of $h_\lambda(v_i)$ is then expanded in terms of the symbolically perturbed coordinates and the necessary if-else statements to robustly resolve the predicate are derived.

Reeb spaces. The Reeb space generalizes the Reeb graph by encoding the connectivity of connected components of level sets of multivariate PL maps. These level sets are known as *fibers* [33]

and their connected components are called *fiber components*. In the bivariate case over a triangulation of a 3-manifold, the fibers are generically a set of one-dimensional PL curves, except when they intersect Jacobi edges. Fiber components intersecting Jacobi edges are either isolated points (analogous to local maxima and minima in the scalar case) or collections of PL curves all meeting at a single point (analogous to saddles in the scalar case).

For a multifield with m components, the Reeb space is locally an m-manifold, except at Jacobi edges, where it becomes singular [15]. In the bivariate case, the Reeb space consists of two-dimensional sheets glued together at the Jacobi edges in possibly complicated ways; see Fig. 1. For a comprehensive introduction to Reeb spaces, we refer the reader to the following works [22, 15]. Throughout this paper, we will only consider 2-dimensional Reeb spaces, although the concept and computation extend to higher dimensions [15, 21].

There are four algorithms for computing the Reeb space [15, 38, 21, 7]. Two of them [15, 21] construct an *arrangement* in the range, then proceed in a fully combinatorial manner. As a result, the geometric predicates they rely on are the same as those for computing the arrangement: lexicographic order, orientation of points, and order of segments. The other two algorithms [7, 38] avoid building the full arrangement, but still use similar geometric predicates. The first [7] computes segment intersections and their relative order. The second [38] also identifies segment intersections to determine where Jacobi fiber surfaces intersect [7]. It also requires ordering the intersections in the domain, which corresponds to the order of their respective segment intersections in the range due to the barycentric interpolation of the multivariate PL map.

The robustness of Reeb space algorithms has not been fully addressed so far, even though it is a crucial step in deploying these algorithms in practice, in which accuracy must be guaranteed. Robustness with respect to numerical errors is addressed [21] by computing arrangements with exact geometric computation using CGAL. However, all four Reeb space algorithms assume a generic input map with no degenerate cases. Two of them [21, 38] apply random numerical perturbations with finite precision arithmetic, which is not guaranteed to resolve all degenerate cases. Most Reeb space algorithms mention that SoS [16] can be used to ensure that the input map is generic, without providing explicit details. In this paper, we address this gap.

Generic Bivariate PL Maps. Generalizing the notion of genericity beyond the scalar case is a challenge. Although injectivity on the vertex set is a sufficient condition for a scalar PL function to be generic, it does not eliminate all degenerate cases [38] in multifields. Furthermore, there is no known PL analogue of stable maps from fiber topology [33]. As a result, various definitions of generic PL maps have emerged in the literature, each tailored to the requirements of a specific algorithm.

The first definition of a generic PL map [15] is such that *"the images of the vertices have no structural properties that can be removed by arbitrarily small perturbations ..."*. Although the term *structural properties* is not explicitly defined in that paper, its meaning is understood from the context. By considering the Reeb space as representing the topological structural properties of a PL map, follow-up work formalizes this definition by requiring that no three points are collinear and no three segments are concurrent at their interiors [22]. Both definitions imply that f is injective on the edges and triangles of the input mesh because degenerate segments and triangles (see Fig. 2) in the range introduce collinear points.

This injectivity condition has also been used independently as a definition of genericity for bivariate PL maps [7]. Another injectivity condition used in defining generic bivariate PL maps requires that the images of no two edges be collinear [38]. While these definitions [7, 38] are sufficient for those Reeb space algorithms, they do not strictly guarantee that the Reeb space of a generic map remains the same under arbitrarily small perturbations [22].

3 Algebraic Formulation of Required Predicates

As discussed in Sec. 2, the predicates required for computing bivariate Jacobi sets and Reeb spaces are lexicographic ordering, identifying upper and lower links of edges, and determining the order of segment intersections. In this section, we will discuss which algebraic formulation we will use for each predicate. Throughout the rest of the paper, we assume that the input mesh M is a triangulation of a 3-manifold and that the *PL* map $f = (f_1, f_2) : |M| \to \mathbb{R}^2$ is bivariate and not necessarily generic.

Unlike previous robust implementations of Jacobi set computation that use the definition of h_λ to identify upper and lower links of edges, we will use the alternative formulation based on orientation tests (Eq. (1)). This choice is motivated by the fact that orientation tests are more widely supported in geometric software libraries, and their robust perturbation techniques are well-studied across all dimensions, including homogeneous coordinates.

The predicate for ordering segment intersections can also be handled using orientation tests. However, this approach requires explicitly computing the coordinates of the intersection points. As discussed in Sec. 2.1, this is significantly less efficient than parameterizing the segments and directly comparing the values of the scalar parameters. However, the resulting expression is not a determinant, which is a requirement for most symbolic perturbation schemes [16, 17, 2]. Although this issue can be addressed, as demonstrated in Sec. 4.1, we introduce an alternative formulation of this predicate based on orientation tests in the dual projective plane. Having both formulations enables a comparative evaluation of their correctness and practical efficiency.

