Layered Reeb Graphs for three-dimensional Manifolds in Boundary Representation

B. Strodthoff, B. Jüttler

Johannes Kepler University, Linz, Austria

Abstract

Reeb graphs are topological graphs originating in Morse theory, which represent the topological structure of a manifold by contracting the level set components of a scalar-valued function defined on it. The generalization to several functions leads to Reeb spaces, which are thus able to capture more features of an object. We introduce the layered Reeb graph as a discrete representation for Reeb spaces of 3D solids (embedded three-dimensional manifolds with boundary) with respect to two scalar-valued functions. After that we present an efficient algorithm for computing the layered Reeb graph, which uses only a boundary representation of the underlying three-dimensional manifold. This leads to substantial computational advantages if the manifold is given in a boundary representation, since no volumetric representation has to be constructed. However, this algorithm is applicable only if the defining functions satisfy certain conditions.

Keywords: Reeb graph, Reeb space, boundary representation, 3D solid

1. Introduction

A *Reeb graph* [1] is a topological data structure which is able to capture the topology of shapes. Considering a scalar-valued function defined on the domain of interest, the level sets of this function may consist of several components, which evolve while sweeping through the function values. The Reeb graph encodes the evolution of these level set components, which are sometimes referred to as *contours* in the literature, by collapsing each contour to a point. Typical applications of Reeb graphs include shape abstraction [2, 3, 4], shape recognition [5, 6, 7, 8, 9] and shape decomposition [10, 11], but they are also applied e.g. to loop detection [12], landscape modeling [13], to guide hex-meshing [14] or in the analysis of trajectories [15] and point data [16].

Generalizing this approach to vector-valued defining functions (or, equivalently, to several scalar-valued functions) leads to *Reeb spaces* [17], which are thus able to capture more features of a shape. However, they are a more complicated structure. Consider for example the Reeb space of a three-dimensional manifold with respect to two functions. In general it consists of several connected surface patches, which makes its computation and storage more difficult. As our first goal, we propose the layered Reeb graph for this situation, which encodes the information captured by the Reeb space using two layers of Reeb graphs, see Figure 1.

We demonstrate that the Reeb graph of a general solid object does not capture the topology of the object. In contrast, the Reeb space with respect to two functions, and therefore also the layered Reeb graph proposed in this paper, is able to do so. Besides, the layered Reeb graph can easily be embedded into the object. From this embedding, a skeletal structure of the object can be derived. To our knowledge, no algorithm for the computation of Reeb spaces has been proposed in the literature, and our algorithm for the layered Reeb graph is a first step in this direction. We expect that the Reeb space will find various applications in shape recognition, classification and decomposition.

The existing literature on Reeb graphs describes several algorithms, which compute the Reeb graph of a manifold with boundary with respect to a given scalar-valued function, typically using a simplex mesh of the manifold as input, or using a voxel-based description, as in [18]. However, if a three-dimensional manifold is given in a boundary representation, a volumetric representation has to be generated to apply these approaches, since the Reeb graph of the manifold and the Reeb graph of its boundary surface are, in general, different objects.

As our second goal, we describe a construction algorithm for the layered Reeb graph (and, implicitly, for the Reeb graph), which uses only a boundary representation of the manifold. This leads to substantial computational advantages, since the generation of a volume representation is costly, and a boundary representation typically has a smaller data volume, e.g. comparing the number of elements in a surface- and volume mesh. However, the class of defining functions has to be restricted.

The remainder of this paper is structured as follows. In the next section, we will review related work on Reeb graphs before defining the layered Reeb graph in Section 3. In Section 4 we identify the conditions on the defining functions which we will need to construct the layered Reeb graph from a boundary representation. After that, we study the Jacobi set and its relevance for the layered Reeb graph in Section 5. Finally, our construction algorithm is described in Section 6 and we present some results in Section 7.

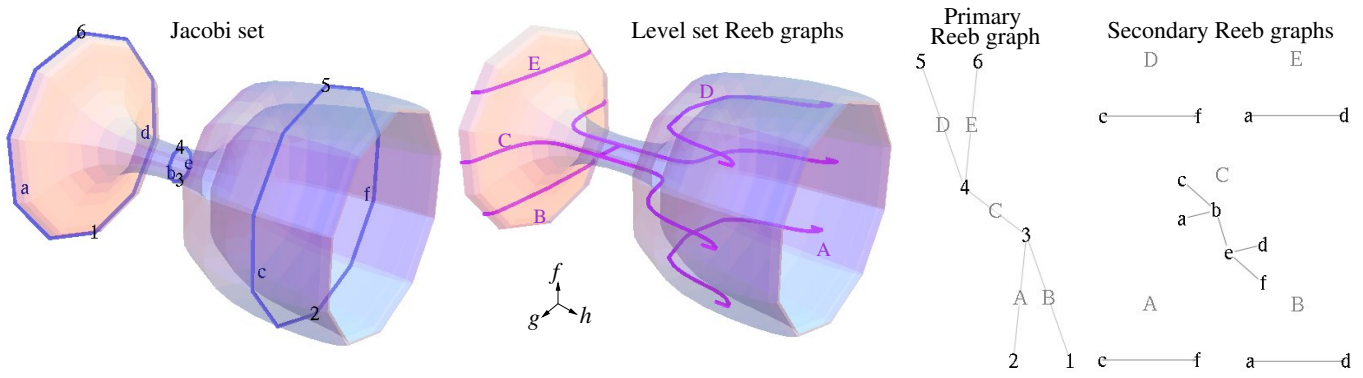


Figure 1: Example of a layered Reeb graph. Left: Object with the blue curves indicating the Jacobi set, which guides our construction, see Section 5. Center-left: For each level set component of the height function, a Reeb graph with respect to a second function is shown, which is embedded into the level set, see Definition 3. First, the level set components A and B occur, which are joined to form C, and so on. Center-right: The Reeb graph with respect to the height function, where arcs are labeled by the capital letters of the corresponding level set component. Right: The Reeb graphs of each level set component with respect to a second function. The primary and secondary graphs together form the layered Reeb graph, see also Definition 3. For a detailed explanation of the labels, see Section 7.1.

2. Related Work

Several algorithms for the computation of Reeb graphs are described in the literature, see e.g. Biasotti et al. [19, 20] or Edelsbrunner and Harer [21] for overviews.

The first known algorithm by Shinagawa and Kunii [22] constructs the Reeb graph in any dimension by explicitly *maintaining the level sets* of the function while sweeping through the function values. The level sets are updated at every vertex, which results in an $O(n^2)$ runtime. This approach was improved by using more efficient data structures for storing the level sets, by Cole-Mclaughlin et al. [23] and Brandolini and Piastra [24] for two-dimensional manifolds and by Doraiswamy and Natarajan [25] and Parsa [26] for higher dimensions.

Sampling-based algorithms analyze the evolution of level sets by dissecting the domain of interest at certain (e.g. uniformly distributed) function levels and analyzing their connectivities, e.g. by Hilaga et al. [8], Attene et al. [2], Berretti et al. [10] and Biasotti et al. [6]. They are, in principle, able to produce a multilevel representation of a shape by decreasing the distance between analyzed function levels in every step. However, they are not guaranteed to compute the correct Reeb graph, since they may loose important features if their resolution is chosen too coarse unless an additional check is included.

Other approaches exploit the vital relation between the Reeb graph and the *critical points* of the defining function. Patanè et al. [27] successively slice a given two-dimensional manifold at critical function levels and extract the adjacencies between the saddle points by flooding through surface triangles. Berretti et al. [11] successively merge level sets that are represented by the same arc in the Reeb graph. Other authors first identify all critical points and then find connections between them using monotone paths. For example, Doraiswamy and Natarajan [28] use paths of adjacent triangles of a simplex mesh of arbitrary dimension, and Strodthoff et al. [29] use monotone edge paths to construct the Reeb graph of a three-dimensional manifold in boundary representation.

The loop-free variant of Reeb graphs, called *contour trees*, can be constructed more efficiently than the Reeb graph, and there are many applicable algorithms described in the literature. For example, Carr et al. [30] first sweep through the function values twice to construct the split- and join-tree, which are then put together to form the contour tree. Chiang et al. [31] improved this approach by avoiding to sort all input vertices. Instead, they first identify component-critical points, which they connect to split- and join-trees using monotone paths. This results in a construction time of $O(n + t \log t)$ with t denoting the number of component-critical points.

