Using Foliation Leaves To Extract Reeb Graphs On Surfaces

Shaodong Wang, Wencheng Wang, *Member, IEEE,* and Hui Zhao

Abstract—For Reeb graph extraction on surfaces, existing methods always use the isolines of a function defined on the surface to detect the surface components and the neighboring relationships between them. Since such detection is unstable, it is still a challenge for the extracted Reeb graphs to stably and concisely encode the topological information of the surface. In this paper, we address this challenge by using foliation leaves to extract Reeb graphs. In particular, we employ a method for generating measured harmonic foliations by defining loops for foliation initialization and diffusing leaves from loops over the surface. We demonstrate that when the loops are determined, the neighboring relationships between the leaves from different loops are fixed. Thus, we can use loops to represent surface components for robustly detecting the interrelationships between surface components. As a result, we are able to extract stable and concise Reeb graphs. We developed novel measures for loop determination and improved foliation generation, and our method allows the user to manually prescribe loops for generating Reeb graphs with desired structures. Therefore, the potential of Reeb graphs for representing surfaces is enhanced, including conveniently representing the symmetries of the surface and ignoring topological noise. This is verified by our experimental results which indicate that our Reeb graphs are compact and expressive, promoting shape analysis.

Index Terms—Reeb graph, topology, foliation

1 Introduction

Reeb graphs are widely used to encode surface topological information to facilitate geometry processing, such as topology-aware shape matching [1], [2], skeleton extraction [3], and nontrivial loop computation [4]. Given a scalar function $f : \mathcal{M} \rightarrow \mathbb{R}$ defined on a manifold surface M, the connected components of the level sets of f (i.e., $\{p \in \mathcal{M}, f(p) = \alpha\}$) are contracted to single points of the Reeb graph, where the points that are contracted from the level sets through critical points (i.e., $\{p \in \mathcal{M}, \nabla f(p) = 0\}$) form the nodes of the Reeb graph, and the other points form the arcs of the Reeb graph. Clearly, the Reeb graph is constructed by the topological changes of the level sets of f, and the critical points correspond to the topological changes. In general, the critical points can be classified into the following two categories: extrema that cause a surface component to be created or destroyed; and saddles that reflect the transition between surface components, where the *surface components* are obtained by segmenting the surface along the level sets through the saddles, and they are correspondingly classified into topological disks and cylinders [5].

According to the above discussion, Reeb graphs represent the neighboring relationships between surface components. To facilitate shape analysis and processing, Reeb graphs should capture the significant surface components to provide compact representation, and robustly represent the neighboring relationships between these surface components, regardless of whether the shapes have various poses or contain noise. Otherwise, unstable and complicated Reeb graph extraction would prevent the implementation of downstream tasks. For this, existing methods have studied many functions, such as height functions [6], geodesic distances [1], [3], and harmonic functions [7], [8], to generate isolines (level sets of $2D$ functions) to extract the Reeb graph on the surface. Unfortunately, these methods have various shortcomings regarding generating critical points, so more surface components than required may be generated or the neighboring relationships between the surface components may be incorrectly determined, as illustrated in Fig. 1(a)(b). A detailed discussion on this is provided in Section 2.

In this paper, we address the challenge of extracting stable and concise Reeb graphs that capture only the required surface components and robustly represent the neighboring relationships between them using *measured foliations*. Simply speaking, the measured foliation on a two-manifold decomposes it into lower-dimensional submanifolds, which are called its *foliation leaves* (or leaves for short), where the 1-dimensional leaves are curves on the surface, while the 0-dimensional leaves are points called the *singularities* [10]. Theoretically, the measured foliation is equivalent to a quadratic differential, and when its leaves are *closed*, the leaves through saddles can segment the surface into topological disks and cylinders [**?**], [11]. Motivated by this, we propose a method that involves extracting surface components using the measured foliation and then generating the Reeb graphs. To the best of our knowledge, no implementational method has been proposed for this purpose. In our method, the input is a triangular mesh that represents the closed orientable manifold surface, and we adopted the methods proposed by Palmer [12] and improved by Zhao et al. [**?**] to generate closed measured foliations. With the methods of [**?**], [12], some loops of triangles of the mesh are

• *S. Wang, W. Wang, and H. Zhao are with the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences and the University of Chinese Academy of Sciences, Beijing, China.*
E-mail: {wangsd, whn, huizhao}@ios.ac.cn.
corresponding author: Wencheng Wang

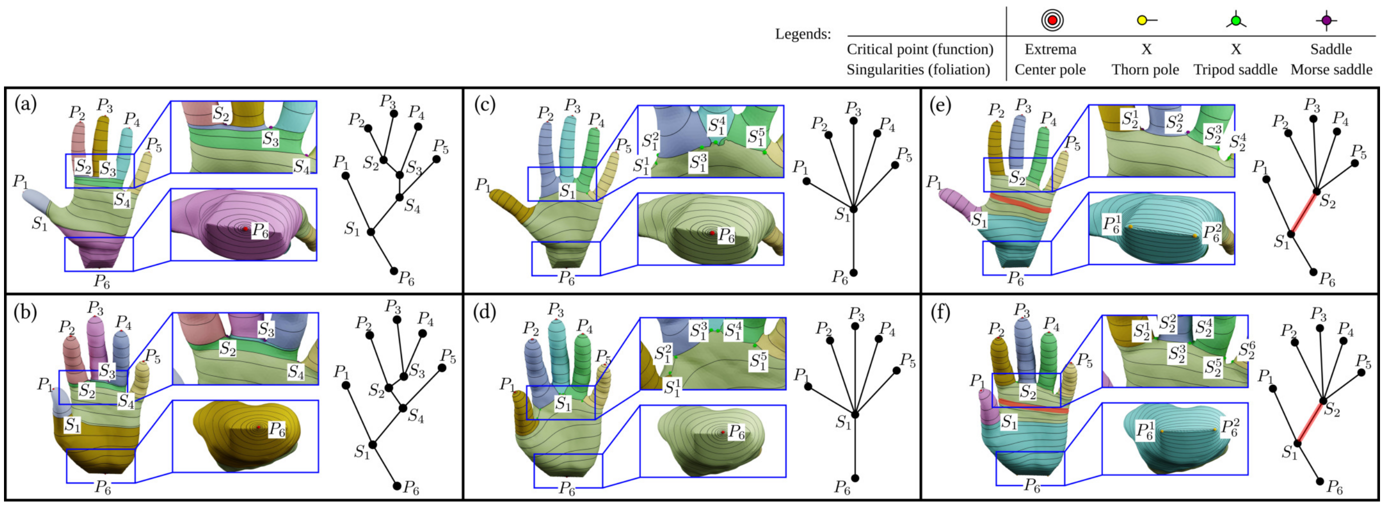


Fig. 1. Comparison of the extracted Reeb graphs using a harmonic function [9] (a)(b) and our method (c)(d) with the same extrema/poles determined on the hand in different poses. Clearly, our results are stable, but the results of the harmonic function are not. Our method also allows the user to insert a loop in red to generate one's desired Reeb graphs in (e)(f), where the thumb is not directly connected to the other four fingers. Here, different surface components are illustrated in different colors. Meanwhile, different types of critical points and singularities are marked in differently colored spheres as shown in the legend, which are also used in the following figures throughout this paper. We denote a pole/extrema as P and a saddle as S. Using the harmonic function, center poles and More saddles can be represented by its extrema and saddles respectively, but not for thorn poles and tripod saddles [10]. As for our method, all these critical points and singularities can be represented.

first defined to initialize a closed measured foliation whose leaves are contained in the loops, and then the leaves of the loops are diffused over the mesh to obtain a harmonic closed measured foliation. As a result, the mesh can be segmented into disk and cylinder components along the leaves through the singularities of the foliation, and the Reeb graph can be extracted thereof. In Section 4.1, we demonstrate that there is a one-to-one correspondence between the surface components and input loops, so our Reeb graph can concisely represent the surface components as intended. Meanwhile, the neighboring relationships between these surface components are fixed as long as the loops for representing surface components are determined, i.e., meaning stable Reeb graph extraction. This method is superior to the existing methods. As illustrated in Fig. 1, with a harmonic function [9], the Reeb graph arcs between saddles S_2, S_3, S_4 in (a)(b) do not correspond to significant parts of the shape, and the Reeb graphs in (a)(b) are not consistent with each other for these two similar models, due to the instability of $S_2 \sim S_4$. Using our method, the resulting Reeb graphs in (c)(d) are compact and the same for both models.

To effectively implement our method, we developed the following novel measures:

- Novel measures for determining loops are developed to effectively capture significant surface components, by which Reeb graphs can be robustly extracted, as discussed in Section 4.2. This also helps the user generate suitable Reeb graphs for improving shape analysis. As illustrated in Fig. 1(e)(f), the user can manually insert a loop located in the middle of the palm so that the thumb is not directly connected to the other four fingers in the extracted Reeb graph.
- Novel measures are developed to improve two aspects of the methods of [**?**], [12]. The first is to use mean value weights to enable foliation generation without extra poles on non-Delaunay meshes, while this cannot be guaranteed by the methods of [**?**], [12] due to its use of cotangent weights, which will be discussed in Section 5.1. The second is to explicitly generate leaves to detect the saddle leaves to determine the neighboring relationships between surface components, while the methods of [**?**], [12] only implicitly encode leaves on the edges, as discussed in Section 5.2. Benefitting from this, the extracted Reeb graphs can also be easily embedded in $\mathbb{R}^3$ for skeleton extraction.

As our loops can be determined intuitively, our method can potentially produce Reeb graphs with enhanced surface representation, such as effectively revealing surface symmetries and removing topological noise disturbances, which will be discussed in Section 6.2.

In sum, we have the following contributions:

- A novel method to extract stable and concise Reeb graphs using foliation leaves is proposed, which has the potential of Reeb graphs enhanced for shape analysis.
- The methods of [**?**], [12] are improved to more effectively generate foliations for Reeb graph extraction.

2 Related Works

2.1 Reeb Graphs

Many methods have been proposed for extracting Reeb graphs on surfaces. When only the combinatorial structure is required, an efficient minimal contouring method is proposed [13] that only uses isolines of the saddles without sweeping, which is enough for applications such as parameterization. When the geometric characteristics of the surface must be well represented, Cole-McLaughlin et al. [14] suggested sweeping the isolines of the function over the surface to construct a Reeb graph that can be embedded in $\mathbb{R}^3$ and act as a skeleton of the surface. Since Reeb graphs were introduced in computer graphics by Shinagawa et al. [6] in 1991, they have been applied in a wide variety of applications, including scientific visualization [15], shape matching [1], [2], shape chartification [8], [9] and nontrivial loop computation [4], and many studies for effectively extracting Reeb graphs have been performed, e.g., handling high-dimensional manifolds [16], simplicial complexes [17], [18], [19], [20], [21], time-varying scalar functions [22], and parallel computation [23], [24]. It is beyond the scope of this paper to survey the existing methods. For a comprehensive understanding, please refer to [24], [25], [26]. Here, we mainly briefly discuss the extraction of stable and concise Reeb graphs on orientable 2-manifolds.

