

# Isocontour based Visualization of Time-varying Scalar Fields

Ajith Mascarenhas\*, and Jack Snoeyink\*\*

Department of Computer Science,  
University of North Carolina at Chapel Hill,  
Chapel Hill, North Carolina.

**Summary.** Time-varying scalar fields are produced by measurements or simulation of physical processes over time, and must be interpreted with the assistance of computational tools. A useful tool in interpreting the data is graphical visualization, often through *level sets*, or *isocontours* of a continuous function derived from the data. In this paper we survey isocontour based visualization techniques for time-varying scalar fields. We focus on techniques that aid selection of meaningful isocontours, and algorithms to extract chosen isocontours.

## 1.1 Introduction

Physical processes that are measured over time, or that are modeled and simulated on a computer, can produce large amounts of time-varying data that must be interpreted with the assistance of computational tools. Such data arises in a wide variety of studies including computational fluid dynamics [15], oceanography [6], medical imaging [60], and climate modeling [46]. The data typically consists of finitely many points in space-time and measured or computed values for each point. Often the values are scalar, with perhaps several scalar values for each sample point. E.g: Pressure, temperature, density. A study of motion or velocity of some kind will result in vector-valued data. In this paper, we focus on scalar-valued data, often called *scalar fields*. Irrespective of how the points are sampled, 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; multilinear interpolation is also used for regular grids.

The goal is to understand the data, usually by exploring it for important features. A medical researcher might be interested in tumors, while a climatologist might be interested in regions of high pressure. Because humans possess a highly developed visual system, transforming the data into images and movies that can be displayed, and providing the scientist with tools to control them can be a powerful *visualization* method [47]. Popular techniques employed to create such visualizations are *direct volume rendering*, *slicing*, and *isocontouring*. Direct volume rendering employs two classes of algorithms to display all the data: image-space projection and volume-space projection. In image-space projection algorithms we cast

---

\* ajith@cs.unc.edu

\*\* snoeyink@cs.unc.edu

rays into the 3-dimensional volumetric data, map the data at each volume element to a user determined color and opacity value, accumulate these values in front-to-back order and display them on screen [38, 41, 39]. In volume-space projection algorithms, we traverse the volume, compute the color and opacity contribution for each volume element and project it onto the image screen [32, 63]. Direct volume rendering displays all the data simultaneously and requires recomputing the image for each new viewing direction. On the other hand, techniques for computing slices and isocontours (also called level sets) of the data have been used to reduce 3-dimensional volumetric data to a 2-dimensional form suitable for interactive display. In slicing, we restrict the data to a suitable plane, and display this restriction on a computer screen. By mapping values to colors, we can view the variation on the plane, and by varying the plane of slicing we can explore the variation of data in space. In isocontour based visualization, we fix a scalar value  $s$ , compute the points in space with that value and display the results [43]. By varying  $s$  we can explore the variation in the data.

Because time-varying data is 4-dimensional, we restrict it along two dimensions; one choice is to create time-slices to study temporal behavior, and isocontouring within the time-slice to study variation over space. Although there can be other choices for dimension reduction, most of the research on time-varying visualization use this choice. This is perhaps because visualization of volumetric data preceded that of time-varying data, and a natural step to apply existing algorithms is to consider each time-step as a static volume.

In this paper we survey theory and algorithms for isocontour-based visualization for time-varying scalar fields. We are guided by two goals. The first, given slicing and isovalue parameters extract isocontours for display. The second, and in our opinion more interesting goal, is find “interesting” parameters to aid the scientist in the visualization.

Because visualization is an interactive process, algorithms for isocontour extraction focus on efficiency. Moreover time-varying scalar fields are usually very large and may not fit into physical memory. Therefore reducing storage overhead and improving I/O efficiency are also important.

To help us address the second goal we pose some questions: How do the different components of an isocontour interact as the time and isovalue are continuously varied? At what time and isovalues do new components appear, disappear, merge, split or change genus? What tools exist that can provide this topological information without actually computing each possible isocontour? As we will see in this survey, some of the answers to these questions have resulted in algorithms to compute topological structures such as the Reeb graph that are useful aids in visualization. The Reeb graph encodes isocontour topological features such as number of components, component merge, split and genus change. We will survey algorithms for the Reeb graph of static functions and its extension to time-varying functions. These algorithms are based on the mathematical fields of topology and Morse theory, and work on the real-world piecewise linear approximations of the smooth spaces required by the theory. We see this trend as an important development; there is a rich and well developed body of work in these mathematical fields that visualization research can benefit from.

### *Outline.*

This survey is structured into two parts. The first part, consisting of Section 1.2, reviews algorithms for extracting isocontours from time-varying data. The second part, consisting of Sections 1.3 and 1.3.3, considers topological structures that aid the visualization process by presenting high dimensional data in a succinct visual form. Our focus is mainly on the Reeb graph, its extension to time-varying data, and algorithms to compute this extension.

## 1.2 Isocontour Extraction

In this section we present techniques for isocontour extraction from time-varying scalar fields. Because most research on isocontour extraction from time-varying scalar fields builds on the techniques for static fields, we briefly survey them first.

The volumetric data consists of finite point samples of a scalar function  $f: \mathbb{R}^3 \rightarrow \mathbb{R}$ . The sample points are connected into a mesh. We refer to the sample points as *vertices* and each mesh element as a *cell*. The function value is extended to the entire domain by interpolation. With linear or multilinear interpolation, the *min-max interval* for each cell  $c$ , is the interval  $[\text{minval}(c), \text{maxval}(c)]$ , where  $\text{minval}(c)$ , and  $\text{maxval}(c)$  are the minimum and maximum scalar values at the vertices of  $c$ .

Research in isocontour extraction arose in the medical imaging field, with the use of sampled medical images to extract boundaries of organs [33, 2, 28]. In the graphics and animation community, isocontours were used to model and render implicit surfaces [8], and for modeling and animating objects [65, 66].

Lorensen & Cline [43] introduce the *marching cubes* algorithm for isocontour extraction from data sampled regularly and connected into a cubical mesh. The algorithm iterates through all cubes in the volume, and extracts a piece of the isocontour from each cube. The marching cubes algorithm has a flaw [22]; it sometimes generates isocontours with holes. Subsequent research provide algorithms that correct this problem [30, 45, 49, 51, 52]. Although simple, the marching cubes algorithm can be inefficient because it examines the entire volume, while an isocontour may intersect only a small fraction of this volume. Techniques that speed-up isocontour extraction build search structures to efficiently find cells intersecting the isocontour, and can be classified into spatial techniques, span space techniques, and topological techniques.

### 1.2.1 Spatial Techniques

Spatial techniques subdivide the volume using an octree based hierarchy [64]. Each octant is equipped with the min-max interval of function values contained in it, enabling a search through the octree hierarchy to terminate at the octant if the query isovalue does not belong to the interval. This technique works because most data sets have large regions with function values that are distributed close together and can be quickly pruned in a search. Next we look at an extension of the octree technique to time-varying data.

#### Temporal branch-on-need tree

The Temporal Branch-on-Need Tree (T-BON) [59] extends the three dimensional branch-on-need octree for time-varying isocontour extraction. It aims at efficient I/O while retaining the accelerated search enabled by a hierarchical subdivision. Unlike the algorithms in Sections 1.2.2 and 1.2.3 this algorithm does not exploit temporal coherence and considers each time-step as a static volume.

##### *Construction.*

The input to the T-BON construction is regularly sampled points in space-time. The output is a branch-on-need octree for each time step, stored on disk in two sections. Store information common to all trees, such as branching factor, pointers to children and siblings, once for all trees. Store the min-max intervals associated with nodes of an octree separately as a linear array, one per time-step, each packed in depth-first order.

*Search and extraction.*

The input is a query: Extract the isocontour for isovalue  $s$  at time  $t$ . The output is the isocontour represented as a triangulated mesh. As a first step, read the octree infrastructure from disk and re-create it in main memory. Resolve the query using the octree for  $t$  and demand-driven paging [19]. Read the min-max interval of the root from disk, and if  $s$  belongs to this interval read the child nodes. Proceed recursively, stopping at the leaf nodes. If the min-max interval of a leaf contains  $s$  add the disk block containing the corresponding cells values onto a list. After completing the octree traversal read all disk blocks from the list into memory. Repeat the tree traversal, this time extracting the isocontour using the cell data read from disk.

