Time-varying Reeb Graphs for Continuous Space-Time Data [*]

Herbert Edelsbrunner[†], John Harer[‡], Ajith Mascarenhas[§] and Valerio Pascucci[¶]

ABSTRACT

We study the evolution of the Reeb graph of a time-varying continuous function defined in three-dimensional space. While maintaining the Reeb graph, we compress the evolving sequence into a single, partially persistent data structure. We envision this data structure as a useful tool in visualizing real-valued space-time data obtained from computational simulations of physical processes.

Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

General Terms: Algorithms, Theory.

Keywords: Differential and computational topology, Morse functions, critical points, level sets, Reeb graph, triangulations, combinatorial algorithms.

1. INTRODUCTION

This paper studies the evolution of the Reeb graph of a continuous function that varies with time. We begin by motivating the topic, reviewing related prior work, and stating our results.

Motivation. Physical processes that are measured over time, or modeled and simulated on a computer, can produce large amounts of data that must be interpreted with the assistance of computational tools. Such data arises in a wide variety of studies, including computational fluid dynamics [8], oceanography[5], and climate modeling [17]. The data typically consists of finitely many points in space-time and a measured value for each. Independent of whether or not the points are sampled on a regular grid, we can connect them into a mesh and interpolate the values to obtain a continuous function over the entire domain. Piecewise linear interpolation is common for large amounts of data, because of its relative ease; multi-linear interpolation is also used for regular grids.

A useful tool in interpreting the data is graphical visualization, often through level sets, or iso-surfaces of a continuous function derived from the data. Fixing a real number s, the level set consists of all points in the domain whose function values are equal to s. In $\mathbb{R}^3$, this is generically a surface that is then displayed. By varying s, we can study the variation in the data. Topological features of the level sets, such as connected components, handles, and voids, can be important aids in interpreting the data. By encoding the evolution of these features, the Reeb graph compactly represents topological information of all level sets. As we pass through time, the Reeb graph goes through an evolution of its own, undergoing structural changes at birth-death points and at interchanges of critical points. The evolution of the Reeb graph thus represents a 2-parameter family of level sets. We suggest that this 2-parameter family encoded in a compact data structure is a useful representation of space-time data.

Related prior work. In the interactive exploration of scientific data, Reeb graphs are used to select meaningful level sets [3] and to efficiently compute them [15]. An extensive discussion of Reeb graphs and related structures in geometric modeling and visualization applications can be found in [14]. All published algorithms for Reeb graphs take as input a function defined on a triangulated manifold. We express their running time as functions of n, the number of simplices in the triangulation. The first algorithm for functions on 2-manifolds due to Shinagawa and Kunii [22] takes time $O(n^2)$ in the worst case. The case of loop-free Reeb graphs, also known as contour trees, has received special attention because the algorithms are simpler and Reeb graphs of the practically important class of simply-connected domains have no loops. An algorithm that constructs contour trees of functions on simply-connected manifolds of constant dimension in time $O(n \log n)$ has been suggested in [7]. For the case of 3-manifolds, this algorithm has been extended to include information about the genus of the level surfaces [20]. Recently, Cole-McLaughlin et al. [9] returned to the general case, giving tight bounds on the number of loops in Reeb graphs of functions on 2-manifolds and describing an $O(n \log n)$ time algorithm to construct them.

[*]Research of the first author is partially supported by NSF under grants EIA-99-72879 and CCR-00-86013. Research of the second author is partially supported by NSF under grant DMS-01-07621. Research of the third author is partially supported by NSF under grant 0128426 and by subcontracts from Lawrence Livermore National Labs. This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48

[†]Department of Computer Science and Mathematics, Duke University, Durham, and Raindrop Geomagic, Research Triangle Park, North Carolina.
[‡]Department of Mathematics and Computer Science, Duke University, Durham, North Carolina.
[§]Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina.
[¶]Center for Applied Scientific Computing, Lawrence Livermore National Labs, Livermore, California.

Results. In this paper, we study the details of how the Reeb graph of a smooth function on three-dimensional space evolves over time. It will be convenient to compactify the space to a closed manifold, which we do by adding the point at infinity, effectively creating the topology of the 3-sphere, denoted by $\mathbb{S}^3$. Our first contribution is a complete enumeration of the type of combinatorial changes the Reeb graph experiences:

- nodes disappear in pairs, contracting arcs to points (inversely, node appear in pairs, anti-contracting arcs);
- nodes swap their positions along the arcs of the graph.

Perhaps surprisingly, the second type of change is more interesting, falls into more sub-types, and is algorithmically more difficult to handle than the first type. Based on this classification, we give an algorithm that maintains the Reeb graph through time and stores the evolution in a partially persistent data structure. The size of this data structure is proportional to the size of the initial Reeb graph, at time zero, plus the number of changes it experiences through time.

Outline. Section 2 reviews background material. Section 3 enumerates the types of changes experienced by a Reeb graph. Section 4 and 5 describe the algorithm for maintaining the Reeb graph. Section 6 concludes the paper and states a few open questions.

2. MATHEMATICAL BACKGROUND

We need some background from Morse theory [16, 18] and from combinatorial and algebraic topology [2, 19].