As discussed in Section 1, critical points play a key role in extracting Reeb graphs. For this, there are many studies in the existing methods, as discussed in the following.

Many methods use height functions [4], [27], [28] and geodesic distances [1], [2], [3]. The height functions are easy to compute. However, it is not an easy task to define a height function with its upright direction meaningfully aligned with the complex surfaces under investigation, and the Reeb graph may change dramatically when the upright direction changes. Meanwhile, height functions are sensitive to the geometrical details on the surface; therefore, more extrema than desired may be produced. Thus, the Reeb graphs extracted by height functions cannot handle complex surfaces well. Geodesic distances are intrinsic to the surface and invariant to rigid motions; thus, they can handle complex surfaces. Unfortunately, geodesic distance computation is very sensitive to local geometric perturbations; thus, many auxiliary extrema would be generated, which leads to overcomplicated Reeb graphs that prevent shape understanding. To suppress the influences of local geometric perturbations, the overall characteristics of the surface are taken into account by the use of spectral distances [29] and Laplacian eigenfunctions [2], [30] to avoid producing too many extrema; however, these methods cannot completely solve this problem because they cannot guarantee that no auxiliary extrema are produced.

As the harmonic function can generate extrema only at prescribed points due to the maximum principle [31], some methods [7], [9], [30] suggest using the harmonic function to extract structurally-simple Reeb graphs. Here, the harmonic function is computed by the Laplace equation with Dirichlet boundary conditions, which can be set on the feature points of a mesh [30]. However, saddle points with similar but different function values would have more isolines through the saddles, which would create extra surface components, causing cluttered and unstable Reeb graphs, as illustrated in Fig. 1(a)(b).

For handling the problems caused by nearby saddles, some methods, such as the extended Reeb graph [32] and multi-resolution Reeb graphs [1], approximate the Reeb graph by discretely sampling some isolines rather than sweeping all the isolines. The region between two neighboring isolines forms a strip, and each strip is contracted to a single point of the Reeb graph. In this way, the nearby saddles could be placed in one strip, often called a critical area [32], and so only one node in the Reeb graph is produced. This is also helpful for simplifying the Reeb graph and handling degenerate critical points. However, properly sampling the isolines is not a trivial task.

In general, Reeb graphs are expected to facilitate shape analysis and processing, so that Reeb graphs should represent the topology of the surface as succinctly as possible. As Reeb graphs may be generated that are too large or complex to be used conveniently, further abstraction of the Reeb graphs is required. For this, some methods propose constraining the number of critical points by numerically approximating the function [28], [33], while some methods abstract the Reeb graph by eliminating some critical points to enable a compact representation [34], [35], [36]. As these methods cannot avoid extra surface components caused by saddles with similar but different values, they cannot promise the extraction of stable and concise Reeb graphs when the saddles are not well determined. For removing small surface components, the approach of [8] suggests that the user can interactively collect saddles with similar values into the same saddle isoline. Unfortunately, handling complex surfaces in such a way could be time-consuming because the approach of [8] needs to edit the saddles in pairs and each editing requires multiple user interactions.

2.2 Foliation Generation

In computer graphics, foliation has been used for improvement, such as using geodesic foliation to guide weaving pattern designs for easy manufacturing [37] or constructing a bijective map between two surfaces or volumes by integrating the mappings between the corresponding leaves of the two foliations for the surfaces or volumes using simplicial foliations [38]. In our treatment, we need harmonic measured foliations; thus, a brief discussion of harmonic measured foliations follows.

Harmonic measured foliations are closed and can be conveniently implemented on surface meshes [12], [39], [40], and among them, the method proposed by Palmer [12] provides an easy way for us to adaptively capture surface components, as discussed in Section 1. Thus, the method of [12] is a good choice for our foliation generation. Considering that the method in [12] may generate poles, which is not desired for conformal parameterization, Zhao et al. [**?**] developed an improvement to avoid such poles. Therefore, our method for foliation generation is actually based on the improved method of [**?**].

Unfortunately, the foliations generated with the methods of [**?**], [12] cannot be directly used for our Reeb graph extraction, as discussed in the following. 1) According to the study in [8], the surface components that Reeb graphs represent can be classified into two categories: disk components and cylinder components. For representing disk components, poles are required [**?**], [11]. Although [12] may generate poles, the positions and the number of poles are not controllable. This is not desired for extracting Reeb graphs, as we need to control poles to represent the desired disk components. 2) In [**?**], [12], loops are allowed to be set manually, but manually prescribing many loops on complex surfaces is a difficult task. 3) The methods of [**?**], [12] only implicitly encode the leaves of the foliation, which is not sufficient for detecting saddle leaves for constructing Reeb graphs, as we need explicitly presented saddle leaves to distinguish surface components. 4) The methods of [?], [12] take Delaunay meshes as input, but the surface meshes in applications cannot always meet such a requirement.

To solve the above problems so that foliation leaves can be used effectively for our Reeb graph extraction, we developed many novel measures, which are discussed in Sections 4 and 5.

3 Preliminaries

Here, we first review some background regarding measured foliations and then describe the methods of [?], [12] for foliation generation on meshes.

3.1 On Measured Foliations

Measured foliations. A *measured foliation* $\mathcal{F}$ on a closed orientable surface $\mathcal{M}$ is a foliation with finite singularities together with a transverse measure that is invariant under transverse homotopy [?]. It has been proved that for any measured foliation $\mathcal{F}$, there is a unique (meromorphic) quadratic differential $\varphi(z)dz^2$ that induces $\mathcal{F}$ [?], [11]. Thus, the measured foliation can be studied using the coordinate functions associated with the quadratic differential.

Quadratic differentials. A *quadratic differential* on a Riemann surface is a set of complex-valued functions $\varphi_i(z_i)$ defined in the local charts U_i, and on the overlapping domains between two charts U_i, U_j, the local parameters z_i, z_j for these functions transform accordingly with the ”quadratic” rule, as follows [11]:

$$\varphi_j dz_j^2 = \varphi_i(\frac{dz_i}{dz_j})^2 \tag{1}$$

Singularities. The *singularities* of the measured foliation correspond to the critical points of the quadratic differential, which are classified into zeros and poles. A point $z_i \in \mathbb{C}$ is called a *zero* (also called a *saddle*, which is used throughout this paper) of the quadratic differential if $\varphi_i(z_i) = 0$, and a *pole* if $1/\varphi_i(z_i) = 0$ [11].

Leaves. The *leaves* of the measured foliation correspond to the horizontal trajectories of the quadratic differential, which are lines on the surface where $\text{Im}(\int \sqrt{\varphi(z)}dz) = const$ [11]. Intuitively, it locally resembles an isoline $y = const$ in the coordinate chart U_i, but may correspond to different isovalues in different charts. We call a leaf a *regular leaf* if it does not cross a singularity, a *saddle leaf* if it connects to saddles, or a *pole leaf* if it connects to poles (including isolated center poles). Note that a saddle leaf or a pole leaf can connect multiple singularities together. Saddle leaves and pole leaves are collectively referred to as *critical leaves*.

Closed measured foliation. A measured foliation is said to be *closed* if its regular leaves are closed curves [10]. According to [?], [11], the surface can be segmented into topological disks or cylinders using the saddle leaves of the closed measured foliation, where the regular leaves in a disk or cylinder are freely homotopic to each other, and not freely homotopic to the regular leaves in the other disks and cylinders. The segmented disks and cylinders are regarded as the surface components in our paper.

Based on the above discussion, similar to the isolines of a scalar function, we can use the leaves of a closed measured foliation to extract the Reeb graph. Specifically, each critical leaf corresponds to a node of the Reeb graph, while each surface component corresponds to an arc of the Reeb graph.

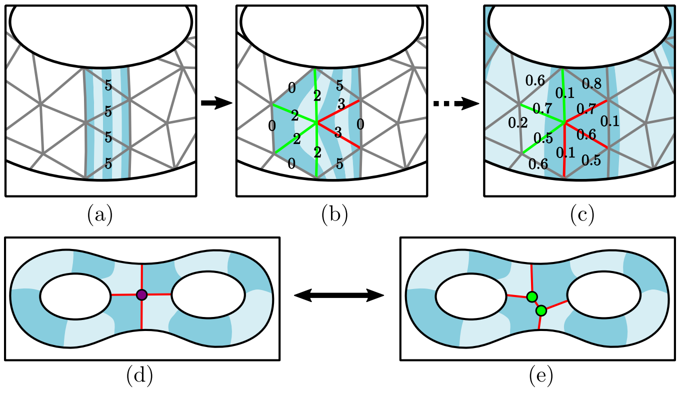


Fig. 2. The pipeline of the methods of [?], [12]. (a) A loop of triangles is defined for foliation initialization, where the same foliation value, say 5, is assigned to the consecutive edges of the loop, meaning ”5 leaves” (illustrated by colored stripes) crossing the edges. (b) The edges incident to a vertex is divided into several sectors, say 2 sectors in red or green, where the leaves crossing them are correspondingly illustrated. (c) The harmonic foliation is obtained by updating the foliation values on edges until convergence. (d) Singularities (marked in a purple point) may occur as the leaves from loops diffuse. (e) A Whitehead move is applied for splitting a singularity in (d) into two singularities (marked in two green points), and vice versa.

Whitehead equivalence. Two measured foliations are said to be *Whitehead equivalent* if one can be deformed into the other either by 1) isotopic deformation of leaves or 2) splitting a singularity into two that are connected by a leaf, and vice versa [41]. These two operations are called *Whitehead moves*.

3.2 Generating Harmonic Measured Foliation on Triangular Meshes

In the discrete setting, the surface $\mathcal{M}$ is represented by a triangular mesh $M(V, E, T)$, where V, E and T are the sets of vertices, edges and triangles for mesh M.

Definitions. For harmonic measured foliation generation on M, Palmer [12] proposes the *discrete measured foliation* F as a map $F : E \to \mathbb{R}^{\geq 0}$, where a nonnegative value w_{ij}, called the *foliation value*, is assigned to each edge e_{ij} of the mesh, meaning ”the number of leaves” crossing the edge, as illustrated in Fig. 2(a-c).

According to [12], the singularities of F can be found on the vertices and triangles of the mesh by examining the *index* of F as follows. A corner $\angle ijk$ between edges e_{ij} and e_{jk} is said to be *closed* if $w_{ij} + w_{jk} = w_{kj}$. Then, the index of a vertex v is $i_v = 2 - O(v) + Z(v)$, where $O(v)$ denotes the number of closed corners around v, and $Z(v)$ denotes the number of edges adjacent to v with zero foliation values. Accordingly, the vertex represents a regular point, saddle, or pole of F if i_v is zero, negative, or positive respectively. Similarly, the index of a triangle t is $i_t = 2 - X(t) - Z(t)$, where $X(t)$ denotes the number of nonclosed corners in t, and $Z(t)$ denotes the number of edges in t with zero foliation values. If i_t is negative, it means that the triangle contains a saddle (poles are not generated in triangles).