Due to the efficient possibilities for constructing contour trees, there have been efforts to reduce the construction of Reeb graphs to the computation of contour trees. Tierny et al. [32] use the Reeb graph of an object to find all loops in the object, and introduce cuts to open the loops of the Reeb graph. Thus they produce one manifold with a loop-free Reeb graph, which can be handled by a contour tree algorithm. Similarly Doraiswamy and Natarajan [33] identify saddle points where they cut the object into several parts with loop-free Reeb graphs to which the algorithm of [30] is applied individually.

Some recent computation algorithms for the Reeb graph do not fit into one of these frameworks. Pascucci et al. [34] introduce an *on-line algorithm*, which constructs the Reeb graph of a simplex mesh while streaming the input data. This algorithm performs well in practice and is applicable to large meshes since it does not require the whole input to be stored in memory. Harvey et al. [35] propose a *randomized algorithm* for simplex meshes which computes the Reeb graph in $O(n \log n)$ expected time by successively collapsing triangles in random order. Furthermore Dey and Wang [36] study the approximation of the Reeb graph for a manifold given by a point sample, and a simplified Reeb graph was introduced in [24]

Reeb graphs are defined via a scalar function on a given domain of interest. The extension to several functions leads to *Reeb spaces*. They were introduced by Edelsbrunner et al. [17], including a construction algorithm for simplex meshes, but appear to be little researched by now. Reeb spaces are able to capture more features of an object than Reeb graphs, which makes them an interesting object of study. However, as mentioned before, they possess a more complicated structure than Reeb

graphs, and are, therefore, harder to handle. We introduce a different representation for a special class of Reeb spaces using two layers of Reeb graphs, which we call *layered Reeb graphs*, and an algorithm to compute this object.

Another possible extension considers the evolution of Reeb graphs over time. Edelsbrunner et al. [37, 38] give a full characterization of the changes which may occur in the Reeb graph while the defining function evolves. This work has some similarity to the evolution of the second layer Reeb graphs in our setup of the layered Reeb graph.

All the above algorithms rely on a full-dimensional representation of the input shape, typically using triangular meshes to analyze surfaces or tetrahedral meshes to analyze volumes. In contrast, we will consider three-dimensional manifolds in space which are given by an oriented triangular surface mesh that represents their boundary. The boundary-based construction algorithm for Reeb graphs of three-dimensional manifolds with respect to the height function was introduced in [29]. Here we will extend this approach to the computation of layered Reeb graphs. Furthermore we will define a feasible function class for the defining functions, for which the layered Reeb graph can be computed using only a boundary representation of the three-dimensional manifold in space. This leads to computational advantages, since we can work with a lower-dimensional object, its boundary surface. If the domain is given in a boundary representation in the first place, for example because it was exported from a CAD program, we do not need to construct a volume representation, which saves an expensive computation step. We will restrict ourselves to triangular surface meshes, since several steps of the algorithm are more efficient on meshes than on other surface representations. This work extends the extended abstracts [39]and [40].

3. The layered Reeb graph (LRG)

We begin by recalling the definition of Reeb graphs (see also [20] for an introduction). As typical in Morse theory, structural information about a manifold is obtained by analyzing a scalar-valued function defined on it. Figure 2 gives two simple examples for Reeb graphs.

Definition 1. Consider a scalar-valued function f defined on a d-dimensional manifold M with boundary. Points mapped to the same function value form a *level set*, connected parts of a level set are called *level set components* or sometimes *contours*. The *Reeb graph* of M with respect to f is obtained by contracting every level set component to a point, maintaining adjacency between level sets.

In the following, we will restrict ourselves to the cases $d = 2$ and $d = 3$ as shown in Figure 2, and assume that M is embedded in $\mathbb{R}^3$.

We can observe that, while the Reeb graph nicely reflects the main structure of a two-dimensional manifold, it may miss certain features of a three-dimensional manifold, like the inner void in Figure 2(b). The usage of more functions leads to Reeb spaces, which are thus able to capture more features.

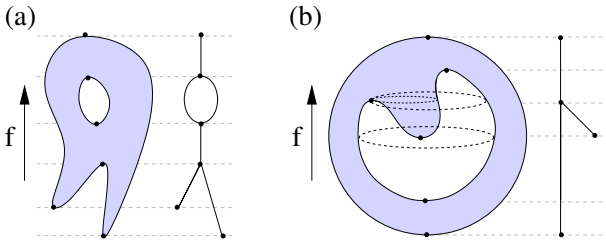


Figure 2: Reeb graphs with respect to the height function, which maps each point to its last Cartesian coordinate, of (a) a two-dimensional manifold in the plane, and (b) a three-dimensional manifold in space, containing an inner void shaped like an indented ball. For a three-dimensional image of this object, see Figures 4 and 18.

Reeb spaces were considered in 2008 by [17], generalizing Reeb graphs by considering up to $d - 1$ scalar-valued functions on a d-dimensional manifold. We will consider the case of two functions on a three-manifold with boundary. From now on, we therefore consider two scalar-valued, C^1-smooth functions f and g, defined on a three-dimensional manifold M with boundary, see also Figure 3.

Definition 2. *Level sets* consist of all points with constant f- and g-value. The *Reeb space* of M with respect to f and g is obtained by contracting every level set component to a point, maintaining adjacency between level sets.

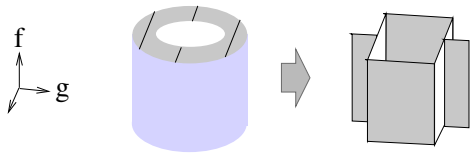


Figure 3: Reeb space for $(f, g) = (z, y)$. Level sets are line segments parallel to the x-axis, contracting them leads to the structure on the right.

As shown in Figure 3, the Reeb space captures the structure of a three-dimensional manifold better than the mere Reeb graph. However, it is a more complicated object to compute, store and handle, since it consists of surface patches. Therefore, we introduce the layered Reeb graph as a discrete representation for the structure of a Reeb space. It captures the information stored in the Reeb space using two layers of Reeb graphs, see also Figure 4.

Definition 3. For a constant value c attained by f, the Reeb graph of the level set $L_c := \{x \mid f(x) = c\}$ with respect to $g|_{L_c}$ is called a *level set Reeb graph*. To obtain the *layered Reeb graph*, the arcs of the Reeb graph with respect to f are subdivided into parts of equivalent level set Reeb graphs. Then, the corresponding level set Reeb graphs are added to these parts as a secondary structure. The *primary Reeb graph* is the Reeb graph with respect to f, enhanced by adding vertices of valency two where the structure of the level set Reeb graph changes. The level set Reeb graphs will be referred to as *secondary Reeb graphs*.

A level set with respect to f consists of points with the same f-value. Of these, the level set Reeb graph identifies all points

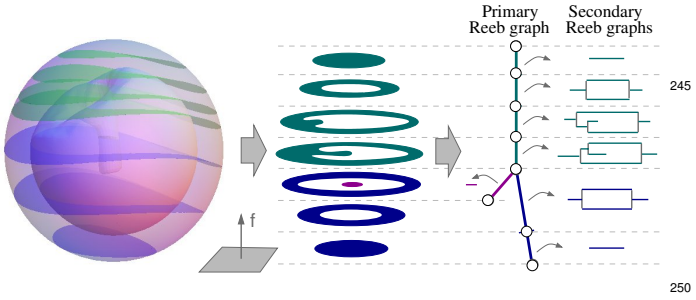


Figure 4: Layered Reeb graph. Left: Spherical object with an inner void like in Figure 2(b) with some representative level sets of f. Center: detailed view of the f-level sets in the left picture. The depicted levels are chosen such that every change in the level set Reeb graphs is captured. Right: layered Reeb graph. As compared to the Reeb graph shown in Figure 2(b), the layered Reeb graph captures the topology of this object. Here, some level set Reeb graphs contain circles, which encodes the existence of an inner void in the object.

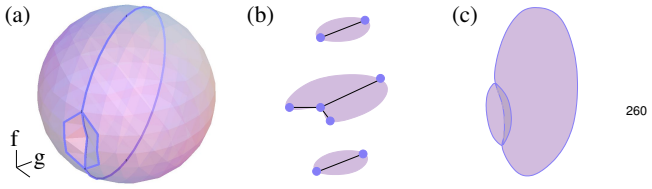


Figure 5: For a simple example: (a) Object (b) level sets with level set Reeb graphs and (c) Reeb space.

which additionally share the same g-value. Thus, each point on a level set Reeb graph represents a connected component of points of M with constant f- and g-value, same as a point on the Reeb space. Therefore, the Reeb space basically consists of the level set Reeb graphs for all f-values, stacked together with intact adjacency, see Figure 5. In this way, the layered Reeb graph actually captures the structure of the Reeb space by grouping parts with equivalent level set Reeb graphs.