**1.2.2 Span-space Techniques**

Livnat et al. [42] define the *span space* as the two dimensional space spanned by the minimum and maximum values of the cells of the volume. A cell  $c$  with minimum value  $min_c$  and maximum value  $max_c$  maps to a point  $(min_c, max_c)$  in the span space. See Figure 1.1. A variety of search structures on the span space have been used to speed-up finding cells that intersect an isocontour. Gallagher [29] uses bucketing and linked lists, Livnat et al. [42] use k-d trees [7], van Kreveld [61] and Cignoni et al. [16] use the interval tree for two- and three-dimensional data respectively. Chiang et al. [14] use a variant of the interval tree that enables out-of-core isocontour extraction, and use the algorithm of Section 1.2.2 to extend this work to time-varying data [13]. Unlike the spatial techniques that use the octree, which requires regularly gridded data, span space techniques have the advantage of being also applicable to irregularly sampled data. Next we look at an algorithm that uses span space techniques for isocontour extraction from time-varying data.

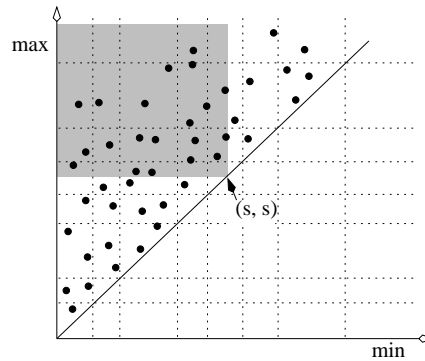


Fig. 1.1: Points in shaded area of span space correspond to cells that intersect isocontour for value  $s$ .

**Temporal hierarchical index (THI) tree**

Shen's algorithm [55] for the THI tree analyzes the span space of the time-varying data, and classifies and stores each cell in one or more nodes of a binary tree based on the temporal

variation of its values. By placing a cell possessing a pre-defined small temporal variation in a single node of the THI tree, along with a conservative min-max interval of its variation over a large time span, this algorithm achieves savings in space. Cells with greater temporal variation are stored at multiple nodes of the tree multiple times, each for a short time span.

*Temporal variation.*

Shen uses the span space to define the temporal variation of a cell’s values. The area over which the points corresponding to a cell’s min-max values over time are spread out give a good measure of its temporal variation; the larger the area of spread the greater the variation. In particular, subdivide the span space into  $\ell \times \ell$  non-uniformly spaced rectangles called *lattice elements* using the *lattice subdivision* scheme [56]. To perform the subdivision, lexicographically sort all extreme values of the cells in ascending order, find  $\ell + 1$  values to partition the sorted list into  $\ell$  sublists, each with the same number of cells. Use these  $\ell + 1$  values to draw vertical and horizontal lines to get the required subdivision. Note that this subdivision does not guarantee that each lattice element has the same number of cells. Given a time interval  $[i, j]$ , a cell is defined to have low temporal variation in that interval if its  $j - i + 1$  min-max interval points are located over an area of  $2 \times 2$  lattice elements.

*Construction.*

The input of the THI algorithm is a fixed mesh whose vertices are points in space. Each point has a data value for each time step in  $[0, T - 1]$ . Each cell has  $T$  corresponding min-max intervals. The output of the THI algorithm is a binary tree constructed as follows. In the root node  $N_0^{T-1}$ , store cells whose min-max intervals have low temporal variation in the time interval  $[0, T - 1]$ . The root has two children,  $N_0^{T/2}$  and  $N_{T/2+1}^{T-1}$  defined recursively on cells that are not stored in the root. Recursion stops at leaf nodes  $N_t^t$ , with  $t \in [0, T - 1]$ . Cells that fall into leaf nodes have the highest temporal variation. See Figure 1.2.

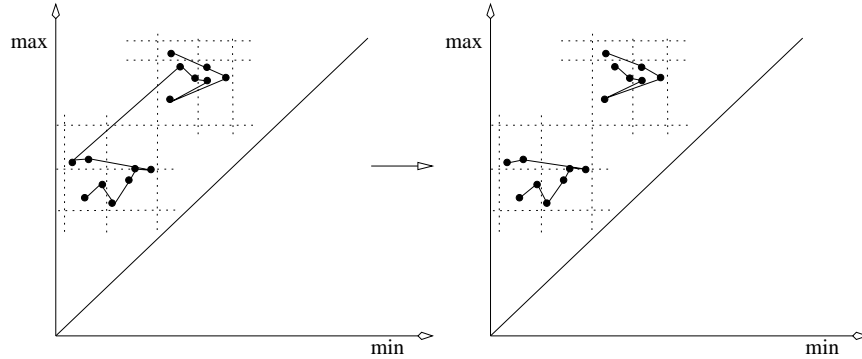


Fig. 1.2: The min-max intervals of a cell over a time interval are shown in the span-space as points with a path connecting them in order of time. The points on the left are spread outside a  $2 \times 2$  lattice area. On breaking the time interval into two halves, on the right, the respective points fall inside a  $2 \times 2$  area.

Represent each cell that falls into an internal node  $N_i^j$  by a conservative min-max interval, called the *temporal extreme values*, which contains all the cells min-max intervals for the time span  $j - i + 1$ . Because the temporal extreme values are used to refer to a cell for more than one time step, we get a reduction in the overall index size.

Within each tree node, organize the cells using one of the span-space based techniques; Shen uses a modified ISSUE algorithm [56]. Use the lattice subdivision scheme described above, and sort cells that belong to each lattice row, excluding the cells in the diagonal lattice element, in ascending order, based on their minimum temporal extreme value. Similarly, sort the cells into another list, in descending order, based on their maximum temporal extreme value. Build an interval tree, as in [16], for cells in the lattice elements along the diagonal.

#### *Search and extraction.*

As in the T-BON algorithm, the input is a query: Extract the isocontour for isovalue  $s$  at time  $t$ , and the output is the isocontour represented as a triangulated mesh. First collect all nodes in the THI-tree whose time span contains  $t$ . Traverse the tree, starting at the root node. From the current node, visit the child  $N_i^j$ , with  $i \leq t \leq j$ , stopping at leaf node  $N_t^t$ .

At each node in the traversal path, use the lattice index structure to locate candidate isocontour cells. Locate the lattice element that contains the point  $(s, s)$ . Because of the symmetry of the lattice subdivision this element is the  $k^{th}$  row in the  $k^{th}$  column, for some integer  $k$ . The isocontour cells are contained in the upper-left corner bounded by the lines  $x = s$  and  $y = s$ , as shown in Figure 1.1. Collect these cells as follows:

- From each row  $r = k + 1$  to  $\ell - 1$ , collect cells from the beginning of the list sorted on the minimum temporal extreme value until the cell whose minimum is greater than  $s$ .
- From row  $k$ , collect cells from the beginning of the list sorted on the maximum temporal extreme value until a cell whose maximum is lesser than  $s$ .
- From the lattice element containing  $(s, s)$ , collect cells by querying the interval tree.

Recall that because we store a conservative min-max interval with cells, some collected cells may not actually intersect the isocontour. For all candidate cells, read the actual data at time  $t$  to extract the isocontour.

### 1.2.3 Topological Techniques

Topological techniques for efficient isocontour extraction typically analyze the data in a pre-process step, and compute a subset of cells called the *seed set*. A seed set contains at least one cell, called a *seed*, intersecting each component of each isocontour. To extract an isocontour first search the seed set, which is stored in an appropriate search structure, to find a seed for each connected isocontour component. To extract each component, start at the seed and visit all intersecting cells by a breadth first search of the mesh. This method of extraction, by performing a breadth first search of the mesh, is called *continuation* by Wyvill et al. [65], *mesh propagation* by Howie & Blake [34], and *contour-following* by [10]. Next we look at an algorithm that uses topological techniques for isocontour extraction from time-varying scalar fields.

#### **Progressive tracking**

Bajaj, Shamir & Sohn [4] extend seed set based techniques to time-varying data. They use temporal coherence to compute an isocontour at time step  $t + 1$  by modifying the isocontour

computed at the time step  $t$ . New components at  $t + 1$  are separately computed from the seed set for that time step. Unlike the T-BON and THI algorithms they accept a range of time-steps and a single isovalue as arguments, and track components over the range of time-steps.