Algorithm. Palmer [12] proposes an algorithm for generating harmonic measure foliations on M, which consists of two steps, as discussed in the following.

1) Initialization. A set of disjoint, nontrivial triangle loops that are not freely homotopic to each other L and real values K are defined to initialize the foliation $F^{(0)}$. A triangle loop $l_i \in L$ is a cycle of triangles with shared edges between them, and the foliation value w_{ij} on the shared edge e_{ij} is set as $k_i \in K$. Meanwhile, the foliation values on the other edges of the mesh are each assigned zero values. Intuitively, $F^{(0)}$ is initialized as a closed measured foliation with its leaves contained in the input loops, as illustrated in Fig. 3(a).

2) Optimization by discrete Whitehead moves. The initialized foliation $F^{(0)}$ is updated iteratively with a gradient-descent like algorithm until a harmonic foliation F is obtained. We denote the edges incident to a vertex v_i as $\mathcal{E}_i$, and they are divided into *sectors* by the closed corners. In each iteration, the foliation values on the edges in $\mathcal{E}_i$ are updated by the *discrete Whitehead move*, as shown in Fig. 2(b). Each discrete Whitehead move runs by subtracting a gradient value δ from w_{ij} of e_{ij} in one sector, say the sector S marked in red edges, and adding δ to w_{ij} of e_{ij} in the other sectors marked in green edges, where the gradient value δ is computed by $(\sum_{e_{ij}\in S}\alpha_{ij}w_{ij} - \sum_{e_{ij}\in\mathcal{E}_i\setminus S}\alpha_{ij}w_{ij})/\sum_{e_{ij}\in\mathcal{E}_v}\alpha_{ij}$, and α_{ij} are the cotangent weights [42] of the edges. With Whitehead moves applied iteratively, the gradient values over the mesh decrease until convergence, so the harmonic measured foliation is generated, as shown in Fig. 2(c).

With the above algorithm, the nonzero foliation values on the edges of the initial foliation are diffused over the mesh, meaning that the leaves are diffused over the surface. Here, with a discrete Whitehead move, a singularity can be split into two singularities that are connected by a leaf, and vice versa, as illustrated in Fig. 2(d)(e).

Improving the algorithm of [12]. The algorithm of [12] may produce poles at unintended positions. A triangle t_{ijk} is said to be uncovered if $w_{ij} = w_{jk} = w_{ki} = 0$, and the connected uncovered triangles form an *uncovered region*. If an uncovered region forms a topological disk, then a pole may be created inside this region [12]. To avoid poles for conformal parameterization, Zhao et al. [?] proposed an improvement to prevent uncovered regions from forming topological disks, as discussed below. A Whitehead move is defined as a *critical move* if it turns any neighboring vertex from a pole to a regular vertex [?]. Considering that a critical move will potentially cause some uncovered regions to form a topological disk, Zhao et al. [?] divide the optimization algorithm of [12] into multiple stages. In the first stage, critical moves are prevented until any vertex of the uncovered triangles causes a critical move when it is processed. Then, critical moves are applied to these vertices, and Zhao et al. [?] prove that no pole can be created at this stage. In this way, the enhanced algorithm can generate pole-free harmonic measured foliations.

4 Using Loops To Capture Surface Components

A Reeb graph is constructed by detecting the surface components and the neighboring relationships between them. As discussed in Section 1, we use the methods of [?], [12] to generate the harmonic measured foliation and then distinguish surface components using the saddle leaves of the foliation.

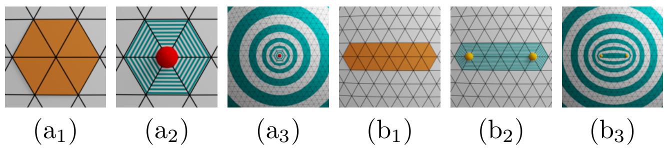


Fig. 3. Our pole loops (in orange) around a point (a_1) or a line segment (b_1) are generated, and then used for foliation initialization (white-teal stripes) $(\mathrm{a}_2, \mathrm{b}_2)$, by which the corresponding harmonic foliations are generated $(\mathrm{a}_3, \mathrm{b}_3)$.

In Section 4.1, we first demonstrate that our method can concisely capture the surface components, and stably represent their neighboring relationships as long as the loops for foliation generation have their homotopy types determined. Then, in Section 4.2, we introduce our measures to automatically generate loops to conveniently capture surface components.

4.1 Capturing Surface Components and the Neighboring Relationships Between Them

Capturing surface components concisely. As introduced in Section 3, the methods of [?], [12] generate foliations on the surface by Whitehead moves, which only deform the leaves isotopically, so the diffused leaves from a loop are homotopic to each other. As a result, with a nontrivial loop, which has been used in [?], [12], the diffused leaves from this loop will represent a cylinder region. Similarly, with a trivial loop around a vertex or line segment, the diffused leaves from this loop will represent a disk region, where a pole leaf will be generated at the vertex or line segment, as shown in Fig. 3. Such a loop is called a *pole loop*. Therefore, in our method, we use the nontrivial loops to capture cylinder components and use the pole loops to capture disk components. According to the discussion in Section 3.1, these surface components can be separated by the saddle leaves of the foliation.

To concisely represent surface components, we require that each surface component is represented by only a loop. Here, we regard each pole leaf as a puncture, so that the loops for representing different surface components should be of different homotopy types. Thus, the leaves from one loop must correspond to a unique surface component. Meanwhile, a surface component whose leaves are not homotopic to any loop cannot appear. As a result, the surface components always have a one-to-one correspondence with the input loops, as long as the loops are disjoint and not freely homotopic to each other. This means that we can *concisely* capture the surface components.

Representing neighboring relationships stably. In our method, several surface components are considered neighbors of each other if they are adjacent to the same saddle leaf. For a set of input loops, if we alter the initial foliation values of the loops or shift the positions of the loops homotopically, the shapes of the surface components would be different. Even so, their neighboring relationships will remain the same. Let us prove this by contradiction as follows.

Assume a surface component C and a saddle leaf S were originally adjacent (Fig. 4(a)), but are no longer adjacent when we change the loops as described in the above

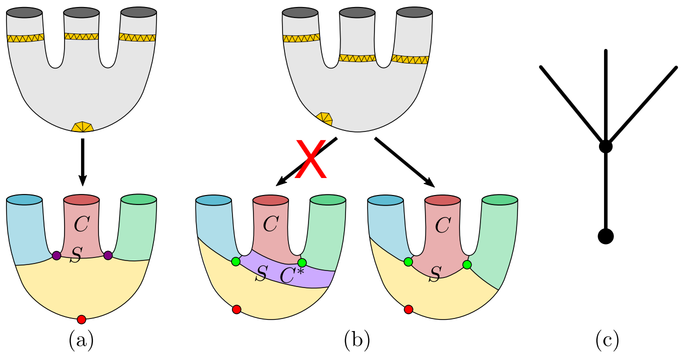


Fig. 4. Stably representing the neighboring relationships between surface components when the related loops are determined, as shown in (a) and the right one of (b), because the left one of (b) is impossible to occur. The obtained Reeb graph is in (c).

paragraph (Fig. 4(b, left)). Then, there must be some surface component C^* between C and S to separate them. Clearly, C^* should not be homotopic to any adjacent surface components; otherwise, they should be combined into one component. Therefore, the loop used to capture C^*, denoted as l^*, should not be homotopic to the loops to represent these adjacent surface components as well. From the original configurations, we know that C^* was not in between C and S before the loops are changed. In order for C^* to separate C and S after the loops are changed, the leaves diffused from l^* must overlap with some leaves of the loops that are originally adjacent to S. However, by the mechanism of Whitehead moves [12], the leaves diffused from different loops cannot overlap each other. Thus, there is a contradiction. As a result, such a surface component C^* cannot occur, and the neighboring relationship between C and S must be fixed as long as the loops do not change their homotopy types (Fig. 4(b, right)), which would result in the same Reeb graph (Fig. 4(c)).

4.2 Loop Determination

In our method, the loops can be manually prescribed by the user interactively. However, this will be troublesome in some cases, such as when handling complex or high-genus surfaces. Thus, we develop measures for automatically generating loops to effectively capture the surface components, as discussed in the following subsections.

4.2.1 Automatic loop generation

Pole loops are used to capture disk components, and the disk components are generally similar to protrusive parts of a shape. Thus, we can apply existing methods to detect feature points [43] and lines [44] on the surface, and then generate the pole loops around them, as discussed in Section 4.1.

Nontrivial loops are used to capture cylinder components, which inevitably represent the handles or tunnels of holes on high-genus surfaces. For this, we develop a measure to automatically generate nontrivial loops after the pole loops are defined. By investigating the leaf diffusion procedure, we can detect the topological changes of the diffused leaves that are caused by holes of the surface. Thus, we can generate new nontrivial loops to represent the surface component around the holes, as discussed in the following.

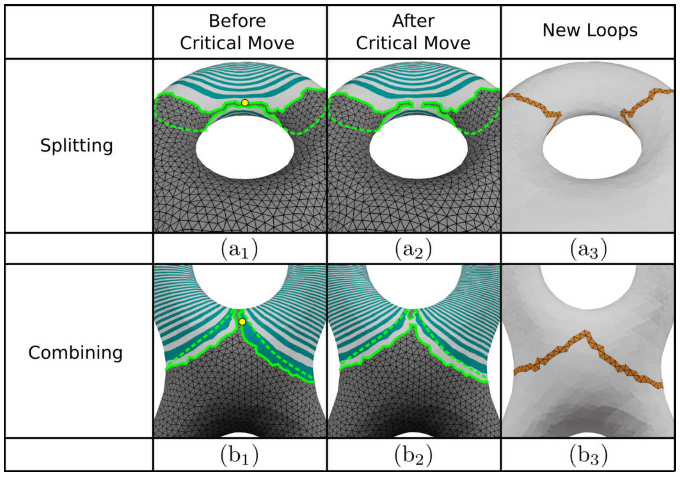


Fig. 5. Splitting $(a_1\text{-}a_3)$ and combining $(b_1\text{-}b_3)$ for generating loops adaptively. With a critical move applied to the yellow vertex on the boundaries (highlighted in green) of the regions covered by the leaves (a_1, b_1), the topology of the boundaries would change (a_2, b_2). By this, new loops (in orange) can be adaptively generated (a_3, b_3).