4. Feasible functions

In this section, we will define a class of functions for which the layered Reeb graph can be computed using a boundary representation of the domain of interest.

Critical points.
First, we consider the behavior of the level sets of f and g in a point.

Definition 4. A point x in M is called *regular*, if there exists a neighborhood $U(x)$ in which the level sets of f and g are two-dimensional manifolds that dissect $U(x)$ into exactly four parts. Otherwise, x is called *critical*, see Figure 6.

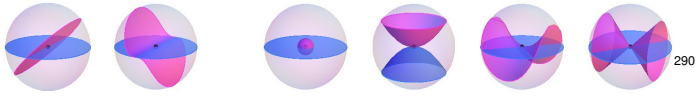


Figure 6: Neighborhood of two regular (left) and four critical (right) points.

Since we want to work only with a boundary representation of the model, we will consider functions without critical points in M.

Lemma 5. *Two C^1-smooth functions f and g with $|\nabla f \times \nabla g| \neq 0$ on a neighborhood[1] of M induce no critical points in M.*

Proof. Consider a small sphere $U(x)$ centered in a point x in M. If x is critical according to our definition, the level set surfaces of f and g dissect $U(x)$ into either more or less than four parts.

To achieve less than four parts, either at least one level set surface has to degenerate, or the level set surfaces must not change sides while intersecting. In the first case, the gradient of the function whose level set degenerates becomes zero. In the second case, the surfaces touch in x, thereby causing their gradients to be linearly dependent. For getting more than four parts, the level set surfaces have to intersect non-trivially in x, which also requires the gradients of f and g to be linearly dependent in x. Thus, in every critical point we have $|\nabla f \times \nabla g| = 0$. □

Note that Lemma 5 gives a sufficient, but non-necessary condition for critical points. For example in the second picture in Figure 6, the gradients of f and g are linearly dependent, but the point is still regular according to our definition.

Feasibility condition.
Once we assumed the gradients of f and g to be linearly independent, we additionally assume to know a third function h such that (f, g, h) form a reparametrization of space.

Definition 6. Two functions f and g on M are called *feasible* if $|\nabla f \times \nabla g| \neq 0$ and if additionally a function h is available such that $\det(\nabla f, \nabla g, \nabla h) \neq 0$ on a neighborhood of M.

In the (f, g, h)-space, level set surfaces are planes. Thus, geometric decisions in the level set surfaces, like the relative position of a point to a curve or the orientation of a curve, are reduced to decisions in the plane. These decisions are important in order to work with boundary representations, since they allow us to reconstruct the structure of a level set surface when we know only its boundary curves along the surface of the manifold. Additionally we know that the function h is monotonic along the common level set curves of f and g, so we can see it as a parameter along these curves.

Existence of function h.
Note that a function h fulfilling our requirements does not always exist. For example, consider the case where $f(x, y, z) = z$ and $g(x, y, z) = r_{x,y}$, with $r_{x,y}$ denoting the distance of a point to the z-axis, and consider a torus M centered at the origin. Then, M contains common level set curves of f and g which are closed curves. Thus, we cannot hope to find a function h that is monotone along these level set curves.

On the other hand, the existence of h is guaranteed if we find a gradient field H which is linearly independent of ∇f and

[1] We consider two functions f and g which are defined on an open set Ω, $M \subset \Omega \subseteq \mathbb{R}^3$.

∇g. So, if we can find parameters $\alpha, \beta, \gamma : \mathbb{R}^3 \to \mathbb{R}, \alpha > 0$ such that

$$H = \alpha(\nabla f \times \nabla g) + \beta \nabla f + \gamma \nabla g$$

is a gradient field, we can choose h as the potential of H, i.e. such that $\nabla h = H$. A sufficient condition for H to be a gradient field is if it is curl-free, i.e. curl$H = 0$, on an open, simply connected domain containing our manifold M, e.g. on a neighborhood of its convex hull. A more detailed discussion of this topic is beyond the scope of this paper.

Piecewise linear setup.
So far we have considered C^1-smooth functions. However, we will work with piecewise linear approximations of these functions in the implementation. We assume the manifold to be given as a sufficiently fine surface mesh. We then assume that there exists a sufficiently fine, conforming tetrahedral mesh, which approximates also the gradients of the functions sufficiently well. Then, the conditions for feasible functions carry over to their piecewise linear approximations. Additionally we assume that the level set surfaces of the functions intersect the boundary in a generic way, such that the level sets of each function on the boundary are curves (without 2D patches) which intersect only in points.

5. Jacobi set

According to the previous section, we consider functions without critical points inside the manifold. In this section, we will consider the functions' behavior on the manifold's boundary. We will introduce the Jacobi set of two functions on a surface and explain its relation to the layered Reeb graph.

5.1. Definition of the Jacobi set

We now consider the restrictions of f and g to the boundary of M. For them, we define the Jacobi set as considered in [41], see also Figures 7 and 8.

Definition 7. The *Jacobi set* of two functions on the boundary of M consists of all points on the boundary in which the level set curves of the two functions do not cross.

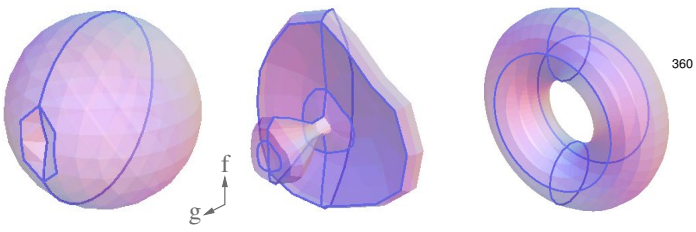


Figure 7: Jacobi sets for three piecewise linear examples.

In [41], the Jacobi set was introduced for a smooth setup as the set of points where $|\bar{\nabla} f \times \bar{\nabla} g| = 0$, with $\bar{\nabla}$ denoting the gradient vector in the boundary surface. One can verify that, in the smooth case, our definition coincides with their definition of the Jacobi set. However, it is applicable directly to the piecewise linear case we consider. In this section, we will denote the piecewise linear approximations of f and g on the boundary which are induced by the given triangular surface mesh by $\bar{f}$ and $\bar{g}$, respectively.

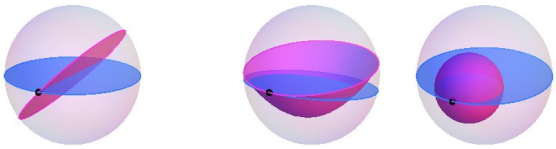


Figure 8: Points on the boundary of a sphere M. Left: the black point does not lie on the Jacobi set, since the level sets of f and g cross on the boundary. Middle and right: the black point lies on the Jacobi set, since in the restriction to the boundary the level sets of f and g do not cross.

The authors of [41] also describe some properties of the Jacobi set, for example that, in our setup of two functions on a boundary surface, it forms a network of curves in which every vertex has even degree, see Figure 7.

5.2. Relation to level set Reeb graphs

Inspired by [37, 38], we will use the relation between Jacobi sets and Reeb graphs, but our setup is slightly different. While there the Jacobi set on the domain of interest is used in order to track the vertices of the Reeb graph, we ruled out these critical points inside the domain. Instead, we use the Jacobi set of the functions restricted to the boundary to trace changes in the level set Reeb graphs which are induced by the boundary. The following lemma states the relation between the Jacobi set on the boundary and the level set Reeb graphs, see also Figure 5.