*Construction.*

The input can be a fixed regular or irregular mesh whose vertices are points in space. Each point has a data value for each time step in  $[0, T - 1]$ . The output is a collection of  $T$  seed sets, one for each time step. Treat each time-step as a static scalar field and compute a seed-set using one of the algorithms proposed in [10, 62]. Organize each seed-set in an interval tree [16].

*Search and extraction.*

The query for this algorithm is different from the T-BON and THI algorithms. Find isocontours for isovalue  $s$  over the time range  $[t_0, t_1]$ . To extract an isocontour, first extract all isocontour components at time  $t_0$  using contour propagation from the seeds for that time. Extract all isocontour components for subsequent discrete time steps,  $t \leq t_1$  by a combination of modifying current isocontour components over time, and by extracting new components from the seed sets at  $t$ .

To modify components, at each time-step  $t$ , with  $t_0 \leq t \leq t_1$ , maintain, in one list per isocontour component, the set of intersecting cells. These cell lists are used to track the evolution of the isocontour for  $s$  over time. As in the marching cubes case, we can label each cell vertex as “above” if its value at time  $t$  is greater than  $s$ , and “below” otherwise. A cell edge joining opposite labels contains an isocontour vertex. To track an isocontour component, consider the label change for each cell edge at  $t + 1$ . If the labels of the end vertices of a cell edge do not change then the isocontour vertex that lies on that edge just changes position, which can be found by linear interpolation. A cell vertex label can change, in which case the isocontour experiences a local connectivity change. The changes include the contour changing its geometry but not topology, two or more components merging, or a component splitting into two or more components. All these changes can be applied to the contours by enqueueing the cell lists, extracting each cell and examining its neighborhood, and propagating the isocontour spatially. See [4] for details.

### 1.2.4 Comparison

The TBON algorithm discussed in Section 1.2.1 can be applied to regularly gridded data and is designed to be I/O-efficient by using demand driven paging; load data only when it is required. It does not exploit temporal coherence and treats each time-step as a static volumetric scalar field. The THI algorithm discussed in Section 1.2.2 can be applied to both regularly and irregularly sampled data. It reduces the storage overhead of the search structure, but it requires all data to be loaded into memory, a serious drawback for time-varying data which can be large. The progressive tracking algorithm discussed in Section 1.2.3 can also be applied to regularly and irregularly sampled data. Since this algorithm computes seed sets the storage overhead is not significant. Bajaj et al. [4] show seed set sizes of less than 2% of the total number of cells. Moreover, extracting isocontours by propagation produces coherent triangulations that are amenable to compression [35], simplification [37], and streaming [36]. The other extraction algorithms do not produce coherent triangulations.

### 1.3 Topological Structures for Supporting Visualization

In this section we review a topological structure called the Reeb graph that encodes the number of isocontour components at each isovalue, and the topological changes experienced by components. The Reeb graph can be used to display this information succinctly to aid visualization. Section 1.3.2 reviews an algorithm that extends the Reeb graph for visualizing time-varying scalar fields. Section 1.3.3 reviews a more systematic study of the evolution of Reeb graphs over time, and an algorithm to compute this evolution.

The algorithms discussed in this section are based on Morse theory [44, 48] and combinatorial and algebraic topology [1, 50]. Note that in deference to the terminology used in this mathematical literature, we sometimes use the term level set instead of isocontour. We begin with a review of smooth maps and critical points. Because algorithms work on piecewise linear data, we discuss how to translate the concepts from the smooth setting to the piecewise linear setting.

#### *Smooth maps on manifolds.*

Let  $\mathbb{M}$  be a smooth, compact  $d$ -manifold without boundary and  $f: \mathbb{M} \rightarrow \mathbb{R}$  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  $x$  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: the *minima* with index 0, the *1-saddles* with index 1, the *2-saddles* with index 2, and the *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 [31, 44]. 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 information about the manifold on which the function is defined. For example, the Euler characteristic of the manifold  $\mathbb{M}$  equals the alternating sum of critical points,  $\chi(\mathbb{M}) = \sum_x (-1)^{\text{index } x}$ .

#### *Piecewise linear functions.*

A *triangulation* of a manifold  $\mathbb{M}$  is a simplicial complex,  $K$ , whose underlying space is homeomorphic to  $\mathbb{M}$  [1]. Given values at the vertices, we obtain a continuous function on  $\mathbb{M}$  by linear interpolation over the simplices of the triangulation. The Euler characteristic of  $\mathbb{M}$  can also be computed from any triangulation of  $\mathbb{M}$  as the alternating sum of simplices,  $\chi(\mathbb{M}) = \sum_\sigma (-1)^{\dim \sigma}$ . We need some definitions to talk about the local structure of the triangulation and the function. The *star* of a vertex  $u$ , denoted  $\text{St } u$ , consists of all simplices that share  $u$ , including  $u$  itself, and the *link*, denoted  $\text{Lk } u$ , consists of all faces of simplices in the star that are disjoint from  $u$ . The *lower link*, denoted  $\text{Lk}_- u$ , is the subset of the link induced by vertices with function value less than  $u$ :

$$\begin{aligned} \text{St } u &= \{\sigma \in K \mid u \subseteq \sigma\}, \\ \text{Lk } u &= \{\tau \in K \mid \tau \subseteq \sigma \in \text{St } u, u \notin \tau\}, \\ \text{Lk}_- u &= \{\tau \in \text{Lk } u \mid v \in \tau \Rightarrow f(v) \leq f(u)\}. \end{aligned}$$

Banchoff [5] introduces the critical points of piecewise linear functions as the vertices whose lower links have Euler characteristic different from unity.

A classification based on the reduced Betti numbers of the lower link is finer than that defined by Banchoff. 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:  $\tilde{\beta}_k = \text{rank } \tilde{H}_k$ . 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 [50]. The first three un-reduced Betti numbers are the number of connected components, the number of tunnels, and the number of voids respectively. For example, a 2-torus has un-reduced Betti numbers:  $\beta_0 = \beta_2 = 1$ , and  $\beta_1 = 2$ . For  $d = 3$  the link is a 2-sphere and the Betti numbers can be computed as follows: Compute the Euler characteristic  $\chi$  of the lower link as the alternating sum of vertices, edges and faces in the lower link. Compute  $\beta_0$ , the number of connected components in the lower link, by using the union-find data structure [18]. If all the link vertices are also in the lower link then  $\beta_2 = 1$  else  $\beta_2 = 0$ . Compute  $\beta_1$  using the relation  $\beta_1 = \beta_0 + \beta_2 - \chi$ . The reduced Betti numbers can be computed from the definitions. For an algorithm to compute Betti numbers of simplicial complexes on the 3-sphere see [20].

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 1.1 and Figure 1.3. The first case in which this definition differs from Banchoff's is a double saddle

	$\tilde{\beta}_{-1}$	$\tilde{\beta}_0$	$\tilde{\beta}_1$	$\tilde{\beta}_2$
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 1.1: Classification of vertices into regular and simple critical points using the reduced Betti numbers of the lower link.

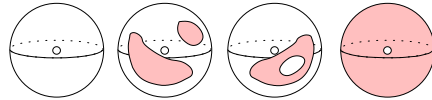


Fig. 1.3: Lower links, shown shaded, for  $d = 3$ . From left to right, a minimum, a 1-saddle, a 2-saddle, and a maximum.

obtained by combining a 1- and a 2-saddle into a single vertex. The Euler characteristic of the lower link is 1, 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 1.1 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 [11, 25]. This allows us to develop algorithms assuming that all critical points are simple.

### *The Reeb graph.*

A level set of a function  $f$  consists of all points in the domain whose function values are equal to a chosen real number  $s$ . A level set of  $f$  is not necessarily connected. If we call two points  $x, y \in \mathbb{M}$  *equivalent* when  $f(x) = f(y)$  and both points belong to the same component of the level set, then 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 [54]. Figure 1.4 illustrates the Reeb graph for a function defined on a 2-manifold of genus two. We call a point on the Reeb graph a *node* if the corresponding level set component passes

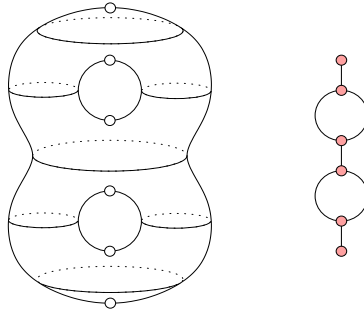


Fig. 1.4: The Reeb graph of the function  $f$  on a 2-manifold that maps every point of the double torus to its distance above a horizontal plane below the surface.