We observe that with the leaves diffusing from a loop, a critical move would occur when the leaves approach the hole, as illustrated in Fig. 5(a_1). This is because there are few mesh edges to be reached in the region around the hole when the diffused leaves arrive at the hole. Similarly, when two loops for two handles of a hole have their leaves diffused, there is also a critical move occurring when the leaves from the two loops are met during their diffusion, as illustrated in Fig. 5(b_1). Therefore, we can apply a critical move to detect topological changes for the diffused leaves. For the case in (a_1), with a critical move applied, the boundary around the region covered by the diffused leaves would be split into two boundaries, as illustrated in Fig. 5(a_2), and these two boundaries can be used to form two loops, as illustrated in Fig. 5(a_3). Similarly, for the case in (b_1), with a critical move applied, the two boundaries for the region covered by the leaves of the two loops would be combined into one boundary so that this boundary can be used to form a new loop. Correspondingly, we develop two measures for adaptively generating loops, as listed below.

- **Splitting.** The region covered by the diffused leaves from a loop has one boundary before a critical move (a_1) and two boundaries after the critical move (a_2). Thus, two new loops are formed (a_3).
- **Combining.** The regions covered by the diffused leaves from two loops have two boundaries before a critical move (b_1) and one boundary after the critical move (b_2). Thus, one new loop is formed (b_3).

In the above paragraph, we discuss how to apply a critical move for a splitting or combining operation to generate new loops, by which the neighboring relationships between three surface components can be represented. When more than three surface components are neighboring with each other, many critical moves can be found, and we apply them simultaneously to generate the corresponding new loops, e.g., a loop can be split into more than two loops, or more than two loops are combined into one. Thus, the neighboring relationships between more than three surface components can be well represented.

Using the above measures, we may produce superfluous loops that are homotopic to the existing loops. For this, we developed measures to remove them, as discussed in Section 4.2.2.

In our measures for automatic loop generation, combining loops in different orders may produce different newly generated loops, causing different Reeb graphs, as illustrated in Fig. 6. This is a limitation of our method and further study on how to define loops to extract the desired Reeb graphs is needed.

4.2.2 Generating Loops Concisely

In our method, we employ loops to represent surface components, so we try to avoid generating more loops than required for extracting concise Reeb graphs. For the newly formed loops, we will check whether they are superfluous and remove them if they are, as discussed in the following.

In our method, initially, the pole loops representing disk components are determined, so we will discard any newly formed loops that would correspond to a disk component. Considering that a disk component is homotopic to a point on the surface, we check whether the newly formed loop can be contracted to a point. This is done by using the method in [45], where the mesh and the loop are first cast to a system of quads, then the loop is iteratively shortened to check whether it can be reduced to a point. If so, the loop is removed.

In Section 4.2.1, we discuss how to generate loops around holes. As the loops are generated locally with diffusing leaves from the existing loops, we will check whether the newly formed loops are freely homotopic to an existing loop. This is also done by using the method in [45], where the mesh and the loop are first cast to a system of quads, and

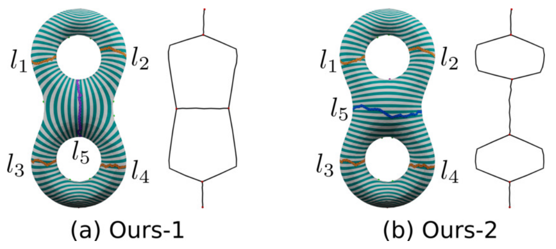


Fig. 6. The automatically generated loop l_5 may be different when the orders for combining existing loops are different. (a) l_1 and l_3 are combined earlier, and (b) l_1 and l_2 are combined earlier.

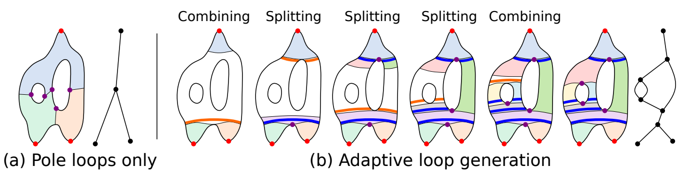


Fig. 7. Adaptively generating nontrivial loops for representing the holes on a high-genus surface. (a) The Reeb graph from pole loops alone cannot represent the holes of the surface. (b) We can adaptively generate new loops in orange lines and determine the usable ones in solid blue lines for extracting the Reeb graph to represent the holes.

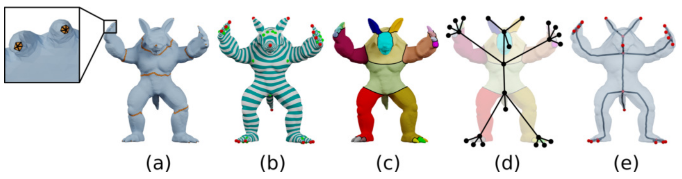


Fig. 8. The pipeline of our algorithm. (a) Loop input. (b) Harmonic measured foliation generation. (c) Detection of critical leaves (black lines) and surface components (colored regions). (d) Reeb graph extraction. (e) Reeb graph embedded in $\mathbb{R}^3$.

the loop is transformed to a canonical loop in its homotopic class. If two loops are transformed into the same canonical loop, they are freely homotopic to each other and one loop would be discarded. As a result, we can adaptively and concisely generate nontrivial loops to represent the handles of the holes.

With our measures to generate nontrivial loops, the Reeb graphs on high-genus surface can be conveniently extracted. As illustrated in Fig. 7(b), after the pole loops are determined, the nontrivial loops can be adaptively generated to represent the cylinder components, by which the extracted Reeb graph well represents the topological structure of the surface. Otherwise, the extracted Reeb graph can only represent the interrelationships between the disk components corresponding to pole loops, as shown in Fig. 7(a).

5 Algorithm

In this section, we present our algorithm for Reeb graph extraction on a triangular mesh for a closed manifold surface; the steps of the algorithm are illustrated in Fig. 8 and listed in Algorithm 1. To effectively implement these steps, we developed some novel measures, as discussed in the following subsections.

Algorithm 1 Reeb Graph Extraction Using Foliation Leaves.

```
Input: Mesh $M$, triangle loops $L$.
Output: Reeb graph $RG(N, A)$ ($N$ denotes nodes, $A$ denotes arcs).
1: Generate the foliation $F$ using $L$.                              ▷ Section 5.1
2: Find the critical leaves $\mathcal{L}_\mathcal{C}$ of $F$, and decompose $M$ into
   surface components $C$ using $\mathcal{L}_\mathcal{C}$.            ▷ Section 5.2
3: Construct the nodes $N$ and arcs $A$ of $RG(N, A)$ based
   on $\mathcal{L}_\mathcal{C}$ and $C$.                               ▷ Section 5.3
4: (optional) Embed $RG(N, A)$ in $\mathbb{R}^3$.                      ▷ Section 5.4
```

5.1 Improved Foliation Generation

As mentioned in Section 3.2, the algorithm of [?], [12] uses the Laplacian weights α_{ij} to generate the harmonic foliation. Unfortunately, their used weights are the cotangent weights $\alpha_{ij} = \frac{1}{2}(\cot\theta_p + \cot\theta_q)$ [42], as illustrated in Fig. 9(a), which may be negative on non-Delaunay meshes. Thus, the non-Delaunay meshes must be remeshed to be Delaunay as a preprocessing step in [?], [12]. As such, remeshing might change the underlying geometry or add many vertices to the original mesh [46], and so the algorithm of [?], [12] is prevented from handling non-Delaunay meshes well.

To solve this problem, we choose the *mean value weights* $\alpha_{ij} = (\tan(\psi_{ij}^1/2) + \tan(\psi_{ij}^2/2))/||e_{ij}||$ [47], as illustrated in Fig. 9(b). Since the angle ψ_{ij} of a triangle cannot be over 180 degrees, the mean value weights must be positive. As also discussed in [48], the mean value weights can be used to mimic the Laplace-Beltrami operator Δ. Although this approximation is not as good as that obtained by the cotangent weights, this only affects the geometry and smoothness of the leaves and does not affect the topology of the foliation, as illustrated in Fig. 10. Therefore, our improved foliation generation can handle non-Delaunay meshes well.

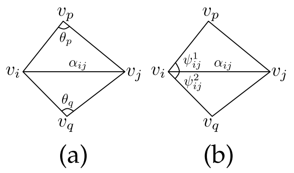


Fig. 9. Laplacian weight computation for foliation generation. (a) Cotangent weights. (b) Mean value weights.

The mean value weights are intrinsic to the mesh [47], so our algorithm is robust to models with nonrigid deformation. In addition, the mean value weights are always positive regardless of how the surface is triangulated, so our algorithm can effectively handle meshes with different resolutions and noisy meshes. The corresponding experimental results are provided in Section 6.1.

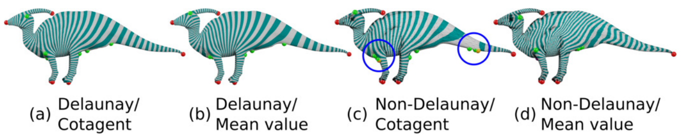


Fig. 10. Comparison between using cotangent weights and mean value weights for foliation generation. On the Delaunay mesh, the obtained foliation using cotangent weights (a) is comparable to that using mean value weights (b). On the non-Delaunay mesh, the obtained foliation using cotangent weights (c) has smoother leaves than that using mean value weights (d), but it has extra singularities (marked in yellow points in the blue circles), different from those in (a)(b)(d).

5.2 Extracting Critical Leaves and Surface Components

With the obtained foliation on the mesh, we first extract the critical leaves, and then segment the mesh along the critical leaves into disk and cylinder components, following the steps listed in Algorithm 2. The general idea is to convert the foliation into a differential one-form and then integrate it into a scalar function, whose isolines would be the leaves of our foliation, as discussed in Section 3.1. Here, we propose a novel measure in Step 4 to explicitly extract the leaves of the foliation. This differs from the methods of [?], [12] that implicitly encode leaves, which is enough for parameterization but not for extracting the Reeb graph. The other steps are generally covered in previous works, and we briefly discuss them here to be self-contained.

5.2.1 Locating Singularities

Here, we iterate over all the vertices and triangles of the mesh and check their indices to determine whether they are singularities, as described in Section 3.2. For integration and leaf tracing, all the singularities should be at the vertices of the mesh. For this, we follow the method of [?] to subdivide the saddle triangle into three subtriangles with three edges and one vertex inside. The foliation values on the new edges are set based on some closed-form formula [?], such that the new triangles are regular while the new vertex is a saddle. In this way, the saddle triangle is turned into a saddle vertex.