Smooth maps on manifolds. Let $\mathbb{M}$ be a smooth, compact d-manifold without boundary and $f : \mathbb{M} \to \mathbb{R}$ a smooth map. Assuming a local coordinate system in its neighborhood, $x \in \mathbb{M}$ is a *critical point* of f if all partial derivatives vanish at x. If x is a critical point, $f(x)$ is a *critical value*. Non-critical points and non-critical values are called *regular points* and *regular values*, respectively. The *Hessian* at x is the matrix of second-order partial derivatives. A critical point x is *non-degenerate* if the Hessian at x is non-singular. The *index* of a critical point x, denoted by $\operatorname{index} x$, is the number of negative eigenvalues of the Hessian. Intuitively, it is the number of mutually orthogonal directions at x along which f decreases. For $d = 3$ there are four types of non-degenerate critical points: *minima* with index 0, 1-*saddles* with index 1, 2-*saddles* with index 2, and *maxima* with index 3. A function f is *Morse* if

I. all critical points are non-degenerate;
II. $f(x) \neq f(y)$ whenever $x \neq y$ are critical.

We will refer to I and II as Genericity Conditions as they prevent certain non-generic configurations of the critical points. This choice of name is justified because Morse functions are dense in $C^\infty(\mathbb{M})$, the class of smooth functions on the manifold. In other words, for every smooth function there is an arbitrarily small perturbation that makes it a Morse function.

The critical points of a Morse function and their indices capture a whole lot of information about the manifold on which the function is defined. For example, the Euler characteristic of the manifold $\mathbb{M}$ is equal to the alternating sum of critical points, $\chi(\mathbb{M}) = \sum_x (-1)^{\operatorname{index} x}$. Another useful tool in the study of manifolds is the incremental construction by adding one cell at a time. Call a space homeomorphic to the k-dimensional ball a *k-cell*. Let $f : \mathbb{M} \to \mathbb{R}$ be a Morse function and define $\mathbb{M}_s = f^{-1}(-\infty, s]$. The boundary of $\mathbb{M}_s$ is the *level set*, $f^{-1}(s)$, defined by $s \in \mathbb{R}$. If s is a regular value then $f^{-1}(s)$ is a $(d-1)$-manifold. For positive ε consider $\mathbb{M}_{s+\varepsilon}$ and assume that f has a single critical point with function value in $(s, s+\varepsilon]$. If the index of that critical point is k then $\mathbb{M}_{s+\varepsilon}$ is homotopy equivalent to $\mathbb{M}_s$ with a single k-cell attached.

Reeb graph. A level set of f is not necessarily connected. Calling two points $x, y \in \mathbb{M}$ *equivalent* if $f(x) = f(y)$ and both points belong to the same component of the level set, we obtain the *Reeb graph* as the quotient space in which every equivalence class is represented by a point and connectivity is defined in terms of the quotient topology [21]. Figure 1 illustrates the definition for a 2-manifold of genus two. We call a point on the Reeb graph a *node*

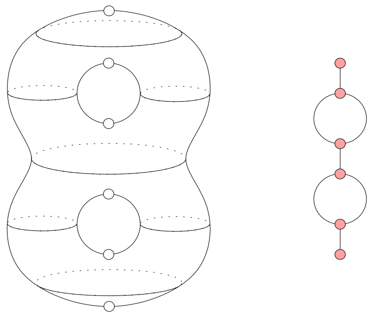

Figure 1: The Reeb graph of the function f on a 2-manifold, which maps every point of the double torus to its distance above a horizontal plane below the surface.

if the corresponding level set component passes through a critical point of f. The rest of the Reeb graph consists of arcs connecting the nodes. The *degree* of a node is the number of arcs incident to the node. A minimum creates and a maximum destroys a level set component and both correspond to degree-1 nodes. A saddle that splits one level set component in two or merges two to one corresponds to a degree-3 node. There are also saddles that alter the genus but do not affect the number of components, and they correspond to degree-2 nodes in the Reeb graph. Nodes of degree higher than three occur only for non-Morse functions.

In mathematics, the Reeb graph is often used to study the manifold $\mathbb{M}$ that forms the domain of the function. For example, the Reeb graph in Figure 1 reveals that the function is defined on a double torus, assuming we know it is an orientable 2-manifold without boundary. In contrast, we will use the Reeb graph to study the behavior of functions. The domain of interest is $\mathbb{R}^3$ but it is convenient to compactify it and consider functions on the 3-sphere, $\mathbb{S}^3$. All our Reeb graphs will reveal the (un-exciting) connectivity of $\mathbb{S}^3$ by being trees, but the structure of the tree will tell us something about the chosen function f.

Jacobi curves. We use Reeb graphs to understand a function at moments in time and Jacobi curves, as introduced in [12], to get a glimpse of its evolution through time. We introduce this concept for the slightly more general case of two Morse functions, $f, g : \mathbb{M} \to \mathbb{R}$; the specific case of a time-varying function, f, is obtained by adding time as an extra dimension to the domain and letting g represent time. For a regular value $t \in \mathbb{R}$, we have the level set $g^{-1}(t)$ and the restriction of f to this level set, $f_t : g^{-1}(t) \to \mathbb{R}$. The *Jacobi curve* of f and g is the closure of the set of critical points of the functions f_t, for all $t \in \mathbb{R}$. The closure operation adds the critical points of f restricted to level sets at critical values, as well as the critical points of g, which form singularities in these level sets. Figure 2 illustrates the definition by showing the Jacobi curve of two smooth functions on a piece of the two-dimensional plane.