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.

### *Visualization interfaces.*

Topological structures, such as the Reeb graph, provide a succinct summary of the underlying function, and are used in visualization interfaces. Consider Figure 1.5. It shows a screenshot of the *contour spectrum* [3] interface; a window displays isocontour properties, such as surface area and volume, using graphs, and topological properties using the contour tree, to aid selection of interesting isocontours which are displayed separately.

Figure 1.6 shows the safari interface [40], which extends the ideas of the contour spectrum to time-varying data, provides the user with a *(time, value)* control plane for isovalue selection, and extracts an isocontour from a time-slice for display. The control plane on the right displays the number of isocontour components for each time-step ( $x$ -axis) and isovalue ( $y$ -axis), which can be computed from the contour tree for each time step.

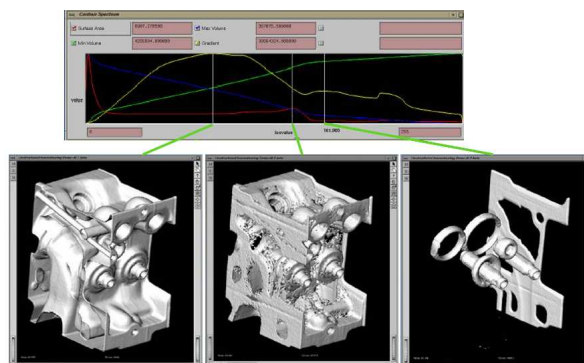


Fig. 1.5: The contour spectrum interface [3]. On the top, graphs of isocontour properties, such as surface area, and volume, versus isovalues. On the bottom, three isocontours chosen using the contour spectrum.

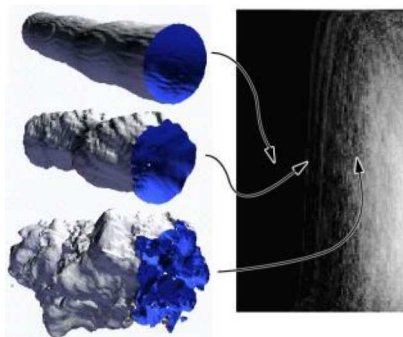


Fig. 1.6: A portion of the safari interface [40]. The control plane on the right displays the number of isocontour components for each time-step( $x$ -axis) and isovalue( $y$ -axis). The user selects isocontours for display by clicking on the contour plane.

Recently, Carr et al. [10] have used the contour tree to compute *path seeds*, a set of edges in the triangulation that can be used to quickly find seeds for any isocontour component, and use these path seeds to create a *flexible isocontour* interface. They provide the user with an interface to select individual arcs in the contour tree and can extract chosen isocontour components for display. Building on this work, they also present an algorithm to simplify the contour tree using local geometric measures to capture the essential features of the underlying data in the presence of noise [12].

### 1.3.1 Reeb Graph Algorithms

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 1.4 reveals that the function is defined on a double torus, assuming we know it is an orientable 2-manifold without boundary. In

visualization, on the other hand, the Reeb graph is used to study the behavior of the function. 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 the Reeb graphs for such functions will reveal the (un-exciting) connectivity of  $\mathbb{S}^3$  by being trees, but the structure of the tree will tell us how the level sets of the chosen function  $f$  change topology.

### *History.*

The Reeb graph was first introduced in [54]. In the field of visualization, Boyell and Ruston [9] introduced the contour tree to summarize the evolution of contours on a map. In the interactive exploration of scientific data, Reeb graphs are used to select meaningful level sets [3] and to efficiently compute them [62]. An extensive discussion of Reeb graphs and related structures in geometric modeling and visualization applications can be found in [27].

Published algorithms for Reeb graphs take as input a function defined on a triangulated manifold. We express their running times as functions of  $n$ , the number of simplices in the triangulation. The first algorithm for functions on 2-manifolds due to Shinagawa and Kunii [57] takes time  $O(n^2)$  in the worst case. Reeb graphs of simply-connected domains are loop-free, and are also known as contour trees. They have received special attention because of the practical importance of these domains, and because the algorithms to compute them are simpler. 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 [11]. For the case of 3-manifolds, this algorithm has been extended to include information about the genus of the level surfaces [53]. Cole-McLaughlin et al. [17] return 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.

### *Computing a contour tree.*

We sketch the algorithm proposed by Carr et al. [11] to compute the contour tree of a function defined on a simply-connected domain. Although their algorithm works for any dimension we restrict our description to  $d = 3$ . This algorithm is used in Sections 1.3.2 and 1.3.3.

The input to the contour tree algorithm is a triangulation  $K$  of a simply-connected 3-manifold, with each vertex equipped with a distinct scalar function value. The output is the contour tree of the function represented as a collection of nodes and arcs that connect them. The algorithm proceeds in two passes: the first computes the *Join tree*, and the second the *Split tree*. Since these passes are similar we describe the construction of the Join tree. The Join tree encodes the merges experienced by the isocontour components as we sweep the isovalue from  $-\infty$  to  $\infty$ ; the Split tree does this for the sweep in the opposite direction.

To implement the sweep sort the vertices of  $K$  in ascending order of function value, and iterate through the vertices. At each step maintain the collection of vertices visited in a union-find (UF) data structure [18]. For each connected component in the collection, maintain a tree encoding the merge history of the component. Classify each vertex  $v$  based on its index and handle each case as follows. Add a regular point to the set that contains its lower link vertex. A minimum creates a new component; start a UF set, and a new Join tree arc. An index-1 critical point locally merges two components. If the components are not connected globally, then create a join node that merges two arcs and starts a new one, else create a join node that ends a single arc, and starts a new one. The latter case corresponds to the component experiencing a genus change. An index-2 critical point is handled in the split sweep. Only the global maximum appears in the Join tree; it ends the arc corresponding to the component that disappears at the maximum.

Finally, construct the contour tree by merging the Join and Split tree [11]. In Figure 1.7 we see two stages during the construction of the Join tree for a function defined on a 2-manifold.

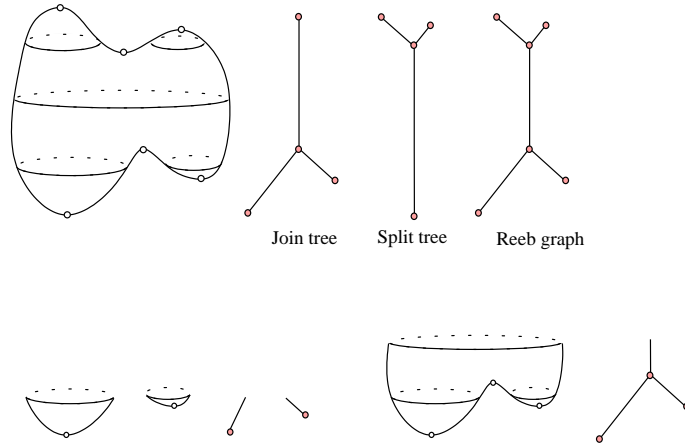


Fig. 1.7: In the top row, a 2-manifold shown with three isocontours for the height function defined on it, and the Join, Split tree and Reeb graph. In the bottom row, two stages during the sweep to construct the Join tree.

### 1.3.2 Time-varying Contour Topology

An important problem in visualizing time-varying scalar fields is computing the correspondence of isocontour components for a fixed isovalue over time. Another problem is detecting when and how these components change topology. Sohn & Bajaj [58] address these problems by computing the correspondence of contour trees over time. They assume that the scalar field can change unpredictably between two successive time steps, and define temporal correspondence of contour tree arcs for successive time steps using a notion of an overlap between an isocontour at time  $t$  with an isocontour at time  $t + 1$ . They develop an algorithm to compute correspondence between successive contour trees based on this definition, and use the correspondence to track isocontour components and their topology over time.