Lemma 8. *For feasible functions f and g, a point p on ∂M lies on the Jacobi set of $\bar{f}$ and $\bar{g}$ exactly if it induces a vertex or leaf in the level set Reeb graph of the f-level set containing p.*

Proof. A point p lies on the Jacobi set exactly if the level sets of $\bar{f}$ and $\bar{g}$ do not cross in p. This is equivalent to saying that in a neighborhood of p, the points on the $\bar{f}$-level set either have all higher, or all lower $\bar{g}$-value than p. Therefore, on the boundary curve of the f-level set, p is a local extremum with respect to $\bar{g}$. If we denote the restriction of g to a level set $\{x \mid f(x) = c\}$ of f by g_c, this means that the level sets of g_c touch (without crossing) the boundary curve of $\{x \mid f(x) = c\}$ exactly in the points which lie on the Jacobi set. Therefore, these are the points where the level sets of g_c may collapse or connect, i.e. the vertices or leafs of the level set Reeb graph. □

From Lemma 8, we get that the vertices and leafs of the level set Reeb graphs move along the Jacobi set on the boundary. If we consider adjacent f-level sets, changes in the level set Reeb graph will only occur if we pass specific configurations on the Jacobi set, see Table 1 for an overview of the generic cases. For example, if we pass a local extremum of the Jacobi set curve, two vertices in the level set Reeb graph will appear or disappear. Additionally, the level set Reeb graph may change if two arcs of the Jacobi network swap their relative position with

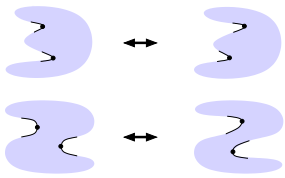
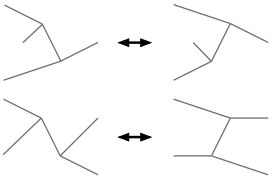
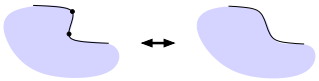
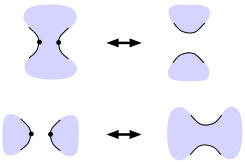
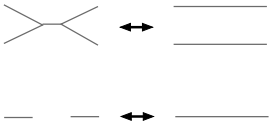
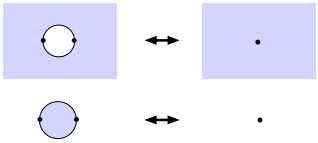
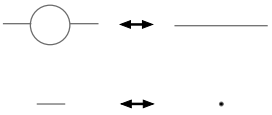
case	change	Jacobi set	level set component	level set Reeb graph
1	swap g position of two vertices	swap		
2	arc (dis)appears	extremum		
3	connectivity change	extremum		
4	contract circle or arc	extremum		

Table 1: Generic changes in adjacent Reeb graphs induced by the Jacobi set. For local extrema of the Jacobi set, the transition from the left to the right setup corresponds to passing a local maximum, while the other direction occurs when passing a local minimum.

respect to g. This means that a vertex of the level set Reeb graph which was below (with respect to g) another vertex in one level set of f passes the g-level of the other vertex and lies above it (with respect to g) in a neighboring f-level set. Of course, the level set Reeb graph also undergoes changes if we pass a vertex of the Jacobi set. However, since the Jacobi network consists of a disjoint set of topological circles in the generic case, and vertices occur only if local extrema collapse with other curve components, we will not include these cases in the discussion. Our implementation, however, works also on examples where the Jacobi set contains vertices.

5.3. Computation of the Jacobi set

For computing the Jacobi set, we use a similar approach as in [41]. As an input, we consider a triangular surface mesh, and we assume the piecewise linear approximations of the functions induced by the mesh. In this setup, the Jacobi set consists of edges of the surface mesh. Therefore it suffices to test every edge of the input mesh whether it forms part of the Jacobi set. For this test, we use an extension of the corresponding criterion introduced in [41].

In short, for every edge, the method proposed in [41] finds a linear combination $h_\lambda = f + \lambda g$ of f and g which is constant along the edge. Then, h_λ is evaluated in the opposite vertices in the two triangles incident to the edge. If these vertices both have a higher or lower function value than the edge itself, the edge is on the Jacobi set. While this criterion is very efficient in most cases, it encounters problems when the opposite vertices of the incident triangles share the same h_λ-value as the tested edge. However, these cases are not ruled out by simply prescribing different f- and g-values on all vertices, for example. They occur in the special third case shown in Figure 9.

When encountering these cases, we therefore use a different criterion. Instead of computing an additional function h_λ, we work with the f- and g-level sets directly, see also Figure 9. For the tested edge, we determine the four points in the incident triangles which lie on the same f- or g-levels sets as the edge midpoint. Then, the edge lies on the Jacobi set if these level sets do not cross each other. While this approach also has problems with the special case as in Figure 9, it is more intuitive in this setup to resolve these cases by a lexicographic ordering in f and g. Under the conditions mentioned at the end of Section 4, we can thus achieve consistent results, assuming a well-behaved mesh where, for example, there are no thin triangles whose area is near machine-precision.

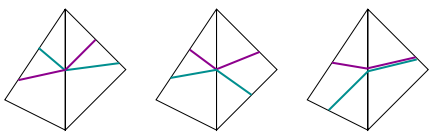


Figure 9: Piecewise linear level sets of f and g (colored) on boundary triangles (black) incident to the tested edge. Left: the usual case, level sets intersect. Middle: level sets touch without crossing, the edge in the middle is part of the Jacobi set. Right: needs special care, for example by reducing this case to a kind of lexicographic ordering. We will skip the technical details here.

From now on, we will use f and g to refer to the piecewise linear approximations of the given, C^1-smooth functions. We assume that neither f nor g are constant on any triangle of the surface mesh and that there is no edge on which both f and g are constant.

6. Boundary-based computation of the LRG

The layered Reeb graph is computed by sweeping through the f-level sets. First, all *events* are identified, i.e. points of M where changes occur in the structure of level sets of f or in their level set Reeb graphs. Then, we sweep through the level sets of f, starting at the lowest value. Information about the current level set is stored in the *status*. At each event, the *event handler* decides which level set components in the status are influenced by the event, and implements these changes. We will now describe these steps in more detail. More details are available in the doctoral thesis [42]..

6.1. Status

The status contains information about the current f-level set by storing a list of all its components, see Figure 10. Each level set component is an object which stores its current level set Reeb graph and a list of its boundary components. Each boundary component basically consists of a list of references to the Jacobi curves it intersects, sorted by their sequence along the boundary curve. Here we assume the curves carry an orientation which is derived from the orientation of the boundary surface.

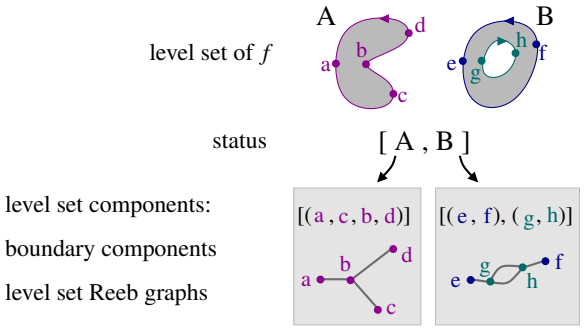


Figure 10: Information stored in the status. For the level set depicted in the top of the figure, the status stores two level set components A and B, where each of them stores its level set Reeb graph and a list of its boundary components.

For feasible functions, we can then reconstruct a level set component from the information stored in the status when necessary. First, we trace all oriented boundary curves of the level set component, starting from the Jacobi-references stored in the status. For every curve, its orientation in the (g,h)-plane then tells us whether its inside or outside contains the level set component. Finally, we can determine the relative position of the curves using a point-in-curve operation in the (g,h)-plane, if necessary.

6.2. Events

Considering the information stored in the status, there are basically four types of events:

① *Level set events* where the number or connectivity of level set components changes.

② *Boundary events* where the number or connectivity of boundary components changes.

③ *Jacobi events* where the Jacobi references of a boundary change.

④ *Swap events* where a level set Reeb graph changes, the motivation for this name will follow.

One can observe that by this definition, these types are not disjoint, since level set events are also boundary events, boundary events are also Jacobi events, and Jacobi events always induce changes to the level set Reeb graph. In the following, we will categorize an event by the smallest set it belongs to, see Figure 11.

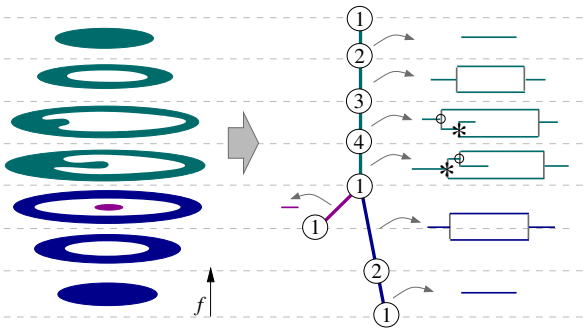


Figure 11: Example from Figure 4. Left: level set components. Right: Layered Reeb graph including categorization of events.