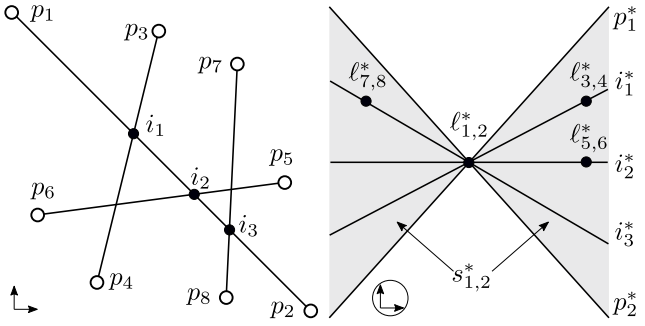
3.1 Order of Segments in Homogeneous Coordinates

An alternative method to determine the intersection order of three segments $s_{1,2}, s_{3,4}$, and $s_{5,6}$ is to use homogeneous coordinates in the dual projective plane. Given two points in homogeneous coordinates $[p_{1,1}, p_{1,2}, 1]$ and $[p_{2,1}, p_{2,2}, 1]$, the line passing through them is represented as $\ell_{1,2} = [p_{1,2} - p_{2,2}, p_{2,1} - p_{1,1}, p_{1,1}p_{2,2} - p_{1,2}p_{2,1}]$. Using the point-line duality in projective geometry [8, Ch. 8.2], this line corresponds to a projective point $\ell^*_{1,2}$ with the same coordinates in the dual plane. Since this transformation preserves incidence and order, the orientation of the three points in homogeneous coordinates $\ell^*_{1,2}, \ell^*_{3,4}$ and $\ell^*_{5,6}$ in the dual plane encodes the order of the original three segments $s_{1,2}, s_{3,4}$ and $s_{5,6}$.

An example in Fig. 7 illustrates this construction. Let $s_{1,2}$ be a segment on the line $l_{1,2}$ in the primal plane, whose dual point in the dual plane is $l^*_{1,2}$. Each point on the segment corresponds to a line in the dual plane that passes through the point $l^*_{1,2}$. These lines sweep out two wedges, shown in gray in Fig. 7b (see also Fig. 8). The three intersection points i_1, i_2, and i_3 on $s_{1,2}$ map to three of these dual lines denoted as i^*_1, i^*_2, and i^*_3. Meanwhile, the lines $\ell_{3,4}$ and $\ell_{5,6}$ (containing the segments $s_{3,4}$ and $s_{5,6}$) are mapped to the dual points $\ell^*_{3,4}$ and $\ell^*_{5,6}$, which lie on the lines i^*_1 and i^*_2, respectively. The orientation of the dual points $l^*_{1,2}$ and $l^*_{3,4}$ matches the order of intersection points i_1 and i_2 along $s_{1,2}$ in the primal plane, which can be evaluated via the sign of the determinant:

$$\triangle_s = \det \begin{pmatrix} p_{2,2} - p_{1,2} & p_{1,1} - p_{2,1} & p_{2,1}p_{1,2} - p_{1,1}p_{2,2} \\ p_{4,2} - p_{3,2} & p_{3,1} - p_{4,1} & p_{4,1}p_{3,2} - p_{3,1}p_{4,2} \\ p_{6,2} - p_{5,2} & p_{5,1} - p_{6,1} & p_{6,1}p_{5,2} - p_{5,1}p_{6,2} \end{pmatrix} \quad (2)$$

However, there is no guarantee that the lines corresponding to two segments will map to the same wedge in the dual plane. For example, the lines of the segments $s_{5,6}$ and $s_{7,8}$ map to two points $l^*_{5,6}$ and $l^*_{7,8}$ that lie in different wedges in Fig. 7. In such cases, the orientation of the dual points is reversed relative to the order of the



(a) Segments in the plane. (b) Dual Homogeneous Coordinates.

Figure 7: The order of intersections of four segments $s_{1,2}$, $s_{3,4}$, $s_{5,6}$, and $s_{7,8}$ relative to the first segment $s_{1,2}$ can be computed with an orientation test in the dual projective plane. The segment $s_{1,2}$ maps to two wedges that meet at a point, as shown shaded in (b). We must flip the sign of the result if the points are not in the same wedge.

intersection points in the primal plane. This can be resolved by two additional sign checks to determine whether the lines are mapped to the same wedge. This can be achieved by substituting their coordinates in the equation of the line given by p_1^*. Geometrically, the wedge to which a line's dual point maps to is determined by the line's orientation with respect to the line connecting the corresponding intersection point (in the primal plane) to the origin; see Fig. 8. We refer the reader to Sec. B for an additional discussion on special cases when one or more of the line segments pass through the origin in the primal plane result in ideal points at infinity in the dual plane.

4 Symbolic Perturbation of Required Predicates

Having established the algebraic formulations of the predicates for computing Jacobi sets and Reeb spaces in Sec. 3, we now demonstrate how to perturb a possibly degenerate PL map $f : |M| \to \mathbb{R}^2$ to a generic PL map f_ε. To ensure that f_ε is well-defined, we will only perturb the points (images of vertices) of f (see Sec. 2.1.3). We show that this resolves all degenerate cases for points and segments (images of edges), shown in Fig. 2. We investigate perturbations using Edelsbrunner and Mücke's SoS (E) [16] in Sec. 4.1, Alliez et al.'s perturbation of the world (A) [2] in Sec. 4.2, and Yap's general symbolic perturbation scheme (YL, YT) [42] in Sec. 4.3. Since floating-point arithmetic cannot guarantee robustness for our predicates, we assume the use of exact arithmetic (see Sec. 2.1).

4.1 Simulation of Simplicity (E)

In the SoS framework, each point $p_i \in P$ is perturbed via an ε-expansion, defined as $p_i(\varepsilon) = (p_{i,1} + \varepsilon_{i,1}, p_{i,2} + \varepsilon_{i,2})$ (see Sec. 2.1). This ensures that we can perform lexicographic point comparisons solely based on the first perturbed coordinates because they are unique. To compare two points p_i and p_j, where $i \neq j$, we first compare their unperturbed first coordinates $p_{i,1}$ and $p_{j,1}$. If they are equal, we break the ties using their indices i and j, analogously to the comparison test widely used in the scalar case. Orientation tests of three perturbed points are done by evaluating a perturbed determinant, which is discussed in the original SoS paper (see [16, Table-ii in the Appendix]). We show the number of terms and arithmetic operations in each row of the evaluation table of this scheme in Tab. 2 (column E). Next, we discuss the implementation of the segment order predicate through parametrization and dualization.