Algorithm 2 Extracting Surface Components.

```
Input: Mesh $M$, foliation $F$.
Output: Critical leaves $\mathcal{L}_\mathcal{C}$, surface components $C$.
1: Locate the singularities $Sing$ of $F$.                     ▷ Section 5.2.1
2: Cut $M$ into a topological disk $M_F$ with no singularity
   inside the disk.                                           ▷ Section 5.2.2
3: Integrate $F$ on $M_F$ into a scalar function $f$.
                                                              ▷ Section 5.2.3
4: Trace critical leaves $\mathcal{L}_\mathcal{C}$ of $F$ using $f$.             ▷ Section 5.2.4
5: Segment $M$ along $\mathcal{L}_\mathcal{C}$ into surface components $C$.
                                                              ▷ Section 5.2.5
```

5.2.2 Mesh Cutting

For the foliation on the surface mesh M to be integrable, M should be cut into a topological disk M_F with the singularities of F on its boundary. Here, we adopt two measures for handling genus-0 meshes and high-genus meshes, similar to [49]. For the genus-0 mesh, we simply connect all its singularities using a spanning tree and use the tree edges to cut the mesh, as shown in Fig. 11(a). The spanning tree is constructed using a union-find data structure with the singularities as seeds. For the high-genus mesh, we first use the homotopy basis of the mesh to cut it into a topological disk. The homotopy basis is computed by the greedy algorithm in [50], which is based on the classic tree-cotree decomposition [?]. Then, we build a spanning tree to connect the singularities to the boundary of the disk using the same measure for handling genus-0 meshes as above, as shown in Fig. 11(b). In this way, a closed mesh of any genus can be cut open to form a topological disk, and the singularities are on the boundary of the disk.

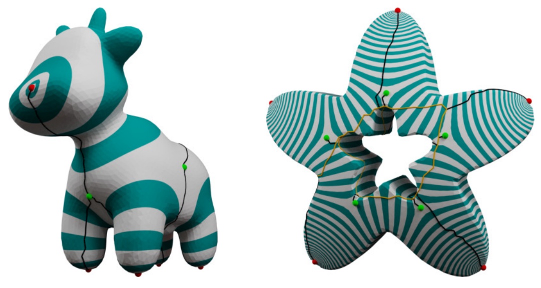
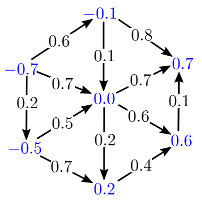


Fig. 11. Cut the mesh into a topological disk. (a) The genus-0 mesh is cut along the black edges that connect the singularities together. (b) The high genus mesh is first cut along the yellow edges which represent the homotopy basis, then cut along the black edges that connect the singularities to the homotopy basis.

5.2.3 Foliation Integration

In this step, we adopt the corresponding measures in [12]. As the foliation has no singularities inside M_F, its foliation values w_{ij} can be treated as an exact differential one-form on M_F. Thus, a piecewise-linear function $f : V \to \mathbb{R}$ can be obtained by integrating the one-form along the oriented edges.



If an edge e_{ij} is oriented from v_i to v_j, the function value at vertex v_j is computed as $f(v_j) = f(v_i) + w_{ij}$; otherwise, $f(v_j) = f(v_i) - w_{ij}$. We first arbitrarily assign an orientation to an arbitrary edge and then orient all the remaining edges with respect to the one-form using a breadth-first search on the triangles. Due to the closedness of the one-form, all the edges can be oriented consistently [12]. After that, we arbitrarily pick a vertex v_0 as the origin and set its function value $f(v_0) = 0$, and then integrate the function values on all the other vertices using a breadth-first search on the vertices. An example is shown in the inset, where the black values on the edges stand for the one-form on these edges, the arrows stand for the orientation for integration, and the blue values on the vertices indicate the scalar function after integration.

5.2.4 *Tracing Leaves*

As mentioned in Section 3, the leaves of a measured foliation on a disk chart U_i without singularities inside it correspond to the isolines of the induced scalar function. On the triangle mesh, the induced function is piecewise-linear, whose isoline is simply a polyline, which can be easily computed by tracing along the triangles that are crossed by the isoline, as described by many previous works, such as [28].

In our treatment, the mesh is cut by some edges to form a single disk chart M_F, and the cut edges form the boundary of M_F. Clearly, such an edge e_{ij} on M would correspond to two edges e^0_{ij}, e^1_{ij} on the boundary of M_F; thus, a leaf on the mesh may be separated into several isolines on the disk, and they should be glued to form a closed leaf on the original mesh M, as illustrated in Fig. 12. For this, we resort to the transition function φ of the measured foliation [41] to find the corresponding isolines of a leaf. The transition function is defined for each cut edge individually; for example, φ_{ij} for the cut edge e_{ij}. If the isoline on the triangle containing e^0_{ij} has function value y, its corresponding isoline on the triangle containing e^1_{ij} would have the function value $\varphi_{ij}(y)$, by which the corresponding isoline can be obtained for the gluing. We denote the vertices of e^0_{ij}, e^1_{ij} as $v^0_i, v^0_j, v^1_i, v^1_j$ and denote their function values as $y^0_i, y^0_j, y^1_i, y^1_j$, respectively. Then, φ_{ij} for the cut edge e_{ij} is defined as follows, which is by one of the two vertices of edge e_{ij}, and the vertex can be randomly selected to form the transition function because $w_{ij} = |y^0_i - y^0_j| = |y^1_i - y^1_j|$. If the edges e^0_{ij}, e^1_{ij} have opposite orientations with respect to the one-form,

$$\varphi_{ij}(y) = (y^1_i + y^0_i - y), \tag{2}$$

otherwise,

$$\varphi_{ij}(y) = (y^1_i - y^0_i + y). \tag{3}$$

As illustrated in Fig. 12, we obtain the green leaf of the foliation in (a) by gluing the green isoline on the left and the two green isolines on the right in (b). Let us assume that the green isoline on the left has function value y. When the isoline is traced to arrive at triangle t_{adb}, its corresponding isoline across triangle t_{abc}, which shares the cut edge with t_{abd}, would have the value $\varphi_{ab}(y)$. Thus, the green isoline in t_{abd} is further traced in t_{abc}.

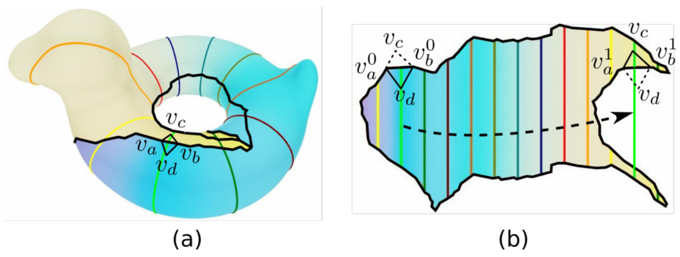


Fig. 12. For a mesh (a), it is cut open along the black edges into a topological disk (b). Then, a function is obtained by the generated foliation on the mesh to produce isolines on the disk, illustrated by some parallel lines in different colors in (b). At last, the isolines on the disk are glued to generate the leaves on the mesh, illustrated by the green closed line in (a).

By the measures discussed above, a critical leaf can be traced from a singular vertex. Note that the center pole is a standalone point and is taken as a critical leaf by itself.

5.2.5 *Extracting Surface Components*

In this step, we simply segment the mesh along the obtained saddle leaves to produce disjoint surface components, as illustrated in Fig. 8(c). Here, pole leaves are not used because they are inside disk components and are not necessary for surface segmentation.

5.3 Reeb Graph Extraction

With the extracted critical leaves and surface components, the nodes and arcs of the Reeb graph are obtained, by which the Reeb graph is constructed, as shown in Fig. 8(d). Here, we first form a Reeb graph node for each critical leaf, and then, for each surface component, we examine which two critical leaves are on its boundaries and add an arc between the nodes that represent these two critical leaves.

5.4 Embedding Reeb Graph in $\mathbb{R}^3$

By the measures in Section 5.3, using the critical leaves suffices to build the combinatorial structure of the Reeb graph. For other tasks such as visualization and skeleton animation, the Reeb graph needs to be embedded in the Euclidean space. Therefore, we use the centroids of each critical leaf as the Euclidean positions for the nodes of the Reeb graph. For the arcs, we sample some leaves uniformly within each surface component and use the centroids of these leaves as the Euclidean positions to form the arcs.

Since the integrated scalar function is discontinuous, the function values for a surface component may not be monotonic. Thus, to uniformly sample the leaves, we re-evaluate the integration in each surface component. For a disk component, we use the pole within it as the origin and integrate the foliation; while for the cylinder component, we first segment it to form a topological disk along a path that connects its two boundaries, and then use a vertex on the boundary as the origin and integrate the foliation. In this way, the function values in each surface component are in the range $[0, f_{max}]$, where 0 and f_{max} are achieved on the boundaries (pole) of the surface component. On each surface component, we uniformly sample some values f_i within the interval $(0, f_{max})$, trace the leaves corresponding to f_i, and use the centroids of the leaves to embed this arc. The final result is demonstrated in Fig. 8(e).

5.5 Analysis

5.5.1 Efficiency

In our paper, the mesh is implemented with a halfedge data structure. Using our method, most of the running time is spent on generating the harmonic measured foliation using [?], [12]. Zhao et al. [?] state that the running time of the algorithm ranges from a few seconds to a few minutes for meshes with fewer than 30,000 vertices. The other parts of our method take much less time than generating the harmonic foliation. For example, handling the meshes (all of which have 6869 vertices) in Fig. 15, the average time for foliation generation and Reeb graph extraction after foliation generation for a mesh are 210.77 seconds and 0.02 seconds, respectively. This is because except foliation generation, the other steps of our method are not time-consuming, as discussed in the following.

- Locating the singularities involves checking all the mesh vertices and triangles, and requires $O(n)$ time, where n is the size of the mesh.
- In the cutting phase, the greedy homotopy basis algorithm runs in $O(n \log n)$ time [50], while connecting the singularities runs in $O(n)$ time by the union-find data structure.
- The integration is performed by a breadth-first search algorithm, so its time complexity is $O(n)$.
- Leaf tracing requires $O(l)$ time to trace one leaf, where l is the number of triangles this leaf crosses, which is often much smaller than n in practice. As a result, it requires $O(kl)$ time to trace all the critical leaves and segment the mesh along them, where k is the number of critical leaves of the foliation, and it is often very small.
- Building the Reeb graph is trivial and requires $O(k + c)$ time, where c is the number of surface components.
- For embedding the Reeb graph in $\mathbb{R}^3$, the running time is $O(ml)$ where m is the number of additional leaves to trace.

In our current implementation, the algorithm runs on a single thread of the CPU. For further boosting the performance of the algorithm, several steps could be easily parallelized, including locating the singularities, tracing the leaves, and building the Reeb graph.

5.5.2 Handling Complex Singularities

For Reeb graph extraction, the critical points of the functions used with existing methods have similar effects as the singularities of foliations with our method for detecting the neighboring relationships between surface components. A critical point is non-degenerate if the Hessian of the used function is non-singular at that point. If all the critical points of a function have distinct function values, they are called simple. In general, degenerate critical points or non-simple critical points cannot be produced stably [25], and a fine tessellation of the mesh is always required for representing degenerate critical points, e.g., producing a vertex with a valence of at least six to represent a monkey saddle [31]. Thus, the existing methods generally require the critical points of the used function to be non-degenerate and simple; thus, they resort to a Morse function. As discussed in [14], the non-degenerate and simple critical points of a Morse function on a surface can only be the maximum, minimum, and Morse saddles, where the maximum and minimum correspond to the nodes of valence one, while a Morse saddle corresponds to a branching node of valence three [14]. Thus, the Reeb graph extracted via a Morse function can only represent the neighboring relationship between three surface components. This prevents existing methods from handling complex singularities, such as non-simple and degenerate saddles, which are related to the neighboring relationship between more than three surface components.