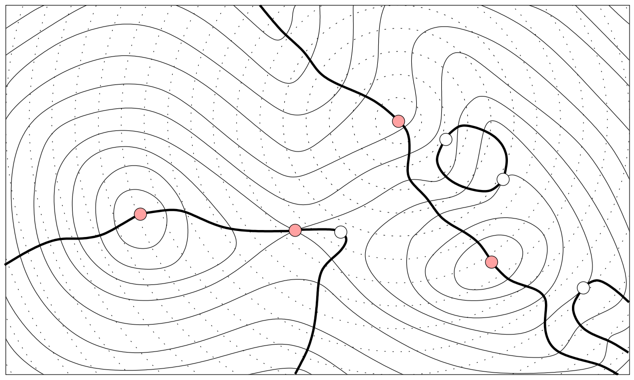

Figure 2: The functions f and g are represented by their dotted and solid level curves. The Jacobi curve is drawn in bold solid lines. The birth-death points and the critical points of the two functions are marked by white and shaded dots, respectively.

We consider a 1-parameter family of Morse functions on the 3-sphere, $f : \mathbb{S}^3 \times \mathbb{R} \to \mathbb{R}$, and introduce an auxiliary function $g : \mathbb{S}^3 \times \mathbb{R} \to \mathbb{R}$ defined by $g(x,t) = t$. A level set has the form $g^{-1}(t) = \mathbb{S}^3 \times t$, and the restriction of f to this level set is $f_t : \mathbb{S}^3 \times t \to \mathbb{R}$. The Jacobi curve of f and g may consist of several components, and in the assumed generic case each is a closed 1-manifold. We can identify the *birth-death points* where the level sets of f and g and the Jacobi curve have a common normal direction. To understand these points, imagine a level set in the form of a (two-dimensional) sphere deforming, sprouting a bud, as we go forward in time. The bud has two critical points, one a maximum and the other a 2-saddle. At the time when the bud just starts sprouting there is a point on the sphere, a birth point, where both these critical points are born. Run this in reverse order to understand a death point. We decompose the Jacobi curve into *segments* by cutting it at the birth-death points. The index of the critical point tracing a segment is the same everywhere along the segment. The indices within two segments that meet at a birth-death point differ by one:

INDEX LEMMA. Let $f : \mathbb{M} \times \mathbb{R} \to \mathbb{R}$ be a 1-parameter family of Morse functions. The indices of two critical points created or destroyed at a birth-death point differ by exactly one.

PROOF. At time t, let f_t have a single birth point. We can choose a small positive ε such that there are no other birth-death points with time in $[t-\varepsilon, t+\varepsilon]$. Denote by x and y the two newly created critical points in $f_{t+\varepsilon}$ and let $k = \operatorname{index} x \leq \operatorname{index} y$. The point x either destroys a homology class of dimension $k-1$ or it introduces one of dimension k. The former case is ruled out as $\operatorname{index} y \geq k$, and a cell of dimension larger than or equal to k cannot compensate for the destroyed dimension $k-1$ class. In the latter case, when x creates a homology class of dimension k, we need a $(k+1)$-cell to cancel the homology class, which implies that $\operatorname{index} y = k+1$. The claim follows. $\square$

Piecewise linear functions. A *triangulation* of a manifold $\mathbb{M}$ is a simplicial complex, K, whose underlying space is homeomorphic to $\mathbb{M}$ [2]. Given values at the vertices, we obtain a continuous function on $\mathbb{M}$ by linear interpolation over the simplices of the triangulation. We need some definitions to talk about the local structure of the triangulation and the function. The *star* of a vertex u consists of all simplices that share u, including u itself, and the *link* consists of all faces of simplices in the star that are disjoint from u. The *lower link* is the subset of the link induced by vertices with function value less than u:

$$\begin{aligned} \operatorname{St} u &= \{\sigma \in K \mid u \subseteq \sigma\}, \\ \operatorname{Lk} u &= \{\tau \in K \mid \tau \subseteq \sigma \in \operatorname{St} u,\ u \not\in \tau\}, \\ \operatorname{Lk}_{-} u &= \{\tau \in \operatorname{Lk} u \mid v \in \tau \Rightarrow f(v) \leq f(u)\}. \end{aligned}$$

Critical points of piecewise linear functions have been introduced by Banchoff [4] as the vertices whose lower links have Euler characteristic different from 1. Our classification is finer than Banchoff's and based on the reduced Betti numbers of the lower link. The *k-th reduced Betti number*, denoted as $\tilde{\beta}_k$, is the rank of the k-th reduced homology group of the lower link: $\tilde{\beta}_k = \operatorname{rank} \tilde{\mathsf{H}}_k$. The reduced Betti numbers are the same as the usual (un-reduced) Betti numbers, except that $\tilde{\beta}_0 = \beta_0 - 1$ for non-empty lower links, and $\tilde{\beta}_{-1} = 1$ for empty lower links [19]. When the link is a 2-sphere only $\tilde{\beta}_{-1}$ through $\tilde{\beta}_2$ can be non-zero. Simple critical points have exactly one non-zero reduced Betti number, which is equal to 1; see Table 1. The first case in which this definition differs from Banchoff's is a double saddle obtained by combining a 1- and a 2-saddle into a single vertex. The Euler characteristic of the lower link is one, which implies that Banchoff's definition does not recognize it as critical. A *multiple saddle* is a critical point that falls outside the classification of Table 1 and therefore satisfies $\tilde{\beta}_{-1} = \tilde{\beta}_2 = 0$ and $\tilde{\beta}_0 + \tilde{\beta}_1 \geq 2$. By modifying the simplicial complex, it can be unfolded into simple 1-saddles and 2-saddles as explained in [13].