#### *Temporal contour correspondence.*

Before we define temporal contour correspondence we need some notation. Define  $\mathbb{X}$  and  $\mathbb{Y}$  for the restrictions of the domain to time  $t$  and  $t + 1$  respectively, function  $f_t$  for the restriction of function  $f$  to  $\mathbb{X}$ , and isocontours  $I = f_t^{-1}(s)$  and  $J = f_{t+1}^{-1}(s)$ . Isocontour  $I$  has connected components  $I = \{I_1, \dots, I_m\}$ , and lies on the intersection of  $\mathbb{X}_{\leq s} = f_t^{-1}(-\infty, s]$  and  $\mathbb{X}_{\geq s} = f_t^{-1}[s, \infty)$ . Identify each  $I_i$  with one component of  $\mathbb{X}_{\leq s}$  and  $\mathbb{X}_{\geq s}$ ;  $I_i$  belongs to their intersection. Sohn & Bajaj call these components the *lower object* and *upper object* of  $I_i$ , respectively. Similar definitions hold for the isocontour  $J$ .

Two contour components  $I_i$  and  $J_j$  exhibit *temporal correspondence* if their corresponding upper objects overlap and their corresponding lower objects overlap. Note, that an overlap

between  $\mathbb{X}_{\leq s}$  and  $\mathbb{Y}_{\leq s}$  makes sense only if we assume that both  $f_t$  and  $f_{t+1}$  are defined on the same domain. See Figure 1.8 for an example.

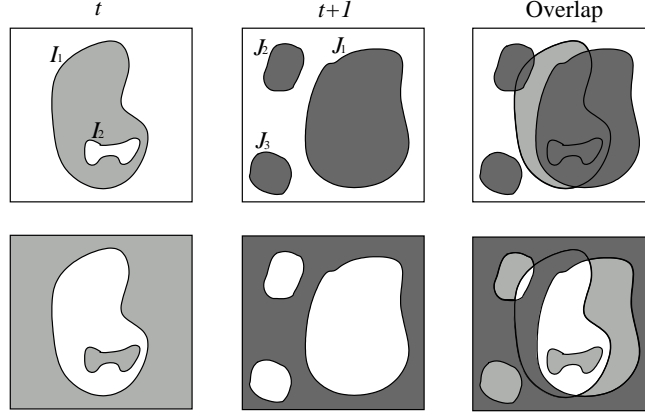


Fig. 1.8: Temporal correspondence between isocontours at successive time steps. In the top row, upper objects for each isocontour and their overlap, in the bottom row, lower objects and their overlap. From the definitions in [58], we get the following correspondence:  $(I_1 \rightarrow J_1)$ ,  $(I_1 \rightarrow J_2)$ ,  $(\emptyset \rightarrow J_3)$ ,  $(I_2 \rightarrow \emptyset)$ .

*Algorithm for contour correspondence.*

Since each isocontour component corresponds to a point on an arc on the contour tree, temporal correspondence between isocontour components can be used to compute a correspondence between arcs of the contour tree at  $t$  with arcs of the contour tree at  $t + 1$ . Sohn & Bajaj compute this correspondence by modifying the contour tree algorithm of Carr et al. [11]. In the description that follows, we use  $JT_t$ ,  $ST_t$ ,  $CT_t$  to denote the join tree, split tree, and contour tree at time  $t$  respectively. The input is a simplicial mesh in  $\mathbb{R}^3$ . Each point of the mesh is equipped with  $T$  scalar values. The output is a collection of  $T$  contour trees, with each arc of  $CT_{t+1}$  labeled with arcs of  $CT_t$ . The labels indicate the temporal correspondence information. Since the join and split trees are symmetric we use the join tree for the description. For each time step  $t$ , pre-compute the contour tree,  $CT_t$  [11]. Label each tree arc with a unique id. The correspondence information is computed as follows.

1. Augment  $JT_t$  with the nodes that appear only in  $ST_t$ . Carr et al. [11] call the resulting join tree the augmented join tree.
2. Equip each arc  $a$  of  $JT_t$  with the ids of the corresponding arcs from  $CT_t$ . An isocontour component corresponding to a point on  $a$ , lies on the boundary of a connected component of  $\mathbb{X}_{\leq s}$  (the lower object), which may contain other isocontour components. Each of these other isocontour components correspond to an arc of  $CT_t$ . Equip  $a$  with the ids of these arcs of  $CT_t$ . This step can be done by simultaneously scanning the nodes of  $CT_t$  and  $JT_t$  in increasing order of  $f_t$ , and incrementally maintaining the arc id lists for  $JT_t$ .

3. This is the step that computes the correspondence information for the arcs of  $JT_{t+1}$ . Simultaneously sweep the functions  $f_t$  and  $f_{t+1}$  in increasing order, as if constructing the join trees  $JT_t$  and  $JT_{t+1}$ . The sweep can be thought of as incrementally generating the lower objects simultaneously in time  $t$  and  $t + 1$ , and detecting overlap. Recall that two isocontour components from successive time-steps exhibit temporal correspondence if their respective lower objects and upper objects overlap. This sweep tests lower object overlap, and the reverse sweep, for  $ST_t$  and  $ST_{t+1}$ , will test overlap for upper objects. During the sweep, maintain a collection of lower objects in  $\mathbb{X}_{\leq s}$  and  $\mathbb{Y}_{\leq s}$ . Each lower object corresponds to an arc of the join tree in its respective time-step. For lower objects in  $\mathbb{X}_{\leq s}$  we know their corresponding contour tree arcs from step 2. When a lower object in  $\mathbb{X}_{\leq s}$  overlaps one in  $\mathbb{Y}_{\leq s}$  we can create the mapping between the arcs of  $JT_{t+1}$  with the corresponding arcs of  $JT_t$ . For a more detailed description of this step see [58].
4. Map labels from the arcs of  $JT_{t+1}$  to the corresponding arcs of  $CT_{t+1}$ . This step is similar to step 2.

After the above steps are repeated for  $ST_{t+1}$ , each arc of the contour tree  $CT_{t+1}$  has two arc lists, one from  $JT_{t+1}$  and the other from  $ST_{t+1}$ . The final arc list is the intersection of the two lists.

#### *Topology change graph.*

Consider isocontour  $I = \{I_1, \dots, I_m\}$  for isovalue  $s$  at time  $t$ . If we keep the isovalue fixed at  $s$  and proceed forward in time to  $t + 1$ , then the isocontour undergoes topological changes in the following possible ways: a component is created or destroyed, two components merge into one, a component splits into two, a component changes genus. The *Topology change graph (TCG)* depicts the change in topology as a graph constructed as follows. At each time step compute the arcs of  $C_t$  that contain the isovalue  $s$ . Create a node in the TCG for each these arcs. Use the correspondence information computed for an arc of  $C_t$  to create a connection between the corresponding nodes in the TCG. See Figure 1.9. Genus change can be detected by examining the Betti numbers of the component at time  $t + 1$ , which can be computed using the algorithm of Pascucci et al. [53].

### 1.3.3 Time-varying Reeb Graph

Edelsbrunner et al. [24] have performed a systematic study of the evolution of the Reeb graph of a function over time. This evolution can be encoded in a data structure that can be used to extract the Reeb graph for any instant of time for display in a visualization interface. Moreover, the data structure can be used to track isocontour components over time, and detect changes in their topology. This information can aid the user in selecting interesting time and isovalue parameters for visualization.

Edelsbrunner et al. enumerate all the combinatorial changes experienced by the Reeb graph of a Morse function on  $\mathbb{S}^3$  over time. Crucial to understanding these changes is the notion of Jacobi sets, which can be used to compute the trajectory of critical points of a function over time.

#### *Jacobi sets.*

Reeb graphs can be used to summarize a function at moments in time and Jacobi curves, as introduced in [23], to get a glimpse of their evolution through time. Edelsbrunner et al.