Events of the first three types occur in vertices or local extrema of the Jacobi set. They can be identified efficiently once the Jacobi set has been determined. Swap events occur if two vertices of the level set Reeb graph (or, equivalently, the Jacobi arcs they move on) swap their relative position with respect to g. In Figure 11, the two vertices marked by $\circ$ and $*$ in the level set Reeb graph are swapped, and the graph connectivity changes. To find these events, we do a preliminary sweep through the Jacobi set, maintaining a list of intersected Jacobi arcs sorted by the g-value of their intersection with the current f-level set.

6.3. Handling level set components

Every event contains a specific event handler, which consists of four parts: the positioner, the associator, the level set adaptor and the level set Reeb graph handler. We will shortly mention the tasks of the first three of these here, and go into some more detail for the level set Reeb graph handler in the next section.

The *positioner* stores relevant information about the event, like references to the swapped Jacobi arcs in a swap event or to the underlying vertex of the Jacobi set in the other types of events. Additionally, it remembers the necessary information to reconstruct the influenced level set component(s).

When handling an event, first of all the *associator* identifies all elements of the status which are influenced by the event. There are different types of associators depending on the character of the event, see also Figure 12.

A1: If the event is not a local minimum of the Jacobi set, its lower Jacobi arcs can be looked up in the status, directly.

A2: For local minima of the Jacobi set which are not local minima of the surface, the associator traces a curve along

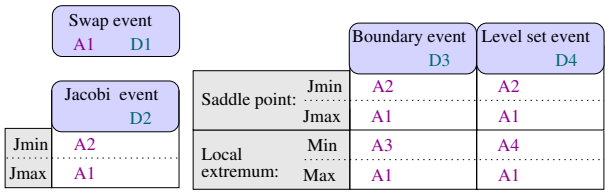


Figure 12: Associators (A1 to A4) and level set adaptors (D1 to D4) for different kinds of events. In the finer classification, *Jmin* and *Jmax* denote the local minima or maxima of the Jacobi set that are no local extrema of the boundary surface, while *Min* and *Max* denote the local minima and maxima of the boundary surface.

the f-level set to the next intersection with a Jacobi arc. Then, the encountered arc can be looked up in the status. For local minima of the boundary surface, there is yet another distinction.

A3: If the event is a level set event, then a new level set component appears, so no association to lower components is necessary.

A4: In a boundary event, on the other hand, a new boundary component appears in an existing level set component. To find the influenced component, the associator successively traces out the boundaries of level set components in the status, until one of the tested level set components contains the minimum.

In the next step, the *level set adaptor* carries out the necessary changes in the status. Again, there are different types of adaptors, see Figure 12.

D1: In a swap event, there are no changes in the level set components.

D2: In a Jacobi event, the lower Jacobi arcs are simply replaced by the upper Jacobi arcs, if any, in the corresponding boundary component.

D3: For a boundary event, the influenced boundary components additionally have to be split or merged in this point.

D4: Finally, in a level set event, the incoming level set components are replaced by new ones. In this case, the remaining boundary components of the lower level set components have to be assigned carefully to the correct new components.

In the end, the *level set Reeb graph handler* carries out necessary changes to the level set Reeb graphs. We would like to go into more detail here, so we will design the next section to this topic.

6.4. Handling level set Reeb graphs

We use a mixture of two approaches for handling the level set Reeb graphs.

In the *brute force approach*, the level set Reeb graph of a level set component is recomputed between each two successive events on the corresponding arc of the primary Reeb graph. In order to do this, the boundary curves of a level set are traced at an intermediate f-level. Then, a sweep with respect to g provides the level set Reeb graph, see the embedded level set Reeb graphs in Figure 1 and in Figures 14 to 16 for some results.

This approach is obviously very costly. Since the computation of the level set Reeb graphs has to be repeated many times, it easily dominates the total computation time. For many events it is, however, rather straightforward how to adapt the level set Reeb graph to reflect the structure of the level set component above the event.

We were able to reduce the number of recomputed graphs by 90% using *adaptation rules* for many of the generic cases in Table 1. We fully implemented adaptation rules for the generic cases 1, 2 in the table, cases 3 and 4 are partly covered by adaptation rules. If no adaptation rule is available, we recompute the level set Reeb graph like in the brute force approach mentioned earlier, like for example in the following cases.

- We do not use adaptation rules for vertices of the Jacobi set, since, in a general setup, they appear by far more seldom than local extrema of the Jacobi set.
- Higher degree saddle points and degenerate cases are not covered by adaptation rules since they are also uncommon in a general setup.
- For saddle points (case 3 in the table), only boundary events are covered. When level set components merge or split, the level set Reeb graphs would have to be merged or split up, which is not implemented at the moment.
- For saddle points (case 3), only the case of a local maximum of the Jacobi set is covered by an adaptation rule, and not the symmetric case for a local minimum. This is because in local minima of the Jacobi set it is harder to identify which part of the status to adapt.

Representatively for the other cases, we now consider the adaptation rule for *swap events* in more detail. Here, two Jacobi arcs swap their relative position with respect to g. There are no changes if the swapped Jacobi arcs belong to different level set components, since they swap vertices on two independent level set Reeb graphs. Otherwise, the corresponding vertices of the level set Reeb graph are swapped in the vertex list. If the level set Reeb graph contains an edge connecting the swapped vertices, we additionally have to swap some incident arcs between the two vertices. This can only occur if both vertices have valency three, see Figure 13 for typical cases.

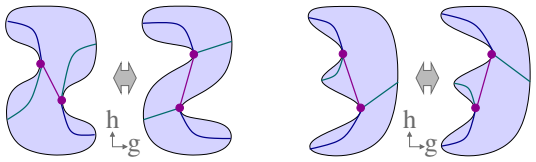


Figure 13: Possible changes in a swap event (type ④).

7. Results

We implemented the algorithm in C++: The manifold is given as triangular surface mesh and the piecewise linear approximations of the functions are considered.

7.1. Some examples

Using Mathematica, an automatic visualization for the output of the construction algorithm is generated, see Figures 1 and 14 to 19. For each example, the layered Reeb graph is visualized by four pictures.

- The first one shows the arcs of the Jacobi set (labeled by lowercase characters). In addition it shows the knots and the local extrema of the Jacobi set (i.e., all events except for swap events), labeled by numbers.
- The next picture shows instances of level set Reeb graphs (see Definition 3) between consecutive events that act on the same level set component. They are labeled by uppercase characters. The level set Reeb graphs have been embedded into three-dimensional space by representing each connected component by its midpoint. This picture has been omitted in Figures x and x, since the visualization is too complex.
- The primary Reeb graph is shown in the next picture. Its arcs correspond to the instances of level set Reeb graphs shown in the previous picture and they are identified by the same labels. Recall that the primary Reeb graph is obtained from the Reeb graph with respect to the first function f by splitting its edges at events that change the level set Reeb graph. These events are exactly the ones labeled by numbers in the first picture plus the swap events (which were not shown there, since one cannot identify them with a single location).
- The final picture shows the secondary Reeb graphs that represent the structure of the level set Reeb graphs. The labels in uppercase characters identify the associated arc of the primary Reeb graph, and the lowercase characters specify the arcs of the Jacobi set, which are traced by the nodes of the level set Reeb graphs.

7.2. Efficiency of adaptation rules

For the three nontrivial objects shown in Figure 20, the layered Reeb graph of the object and of its complement are computed using six different combinations of the coordinate functions. Of these in total 12 setups per object, we represent the smallest, the most complicated, and an average result in Table 2.

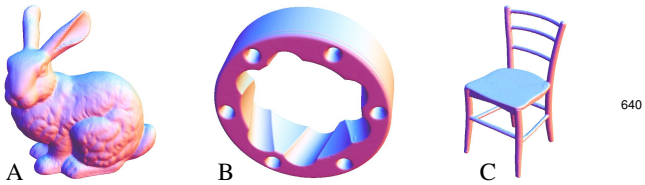


Figure 20: The three objects considered in Table 2: A: Watertight model of the Stanford bunny. B and C: Rolling stage and wooden chair model from AIM@SHAPE shape repository.