	$\tilde{\beta}_{-1}$	$\tilde{\beta}_0$	$\tilde{\beta}_1$	$\tilde{\beta}_2$
regular	0	0	0	0
minimum	1	0	0	0
1-saddle	0	1	0	0
2-saddle	0	0	1	0
maximum	0	0	0	1

Table 1: Classification of vertices into regular and simple critical points using the reduced Betti numbers of the lower link.

3. TIME-VARYING REEB GRAPHS

In this section, we study how the Reeb graph of a function changes with time. Specifically, we give a complete enumeration of the combinatorial changes that occur for a Morse function on $\mathbb{S}^3$.

Jacobi curves connect Reeb graphs. Let R_t be the Reeb graph of f_t, the function on $\mathbb{S}^3$ at time t. The nodes of R_t correspond to the critical points of f_t, and as we vary t, they trace out the segments of the Jacobi curve. The segments connect the family through time, giving us a mechanism for identifying nodes in different Reeb graphs. We illustrate this idea in Figure 3.

Generically, the function f_t is Morse. However, there are discrete moments in time at which f_t violates one or both Genericity Conditions of Morse functions and the Reeb graph of f_t experiences a combinatorial change. Since we have only one varying parameter, namely time, we may assume that there is only a single violation of the Genericity Conditions at any of these discrete moments, and there are no violations at all other times. Condition I is violated iff f_t has a birth-death point at which a cancellation annihilates two converging critical points or an anti-cancellation gives birth to two diverging critical points. Condition II is violated iff f_t

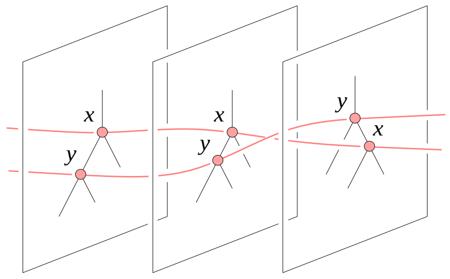

Figure 3: Reeb graphs at three moments in time whose nodes are connected by two segments of the Jacobi curve.

has two critical points $x \neq y$ with $f_t(x) = f_t(y)$ that form an interchange. The two critical points may be independent and have no effect on the Reeb graph, or they may belong to the same level set component of f_t and correspond to two nodes that swap their positions along the Reeb graph. We now analyze the changes caused by birth-death points and by interchanges in detail.

Nodes appear and disappear. When time passes the moment of a birth point, we get two new critical points and correspondingly two new nodes connected by an arc in the Reeb graph. By the Index Lemma, the indices of the two critical points differ by one, leaving three possibilities: 0-1, 1-2, and 2-3. Consider first the 0-1 case in which a minimum and a 1-saddle are born. In the Reeb graph, we get a new degree-1 node that starts an arc ending at a new degree-3 node. In other words, the Reeb graph sprouts a new arc downward from an existing branch; see Figure 4. The 2-3 case is upside-down symmetric to the 0-1 case, with the Reeb graph sprouting a new arc upward from an existing branch.

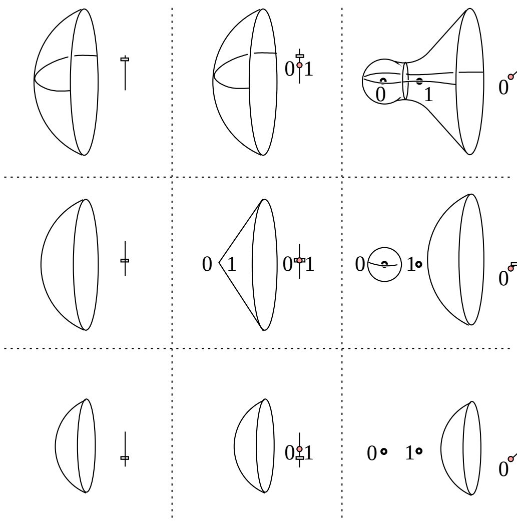

Figure 4: Level sets and Reeb graphs around a 0-1 birth point. Time increases from left to right and the level set parameter, indicated by a rectangular slider bar, increases from bottom to top. Going forward in time, we see the sprouting of a bud, while going backward in time we see its retraction.

Consider second the 1-2 case in which a 1-saddle and a 2-saddle are born. In the Reeb graph we get two new degree-2 nodes that effectively refine an arc by decomposing it into three arcs. As illustrated in Figure 5, this event corresponds to the appearance of a short-lived handle in the evolution of level sets. Turning the picture upside-down does not change anything, which shows that the case is symmetric to itself. We have similar three cases when time

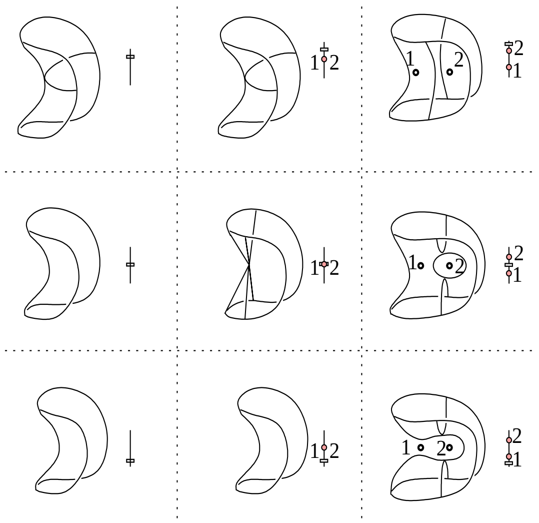

Figure 5: Level sets and Reeb graphs around a 1-2 birth point. Time increases from left to right and the level set parameter increases from bottom to top. Going forward in time, we see a refinement of an arc in the Reeb graph and going backward we see a coarsening.

passes the moment of a death point. Two critical points of f_t converge and annihilate when they collide, and correspondingly an arc of the Reeb graph contracts to a point, effectively removing its two nodes. The 0-1 and 2-3 cases are illustrated in Figure 4, which we now read from right to left, and the 1-2 case is illustrated in Figure 5, which we also read backward, from right to left.