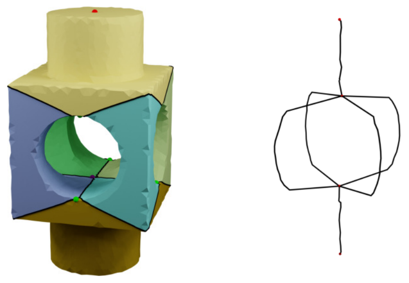


Fig. 13. We can conveniently extract the Reeb graph to represent the neighboring relationships between more than three surface components.

For handling complex singularities, some methods suggest grouping multiple critical points into critical areas [1], [32], so that a critical area containing degenerate or non-simple critical points can represent the neighboring relationships between more than three surface components. However, it is nontrivial to decide how to group critical points into a single critical area, and the resulting Reeb graph may differ greatly due to different decisions.

Fortunately, such difficulties can be naturally resolved using our method. As discussed in Section 4.1, the neighboring relationships between surface components are fixed when the loops are determined to represent surface components. This means the surface components correlated to a saddle leaf are fixed, and the arcs of the surface components are always connected to the node of the saddle leaf in the Reeb graph. As illustrated in Fig. 13, a node of the Reeb graph can have five arcs connected. In our implementation, our foliation is generated by Whitehead moves, which can turn a higher-order singularity into two lower-order singularities, while keeping the saddle leaf correlated to the same surface components. Here, the order of the singularity is determined by its index, e.g., a first-order saddle (tripod saddle in Fig. 1) has an index of -1, a second-order saddle (Morse saddle in Fig. 1) has an index of -2, and so on [?]. Thus, we only require the tessellation of the mesh to be able to represent first-order saddles, as high-order saddles can be Whitehead moved into multiple first-order saddles. This is generally satisfied by most meshes, as a vertex only needs three adjacent corners to be able to represent the first-order saddle, according to the equation for computing the index introduced in Section 3.2. As a result, we enhance the potential of handling complex singularities without having

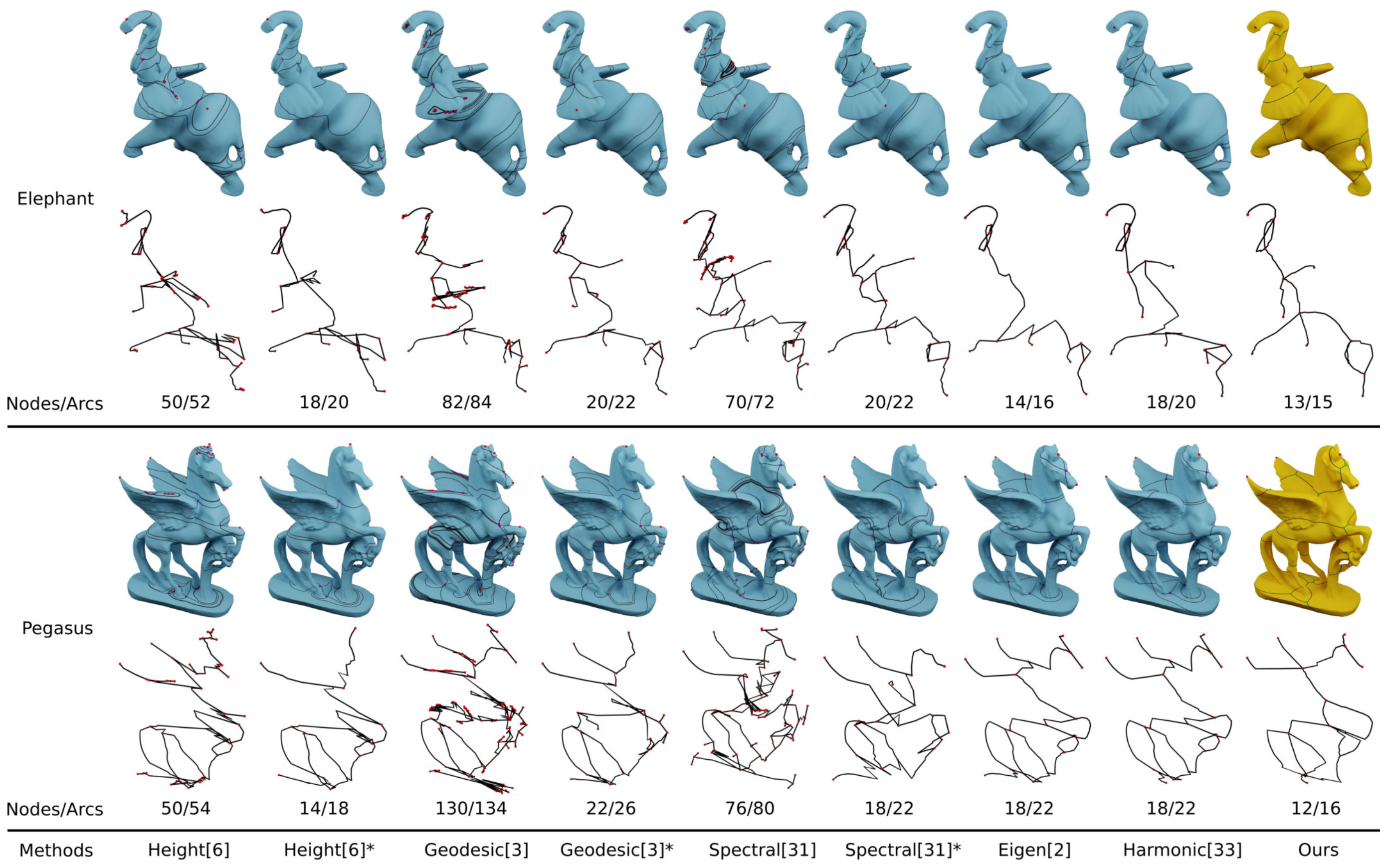


Fig. 14. Comparison of conciseness between the extracted Reeb graphs by our method and those by some functions, including the height function [6], geodesic distance [3], spectral distance [29], Laplace eigenfunction [2], and harmonic function [31]. The functions marked with * mean their detected critical points are post-processed by the method of [51] to simplify the extracted Reeb graphs. On the 1st and 3rd row, the isolines of the functions and our foliation leaves are represented in black curves, while the critical points of the functions and singularities of the foliations are marked in colored spheres. On the 2nd and 4th row, the corresponding extracted Reeb graphs by these methods are illustrated. Clearly, our Reeb graphs are more concise than the Reeb graphs produced by the compared methods.

a high requirement for tessellation of the mesh.

6 Results and Discussion

Many methods have been proposed for extracting Reeb graphs on surfaces. To show our superiority over the existing methods, we implemented our method and some exemplary isoline-based methods for comparison using C++14 and conducted the experiments on a personal computer installed with a 3.5 GHz 3950X CPU and the ArchLinux system. Here, we implemented the height function [6], the geodesic distance to the nearest feature points [3], the spectral distance to the nearest feature points [29], the first non-constant eigenfunction of the Laplace-Beltrami operator [2], and the harmonic function with manually prescribed Dirichlet boundary conditions [9] for generating isolines. Since some of these compared methods may generate too many critical points to extract concise Reeb graphs, we used a post-process of topological simplification to remove excessive critical points. With the critical points obtained, Reeb graphs were extracted. In this paper, we employed the implementations in TTK (0.9.9) [51] for topological simplification and Reeb graph extraction. When using the methods via the geodesic distance, spectral distance, and harmonic function, the users are required to set some points as function extrema; thus, we also set these points as our poles for foliation generation, for fair comparison.

In the experiments, we first compared all these methods for extracting concise Reeb graphs. Then, regarding the stable extraction of Reeb graphs, we mainly compared our method with the harmonic function [9] because it is free of extra critical points and generally superior to the other functions. Afterward, we demonstrate some potential benefits of our Reeb graphs for improving shape analysis.

6.1 Extracting Concise and Stable Reeb Graphs

From the resulting Reeb graphs and the related statistics about their nodes and arcs for the two tested models in Fig. 14, it is clear that we can extract more concise Reeb graphs than the other methods, even if the functions used to extract the Reeb graphs are simplified. Our Reeb graphs well represent the significant surface components, while this is challenging with some existing methods. Note that the Reeb graph extracted for Pegasus by the method [6] with a topological simplification misses the left wing in the graph, due to its over simplification.

To test the stability on extracting Reeb graphs, we performed several tests. First, we tested several human models in many poses, and compared the results with those of the harmonic function method [9]. As illustrated in Fig. 15,