7.3. Computation time depending on output size

Table 2 suggests that the size of the input mesh has a rather insignificant influence on the runtime of the construction algorithm, since for the same input mesh very different runtimes are encountered depending on the chosen functions. When analyzing the numbers in more detail, experimental results suggest that the total runtime depends approximately linearly on the output size, i.e. on the total number of arcs in all level set Reeb graphs, see Figure 21.

object	triangles	events	adapts	total arcs	time using brute force	time using adaptation
A	69 664	2 669	2 423	47 492	3.6s	0.4s
		4 030	3 074	109 877	6.4s	1.4s
		13 265	12 662	1 058 542	49.7s	2.5s
B	382 242	3 995	3 776	81 380	15.6s	2.1s
		6 569	5 647	114 055	14.4s	2.2s
		63 809	60 757	5 004 230	333.9s	19.3s
C	408 398	4 697	4 162	50 647	13.4s	2.1s
		14 222	13 009	529 834	77.6s	8.6s
		56 593	53 202	4 555 288	262.8s	19.2s

Table 2: Some results for three function setups per object. Number of events, the number of events which can be handled by an adaption rule, total number of arcs in all level set Reeb graphs, total time for the pure brute force algorithm and total time for the adaptive algorithm.

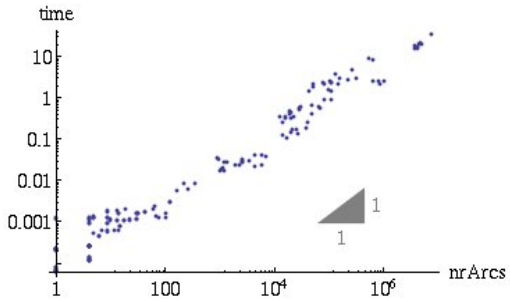


Figure 21: Computation time depending on total number of arcs in all level set Reeb graphs in a doubly logarithmic plot.

8. Conclusion

We defined the layered Reeb graph as a discrete representation of the structure of a Reeb spaces and formulated sufficient conditions on the defining functions to allow a boundary based construction. After that we presented an efficient algorithm to construct the layered Reeb graph using only a boundary representation of the underlying manifold.

When deriving our construction, we assumed that the defining functions do not possess critical points inside the considered manifold. The extension to Morse functions with critical points in the interior is of great interest. A promising approach is the following:

- Subdivide the object into smaller parts which contain no critical points in their interior.
- Compute the layered Reeb graph of these parts.
- Merge the different layered Reeb graphs.

Another possible extension concerns the auxiliary function h. We now assumed to find a function h that fulfills the desired properties globally. Instead, it would also be possible to follow a similar subdivision approach as described in the previous

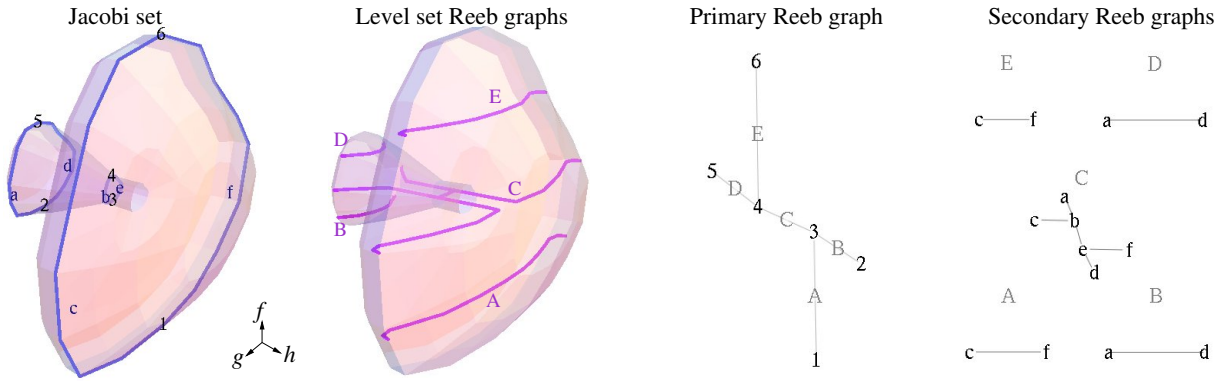


Figure 14: Example of a layered Reeb graph. Left: A mushroom-shaped object whose "cap" has a concave inner boundary. The blue lines show the Jacobi set. Center-left: Level set Reeb graphs embedded at the midpoints of the represented (f, g)-level sets. Center-right and right: Primary and secondary Reeb graphs. The graphs are drawn such that the f-value (or the g-value for the level set Reeb graphs) of a vertex serves as its y (or x) coordinate in the planar picture.

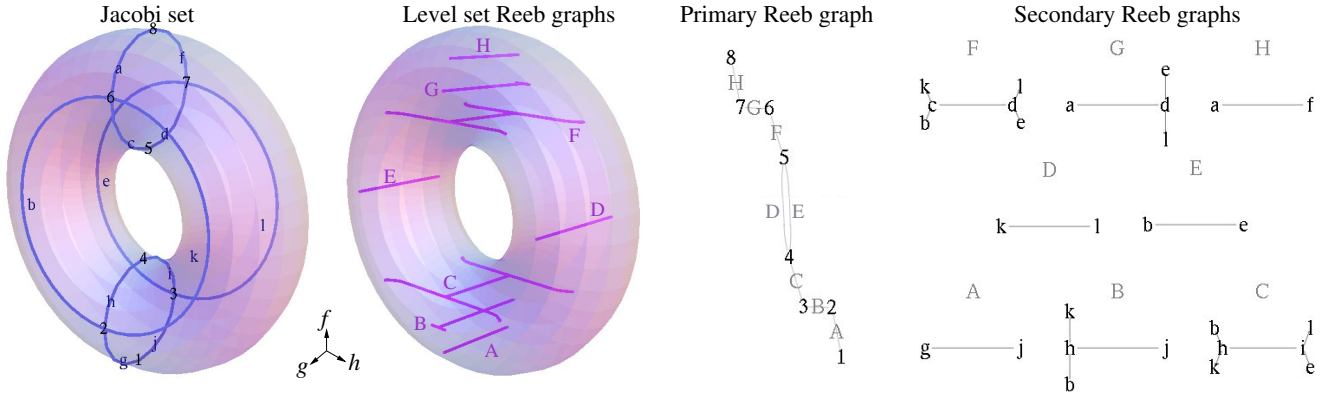


Figure 15: Example of a layered Reeb graph. The level set Reeb graph of the lowest component evolves from A via B to C, and symmetrically the level set Reeb graph of the upper component evolves from F via G to H. These changes are represented by vertices of valency two in the primary Reeb graph.

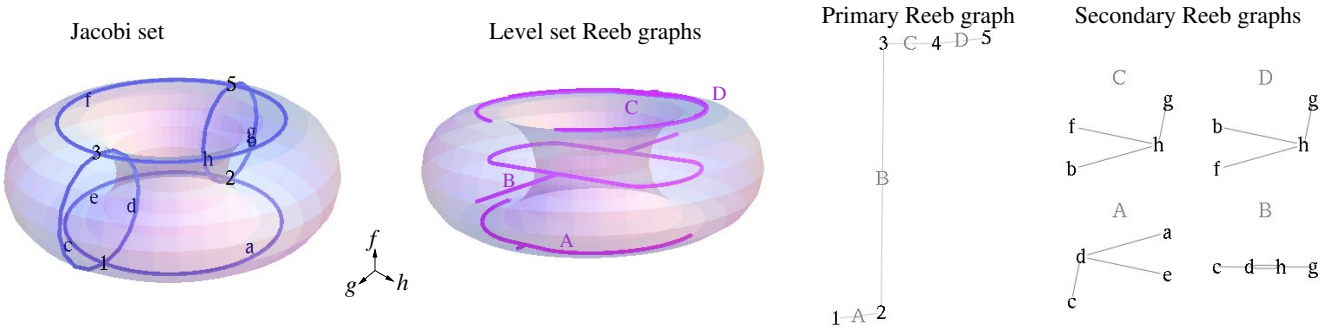


Figure 16: Example of a layered Reeb graph. In this example, the primary Reeb graph with respect to f contains only vertices of valency two. Event 4 is a swap event: In level set graph C, vertex f is slightly to the left of vertex b, and the two vertices are swapped in the transition to graph D.