Nodes swap. Nodes of the Reeb graph swap position in the Reeb graph when the corresponding critical points, x and y, form an interchange and, at that moment, belong to the same level set component. Assume without loss of generality that $f_{t-\varepsilon}(x) < f_{t-\varepsilon}(y)$ and $f_{t+\varepsilon}(x) > f_{t+\varepsilon}(y)$. We have four choices for each of x and y depending on whether they add or remove a handle, merge two level set components or split a level set component. This gives a total of sixteen configurations. We analyze possible before and after combinations and pair them, giving us the cases illustrated in Figure 6. It is convenient to group the cases with similar starting configurations together. We use $+, -, \mathrm{M}, \mathrm{S}$ to mean 'handle addition', 'handle deletion', 'component merge', and 'component split', respectively, and a pair of these to indicate the types of x and y.

`Case 1` $(++, +-, -+, --)$ Both x and y change the genus and their corresponding nodes simply swap their positions in the Reeb graph. [We pair $++$ with itself to get `Case 1a`, $+-$ with $-+$ to get `Case 1b`, and $--$ with itself to get `Case 1c`].

`Case 2` $(+\mathrm{M}, \mathrm{M}+, -\mathrm{M}, \mathrm{M}-)$ We consider two sub-cases.

$(\mathrm{M}+)$ Before the swap, x merges two components and y adds a handle. There are two possible configurations after the swap. Either y involves the two components that were merged by x, so x and y just swap, or y involves only one of the two components, so y goes down one of the branches at x. [In the first configuration, we pair $\mathrm{M}+$ with itself to get `Case 2a`, and in the second we pair $\mathrm{M}+$ with $+\mathrm{M}$ to get `Case 2b`.]

(M$-$) Before the swap, x merges two components and y removes a handle. After the swap, y moves down one of the branches at x. [We pair M$-$ with $-$M to get `Case 2c`].

`Case 3` ($-$S, S$-$, $+$S, S$+$) We consider two sub-cases.

($-$S) Before the swap, x deletes a handle and y splits the component. There are two possible configurations after the swap. Either y breaks a handle and x splits the component into two, so the nodes x and y swap, or x involves only one of the two components split by y, so node x goes up one of the branches at node y. [In the first configuration, we pair $-$S with itself to get `Case 3a`, and in the second we pair $-$S with S$-$ to get `Case 3b`.]

($+$S) Before the swap, x adds a handle and y splits the component. After the swap, x moves up one of the branches at y. [We pair $+$S with S$+$ to get `Case 3c`].

`Case 4` (MM) Three components merge into one, and the only change between before and after is the order of merging. [We pair MM with itself.]

`Case 5` (MS, SM) Before the swap, x merges two components and y splits the merged component. After the swap, y splits one of the components which merge at x before the swap, and x merges one of the split components with the remaining component. [We pair MS with SM.]

`Case 6` (SS) A component splits into three, and the only change between before and after is the order of splitting. [We pair SS with itself.]

The pairing of cases indicates a symmetry between before and after configurations. There is also the symmetry we observe when we exchange inside with outside. Equivalently, we substitute $-f$ for f, which turns the Reeb graph upside-down, exchanging minima with maxima and 1-saddles with 2-saddles.

4. ABSTRACT ALGORITHM

In this section, we introduce an algorithm for maintaining a Reeb graph through time. The algorithm is explained at the abstract level without going into implementation details. Section 5 will describe an adaptation of the algorithm to the piecewise linear case.

Data types. We represent **time** by a conventional priority queue storing *birth-death* and *interchange events* prioritized by the moments in time they occur. At a given moment, t, the time data type supports the following operations:

INSERT(e) : add the future event e (it occurs after time t);

NEXTEVENT : return the earliest, top priority event and delete it from the queue;

DELETE(e) : delete the event e from the queue.

We maintain the **Reeb graph** as a collection of nodes, and arcs that connect the nodes. Each node knows about its incident arcs and about the segment of the Jacobi curve that contains the corresponding critical point. Each arc knows its start-node and end-node and the time when they will die at a death point or swap at an interchange. At a given moment in time, t, the Reeb graph data type supports the following operations:

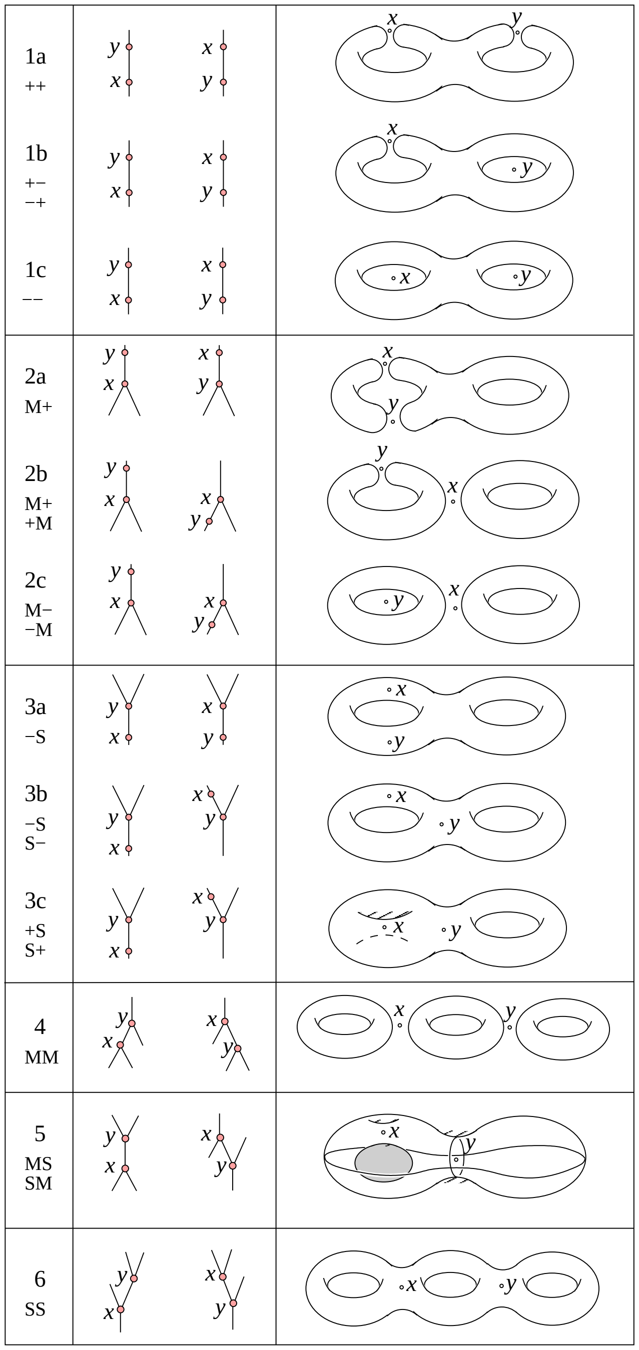

Figure 6: On the left, Reeb graph portions before and after the interchange x and y. On the right, level sets at a value just below the function value of x and y. In each case, the index of a critical point can be inferred from whether the level set merges (index 1) or splits (index 2) locally at the critical point.

SEGMENT(x) : return the segment of the Jacobi curve that contains the critical point that corresponds to node x;

NODES(a) : return the start-node and the end-node of arc a;

REMOVEARC(a) : remove the arc a from the Reeb graph;

ADDARC(x, y) : add an arc connecting the nodes x and y in the Reeb graph.

We have similar operations for removing and adding nodes, which are invoked whenever we remove or add arcs. The **Jacobi curve** is stored as a collection of segments joined at shared birth-death points. Each segment knows its endpoints, the index of its critical point, and the corresponding node in the Reeb graph, if any. Each birth-death point knows its incident segments. At a given moment in time, t, the data type for the Jacobi curve supports the following operations:

NODE(γ) : return the node in the Reeb graph that corresponds to the critical point on the segment γ;

NEXTXING(γ, γ') : return the next interchange (after time t) of the critical points tracing the segments γ and γ'.

Finally, this data type supports the operation BDPOINTS that returns all birth-death points of the function f. We also have a data type for the **input function**, $f : \mathbb{S}^3 \times \mathbb{R} \to \mathbb{R}$. One of the operations it supports connects the level sets with the Reeb graph:

PATH(u) : return a path in the Reeb graph that contains the point representing the level set component of f_t passing through the vertex u in the triangulation.

This operation will be instrumental in distinguishing between the various configurations at an interchange event.

Sweeping time. We use the operations provided by the various data types to maintain the Reeb graph of f_t through time. We assume that data is available in a finite range, from time 0 to 1. Starting with the Reeb graph R_0 at time $t = 0$, we maintain R_t by sweeping forward in time, using the Jacobi curve as a path for its nodes. The time data type is initialized by inserting all birth-death points provided by BDPOINTS. Interchange events are inserted and deleted as arcs appear and disappear in the Reeb graph. Events are processed in the order they are returned by repeated execution of the NEXTEVENT operation.

`Case` birth event. The type is 0-1, 1-2, or 2-3 and can be determined from the indices of the two segments that meet at the birth point u on the Jacobi curve. Next, we determine the arc a on PATH(u) to modify. Finally, for cases 0-1, and 2-3 we refine a and sprout a bud, and for case 1-2 we refine a by decomposing it into three arcs.

`Case` death event. We retrieve the nodes in the Reeb graph that correspond to the two segments γ and γ' that share the death point on the Jacobi curve: $x =$ NODE(γ) and $y =$ NODE(γ'). Then we contract the arc connecting x and y to a point and finally delete this point by removing three arcs and adding one.

`Case` interchange event. We swap the two nodes x and y that correspond to the critical points of the interchange by removing and adding arcs as indicated in Figure 6. Determining which arcs below x or above y to remove is equivalent to deciding between sub-cases of the interchange event. Such a decision is needed in `Cases 2` to `6` and made with the help of the PATH operation.

As mentioned earlier, each arc removal implies the deletion of an interchange event, and each arc addition implies the insertion of one into the time data type. To explain how we distinguish between the various sub-cases of an interchange event, we consider `Case 2` illustrated in Figure 6. In K_t, x and y have the same function value and the lower link of y has two components. Letting u and v be a vertex in each, we compute the arcs a and b incident to and below x in PATH(u) and PATH(v). We have `Case 2a` iff $a \neq b$. We distinguish between `Cases 2b` and `2c` using the index of y and identify $a = b$ as the arc below x to be refined by y in the new Reeb graph. `Case 4` is similar, except that the only decision to be made is which arc below x gets refined by y. `Cases 3` and `6` are upside-down symmetric to `Cases 2` and `4` and decided by calling PATH for vertices in the upper link components of x. Finally, `Case 5` is a bit different and decided by calling PATH for x and for y.