Parametrization

The expression we obtain from the order of segments predicate with parameterization is not a determinant, unlike the typical setup in SoS. However, this is not a problem if we can verify two properties that are guaranteed when we expand a determinant into an expression with SoS. The first property is that at least one ε-product has a constant coefficient. This ensures that the predicate never evaluates to zero. The second property is that no ε-variable is raised to a power other than zero or one. This ensures that they have a well-defined total ordering. To verify these properties, we can explicitly expand the expression, but first we need to rearrange it to eliminate the denominator.

To expand the expression, we substitute all vectors with their perturbed counterparts, as described in Sec. 2.1. The resulting expression is $\frac{\mathbf{q}(\varepsilon)\times\mathbf{s}(\varepsilon)}{\mathbf{r}(\varepsilon)\times\mathbf{s}(\varepsilon)} - \frac{\mathbf{q}'(\varepsilon)\times\mathbf{s}'(\varepsilon)}{\mathbf{r}(\varepsilon)\times\mathbf{s}'(\varepsilon)}$. To eliminate the denominator, we multiply both terms by their common denominator $(\mathbf{r}(\varepsilon) \times \mathbf{s}'(\varepsilon))(\mathbf{r}(\varepsilon) \times \mathbf{s}(\varepsilon))$. Although this changes the value of the expression, we are only concerned with its sign, which remains the same as long as the denominator is positive. Therefore, we must track the sign of both the denominator and the resulting expression:

$$(\mathbf{q}(\varepsilon) \times \mathbf{s}(\varepsilon))(\mathbf{r}(\varepsilon) \times \mathbf{s}'(\varepsilon)) - (\mathbf{q}'(\varepsilon) \times \mathbf{s}'(\varepsilon))(\mathbf{r}(\varepsilon) \times \mathbf{s}(\varepsilon)).$$

When expanded, the final expression has 768 terms, making the evaluation of its sign non-trivial. For this reason, we use an automatic evaluation procedure (Algorithm 1) whose description we postpone to Sec. 6 for clarity of exposition. The evaluation table of the final expression has a maximum depth of 34. We compare the number of terms and arithmetic operations (addition and multiplication) of each row of the evaluation tables of all perturbation schemes in Tab. 3 (column E). Finally, we confirm that no ε-variable in the expression has an exponent different than zero or one, and that there is an ε-product that has a constant coefficient.

Dualization

The second approach we introduced in Sec. 3 for implementing the segment order predicate relies on orientation tests in the dual projective plane. This method fits naturally within the SoS framework, as it can be formulated as a determinant. However, unlike standard orientation predicates, we cannot use the evaluation tables provided by Edelsbrunner and Mücke [16], since the determinant in this case does not directly involve the original input points. We can still use Algorithm 1 to automatically determine the sign of the determinant of the matrix described in Eq. (2) using the perturbed coordinates.

Similar to the parametrized formulation, we need to compute additional signs to decide whether to flip the sign of the final result. This depends on whether the lines of the two segments are mapped to the same wedge in the dual plane (Sec. 3.1). We also need to check for the signs that account for normalizing the points in homogeneous coordinates [16].

The fully expanded expression for the dualized formulation of the segment order predicate contains 768 terms, and its evaluation table reaches a maximum depth of 34 as shown in Tab. 3. These are exactly the same figures as those obtained from the parametrization-based formulation of the predicate. When comparing the two expanded expressions and their corresponding evaluation tables, we observe that they are identical.

4.2 Perturbing the World (A)

Next, we discuss the symbolic perturbation scheme called perturbing the world, which is specifically designed for line segments [2]. This scheme uses a single global epsilon ε and the epsilon expansion is given as $p_i(\varepsilon) = \big(p_{i,1} + \varepsilon p_{i,2}, p_{i,2} + \varepsilon^2 p_{i,1} + \varepsilon^3 (p_{i,1}^2 + p_{i,2}^2)\big)$. It is important to note that this scheme assumes that the input contains no duplicate points. Indeed, from the structure of the epsilon expansion, a lexicographic comparison would be unable to distinguish between two points that share the same coordinates. The paper [2] gives details about evaluating the point orientation predicate, and we can use Algorithm 1 to automatically generate the

	Number of Terms				Number of Operations			
Depth	**A**	**E**	**YL**	**YT**	**A**	**E**	**YL**	**YT**
0	6	6	6	6	11	11	11	11
1	18	2	2	2	47	1	1	1
2	12	2	2	2	35	1	1	1
3		2	2	2		1	1	1
4		1	1	2		0	0	1
5				2				1
6				2				1
7				1				0
Non-zero constant		1	-1	-1				

Table 2: Number of terms and arithmetic operations for each row in the evaluation tables of each symbolic perturbation scheme for the point orientation predicate. The perturbation schemes are labeled according to the initial of their first author: A [25], E [16], YL [42] with lexicographic ordering and YT [42] with total ordering.

evaluation table for the segment order predicate. As we have seen previously, both formulations of that predicate produce the same final expression. When we generate the evaluation table from the final expression, the ordering of all epsilon products is $\varepsilon, \varepsilon^2, \ldots, \varepsilon^n$, where n is the maximum degree of any epsilon in the expression. This scheme produces significantly more terms than the other two schemes, as can be seen in column A of Tab. 2 and Tab. 3.

	Number of Terms				Number of Operations			
Depth	**A**	**E**	**YL**	**YT**	**A**	**E**	**YL**	**YT**
0	48	48	48	48	191	191	191	191
1	240	16	16	16	1199	47	47	47
2	192	16	16	16	959	47	47	47
3	432	16	16	16	2303	47	47	47
4	576	4	8	16	3263	7	15	47
5	192	6	16	16	1151	11	47	47
6		16	8	16		47	15	47
⋮		⋮	⋮	⋮		⋮	⋮	⋮
34		1	6	6		0	11	11
35			4	4			7	7
⋮			⋮	⋮			⋮	⋮
59			1	4			0	7
60				6				11
⋮				⋮				⋮
163				1				0
Non-zero constant		1	-1	-1				

Table 3: Number of terms and arithmetic operations for each row in the evaluation tables of each symbolic perturbation scheme for the segment order predicate. The perturbation schemes are labeled according to the initial of their first author: A [25], E [16], YL [42] with lexicographic ordering, and YT [42] with total ordering.