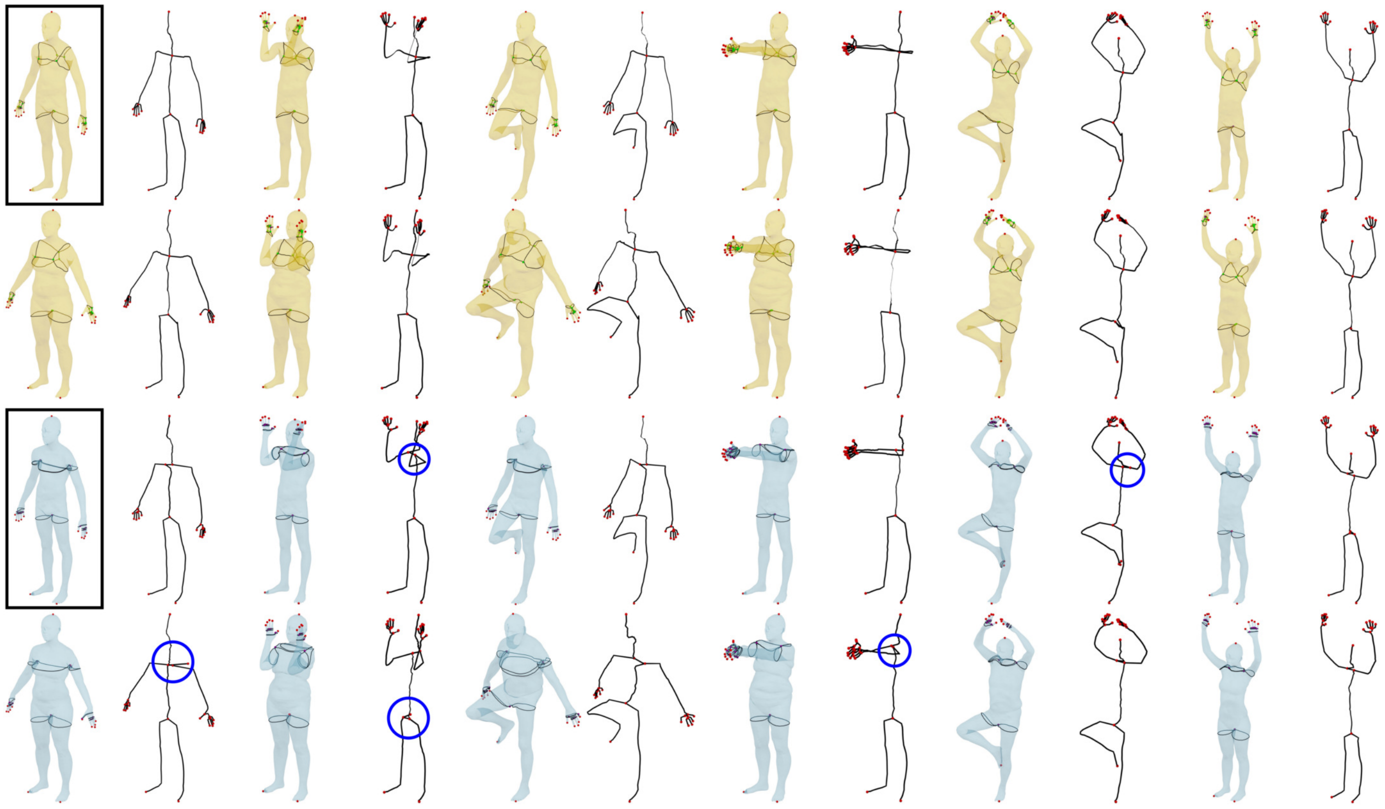


Fig. 15. Comparison of the extracted Reeb graphs for the similar shapes in different poses with our method (the upper 2 rows) and the harmonic function [9] (the lower 2 rows). Our method can extract consistent Reeb graphs with that of the reference shape (marked in a black box), while the harmonic function [9] cannot. Some different Reeb graphs from that for the reference shape are highlighted in blue circles.

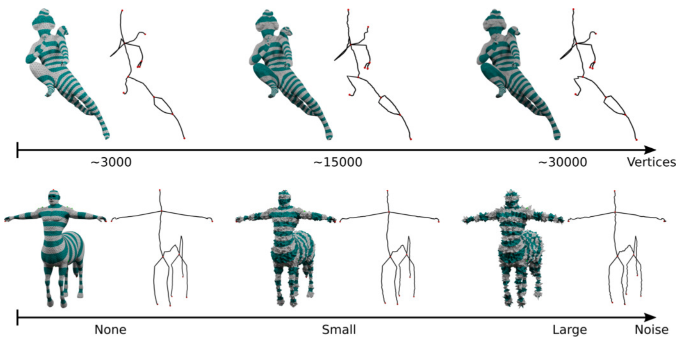


Fig. 16. Our algorithm can stably extract Reeb graphs on meshes in different resolutions (top) and noisy meshes (bottom).

our method can extract the same Reeb graph, while the results using the method of [9] are inconsistent for these human models, especially near the hands and the chest. Second, we tested our method to handle models at different resolutions or with/without noise. As illustrated in Fig. 16, our extracted Reeb graphs are consistent. This greatly benefits from foliation leaves being stably generated over the surfaces. All these results mean that we can robustly extract Reeb graphs that are superior to those of existing methods.

6.2 Enhanced Potential for Shape Representation

Our method employs loops to capture significant surface components for Reeb graph extraction. Here, the loops can be defined intuitively on the surface. Thus, we provide an easy way for the user to extract the Reeb graph as desired, by which the structure of the shape can be more effectively represented, such as more conveniently revealing the symmetries of the shapes, as well as extracting the same Reeb graph for the similar shapes by ignoring the small differences between them, e.g., small holes or small deformations. This enhances the potential of Reeb graphs to represent shapes, in comparison with existing methods, whose potential for representing shapes is limited, as discussed below.

Suitable shape representation for meeting the requirements of applications is always needed. However, this is not easy to achieve with existing methods, as the isolines on the surface are fixed when their used functions are determined, so the nodes and arcs of their extracted Reeb graphs are fixed. Thus, for extracting a Reeb graph with desired structure, the user needs to adjust the function, but this is not easy, as such adjustment is indirect. When using our method, the user can conveniently adjust the loops to configure the structure of the Reeb graph, as demonstrated in Fig. 1(e)(f).

Revealing symmetry is very important for shape analysis. With our method, the symmetries of the shape can be easily presented in the Reeb graphs by setting the loops on the symmetrical parts, as shown in Fig. 17(c), where we found the tips of the three handles to define three loops around them. For the existing methods, obtaining a suitable function for representing the symmetries in the Reeb graph is not an easy task, as illustrated in Fig. 17(a)(b). Using the harmonic function, it is difficult to adjust the

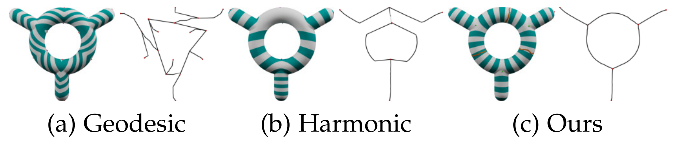


Fig. 17. The symmetries of this shape cannot be well represented in the extracted Reeb graphs using geodesic distance or harmonic function (a)(b), while this can be easily achieved with our method (c).

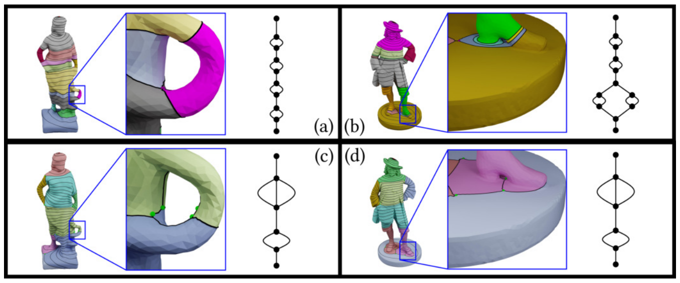


Fig. 18. The existing methods extract different Reeb graphs for the two similar models in (a)(b). But our method can extract similar Reeb graphs for similar shapes in (c)(d), as we can easily avoid generating loops for small holes.

boundary conditions to represent the symmetries. Using the geodesic distance with a topological simplification post-process, the extracted Reeb graph is complicated and does not clearly represent the symmetries, due to its sensitiveness to geometry.

Avoiding topological noise is expected for effective shape analysis, especially for high-genus surfaces. Theoretically, the Reeb graph of a Morse function on a closed orientable 2-manifold of genus g must have g loops in the graph [14]. As a result, Reeb graphs extracted from scalar functions cannot avoid any of the holes in meshes, which prevents them from stably representing meshes that are very similar to each other except for some trivial topological differences. For example, the only differences in the two human models shown in Fig. 18 are the small holes in them, including the wrinkle of the cloth in the left model, and the gaps between the shoes and grounds in the right model. With existing methods, the Reeb graphs for these two models are very different, as shown in Fig. 18(a)(b). To solve this problem, Doraiswamy and Natarajan [19] suggested simplifying the loops in the Reeb graph based on topological persistence. However, this simplification is performed after the Reeb graph is extracted, which does not change the surface components. As a result, it no longer has a one-to-one mapping between the surface components and the arcs of the simplified Reeb graph, preventing shape processing. In contrast, our method can extract the same Reeb graphs for such meshes to represent their prominent structures, as we can simply define no loop for small holes, as shown in Fig. 18(c)(d). In addition, our surface components are clean and simple, and there is a one-to-one mapping between the surface components and arcs of the Reeb graph, which is helpful for tasks like shape chartification and quadrangulation [8], [9].

If all holes of the mesh are ignored, our method can produce Reeb graphs in tree structures for high genus surfaces, as we can generate no loop for holes, as shown in Fig. 7(a). As we know, existing methods can only produce Reeb graphs in tree structures on simply connected domains, called the contour trees [25]. With our method, the contour trees can be extracted similarly as the existing methods, as shown in Fig. 8(d).

7 Conclusions

It is still a challenge for existing methods to extract stable and concise Reeb graphs, as the isolines of their used functions defined on the surface cannot suitably represent the surface components. In this paper, we address this challenge by using foliation leaves to extract Reeb graphs. With a method for foliation generation by determining loops on the surface to initialize the foliation, we develop a series of measures for extracting Reeb graphs, where loops are employed to represent the surface components. We demonstrate that when the homotopy types of the loops are determined, the surface components are fixed and the neighboring relationships between them are also fixed, so that the extracted Reeb graphs are compact and robust. Moreover, with our method, the potential for shape analysis by the Reeb graphs is enhanced, including allowing the user to freely prescribe loops to conveniently adjust the structure of the Reeb graph according to one's desire, easily representing the symmetries of the surface and ignoring the topological noise, while avoiding the post process of topological simplification. As a result, we can have Reeb graphs that more effectively encode the topology of the surface and are conveniently obtained.

There are some future issues for improving our work. First, our method relies greatly on foliation generation, while this is still time-consuming, as discussed in Section 5.5. It is noted that a fast foliation generation method was proposed recently [?]. We will study the integration of this method and our method soon. Second, using the automatic loop generation algorithm, different Reeb graphs may be generated for the same shape when the determined loops are different, as mentioned in Section 4.2. If the loops are not suitably generated, the extracted Reeb graph would not be the one desired. Thus, how to determine loops to extract suitable Reeb graphs to satisfy the applications' requirements need to be studied. Third, our currently extracted Reeb graphs are mainly for reflecting the neighboring relationships between surface components, whose geometric representation may be in low quality, which may prevent using these Reeb graphs for some applications, such as shape segmentation. Therefore, generating Reeb graphs with surface components with high quality geometric representations needs to be studied.

Acknowledgements

This work is partially supported by the National Natural Science Foundation of China under Grant 62072446. The authors really appreciate the editors and anonymous reviewers for their constructive comments and suggestions, and Prof. Jinsong Liu from Institute of Mathematics, AMSS, Chinese Academy of Sciences for the insightful discussion on quadratic differentials. The models used in this paper are credited to Myles et al. [52], Keenan Crane [53], the Three D Scans project [54], and the FAUST dataset [55].

References

[1] M. Hilaga, Y. Shinagawa, T. Komura, and T. L. Kunii, "Topology matching for fully automatic similarity estimation of 3D shapes," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, 2001, pp. 203–212.

[2] V. Barra and S. Biasotti, "3D shape retrieval using kernels on extended Reeb graphs," *Pattern Recognition*, vol. 46, no. 11, pp. 2985–2999, 2013.

[3] J. Tierny, J.-P. Vandeborre, and M. Daoudi, "Enhancing 3D mesh topological skeletons with discrete contour constrictions," *The Visual Computer*, vol. 24, no. 3, pp. 155–172, Mar. 2008.

[4] T. K. Dey, F. Fan, and Y. Wang, "An efficient computation of handle and tunnel loops via Reeb graphs," *ACM Transactions on Graphics*, vol. 32, no. 4, pp. 1–10, Jul. 2013.