5. PL IMPLEMENTATION

In this section, we describe how to implement the algorithm of Section 4 for a piecewise linear function defined on a triangulation of the 3-sphere cross time. We finesse technical difficulties by representing time as a spiraling line forming a universal cover of an oriented circle.

Data structures. We can now describe specific data structures implementing the four abstract data types: time, Reeb graph, Jacobi curve, and input function. A standard implementation of the priority queue will do for time, and a standard graph representation will do for the Reeb graph [1]. The Jacobi curve is represented by cyclic lists of edges in the input triangulation, K. Each cycle is decomposed into segments of maximal linear lists of edges that are monotonous in time. The input function is represented by a standard data structure for four-dimensional complexes, see e.g. Brisson [6], enhanced with function values stored at the vertices.

Let K_t denote the three-dimensional slice at time t of the four-dimensional triangulation K. The vertices of K_t are points on edges of K, the edges are slices of triangles, etc. The most important operation supported by the input function data type is PATH(u): given a vertex u of K_t, return a path in the Reeb graph R_t that contains the point representing the level set component that passes through u. To compute this path, we walk in the 1-skeleton of K_t, in the direction of increasing f_t, until we reach a critical vertex x. Similarly, we walk in the direction of decreasing f_t until we reach another critical vertex y. Observe that x and y are also nodes in the Reeb graph, R_t, and delimit the desired path. As described in Section 4, this operation is instrumental in finding the connection of the new arc in a birth event and for determining the after configuration in most interchange events.

Initialization, sweep, and construction. We begin by constructing the Jacobi curve as a collection of edges in K using the algorithm in [12]. This provides the collection of birth-death points, which we use to initialize the priority queue representation of time. We also construct the Reeb graph at time zero from scratch, using the algorithm in [7], which is similar to the forward-backward sweep algorithm for computing Betti numbers in [10]. The latter algorithm also detects when 1-cycles are created and destroyed, which is the information we need to add the degree-2 nodes to the Reeb graph, which is not part of the former algorithm. The last step in preparation for the sweep through time inserts the interchange events that correspond to arcs in the Reeb graph into the priority queue. Specifically, for each arc a in R_0, we get $(x, y) =$ NODES(a),

$\gamma = \text{SEGMENT}(x)$, $\gamma' = \text{SEGMENT}(y)$ and we insert the interchange returned by $\text{NEXTXING}(\gamma, \gamma')$ for time $t = 0$ into the priority queue.

The sweep is now easy, repeatedly retrieving the next event, updating the Reeb graph, and deleting and inserting interchange events as arcs are removed and added. We think of the sequence as the evolution of a single Reeb graph. Following Driscoll et al.[11], we accumulate the changes to form a single data structure representing the entire evolution, which we refer to as the *partially persistent Reeb graph*. We adhere to the general recipe to construct it, using a constant number of data fields and pointers per node and arc to store time information and keep track of the changes caused by an update. In addition, we construct an array of access pointers that can be used to retrieve the Reeb graph at any moment in time proportional to its size.

Analysis. The running time of the algorithm can be expressed in terms of a small number of parameters, which we now introduce:

$N =$ number of simplices in K, the triangulation of the space-time data;

$n =$ upper bound on the number of simplices in a slice K_t of K;

$E =$ number of birth-death and interchange events;

$k =$ number of edges of the Jacobi curve.

We have $n \leq N$, $k \leq N$, and $E \leq k^2$, assuming the triangulation is fine enough to resolve the Jacobi curve as a disjoint collection of simple cycles. For reasonable input data, the left side will be significantly smaller than the right side in all three inequalities. To construct the Jacobi curve, we compute the reduced Betti numbers of the lower link of each edge in time $O(\ell)$, where ℓ is the size of the link. The total size of all links is some constant times N, which implies a running time of $O(N)$. The birth-death points are inserted into the priority queue in constant time each. The initial Reeb graph is constructed in time $O(n\alpha(n))$, inserting the initial batch of interchange events in time $O(n)$. The sweep iterates through E events, each in time $O(n)$ needed to determine the after configuration of the event. In addition, we use time $O(k)$ to move the nodes of the Reeb graph along the chains of edges representing the segments of the Jacobi curve. In total, the running time is $O(N + En)$. We construct the partially persistent data structure representing the evolution of the Reeb graph in the same time. The size of that data structure is proportional to the size of the initial Reeb graph plus the number of events, which is $O(n + E)$.

An obvious place to improve the running time is to improve the time needed to do a PATH operation. Is there a data structure that can return (the endpoints of a) path in time $O(\log n)$? If so then the total running time would improve to $O(N + E \log n)$, which is perhaps optimal.

6. CONCLUSION

The main contribution of this paper is the classification of the combinatorial changes in the evolution of the Reeb graph of a generic time-varying Morse function on $\mathbb{S}^3$. We establish a connection between the time-series of Reeb graphs and the Jacobi curve defined by the time-varying function. Using this connection, we describe an algorithm that maintains the Reeb graph for piecewise linear data. Letting n be the upper bound on the number of simplices in the triangulation of $\mathbb{S}^3$, this algorithm takes time $O(n)$ per combinatorial change in the Reeb graph. While maintaining the Reeb graph, we construct a partially persistent data structure of size proportional to the initial Reeb graph plus the number of events that represent the entire evolution. Given a moment in time, t, we can use this data structure to retrieve the Reeb graph R_t in time logarithmic in the number of events plus linear in its size.