4.3 General Symbolic Scheme (YL, YT)

The general symbolic scheme proposed by Yap [41, 42] does not require an epsilon expansion and relies on symbolic differentiation, which is present in most symbolic mathematics libraries. The only prerequisite to using this scheme is to choose an admissible ordering of the power products of partial derivatives. Two common admissible orderings are the lexicographic ordering (YL) and the total degree ordering (YT), where power products are first sorted by their total degree and then lexicographically. When applied to the lexicographic order predicate, this scheme produces the same result as SoS — ties in the first coordinate are broken by its index.

We show the difference in the number of terms and arithmetic operations produced by the two admissible orderings for the orientation of points predicate in Tab. 2 and for the order of segments predicate in Tab. 3. In the evaluation table for the lexicographic ordering (YL), rows with fewer terms appear earlier because higher-order derivatives are evaluated before lower-order partial derivatives. As a result, simpler expressions tend to appear earlier in the table, which is advantageous for predicates that can be resolved without requiring deep evaluations. The lexicographic ordering approach also has a smaller maximum depth because partial derivatives of higher order are evaluated first.

5 Genericity

The results from the previous section allow us to perturb any bivariate PL map f to a generic PL map f_ε. We will use the definition of genericity that requires that no three points are collinear and no three segments are concurrent in their interiors [22]. As discussed in Sec. 2.2.2, this implies all other formulations of genericity of bivariate PL maps [15, 7, 38].

Lemma 1. *Let M be a triangulation of a 3-manifold and let $f : |M| \to \mathbb{R}^2$ be a bivariate PL map. The symbolic perturbation f_ε of f with either Simulation of Simplicity or general symbolic perturbation is a generic PL map.*

Proof. As we have shown in Sec. 4 the lexicographic order, orientation of points, and order of segments predicates never evaluate to zero. This is evident from Tab. 2 and Tab. 3 where the predicates reach a non-zero constant term in their evaluation tables. Therefore, no three points are collinear and no three segments are concurrent in their interiors. So, f_ε is generic. □

6 Sign Evaluation of Symbolic Polynomials

Evaluating the perturbed expressions of the parametrized and dualized formulations of the order of segments predicate is challenging because they involve points constructed from the original input. As we have shown in Sec. 4 this leads to symbolic expressions with hundreds of terms, which are impractical to factor manually since it is not straightforward to express them as sums of subdeterminants. In this section, we demonstrate how this factorization can be automated for the SoS and perturbing the world schemes. We provide a concrete example at the end of the section. Note that the general symbolic scheme does not use an explicit symbolic perturbation and can be resolved by directly differentiating the expression of the predicate in terms of the original coordinates.

Consider an algebraic expression q used to evaluate a particular geometric predicate. Typically, q is the expansion of a determinant, but here we consider the general case where it is a multivariate polynomial in the original coordinates. Let q_ε be the multivariate polynomial obtained by evaluating q on the ε-expansion of the input coordinates. We assume that a total ordering is defined on all ε-products in q_ε. Instead of manually factoring q_ε, we use computer algebra systems such as Mathematica [40] or SymPy [28] to automate the steps for producing the relevant evaluation tables.

For each ε-product w in the ordered sequence of all ε-products we define T_w to be the sum of all terms in q_ε which contain exactly w and no other ε-variables. Then $T_w = c \times w$, where c is an expression that only involves the original point coordinates. Note that the first ε-product $w = 1$ contains no ε-variables and corresponds to evaluating the predicate on the non-perturbed coordinates. If c is non-zero, we return its sign. Otherwise, we proceed down the list until a non-zero c is found.

A non-zero coefficient c is guaranteed to be found if there is at least one ε-product in q_ε that is multiplied by a constant. Once such a term is found, there is no need to evaluate further terms, as they are guaranteed to be orders of magnitude smaller. Therefore, our procedure can be used to evaluate predicates more general than

those expressible as determinants [16]. For determinant predicates, we can compute the table of relevant determinants produced by the SoS method by listing all coefficients c in order. We will refer to the table produced by our method more generally as an *evaluation table* since it does not need to include any subdeterminant. We illustrate this procedure in Algorithm 1.

Algorithm 1 Evaluation Table of a Symbolic Polynomial

```
procedure ComputeEvaluationTable(q_ε)
    W ← ordered list of all ε-products in q_ε.
    E ← [ ]                                   ▷ Evaluation Table
    for w ∈ W do
        T_w ← all terms of q_ε that contain exactly w
        c ← T_w/w
        E.add(c)
        if c is a constant then
            return E
        end if
    end for
end procedure
```

Finally, we present an example. Let $p_i(\varepsilon) = (p_{i,1} + \varepsilon_{i,1}, p_{i,2} + \varepsilon_{i,2})$ and $p_j(\varepsilon) = (p_{j,1} + \varepsilon_{j,1}, p_{j,2} + \varepsilon_{j,2})$ be the ε-expansions of two points with SoS. Suppose that we have an expression for a predicate q_ε where the terms are ordered and grouped by their ε-product ordering. Let $q_\varepsilon = (p_{i,1} + p_{j,2}) + (p_{i,1}\varepsilon_{i,1} + p_{i,2}\varepsilon_{i,1}) + 10\varepsilon_{i,1}\varepsilon_{j,2} + p_{i,1}\varepsilon_{j,2}$. First, we evaluate $p_{i,1} + p_{j,2}$. If that is zero, we then evaluate $p_{i,1} + p_{i,2}$, and return its sign if it is nonzero. Finally, $\varepsilon_{i,1}\varepsilon_{j,2}$ has a constant coefficient of 10, so we return a positive sign. We do not evaluate the coefficient of $\varepsilon_{j,2}$, since $\varepsilon_{i,1}\varepsilon_{j,2} \gg \varepsilon_{j,2}$.