[5] J. Tierny, J.-P. Vandeborre, and M. Daoudi, "Partial 3D shape retrieval by Reeb pattern unfolding," *Computer Graphics Forum*, vol. 28, no. 1, pp. 41–55, 2009.

[6] Y. Shinagawa, T. Kunii, and Y. Kergosien, "Surface coding based on Morse theory," *IEEE Computer Graphics and Applications*, vol. 11, no. 5, pp. 66–78, Sep. 1991.

[7] G. Aujay, F. Hétroy-Wheeler, F. Lazarus, and C. Depraz, "Harmonic skeleton for realistic character animation," *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 151–160, Aug. 2007.

[8] J. Tierny, J. Daniels II, L. G. Nonato, V. Pascucci, and C. T. Silva, "Interactive quadrangulation with Reeb atlases and connectivity textures," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 10, pp. 1650–1663, Oct. 2012.

[9] T. Sorgente, S. Biasotti, M. Livesu, and M. Spagnuolo, "Topology-driven shape chartification," *Computer Aided Geometric Design*, vol. 65, pp. 13–28, Oct. 2018.

[10] I. Nikolaev, *Foliations on Surfaces*, ser. Ergebnisse Der Mathematik Und Ihrer Grenzgebiete. 3. Folge / A Series of Modern Surveys in Mathematics. Berlin Heidelberg: Springer-Verlag, 2001.

[11] K. Strebel, *Quadratic Differentials*, ser. Ergebnisse Der Mathematik Und Ihrer Grenzgebiete. 3. Folge / A Series of Modern Surveys in Mathematics. Berlin Heidelberg: Springer-Verlag, 1984.

[12] D. R. Palmer, "Toward computing extremal quasiconformal maps via discrete harmonic measured foliations," Bachelor's Thesis, Harvard College, 2016.

[13] G. Patane, M. Spagnuolo, and B. Falcidieno, "A minimal contouring approach to the computation of the Reeb graph," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 4, pp. 583–595, Jul. 2009.

[14] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci, "Loops in Reeb graphs of 2-manifolds," *Discrete & Computational Geometry*, vol. 32, no. 2, pp. 231–244, Jul. 2004.

[15] C. Wang and J. Tao, "Graphs in scientific visualization: A survey," *Computer Graphics Forum*, vol. 36, no. 1, pp. 263–287, 2017.

[16] J. Tierny, A. Gyulassy, E. Simon, and V. Pascucci, "Loop surgery for volumetric meshes: Reeb graphs reduced to contour trees," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1177–1184, Nov. 2009.

[17] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas, "Robust on-line computation of Reeb graphs: Simplicity and speed," *ACM Transactions on Graphics*, vol. 26, no. 3, pp. 58–es, Jul. 2007.

[18] W. Harvey, Y. Wang, and R. Wenger, "A randomized $O(m \log m)$ time algorithm for computing Reeb graphs of arbitrary simplicial complexes," in *Proceedings of the Twenty-sixth Annual Symposium on Computational Geometry*, ser. SoCG '10. New York, NY, USA: Association for Computing Machinery, Jun. 2010, pp. 267–276.

[19] H. Doraiswamy and V. Natarajan, "Output-sensitive construction of Reeb graphs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, pp. 146–159, 2011.

[20] S. Parsa, "A deterministic $O(m \log m)$ time algorithm for the Reeb graph," *Discrete & Computational Geometry*, vol. 49, no. 4, pp. 864–878, 2013.

[21] H. Doraiswamy and V. Natarajan, "Computing Reeb graphs as a union of contour trees," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 2, pp. 249–262, Feb. 2013.

[22] H. Edelsbrunner, J. Harer, A. Mascarenhas, and V. Pascucci, "Time-varying Reeb graphs for continuous space-time data," in *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, 2004, pp. 366–372.

[23] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny, "Task-based augmented Reeb graphs with dynamic ST-trees," in *Eurographics Symposium on Parallel Graphics and Visualization*, Porto, Portugal, Jun. 2019, pp. 27–37.

[24] M. Hajij and P. Rosen, "An efficient data retrieval parallel Reeb graph algorithm," *Algorithms*, vol. 13, no. 10, p. 258, Oct. 2020.

[25] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno, "Reeb graphs for shape analysis and applications," *Theoretical Computer Science*, vol. 392, no. 1-3, pp. 5–22, Feb. 2008.

[26] S. Parsa, "Algorithms for the Reeb graph and related concepts," Ph.D. dissertation, Duke University, 2014.

[27] Y. Shinagawa and T. L. Kunii, "Constructing a Reeb graph automatically from cross sections," *IEEE Computer Graphics and Applications*, vol. 11, no. 6, pp. 44–51, Nov. 1991.

[28] G. Patanè and B. Falcidieno, "Computing smooth approximations of scalar functions with constraints," *Computers & Graphics*, vol. 33, no. 3, pp. 399–413, Jun. 2009.

[29] R. E. Khoury, J.-P. Vandeborre, and M. Daoudi, "3D-mesh Reeb graph computation using commute-time and diffusion distances," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 8290, pp. 157–166, Feb. 2012.

[30] M. Hajij, T. Dey, and X. Li, "Segmenting a surface mesh into pants using Morse theory," *Graphical Models*, vol. 88, pp. 12–21, Nov. 2016.

[31] X. Ni, M. Garland, and J. C. Hart, "Fair Morse functions for extracting the topological structure of a surface mesh," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 613–622, 2004.

[32] S. Biasotti, B. Falcidieno, and M. Spagnuolo, "Extended Reeb graphs for surface understanding and description," in *Discrete Geometry for Computer Imagery*, ser. Lecture Notes in Computer Science, G. Borgefors, I. Nyström, and G. S. di Baja, Eds. Berlin, Heidelberg: Springer, 2000, pp. 185–197.

[33] U. Bauer, C. Lange, and M. Wardetzky, "Optimal topological simplification of discrete functions on surfaces," *Discrete & Computational Geometry*, vol. 47, no. 2, pp. 347–377, Mar. 2012.

[34] J. Tierny and V. Pascucci, "Generalized topological simplification of scalar fields on surfaces," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2005–2013, Dec. 2012.

[35] J. Tu, M. Hajij, and P. Rosen, "Propagate and pair: A single-pass approach to critical point pairing in Reeb graphs," in *Advances in Visual Computing*, ser. Lecture Notes in Computer Science, G. Bebis, R. Boyle, B. Parvin, D. Koracin, D. Ushizima, S. Chai, S. Sueda, X. Lin, A. Lu, D. Thalmann, C. Wang, and P. Xu, Eds. Cham: Springer International Publishing, 2019, pp. 99–113.

[36] J. Lukasczyk, C. Garth, R. Maciejewski, and J. Tierny, "Localized topological simplification of scalar data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 572–582, 2020.

[37] J. Vekhter, J. Zhuo, L. F. G. Fandino, Q. Huang, and E. Vouga, "Weaving geodesic foliations," *ACM Transactions on Graphics*, vol. 38, no. 4, pp. 1–22, 2019.

[38] M. Campen, C. T. Silva, and D. Zorin, "Bijective maps from simplicial foliations," *ACM Transactions on Graphics*, vol. 35, no. 4, pp. 1–15, Jul. 2016.

[39] N. Lei, X. Zheng, J. Jiang, Y.-Y. Lin, and D. X. Gu, "Quadrilateral and hexahedral mesh generation based on surface foliation theory," *Computer Methods in Applied Mechanics and Engineering*, vol. 316, pp. 758–781, Apr. 2017.

[40] ——, "Quadrilateral and hexahedral mesh generation based on surface foliation theory II," *Computer Methods in Applied Mechanics and Engineering*, vol. 321, pp. 406–426, Apr. 2017.

[41] A. Fathi, F. Laudenbach, and V. Poénaru, *Thurston's Work on Surfaces (MN-48)*. USA: Princeton University Press, Apr. 2012.

[42] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr, "Discrete differential-geometry operators for triangulated 2-manifolds," in *Visualization and Mathematics III*. Springer, 2003, pp. 35–57.

[43] J. Sun, M. Ovsjanikov, and L. Guibas, "A concise and provably informative multi-scale signature based on heat diffusion," in *Proceedings of the Symposium on Geometry Processing*, vol. 28. Wiley Online Library, 2009, pp. 1383–1392.

[44] K. Hildebrandt, "Smooth feature lines on surface meshes," in *Proceedings of the Third Eurographics Symposium on Geometry Processing*, 2005, pp. 85–90.

[45] J. Erickson and K. Whittlesey, "Transforming curves on surfaces redux," in *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2013, pp. 1646–1655.

[46] R. Dyer, H. Zhang, and T. Möller, "Delaunay mesh construction," in *Proceedings of the Fifth Eurographics Symposium on Geometry

Processing, ser. SGP '07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, pp. 273–282.

[47] M. S. Floater, "Mean value coordinates," *Computer Aided Geometric Design*, vol. 20, pp. 19–27, 2003.

[48] M. Wardetzky, S. Mathur, F. Kälberer, and E. Grinspun, "Discrete Laplace operators: No free lunch," in *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, 2007, pp. 33–37.

[49] B. Springborn, P. Schröder, and U. Pinkall, "Conformal equivalence of triangle meshes," *ACM Transactions on Graphics*, pp. 1–11, 2008.

[50] J. Erickson and K. Whittlesey, "Greedy optimal homotopy and homology generators," in *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '05. USA: Society for Industrial and Applied Mathematics, Jan. 2005, pp. 1038–1046.

[51] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux, "The topology toolkit," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 832–842, Jan. 2018.

[52] A. Myles, N. Pietroni, and D. Zorin, "Robust field-aligned global parametrization," *ACM Transactions on Graphics*, vol. 33, no. 4, pp. 1–14, Jul. 2014.

[53] K. Crane, "Keenan Crane - 3D model repository," http://www.cs.cmu.edu/~kmcrane/Projects/ModelRepository/.

[54] "Three D Scans," https://threedscans.com/.

[55] F. Bogo, J. Romero, M. Loper, and M. J. Black, "FAUST: Dataset and evaluation for 3D mesh registration," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 3794–3801.

Shaodong Wang received his B.Sc. degree from the School of Resource and Environmental Sciences at Wuhan University. He is currently a Ph.D. student with the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Science. His research interests include geometry processing and shape analysis.

Wencheng Wang is a professor of the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, where he leads a research group on Computer Graphics and Image Processing. He received his PhD degree in software from Institute of Software, Chinese Academy of Sciences in 1998. His research interests include computational geometry, computer graphics, visualization, and image editing. He is a member of the IEEE and the ACM.

Hui Zhao is a computer graphics scientist. He was a visiting scholar in Harvard University from 2015 to 2016. He published five books on computer graphics. His research interests include discrete differential geometry, discrete conformal geometry, hyperbolic geometry, mesh parameterizations, mesh deformations, and mesh quadrangulations.