Both our case analysis of events and our algorithm are limited to $\mathbb{S}^3$ and to a function on $\mathbb{S}^3$ that varies with time. It would be interesting to extend the analysis and the algorithm to a time-varying function on a general 3-manifold, for which the Reeb graph may have loops. Beyond this extension, it would be interesting to generalize the analysis and the algorithm to a function f restricted to the level sets of another function g defined on the same 4-manifold. Additional problems can be formulated by increasing the number of dimensions and the number of functions.

Acknowledgments

The third author would like to thank Jack Snoeyink for providing the start in working on this problem and for stimulating discussions.

References

[1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Massachusetts, 1974.

[2] P. S. ALEXANDROV. *Combinatorial Topology.* Dover, Mineola, New York, 1998.

[3] C. L. BAJAJ, V. PASCUCCI AND D. R. SCHIKORE. The contour spectrum. In “Proc. IEEE Conf. Visualization, 1997”, 167–175.

[4] T. F. BANCHOFF. Critical points for embedded polyhedral surfaces. *Amer. Math. Monthly* **77** (1970), 457–485.

[5] K. G. BEMIS, D. SILVER, P. A. RONA AND C. FENG. Case study: a methodology for plume visualization with application to real-time acquisition and navigation. In “Proc. IEEE Conf. Visualization, 2000”, 481–494.

[6] E. BRISSON. Representing geometric structures in d dimensions: topology and order. *Discrete Comput. Geom.* **9** (1993), 387–426.

[7] H. CARR, J. SNOEYINK AND U. AXEN. Computing contour trees in all dimensions. In “Proc. 11th Ann. SIAM-ACM Sympos. Discrete Alg., 2000”, 918–926.

[8] T. CHIUEH AND K-L. MA. A parallel pipelined renderer for time-varying volume data. In “Proc. Parallel Arch., Alg., Networks, 1997”, 9–15.

[9] K. COLE-MCLAUGHLIN, H. EDELSBRUNNER, J. HARER, V. NATARAJAN AND V. PASCUCCI. Loops in Reeb graphs of 2-manifolds. In “Proc. 14th Ann. Sympos. Comput. Geom., 2003”, 344–350.

[10] C. J. A. DELFINADO AND H. EDELSBRUNNER. An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere. *Comput. Aided Geom. Design* **12** (1995), 771–784.

[11] J. R. DRISCOLL, N. SARNAK, D. D. SLEATOR AND R. E. TARJAN. Making data structures persistent. *J. Comput. Sys. Sci.* **38** (1989), 86–124.

[12] H. EDELSBRUNNER AND J. HARER. Jacobi sets of multiple Morse functions. In *Foundations of Computational Mathematics*, ed. F. Cucker, Cambridge Univ. Press, England, to appear.

[13] H. EDELSBRUNNER, J. HARER, V. NATARAJAN AND V. PASCUCCI. Morse-Smale complexes for piecewise linear 3-manifolds. In “Proc. 19th Ann. Sympos. Comput. Geom., 2003”, 361–370.

[14] A. T. FOMENKO AND T. L. KUNII, (EDS). *Topological Methods for Visualization.* Springer-Verlag, Tokyo, Japan, 1997.

[15] M. VAN KREVELD, R. VON OOSTRUM, C. L. BAJAJ, V. PASCUCCI AND D. R. SCHIKORE. Contour trees and small seed sets for iso-surface traversal. In “Proc. 13th Ann. Sympos. Comput. Geom., 1997”, 212–220.

[16] Y. MATSUMOTO. *An Introduction to Morse Theory.* Translated from Japanese by K. Hudson and M. Saito, Amer. Math. Soc., 2002.

[17] N. MAX, R. CRAWFIS AND D. WILLIAMS. Visualization for climate modeling. *IEEE Comput. Graphics Appl.* **13** (1993), 34–40.

[18] J. MILNOR. *Morse Theory*. Princeton Univ. Press, New Jersey, 1963.

[19] J. R. MUNKRES. *Elements of Algebraic Topology.* Addison-Wesley, Redwood City, California, 1984.

[20] V. PASCUCCI AND K. COLE-MCLAUGHLIN. Efficient computation of the topology of level sets. *Algorithmica*, to appear.

[21] G. REEB. Sur les points singuliers d'une forme de Pfaff complè-ment intégrable ou d'une fonction numérique. *Comptes Rendus de L'Académie ses Séances, Paris* **222** (1946), 847–849.

[22] Y. SHINAGAWA AND T. L. KUNII. Constructing a Reeb graph automatically from cross sections. *IEEE Comput. Graphics Appl.* **11** (1991), 44–51.

Appendix A

Table 2 provides a list of notation used in this paper.

$\mathbb{M}, \mathbb{R}^3, \mathbb{S}^3$	manifold, Euclidean space, sphere
$f, g : \mathbb{M} \to \mathbb{R}$	Morse functions
γ, γ'	segments of Jacobi curve
K	triangulation
u, v, σ, τ	vertices, simplices
$\mathrm{Lk}\,\sigma, \mathrm{Lk}_{-}\sigma$	link, lower link
$\tilde{\beta}_k$	reduced Betti number
s, t	level, time parameters
x, y	critical points
x, a	node, arc in Reeb graph
N, n, k	#edges of K, section, Jacobi curve
E	#events

Table 2: Notation for geometric concepts, sets, functions, vectors, variables.