7 Implementation

We implemented all predicates and perturbation schemes discussed Sec. 4, as well as Algorithm 1, using the SymPy symbolic mathematics package [28] in Python. Symbolic variables were used to represent the ε-expansion of all coordinates, utilizing SymPy's built-in capabilities for factoring, simplifying, and differentiating symbolic expressions. We used Python's `itertools` library to generate orderings of ε-products and partial derivatives. The output of our code is a set of evaluation tables for all required geometric predicates. Our code is freely available on GitHub [20].

The evaluation tables for each predicate can be used to compute perturbed signs by substituting numerical values for the symbolic coordinates. However, doing this directly in Python is slow. In order to generate high-performance C++ code, we advise the use of SymPy's built-in `ccode` functionality. The rows of each evaluation table can be organized into numerous cases in a `switch` statement and exported as C++ functions in a header file. This header file can then be integrated with existing code for computing Jacobi sets of the Reeb spaces in order to handle the predicates robustly.

8 Evaluation

So far, we have established three important things about the symbolic perturbation of our predicates. The first is that both the parametrized and dualized formulations of the order of segments predicate have the same expression. The second is that the general symbolic scheme could be more efficient in practice when implemented with the lexicographic ordering, rather than with the total ordering, because the earlier rows in the table have fewer terms. The third is that the perturbation scheme that perturbs the world is less practical because it requires significantly more arithmetic operations than the other two schemes.

The goal of this section is to evaluate the practical performance of our proposed predicates with two experiments. First, we would like to understand how often degenerate cases occur in real-world datasets. This gives us an idea of how often perturbation is required to evaluate them. Our second goal is to determine how many terms and arithmetic operations are needed on average to resolve degenerate cases. In order to estimate this, we run our predicates on synthetically generated degenerate cases.

Real-world Data

We have chosen multivariate datasets widely used in the scientific visualization community [5, 24]. All datasets are sampled on the vertices of a regular cubic grid, which we then subdivide using five tetrahedra per cube in ParaView [1]. In the first five columns of Tab. 4 we list the original grid resolution and the number of simplices in the resulting tetrahedral meshes. In the engine, scission, and tooth datasets, one of the scalar fields is the gradient magnitude of the primary scalar field.

To determine the number of degenerate cases for each dataset, we would need to count the number of duplicate points, three or more collinear points, and three or more segments that are concurrent in their interiors (see Fig. 2). Aside from counting duplicate points, which can be done efficiently with a hash map, it is infeasible to count the exact number of degenerate cases for the other two predicates. This is because the number of triplets for up to 16 million vertices and 100 million edges is on the order of 10^{17} to 10^{19}. Generating all such triplets is impractical and using memorization to test pairs instead of triplets requires terabytes of memory.

In our experiment, we estimate the number of degenerate cases by generating 10^{10} random triplets of vertices and edges. We then used CGAL [37] to test for collinearity and concurrency in the corresponding points and segments using an exact kernel. As shown in Fig. 4, the number of duplicate and collinear points is significant in some datasets, particularly those where one scalar field is the gradient of the other. In contrast, there are far fewer degenerate cases for concurrent segments. This can be explained by the fact that, in such cases, six points (the segment endpoints) must align, rather than just two or three. Although we report zero degenerate cases involving concurrent segments in three of the datasets, we cannot guarantee that none exist. We may simply have not sampled them.

We can confirm that degenerate cases of all types (see Fig. 2) do occur in real-world bivariate datasets. In particular, highly correlated scalar fields are more likely to have similarities that could lead to degenerate cases. Although degenerate configurations are, in a sense, local, they can affect the global combinatorial structure of Jacobi sets and Reeb spaces. This clearly demonstrates the need for robust predicates introduced in this paper.

Synthetic Data

Having established that all degenerate cases occur in real-world datasets, we now evaluate how efficiently they can be resolved using the perturbation schemes proposed in Sec. 4. For this evaluation, we used synthetically generated degenerate cases. For collinear points, we randomly generated three rational points along the $y = x$ line and then applied a random rotation and translation. For concurrent segments, we used three random rational numbers t_1, t_2 and t_3 to generate three rational points on the unit circle with a stereographic projection, where $x_i = (1 - t_i^2)/(1 + t_i^2)$ and $y_i = (2t_i)/(1 + t_i^2)$ for $i \in \{1, 2, 3\}$. Taking each point and its antipodal results in three concurrent segments, which we then randomly translated and rotated. Using rational numbers in all cases is crucial to ensure robustness (see Sec. 2.1).

We evaluated Yap's general scheme [42] with both lexicographic (YL) and total ordering (YT), as well as the SoS scheme [16]. We did not evaluate the perturb the world scheme, because, as we saw in Sec. 4, it produces significantly more terms. The results of our experiments are shown in Tab. 5 and Tab. 6. In Tab. 5, we report statistics on the depth and number of arithmetic operations across one million randomly generated degenerate cases, for

Dataset	Dimensions	Tetrahedra	Vertices	Edges	Duplicates	Collinear Points	Concurrent Segments
engine	255×255×109	35,438,625	7,208,960	42,888,814	5,884,231	1,050,940,003	57
enzo	256×256×256	82,906,875	16,777,216	100,074,240	86,344	9,832,907	0
ethane-diol	115×116×134	8,718,150	1,787,560	10,592,843	92	1	0
isabel1	125×125×100	7,611,120	1,562,500	9,253,475	6,631	537,992	0
scission	40×40×66	494,325	105,600	613,106	104,665	412,777,680	16,280
tooth	103×94×161	7,588,800	1,558,802	9,228,973	5,255	35,780,055	1

Table 4: Number of degenerate cases found for each dataset using random sampling with $n = 10^{10}$. Duplicates refer to the number of vertices that map to the same point in the range as another vertex. Collinear refers to the number of collinear triplets of points in the range for n samples of triplets. Collinear refers to the number of collinear triplets of segments in the range for n samples of triplets.

both collinear points and concurrent segments. Arithmetic operations were counted as the number of additions and multiplications present in each symbolic expression. In practice, addition and multiplication may have different costs, depending on library-specific implementation details. All computations were performed using SymPy's rational types to ensure exact arithmetic.