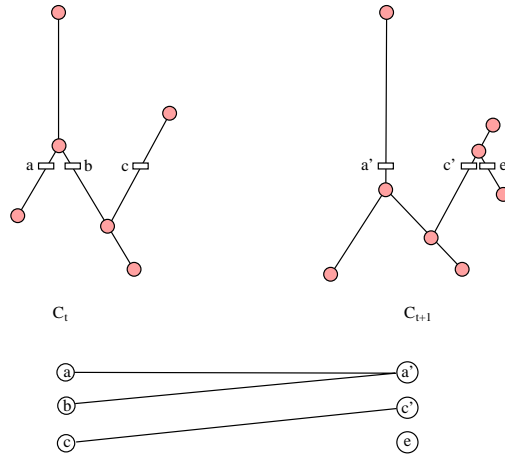


Fig. 1.9: In the top row, a contour tree at successive time steps, with arcs containing isovalue  $s$  labeled. In the bottom row, the topology change graph showing how contour components change topologically.

introduce this concept for the slightly more general case of two Morse functions,  $f, g: \mathbb{M} \rightarrow \mathbb{R}$ ; 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  $t \in \mathbb{R}$ , consider the level set  $g^{-1}(t)$  and the restriction of  $f$  to this level set  $f_t: g^{-1}(t) \rightarrow \mathbb{R}$ . 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. We use Figure 1.10 from [23] to illustrate the definition by showing the Jacobi curve of two smooth functions on a piece of the two-dimensional plane. To understand this picture, imagine  $f$  as a cone-like mountain indicated by dotted level curves, and the solid level curves of  $g$  gliding over that mountain. On the left, we see a circle beginning at a minimum of  $g$  and expanding outwards on a slope. As this circle expands a maximum of the restriction of  $f$  moves up and a minimum moves down from the starting point.

Consider a 1-parameter family of Morse functions on the 3-sphere,  $f: \mathbb{S}^3 \times \mathbb{R} \rightarrow \mathbb{R}$ , and introduce an auxiliary function  $g: \mathbb{S}^3 \times \mathbb{R} \rightarrow \mathbb{R}$  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  $f_t: \mathbb{S}^3 \times t \rightarrow \mathbb{R}$ . The Jacobi curve of  $f$  and  $g$  may consist of several components, and in the assumed generic case each is a closed 1-manifold. 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. 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:

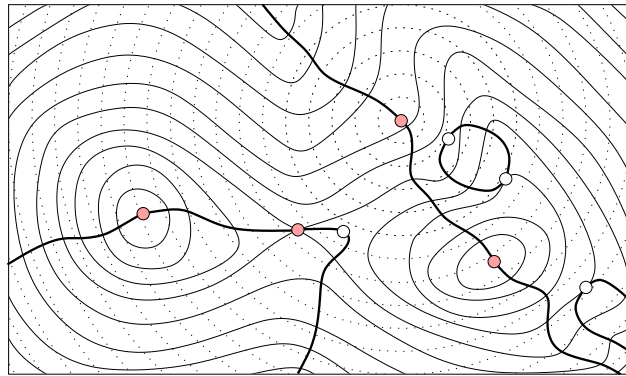


Fig. 1.10: The functions  $f$  and  $g$  are represented by their dotted and solid level curves. The Jacobi curve is drawn in bold solid lines. The birth-death points and the critical points of the two functions are marked by white and shaded dots, respectively.

INDEX LEMMA [24]. Let  $f: \mathbb{M} \times \mathbb{R} \rightarrow \mathbb{R}$  be a 1-parameter family of Morse functions. The indices of two critical points created or destroyed at a birth-death point differ by one.

*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, and provide a mechanism for identifying nodes in different Reeb graphs. Figure 1.11 illustrates this idea.

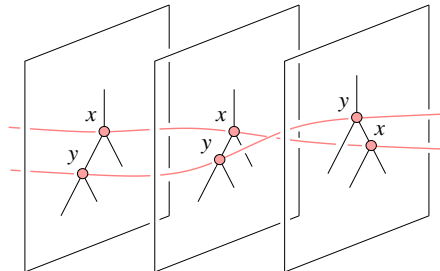


Fig. 1.11: Reeb graphs at three moments in time whose nodes are connected by two segments of the Jacobi curve [24].

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 time is the only varying parameter, one 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  $f_t$  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 briefly sketch the changes caused by birth-death points and by interchanges.

*Birth, death and interchange.*

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. See Figures 1.12 and 1.13, and [24] for details.

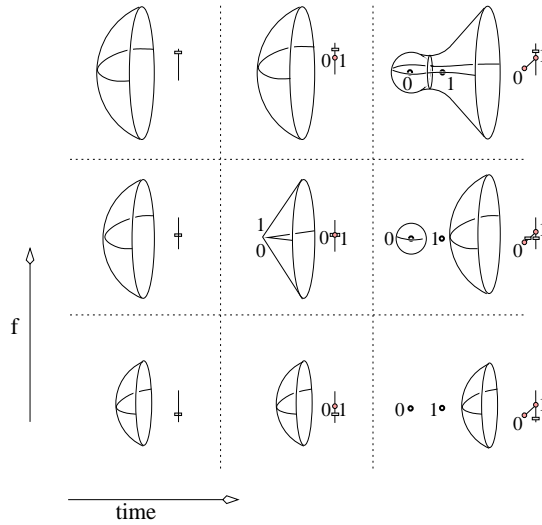


Fig. 1.12: Level sets and Reeb graphs around a 0-1 birth point. The 2-3 case is upside-down symmetric to this case. Time increases from left to right and the level set parameter, indicated by a rectangular slider bar, increases from bottom to top. Going forward in time, we see the sprouting of a bud, while going backward in time we see its retraction [24].

There are three similar cases when time 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 1.12, which we now read from right to left, and the 1-2 case is illustrated in Figure 1.13, which we also read backward, from right to left.

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  $f_{t-\epsilon}(x) < f_{t-\epsilon}(y)$  and  $f_{t+\epsilon}(x) > f_{t+\epsilon}(y)$ . There are 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

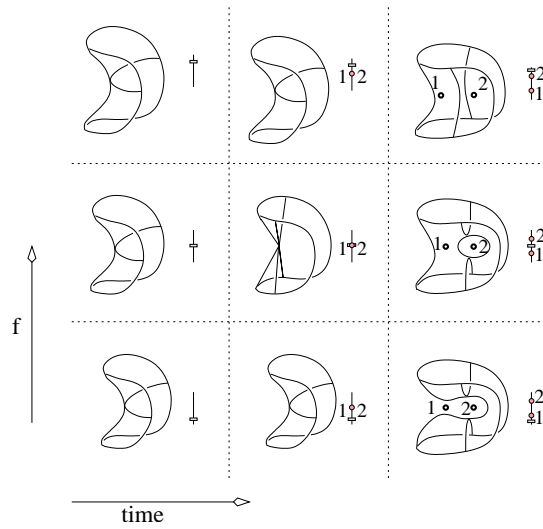


Fig. 1.13: Level sets and Reeb graphs around a 1-2 birth point. Time increases from left to right and the level set parameter increases from bottom to top. Going forward in time, we see a refinement of an arc in the Reeb graph and going backward we see a coarsening [24].

configurations. An analysis of possible before and after combinations and pairing them, gives the cases illustrated in Figure 1.14. It is convenient to group the cases with similar starting configurations together. Edelsbrunner et al. use +, -, M, 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  $y$ . For a more detailed description of these cases see [24].

*Computing time-varying Reeb graphs.*

The input to the algorithm is a piecewise linear function defined on a triangulation  $K$  of 3-sphere cross time. The output is a collection of Reeb graphs, stored in a partially persistent data-structure [21], that captures the evolution of the Reeb graph  $R_t$  over time.

Begin by constructing the Jacobi curve as a collection of edges in  $K$  using the algorithm in [23]. Also construct the Reeb graph at time zero,  $R_0$ , from scratch, using the algorithm in [11]. Maintain  $R_t$  by sweeping forward in time, using the Jacobi curve as a path for its nodes. Implement the sweep by maintaining a priority queue of events sorted on time, repeatedly retrieving the next event, updating the Reeb graph, and deleting and inserting interchange events as arcs are removed and added. The sequence can be thought of as the evolution of a single Reeb graph. Following Driscoll et al.[21], accumulate the changes to form a single data structure representing the entire evolution, which Edelsbrunner et al. [24] refer to as the *partially persistent Reeb graph*. 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.