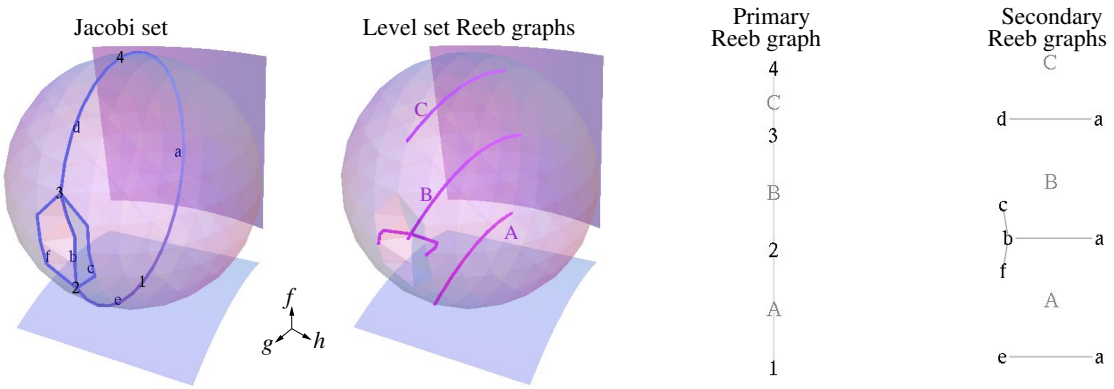


Figure 17: Example of a layered Reeb graph for nonlinear defining functions. A representative level set surface for f and g, each, are shown. With $f(x, y, z) = y^2/5+z$, $g(x, y, z) = x^2/5 + y$ and $h(x, y, z) = -x$ we get $|(\nabla f, \nabla g, \nabla h)| = 1 \neq 0$, so the functions f and g are feasible.

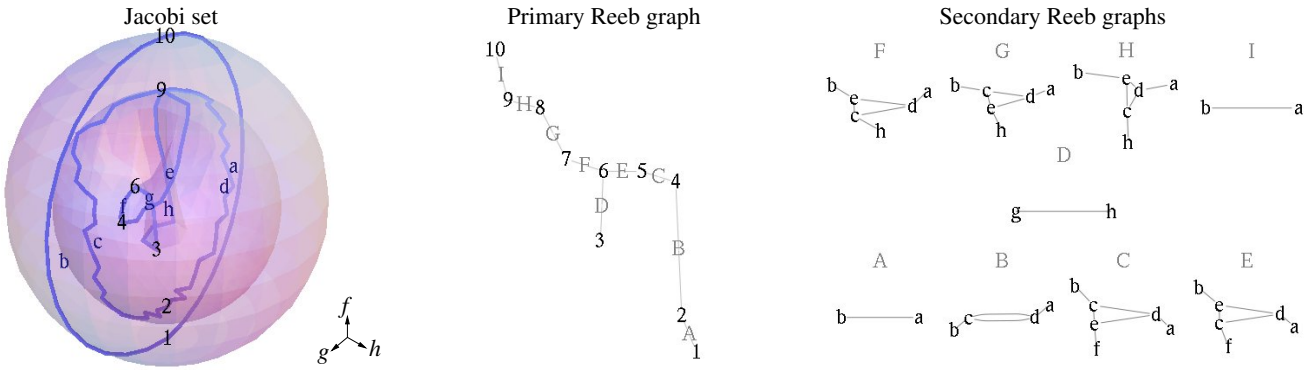


Figure 18: Example of a layered Reeb graph. The object is a sphere with an inner void, similar to the one in Figures 2(b) and 2. Events 5, 7 and 8 are swap events. We do not show the embedded level set Reeb graphs here, since there are too many to see them nicely in the picture.

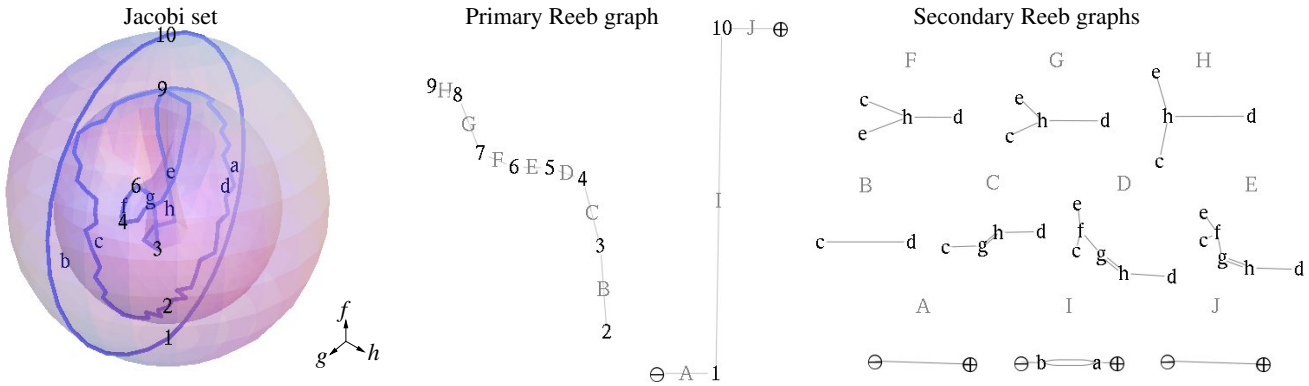


Figure 19: Example of a layered Reeb graph, now of the complement of the shown object. The Reeb graph with respect to f has two components, one for the outer volume, and one for the inner void. The nodes $\ominus$ and $\oplus$ in both the primary graph and the secondary graphs represent the nodes at minus and plus infinity.

paragraph, for example by considering different functions h on different level sets of f.

Additionally, it would be an interesting topic to determine how to choose the functions f and g. One can distinguish two scenarios. On the one hand, the Morse functions may be determined by the application in mind. For example, the Reeb graph with respect to the height function is a valuable tool in the simulation of dip coating processes [29]. A similar situation may occur for specific applications of the Reeb space.

On the other hand, if no functions are given, one may seek to optimize the choice of the Morse functions depending on the given object. For example, one could align the functions with the principal axes of the considered object, so as to make the resulting Reeb graph independent of rotation or translation of the object. Also, one could seek to use functions which result in a simple or in a more complicated layered Reeb graph, depending on the level of detail that should be resolved. The optimization of the Morse functions is a challenging topic for future research.

So far, we considered the layered Reeb graph of a three-dimensional manifold in space. Currently, we are investigating the possible extension to moving three-dimensional manifolds, seen as four-dimensional manifolds in space-time. Sweeping through values of f= time, a level set Reeb graph is now the Reeb graph of the moved manifold at a fixed instance of time, see Figure 22. Many ideas carry over to this setup from the case of three-dimensional manifolds in space, like e.g. the connection to Jacobi sets. While the construction of the Reeb graph with respect to the height function finds an application in the simulation of dip-coating processes [29], the Reeb graph of a moving manifold could be applied in the simulation of the dip-coating process of an object that moves and rotates over time.

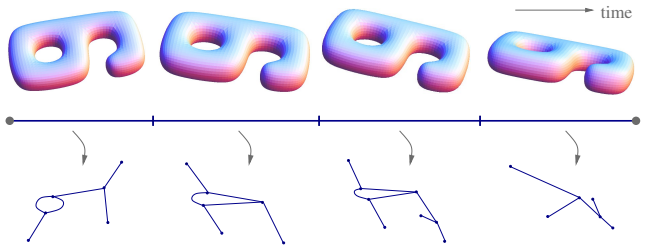


Figure 22: Schematic view of the layered Reeb graph of a moving three-dimensional manifold in space-time.

As a second application of the layered Reeb graph, we are considering its use in the decomposition of solid objects into contractible solids. These can be used as input for other decomposition methods, like [43]. While the Reeb graph of the object surface is enough to find all holes in the object [32], it does not provide sufficient information to cut all these holes. Firstly, if a level set component contains holes, some additional information is needed to identify its inner boundaries for inserting a valid cutting surface. Secondly, cuts can only be performed efficiently along level sets of the defining function. This suffices to make the Reeb graph of the object loop free, which is the main objective in [32]. However, the object itself may still contain holes. Consider, for example, a vertical pipe, which cannot be cut into hole-free pieces by cutting along any level set of the height function. Therefore, we are currently exploring how the layered Reeb graph can be used to provide this additional information.

9. Acknowledgements

This work was supported by the ESF Programme EuroGIGA-Voronoi.

References

[1] G. Reeb, Sur les points singuliers d'une forme de Pfaff completement integrable ou d'une fonction numerique, Comptes Rendus Acad. Science 222 (1946) 847–849.