The performance of all perturbation schemes is almost identical, as we can see from Tab. 5. The average number of operations and standard deviation differ by very small margins. Overall, SoS uses the fewest arithmetic operations, closely followed by the general symbolic scheme with total and lexicographic orderings. Interestingly, lexicographic ordering does not perform better than total ordering as predicted. The smaller maximum depth of the lexicographic ordering is never reached as we can see from Tab. 6. Furthermore, larger expressions earlier in the table, which we see with total ordering, may actually be beneficial, since they can help resolve a case with less depth. Finally, we demonstrate in Tab. 6 that 99.9% of all degenerate cases are resolved with only one additional evaluation (depth one). This demonstrates that the schemes we have proposed are efficient and do not introduce a substantial overhead.

		Depth		#Operations	
Predicate	Scheme	Mean	SD	Mean	SD
Orient	YL	1.001	0.001	11.001	0.030
	YT	1.001	0.001	11.001	0.030
	E	1.001	0.001	11.001	0.030
Order	YL	1.001	0.040	191.051	1.697
	YT	1.001	0.035	191.051	1.646
	E	1.001	0.032	191.046	1.488

Table 5: Mean and standard deviation (SD) of depth and number of arithmetic operations by predicate and scheme.

		Depth Count						
Predicate	Scheme	1	2	3	5	6	7	14
Orient	YL	999,087	913	0	0	0	0	0
	YT	999,087	913	0	0	0	0	0
	E	999,087	913	0	0	0	0	0
Order	YL	998,961	1027	0	0	0	11	1
	YT	998,961	1027	0	11	1	0	0
	E	999,030	959	11	0	0	0	0

Table 6: Depth counts by predicate and scheme.

9 Discussion and Limitations

In our evaluation in Sec. 8 we have confirmed that degenerate cases do occur in real-life data and that they can be handled robustly without substantial overhead. However, symbolic perturbation has faced multiple concerns in the computational geometry literature [34, 35]. First, it complicates the debugging process during algorithm development. For example, visualizing the perturbed arrangement of segments becomes increasingly difficult. Second, we compute the topological structure of the perturbed data, not the original. This is an important distinction, even if we assume that real-world data contains noise, because, to the best of our knowledge, there are no stability results that allow us quantify the effects of a perturbation for Jacobi sets and Reeb spaces. Furthermore, perturbation introduces artificial features and there are currently no established simplification methods that can eliminate them. Our approach also relies on exact arithmetic, and its overall impact on algorithm performance requires further investigation.

Modern computational geometry libraries such as CGAL [37] treat degenerate cases as legitimate parts of the input and handle them explicitly, without relying on perturbation. This approach is generally faster and conceptually cleaner, but only when all possible degenerate cases are known and can be classified. It remains to be seen whether classifying all degenerate cases for Reeb spaces and Jacobi sets is tractable, even in the bivariate case.

Finally, note that our approach applies only to bivariate PL maps. For higher-dimensional PL maps, supporting the geometric predicates required for higher-dimensional geometric arrangements will be necessary. In other cases, such as cubic meshes with a trilinear interpolant, the set of Jacobi edges is not guaranteed to be a subset of the domain's edges. However, this is currently not an issue since all Jacobi set and Reeb space algorithms assume a PL map.

10 Conclusion

In this paper, we presented both the theoretical foundation and practical implementation of robust predicates for bivariate Jacobi sets and Reeb spaces algorithms. We surveyed relevant work in computational geometry on robust geometric predicates and highlighted the unique challenges these predicates face in multivariate computational topology, particularly the need for exact geometric computation and the emergence of complex expressions under symbolic perturbation. To handle these nontrivial expressions, we introduced a general procedure for evaluating them automatically, rather than manually, as is typically done in the literature.

We demonstrated that all predicates required for computing Jacobi sets and Reeb spaces can be handled robustly using various symbolic perturbation schemes, including, but not limited to, the well-known Simulation of Simplicity. Then, we assessed the practical performance of these schemes and confirmed that degenerate cases do, in fact, arise in real-world bivariate datasets. However, we also demonstrated that most degenerate cases are resolved at very shallow depths requiring only a few additional operations. Finally, we outlined a procedure for generating efficient C++ code to evaluate these predicates without using any symbolic arithmetic.

Acknowledgments

This work was partially supported by the Wallenberg Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation; the Swedish e-Science Research Center (SeRC); and the Swedish Research Council (VR) grant 2023-04806.

REFERENCES

[1] J. Ahrens, B. Geveci, and C. Law. Paraview: An end-user tool for large data visualization. In C. D. Hansen and C. R. Johnson, eds., *Visualization Handbook*, pp. 717–731. Elsevier, 2005. 8

[2] P. Alliez, O. Devillers, and J. Snoeyink. Removing Degeneracies by Perturbing the Problem or Perturbing the World. *Reliable Computing*, 6(1):61–79, Feb. 2000. doi: 10.1023/A:1009942427413 4, 5, 6

[3] K. Beketayev, D. Yeliussizov, D. Morozov, G. Weber, and B. Hamann. Measuring the distance between merge trees. *Topological Methods in Data Analysis and Visualization III: Theory, Algorithms, and Applications, Mathematics and Visualization*, pp. 151–166, 2014. 4

[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, Feb. 2008. doi: 10.1016/j.tcs.2007.10.018 1, 4

[5] H. Carr, Z. Geng, J. Tierny, A. Chattopadhyay, and A. Knoll. Fiber Surfaces: Generalizing Isosurfaces to Bivariate Data. 34(3):241–250, 2015. doi: 10.1111/cgf.12636 8

[6] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry Theory and Applications*, 24(2):75–94, 2003. 1, 4

[7] A. Chattopadhyay, Y. Ramamurthi, and O. Saeki. An algorithm for correct computation of reeb spaces for pl bivariate fields. 2024. doi: 10.48550/arXiv.2403.06564 1, 5, 7