There are difficulties in implementing this algorithm. Properties of the Jacobi curve that are valid in the smooth setting need not necessarily hold in the piecewise-linear setting. In particular, the Jacobi curve need not be a 1-manifold; it could have vertices with degree greater than two, and edges corresponding to multiple critical points. Such vertices and edges have

1a ++		
1b +- -+		
1c --		
2a M+		
2b M+ +M		
2c M- -M		
3a -S		
3b -S S-		
3c +S S+		
4 MM		
5 MS SM		
6 SS		

Fig. 1.14: 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 [24].

to be unfolded to simulate the 1-manifold property. Techniques to maintain the integrity of structural properties in the piecewise linear setting are discussed in greater detail in [26].

**1.3.4 Comparison.**

We compare the algorithms of Sections 1.3.2 and 1.3.3. The former considers a discrete set of functions, their Reeb graphs, and maps the arcs of the Reeb graph at  $t$  to the arcs of the graph at  $t + 1$  using a definition of temporal coherence. This algorithm works well when the time sampling rate is high relative to the phenomenon under study so that there is good temporal coherence. Unlike the latter, it provides no understanding of how the Reeb graph changes over time, nor can it produce a Reeb graph for all continuous values of time. The algorithm of Section 1.3.3 assumes a continuous space-time function, uses Jacobi sets to connect all Reeb graphs, systematically enumerates the possible changes that the Reeb graph experiences, and captures the evolution of the Reeb graph over time. The time-varying Reeb graph can be used to compute the topological information computed in Section 1.3.2. The arc mapping information is implicit; each Reeb graph arc at time  $t + 1$  has a sequence of events that maps it to an arc at time  $t$ . For the topology change graph, start with the Reeb graph  $R_t$ , examine each arc containing isovalue  $s$  and compute the time when the Jacobi segments attached to its end-nodes intersects the line  $f = s$ . At this time, the corresponding isocontour component experiences a topological change. See figure 1.15.

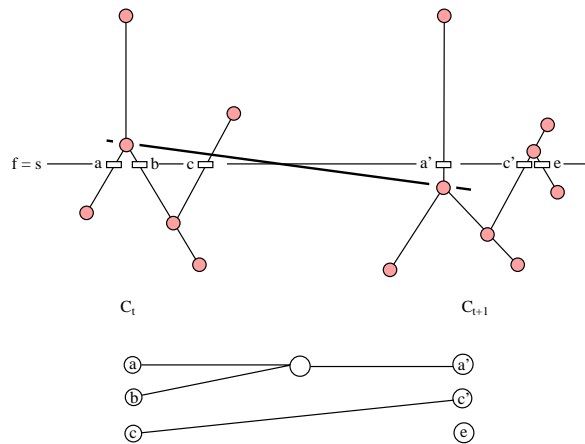


Fig. 1.15: In the top row, a contour tree at successive time steps, with arcs containing isovalue  $s$  labeled. The Jacobi segment for the upper node of the arcs containing  $a$  and  $b$  is shown. When this segment intersect the line  $f = s$  the components  $a$  and  $b$  merge. Note the difference with the TCG in Figure 1.9; in [24] function  $f$  is continuous in space-time and isocontour topology can change at any time.

## 1.4 Conclusions

Isocontour extraction algorithms for time-varying scalar fields use three techniques to increase efficiency: spatial techniques, span space techniques, and topological techniques. Spatial techniques organize space using an octree decomposition, to detect regions that intersect, and reject regions that do not intersect the isocontour. Span space techniques organize cells in the space of function values to efficiently detect cells that intersect the isocontour. Topological techniques organize a subset of cells, called the seed set, in a search structure. The seed set contains one intersecting cell for each connected component of each isocontour, and the component can be extracted by propagation from the intersecting cell. While span space techniques and topological techniques can be used for both regularly and irregularly sampled data, spatial techniques can be used only for regularly sampled data. Unlike spatial techniques and span space techniques, topological techniques allow isocontour extraction using contour propagation and produce coherent triangulations that are amenable to compression [35], simplification [37], and streaming [36]. This feature is useful for large data set visualization when the isocontours themselves might be too large to fit in memory.

Two algorithms extend the Reeb graph to time-varying functions to aid the user in selecting interesting time and isovalue parameters for visualization. The first algorithm, by Bajaj & Sohn [58], uses an overlap heuristic to connect the Reeb graph for each time slice; it works well when the time sampling rate is high relative to the phenomenon under study so that there is good temporal coherence. The second algorithm, by Edelsbrunner et al. [24], determines the actual dynamics of the Reeb graph over all time slices, under a chosen interpolation function. The dynamics depend on the Jacobi curve which is the trajectory of the critical points over time and leads to a classification of combinatorial changes experienced by the Reeb graph over time. The algorithm constructs the Reeb graph for time  $t = 0$  from scratch and sweeps forward in time, modifying the current Reeb graph to compute the evolution of the Reeb graph over time. Because the theory for this algorithm is based on smooth functions and real-world data is not smooth, special care has to be taken during implementation to ensure structural integrity of the Jacobi curve. As a consequence this algorithm is harder to implement than the first one, but it gives a more complete picture of the evolution of the Reeb graph.

Beyond the questions addressed in this paper, we see some interesting research directions that we believe will become increasingly important in visualization. Often the data is noisy and the Reeb graph is itself too large and cluttered to make any sense. We believe it is worthwhile to investigate simplification of time-varying Reeb graphs on the lines of the work of Carr et al. [12]. How can we simplify the time-varying Reeb graph, and maintain consistency with the underlying data? What meaningful measures, geometric and topological, can we devise to guide the simplification? How can we represent several levels of simplification and how can we present the user with a multi-resolution view of time-varying Reeb graphs?

## Acknowledgments

We thank our funding agencies: NSF grant 0128426 and sub-contracts from Lawrence Livermore National Labs. Portions of 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. We thank the anonymous referees for their suggestions.

## References

1. P. S. Alexandrov. *Combinatorial Topology*. Dover, Mineola, New York, 1998.
2. E. Artzy. Display of three-dimensional information in computed tomography. In *Computer Graphics and Image Processing*, volume 9, pages 196–198, 1979.
3. C. L. Bajaj, V. Pascucci, and D. Schikore. The contour spectrum. In *IEEE Visualization*, pages 167–174, 1997.
4. C. L. Bajaj, A. Shamir, and S. Bong-Soo. Progressive tracking of isosurfaces in time-varying scalar fields. Technical report, Univ. of Texas, Austin, 2002. <http://www.ticam.utexas.edu/CCV/papers/Bongbong-Vis02.pdf>.
5. T. F. Banchoff. Critical points for embedded polyhedral surfaces. In *Amer. Math. Monthly*, volume 77, pages 457–485, 1970.
6. 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*, pages 481–494, 2000.
7. J. L. Bentley. Multidimensional binary search trees used for associative searching. In *Communications of the ACM*, volume 18(9), pages 509–517, 1975.
8. J. F. Blinn. A generalization of algebraic surface drawing. In *ACM Transactions on Graphics*, volume 1(3), pages 235–256, 1982.
9. R. L. Boyell and H. Ruston. Hybrid techniques for real-time radar simulation. In *Proc. of 1963 Fall Joint Computer Conference (IEEE)*, pages 445–458, 1963.
10. H. Carr and J. Snoeyink. Path seeds and flexible isosurfaces: Using topology for exploratory visualization. In *Proc. of Eurographics Visualization Symposium*, pages 49–58, 285, 2003.
11. H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *Computational Geometry*, volume 24(2), pages 75–94, 2003.
12. H. Carr, J. Snoeyink, and M. van de Panne. Simplifying flexible isosurfaces using local geometric measures. In *Proc. of IEEE Visualization 2004*, pages 497–504, 2004.
13. Y.-J. Chiang. Out-of-core isosurface extraction of time-varying fields over irregular grids. In *Proc. IEEE Visualization 2003*, pages 217–224, 2003.
14. Y.-J. Chiang, C. T. Silva, and W. J. Schroeder. Interactive out-of-core isosurface extraction. In *Proc. of the Symp. for Volume Vis.*, pages 167–174, 1998.
15. T. Chiueh and K.-L. Ma. A parallel pipelined renderer for time-varying volume data. In *Proc of Parallel Architecture, Algorithms, Networks*, pages 9–15, 1997.
16. P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Trans. on Vis. and Computer Graphics*, 3(2):158–170, /1997.
17. 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.*, pages 344–350, 2003.
18. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1994.
19. M. Cox and D. Ellsworth. Application controlled demand paging for out-of-core visualization. In *IEEE Proc. of Vis. '97*, pages 235–244, 1997.
20. C. J. A. Delfinado and H. Edelsbrunner. An incremental algorithm for betti numbers of simplicial complexes on the 3-sphere. In *Comput. Aided Geom. Design*, volume 12, pages 771–784, 1995.
21. J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. In *J. Comput. Sys. Sci.*, volume 38, pages 86–124, 1989.