[2] M. Attene, S. Biasotti, M. Spagnuolo, Shape understanding by contour-driven retiling, The Visual Computer 19 (2003) 127–138.

[3] R. El Khoury, J.-P. Vandeborre, M. Daoudi, 3D mesh Reeb graph computation using commute-time and diffusion distances, in: Electronic Imaging 2012 Symposium, SPIE, 2012, pp. 8090–16.

[4] A. Pepe, L. Brandolini, M. Piastra, J. Koikkalainen, J. Hietala, J. Tohka, Simplified Reeb graph as effective shape descriptor for the striatum, in: Mesh Processing in Medical Image Analysis 2012, Vol. 7599 of Lecture Notes in Computer Science, 2012, pp. 134–146.

[5] V. Barra, S. Biasotti, 3D shape retrieval using kernels on extended Reeb graphs, Pattern Recognition 46 (2013) 2985–2999.

[6] S. Biasotti, S. Marini, M. Spagnuolo, B. Falcidieno, Sub-part correspondence by structural descriptors of 3D shapes, Computer-Aided Design 38 (9) (2006) 1002–1019.

[7] T. Tung, F. Schmitt, Augmented Reeb graphs for content-based retrieval of 3D mesh models, in: Proc. of the Shape Modeling International, IEEE Computer Society, 2004, pp. 157–166.

[8] M. Hilaga, Y. Shinagawa, T. Kohmura, T. L. Kunii, Topology matching for fully automatic similarity estimation of 3D shapes, in: Proc. Conf. Computer Graphics and Interactive Techniques, ACM, 2001, pp. 203–212.

[9] W. Mohamed, A. BenHamza, Reeb graph path dissimilarity for 3D object matching and retrieval, The Visual Computer 28 (3) (2012) 305–318.

[10] S. Berretti, A. del Bimbo, P. Pala, 3D mesh partitioning for retrieval by parts applications, in: Proc. IEEE Int. Conf. on Multimedia and Expo, 2005, pp. 1210–1213.

[11] S. Berretti, A. del Bimbo, P. Pala, 3D Mesh decomposition using Reeb graphs, Image and Vision Computing 27 (2009) 1540–1544.

[12] T. K. Dey, F. Fan, Y. Wang, An efficient computation of handle and tunnel loops via Reeb graphs, ACM Trans. Graph. 32 (4) (2013) 32:1–32:10.

[13] K. Ohmori, T. Kunii, Three dimensional sketch for a landscape using Morse theory and Reeb graphs, in: Int. Conf. on Cyberworlds (CW), 2012, pp. 178–183.

[14] W. Yu, K. Zhang, S. Wan, X. Li, Optimizing polycube domain construction for hexahedral remeshing, Computer-Aided Design 46 (2014) 58 – 68.

[15] K. Buchin, M. Buchin, M. Kreveld, B. Speckmann, F. Staals, Trajectory grouping structure, in: Algorithms and Data Structures, Vol. 8037 of Lecture Notes in Computer Science, 2013, pp. 219–230.

[16] F. Chen, H. Obermaier, H. Hagen, B. Hamann, J. Tierny, V. Pascucci, Topology analysis of time-dependent multi-fluid data using the Reeb graph, Computer Aided Geometric Design 30 (6) (2013) 557 – 566.

[17] H. Edelsbrunner, J. Harer, A. K. Patel, Reeb spaces of piecewise linear mappings, in: Proc. Sympos. on Comput. Geom., ACM, 2008, pp. 242–250.

[18] D. Shattuck, R. Leahy, Automated graph-based analysis and correction of cortical volume topology, IEEE Transactions on Medical Imaging 20 (11) (2001) 1167–1177.

[19] S. Biasotti, D. Attali, J.-D. Boissonnat, H. Edelsbrunner, G. Elber, M. Mortara, G. Baja, M. Spagnuolo, M. Tanase, R. Veltkamp, Skeletal structures, Mathematics and Visualization, Springer Berlin Heidelberg, 2008, pp. 145–183.

[20] S. Biasotti, D. Giorgi, M. Spagnuolo, B. Falcidieno, Reeb graphs for shape analysis and applications, Theor. Computer Science 392 (1-3) (2008) 5–22.

[21] H. Edelsbrunner, J. Harer, Computational Topology - an Introduction, American Mathematical Society, 2010.
[22] Y. Shinagawa, T. Kunii, Constructing a Reeb graph automatically from cross sections, IEEE Computer Graphics and Appl. 11 (6) (1991) 44–51.
[23] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, V. Pascucci, Loops in Reeb graphs of 2-manifolds, Discrete and Computational Geometry 32 (2) (2004) 231–244.
[24] L. Brandolini, M. Piastra, Computing the Reeb graph for triangle meshes with active contours, in: ICPRAM (2)'12, 2012, pp. 80–89.
[25] H. Doraiswamy, V. Natarajan, Efficient algorithms for computing Reeb graphs, Computational Geometry 42 (6-7) (2009) 606–616.
[26] S. Parsa, A deterministic O(m log m) time algorithm for the Reeb graph, Discrete Comput. Geom. 49 (2013) 864–878.
[27] G. Patanè, M. Spagnuolo, B. Falcidieno, A minimal contouring approach to the computation of the Reeb graph, IEEE Transactions on Visualization and Computer Graphics 15 (4) (2009) 583–595.
[28] H. Doraiswamy, V. Natarajan, Output-sensitive construction of Reeb graphs, IEEE transactions on visualization and computer graphics 18 (1) (2012) 146–159.
[29] B. Strodthoff, M. Schifko, B. Jüttler, Horizontal decomposition of triangulated solids for the simulation of dip-coating processes, Computer Aided Design 43 (2011) 1891–1901.
[30] H. Carr, J. Snoeyink, U. Axen, Computing contour trees in all dimensions, Computational Geometry 24 (2003) 75–94.
[31] Y.-J. Chiang, T. Lenz, X. Lu, G. Rote, Simple and optimal output-sensitive construction of contour trees using monotone paths, Computational Geometry 30 (2) (2005) 165–195.
[32] J. Tierny, A. Gyulassy, E. Simon, V. Pascucci, Loop surgery for volumetric meshes: Reeb graphs reduced to contour trees, IEEE Trans. on Visualization and Computer Graphics 15 (6) (2009) 1177–1184.
[33] H. Doraiswamy, V. Natarajan, Computing Reeb graphs as a union of contour trees, IEEE transactions on visualization and computer graphics 19 (2) (2013) 249–262.
[34] V. Pascucci, G. Scorzelli, P.-T. Bremer, A. Mascarenhas, Robust on-line computation of Reeb graphs: simplicity and speed, ACM Trans. Graph. 26 (2007) 58.1–58.9.
[35] W. Harvey, Y. Wang, R. Wenger, A randomized O(m log m) time algorithm for computing Reeb graphs of arbitrary simplicial complexes, in: Proc. Sympos. Computational Geometry, 2010, pp. 267–276.
[36] T. Dey, Y. Wang, Reeb graphs: Approximation and persistence, Discrete and Computational Geometry 49 (1) (2013) 46–73.
[37] H. Edelsbrunner, J. Harer, A. Mascarenhas, V. Pascucci, Time-varying Reeb graphs for continuous space-time data, in: Proc. Sympos. on Comput. Geom., ACM, 2004, pp. 366–372.
[38] H. Edelsbrunner, J. Harer, Time-varying Reeb graphs for continuous space-time data, Computational Geometry 41 (2008) 149–166.
[39] B. Strodthoff, B. Jüttler, Layered Reeb graphs of a spatial domain, in: Booklet of Abstracts of EuroCG, 2013, pp. 21–24.
[40] B. Strodthoff, B. Jüttler, Computing layered Reeb graphs, in: Booklet of Abstracts of EuroCG, 2014.
[41] H. Edelsbrunner, J. Harer, Jacobi sets of multiple Morse functions, Foundations of Computational Mathematics, Minneapolis 2002 (2004) 35–57.
[42] B. Strodthoff, Algorithms for computing Reeb graphs and spaces for triangulated shapes in boundary representation, Ph.D. thesis (2014).
[43] B. Jüttler, M. Kapl, D.-M. Nguyen, Q. Pan, M. Pauley, Isogeometric segmentation: The case of contractible solids without non-convex edges, Computer-Aided Design 57 (0) (2014) 74 – 90.