[8] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer Berlin Heidelberg, 2008. doi: 10.1007/978-3-540-77974-2 2, 5

[9] O. Devillers, M. Karavelas, and M. Teillaud. Qualitative symbolic perturbation. In S. Fekete and A. Lubiw, eds., *32nd International Symposium on Computational Geometry (SoCG 2016)*, vol. 51 of *Leibniz International Proceedings in Informatics (Lipics)*, pp. 33:1–33:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi: 10.4230/LIPIcs.SoCG.2016.33 4

[10] Edelsbrunner, Letscher, and Zomorodian. Topological Persistence and Simplification. 28(4):511–533, 2002. doi: 10.1007/s00454-002-2885-2 1

[11] H. Edelsbrunner and J. Harer. Jacobi sets of multiple Morse functions. pp. 37–57, 2002. 1, 4

[12] H. Edelsbrunner and J. Harer. Persistent homology – a survey. In J. E. Goodman, J. Pach, and R. Pollack, eds., *Surveys on Discrete and Computational Geometry: Twenty Years Later*, vol. 458, pp. 257–282. AMS Bookstore, 2008. 4

[13] H. Edelsbrunner and J. Harer. *Computational Topology*. American Mathematical Society, Providence, Rhode Island, Dec. 2009. doi: 10.1090/mbk/069 4, 11

[14] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Morse-Smale complexes for piecewise linear 3-manifolds. *Proceedings of the 19th Annual symposium on Computational Geometry*, pp. 361–370, 2003. 4

[15] H. Edelsbrunner, J. Harer, and A. K. Patel. Reeb spaces of piecewise linear mappings. In *Proceedings of the Twenty-Fourth Annual Symposium on Computational Geometry*, pp. 242–250, 2008. 1, 5, 7

[16] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, Jan. 1990. doi: 10.1145/77635.77639 1, 2, 3, 4, 5, 6, 7, 8, 11

[17] I. Z. Emiris, J. F. Canny, and R. Seidel. Efficient Perturbations for Handling Geometric Degeneracies. 19(1):219–242, 1997. doi: 10.1007/PL00014417 2, 3, 5

[18] R. Fuchs and H. Hauser. Visualization of multi-variate scientific data. *Computer Graphics Forum*, 28(6):1670–1690, 2009. 2

[19] J. E. Goodman, J. O'Rourke, and C. D. Tóth, eds. *Handbook of Discrete and Computational Geometry*. Discrete Mathematics and Its Applications. CRC Press, Boca Raton London New York, third edition ed., 2018. 1, 2, 3

[20] P. Hristov. Symbolic perturbation for bivariate computational topology. https://github.com/peter-hristov/symbolic-perturbation-for-bivariate-computational-topology, 2025. Accessed: 2025-08-12. 8

[21] P. Hristov, D. Sakurai, H. Carr, I. Hotz, and T. B. Masood. Arrange and Traverse Algorithm for Computation of Reeb Spaces of Piecewise Linear Maps. *Computer Graphics Forum*, 2025. doi: 10.1111/cgf.70206 1, 4, 5

[22] P. G. Hristov. Hypersweeps, Convective Clouds and Reeb Spaces, 2022. 4, 5, 7

[23] G. Irving and F. Green. A deterministic pseudorandom perturbation scheme for arbitrary polynomial predicates, 2013. doi: 10.48550/arXiv.1308.1986 3

[24] J. Jankowai and I. Hotz. Feature Level-Sets: Generalizing Iso-Surfaces to Multi-Variate Data. *IEEE Transactions on Visualization and Computer Graphics*, 26(2):1308–1319, 2019. doi: 10.1109/TVCG.2018.2867488 8

[25] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap. Classroom examples of robustness problems in geometric computations. 40(1):61–78, 2008. doi: 10.1016/j.comgeo.2007.06.003 3, 7

[26] J. Lukasczyk, C. Garth, R. Maciejewski, and J. Tierny. Localized topological simplification of scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):572–582, 2021. doi: 10.1109/TVCG.2020.3030353 1

[27] K. Mehlhorn, R. Osbild, and M. Sagraloff. A general approach to the analysis of controlled perturbation algorithms. 44(9):507–528, 2011. doi: 10.1016/j.comgeo.2011.06.001 3

[28] A. Meurer, C. P. Smith, M. Paprocki, O. Certik, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017. doi: 10.7717/peerj-cs.103 7, 8

[29] K. Moreland, C. Sewell, W. Usher, L.-T. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K. L. Ma, H. Childs, M. Larsen, C. M. Chen, R. Maynard, and B. Geveci. VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures. *IEEE Computer Graphics and Applications*, 36(3):48–58, May 2016. doi: 10.1109/MCG.2016.48 1

[30] V. Natarajan. Topological analysis of scalar functions for scientific data visuzliation, 2004. 4

[31] C. P. Rourke. *Introduction to Piecewise-Linear Topology*. Springer Study Edition Ser. Springer Berlin / Heidelberg, Berlin, Heidelberg, 1982. 4

[32] C. P. Rourke and B. J. Sanderson. *Introduction to piecewise-linear topology*. Springer Science & Business Media, 2012. 11

[33] O. Saeki. *Topology of Singular Fibers of Differentiable Maps*, vol. 1854 of *Lecture Notes in Mathematics*. Springer Berlin Heidelberg, 2004. doi: 10.1007/b100393 4, 5

[34] P. Schorn. Degeneracy in geometric computation and the perturbation approach. 37(1):35–42, 1994. doi: 10.1093/comjnl/37.1.35 9

[35] R. Seidel. The Nature and Meaning of Perturbations in Geometric Computing. 19(1):1–17, 1998. doi: 10.1007/PL00009330 3, 4, 9

[36] M. Sharma and V. Natarajan. Jacobi Set Driven Search for Flexible Fiber Surface Extraction. In *2022 Topological Data Analysis and Visualization (TopoInVis)*, pp. 49–58, 2022. doi: 10.1109/TopoInVis57755.2022.00012 1, 4