22. M. J. Düst. Additional reference to “marching cubes”. *SIGGRAPH Comput. Graph.*, 22(5):243, 1988.
23. H. Edelsbrunner and J. Harer. Jacobi sets of multiple morse functions. In F. Cucker, R. DeVore, P. Olver, and E. Sueli, editors, *Foundations of Computational Mathematics*, pages 37–57. Cambridge Univ. Press, England, 2002.
24. H. Edelsbrunner, J. Harer, A. Mascarenhas, and V. Pascucci. Time-varying Reeb graphs for continuous space-time data. In *Proc. of the 20th Ann. Sympos. on Comp. geometry*, pages 366–372. ACM Press, 2004.
25. H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Morse-smale complexes for piecewise linear 3-manifolds. In *Proc. 19th Ann. Sympos. Comput. Geom.*, pages 361–370, 2003.
26. H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical morse complexes for piecewise linear 2-manifolds. In *Proceedings of the 17th Annual Symposium on Computational geometry*, pages 70–79. ACM Press, 2001.
27. A. T. Fomenko and e. T. L. Kunii. *Topological Methods for Visualization*. Springer-Verlag, Tokyo, Japan, 1997.
28. H. Fuchs, Z. Kedem, and S. Uselton. Optimal surface reconstruction from planar contours. In *Communications of the ACM*, volume 20, pages 693–702, 1977.
29. R. S. Gallagher. Span filtering: An efficient scheme for volume visualization of large finite element models. In G. M. Neilson and L. Rosenblum, editors, *Proc. of Vis. '91*, pages 68–75, Oct 1991.
30. A. V. Gelder and J. Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics*, 13(4):337–375, 1994.
31. M. Golubitsky and V. Guillemin. *Stable mappings and their singularities*. Springer-Verlag, New York, 1973. Graduate Texts in Mathematics, Vol. 14.
32. P. Hanrahan. Three-pass affine transforms for volume rendering. In *Computer Graphics*, volume 24(5), pages 71–78, 1990.
33. G. T. Herman and H. K. Lun. Three-dimensional display of human organs from computed tomograms. In *Computer Graphics and Image Processing*, volume 9, pages 1–21, 1979.
34. C. T. Howie and E. H. Black. The mesh propagation algorithm for isosurface construction. In *Computer Graphics Forum 13, Eurographics '94 Conf. Issue*, pages 65–74, 1994.
35. M. Isenburg and S. Gumhold. Out-of-core compression for gigantic polygon meshes. In *Proc. of SIGGRAPH 2003*, pages 935–942, July 2003.
36. M. Isenburg and P. Lindstrom. Streaming meshes. In *Manuscript*, April 2004.
37. M. Isenburg, P. Lindstrom, S. Gumhold, and J. Snoeyink. Large mesh simplification using processing sequences. In *Proc. of Vis. 2003*, pages 465–472, Oct 2003.
38. J. T. Kajiya and B. P. V. Herzen. Ray tracing volume densities. In *Computer Graphics*, volume 18(3), pages 165–174, 1984.
39. A. Kaufman and E. Shimony. 3d scan-conversion algorithms for voxel-based graphics. In *1986 Workshop on Interactive 3D Graphics*, pages 45–75, 1986.
40. L. Kettner, J. Rossignac, and J. Snoeyink. The safari interface for visualizing time-dependent volume data using iso-surfaces and contour spectra. *Comput. Geom. Theory Appl.*, 25(1-2):97–116, 2003.
41. M. Levoy. Efficient ray tracing of volume data. In *ACM Transactions on Graphics*, volume 9(3), pages 245–261, 1990.
42. Y. Livnat, H. W. Shen, and C. R. Johnson. A near optimal iso-surface extraction algorithm for unstructured grids. In *IEEE Trans. on Vis. and Computer Graphics*, volume 2(1), pages 73–84, 1996.

43. W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In M. C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proc.)*, volume 21, pages 163–169, July 1987.
44. Y. Matsumoto. *An Introduction to Morse Theory (Translated from Japanese by K. Hudson and M. Saito)*. American Mathematical Society, 2002.
45. S. V. Matveyev. Approximation of isosurface in the marching cube: ambiguity problem. In *Proceedings of the conference on Visualization '94*, pages 288–292. IEEE Computer Society Press, 1994.
46. N. Max, R. Crawfis, and D. Williams. Visualization for climate modeling. In *IEEE Computer Graphics Applications*, pages 481–494, 2000.
47. B. H. McCormick, T. A. DeFanti, and M. D. Brown. Visualization in scientific computing. *Computer Graphics*, 21(6), 1987.
48. J. Milnor. *Morse Theory*. Princeton Univ. Press, New Jersey, 1963.
49. C. Montani, R. Scateni, and R. Scopigno. Discretized marching cubes. In *Proceedings of the conference on Visualization '94*, pages 281–287. IEEE Computer Society Press, 1994.
50. J. R. Munkres. *Elements of Algebraic Topology*. Addison-Wesley, Redwood City, California, 1984.
51. B. K. Natarajan. On generating topologically consistent isosurfaces from uniform samples. *Vis. Comput.*, 11(1):52–62, 1994.
52. G. M. Nielson and B. Hamann. The asymptotic decider: resolving the ambiguity in marching cubes. In *Proceedings of the 2nd conference on Visualization '91*, pages 83–91. IEEE Computer Society Press, 1991.
53. V. Pascucci and K. Cole-McLaughlin. Parallel computation of the topology of level sets. In *Algorithmica*, volume 38(1), pages 249–268, 2003.
54. G. Reeb. Sur les points singuliers d'une forme de pfaff complètement intégrable ou d'une fonction numérique. In *Comptes Rendus de L'Académie ses Séances, Paris*, volume 222, pages 847–849, 1946.
55. H. W. Shen. Iso-surface extraction in time-varying fields using a temporal hierarchical index tree. In *IEEE Proc. of Vis. '98*, pages 159–166, Oct 1998.
56. H. W. Shen, C. D. Hansen, Y. Livnat, and C. R. Johnson. Isosurfacing in span space with utmost efficiency (issue). In *Proc. of Vis. '96*, pages 287–294, 1996.
57. Y. Shinagawa and T. L. Kunii. Constructing a Reeb graph automatically from cross sections. In *IEEE Comput. Graphics Appl.*, volume 11, pages 44–51, 1991.
58. B.-S. Sohn and C. L. Bajaj. Time-varying contour topology. In *Manuscript*, 2004.
59. P. Sutton and C. Hansen. Isosurface extraction in time-varying fields using a temporal branch-on-need tree(t-bon). In *IEEE Proc. of Vis. '99*, pages 147–153, 1999.
60. N. Thune and B. Olstad. Visualizing 4-d medical ultrasound data. In *Proceedings of the 2nd conference on Visualization '91*, pages 210–215. IEEE Computer Society Press, 1991.
61. M. van Kreveld. Efficient methods for isoline extraction from digital elevation model based on triangulated irregular networks. In *Sixth Inter. Symp. on Spatial Data Handling*, pages 835–847, 1994.
62. 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 *The 13th ACM Sym. on Computational Geometry*, pages 212–220, 1997.
63. L. Westover. Interactive volume rendering. In *Chapel Hill Workshop on Volume Visualization*, pages 9–16, 1989.
64. J. Wilhelms and V. Gelder. Octrees for faster isosurface generation. In *ACM Trans. on Graphics*, volume 11(3), pages 201–227, 1992.

65. B. Wyvill, C. McPheeters, and G. Wyvill. Animating soft objects. *Visual Computer*, 2:235–242, 1986.
66. G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *Visual Computer*, 2:227–234, 1986.