[37] The CGAL Project. *CGAL User and Reference Manual*, 2023. Version 5.5. 3, 8, 9

[38] J. Tierny and H. Carr. Jacobi Fiber Surfaces for Bivariate Reeb Space Computation. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):960–969, Jan. 2017. doi: 10.1109/TVCG.2016.2599017 1, 4, 5, 7

[39] 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 1

[40] Wolfram Research, Inc. *Mathematica, Version 13.3*. Wolfram Research, Inc., Champaign, Illinois, 2023. https://www.wolfram.com/mathematica/. 7

[41] C.-K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. 40(1):2–18, 1990. doi: 10.1016/0022-0000(90)90016-E 2, 3, 7

[42] C.-K. Yap. Symbolic treatment of geometric degeneracies. 10(3):349–370, 1990. doi: 10.1016/S0747-7171(08)80069-7 2, 3, 4, 6, 7, 8

A Background on PL Topology

In this appendix we review some foundational definitions from PL topology. For a comprehensive introduction, we refer the reader to relevant textbooks [13, 32].

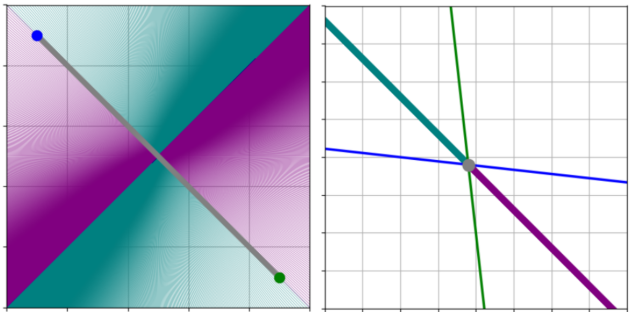
A *simplex* σ of dimension d is a convex combination of d affinely independent points in $\mathbb{R}^{\mathsf{d}+1}$. The points that define σ are called its *vertices*, and we say that a simplex is *spanned* by its vertices. A subset of a simplex spanned by a subset of its vertices is called a *face* of that simplex. For example, simplices of dimension $0, 1, 2$ and 3 are points, line segments, triangles, and tetrahedra respectively.

A simplicial complex K is a collection of simplices where the faces of any simplex are also in K and the intersection of any two simplices is either empty or a face of both. The underlying geometric space of K is denoted as $|K|$. We say that K triangulates a manifold when $|K|$ is homeomorphic to a manifold. The notion of a topological neighbourhood is defined for a simplicial complex by using the *star* and *link* of a simplex σ. The star of σ is the set of simplices that have σ as a face, and the link of σ consists of the faces of all simplices in the star of σ that do not intersect σ.

A scalar function $f : |K| \to \mathbb{R}$ is PL when its values are only defined on the vertices of K and extend to the interiors of the simplices via barycentric interpolation. *Critical vertices* of scalar PL functions are defined using the upper and lower link of a vertex. The *upper link* of a vertex v consists of all simplices σ in the link of v whose vertices have a higher function value than $f(v)$. The *lower link* is defined analogously with all simplices σ in the link of v whose vertices have a lower function value than $f(v)$.

B Order of Segments with Ideal Points

Example of when the lines of segments are be mapped to different wedges (see Sec. 3.1).



(a) Lines in the plane. (b) Dual Homogeneous Coordinates.

Figure 8: Given a segment with blue and green endpoints, we show which lines ℓ intersecting the halfway point of the segment are mapped in either of the two wedges that correspond to the segment in the dual projective plane. As ℓ becomes close to the line defined by the segment, its corresponding point in the dual projective plane becomes closer to the point that corresponds to the line defined by the segment in the primal plane. As ℓ becomes close to the line that connects the halfway point and the origin it moves away from the gray point in the dual projective plane and into infinity.

Orientation Tests with Ideal Points. In the dual formulation of the segment order predicate described in Sec. 3.1, lines that pass through the origin in the primal plane correspond to ideal points at infinity in the dual plane. This requires special handling, as orientation tests are not well defined in that case [16]. This is a consequence of the projective plane being a non-orientable topological space. Nevertheless, the determinant used in orientation tests can still be meaningfully interpreted. When all three points are at infinity, the determinant will be zero, and this case can be perturbed with SoS.

When two points are at infinity, they can be geometrically interpreted as direction vectors in the affine plane at $z = 0$. An orientation between two direction vectors and a finite homogeneous point reduces to computing the determinant of the direction vectors scaled by the last coordinate of the finite point. This is well defined both algebraically and geometrically. Geometrically, it is equivalent to the points $\ell^*_{3,4}$ and $\ell^*_{5,6}$ form Fig. 7 sliding along the lines i^*_1 and i^*_2 respectively to infinity (see Fig. 8). Algebraically, this follows from the expression:

$$\det \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & 0 \\ x_3 & y_3 & 0 \end{pmatrix} = z_1 \det \begin{pmatrix} x_2 & y_2 \\ x_3 & y_3 \end{pmatrix}.$$

We do not need to explicitly account for the sign of z_1 because it is already accounted for in the evaluation of the orientation test in homogeneous coordinates [16]. However, we need to normalize the direction vectors $[x_2, y_2, 0]$ and $[x_3, y_3, 0]$ using their first non-zero coordinate with two additional sign checks.

When one point is at infinity, the orientation test reduces to comparing the orientation of the affine vector defined by the two finite points and the affine vector defined by the point at infinity because:

$$\det \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & 0 \end{pmatrix} = z_1 z_2 \det \begin{pmatrix} \frac{x_2}{z_2} - \frac{x_1}{z_1} & \frac{y_2}{z_2} - \frac{y_1}{z_1} \\ x_3 & y_3 \end{pmatrix}.$$

As in the previous case, we do not need to explicitly account for the signs of z_1 and z_2, because they will be accounted for in the evaluation of the determinant, but we do need to account for the sign of normalizing the direction vector (the point at infinity).