Flexible Fiber Surfaces: A Reeb-Free Approach

Daisuke Sakurai, Kenji Ono, Hamish Carr, Jorji Nonaka, and Tomohiro Kawanabe

Abstract The *fiber surface* generalizes the popular isosurface to multi-fields, so that pre-images can be visualized as surfaces. As with the isosurface, however, the fiber surface suffers from visual occlusion. We propose to avoid such occlusion by restricting the components to only the relevant ones with a new component-wise flexing algorithm. The approach, *flexible fiber surface*, generalizes the manipulation idea found in the *flexible isosurface* for the fiber surface. The flexible isosurface in the original form, however, relies on the contour tree. For the fiber surface, this corresponds to the Reeb space, which is challenging for both the computation and user interaction. We thus take a *Reeb-free* approach, in which one does not compute the Reeb space. Under this constraint, we generalize a few selected interactions in the flexible isosurface and discuss the implication of the restriction.

1 Introduction

An isosurface is defined as the inverse image $f^{-1}(s)$ of some scalar value s in the scalar field $f : M(n) \to \mathbb{R}$, where $M(n)$ is an n-manifold. Though visualizing isosurfaces [20] is a routine task for scalar field analysis, it is often required to understand correlations between multiple quantities (e.g. temperature and pressure). In fact,

Daisuke Sakurai
Zuse Institue Berlin, Germany, e-mail: d.sakurai@computer.org

Kenji Ono
Research Institute for Information Technology, Kyushu University, Fukuoka, Japan;
RIKEN Center for Computational Science, Kobe, Japan

Hamish Carr
School of Computing, University of Leeds, Leeds, UK

Jorji Nonaka · Tomohiro Kawanabe
RIKEN Center for Computational Science, Kobe, Japan

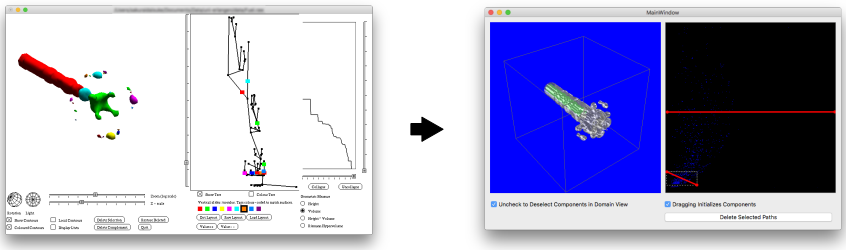

Fig. 1 Our flexible fiber surface generalizes the flexible isosurface to multi-fields without pre-computing the topology of pre-images. The domain is shown on the left and the range on the right.

scalar data analysis itself can be assisted by multi-field analysis when considering the gradient magnitude as the second field [17].

For 3-D multi-fields of the form $f : M(3) \to \mathbb{R}^m$, the range is extended to data tuples $(x_1, .., x_m) \in \mathbb{R}^m$. In particular, the *fiber surface* [5] generalizes the isosurface for $f : M(3) \to \mathbb{R}^2$, which is useful for extracting pre-images in the domain as a surface mesh. The fiber surface is the pre-image $f^{-1}(P)$ of a *control polygon* P, which is a polyline composed by l linear segments each having the endpoints (x_i, y_i) and (x_{i+1}, y_{i+1}) $(1 \leq i \leq l+1;\ i, l \in \mathbb{N})$. Each vertex (x_i, y_i) is called a *control point*. The user can specify the fiber surface by drawing the control polygon in the 2D range. The control point shall be dragged to see how the shape of the fiber surface changes. This allows fiber surfaces to be simple to compute, compact in size, and quantitative as isosurfaces in scalar fields. Since the fiber of a point in an $\mathbb{R}^2$ range is a 1-D structure almost everywhere in the 3-D domain, the fiber surfaces are indeed surfaces in general. We call a connected component of a fiber surface a *fiber surface component*. (A *component* refers to a fiber surface component unless specified otherwise.) However, interacting with the components remains still a challenging task. As with the isosurface, the fiber surface suffers from overlaps and may have too many features for the user to comprehend. In case of the isosurface, one has been able to utilize the *flexible isosurface* [8] to distinguish the components, hide irrelevant ones such as noise or uninteresting phenomena, and even vary the isovalue per component. Such interactions were shown to be useful for visualizing different objects in a CT scan dataset separately, for example.

We thus adapt the component-wise manipulation of the isovalue, as found in the flexible isosurface for multi-fields, as the *flexible fiber surface* (Fig. 1). Where an isosurface component is contracted as a point in the *contour tree* [7], a component of a fiber $f^{-1}(p)$ of some value $p = (x, y)$ is contracted as a point in the *Reeb space* [13]. Hence, each component of a fiber surface $f^{-1}(P)$ finds its contraction in the Reeb space as a connected component.

We generalize the flexible isosurface, or to be more precise, its concept of component-wise manipulation. Our flexible isosurface interface, as implemented [8], pre-computes the contour tree in order to obtain the seed of isosurface components and the connectivity of these components across different isovalues. While the

contour tree is generalized into the Reeb space for multi-fields, in this work we avoid computating the Reeb space (we call such an approach to be *Reeb-free*), and for the following reasons. Firstly, the scalability of analysis is limited by the scale at which the Reeb space can be computed. Secondly, the current standard implementation [28, 29] of the Reeb space computation requires the user to subdivide the domain to a sufficiently fine resolution such that at least one tetrahedron is completely contained in each 3-sheet (which is challenging to achieve). Finally, the Reeb space is hard to be shown to, and to be utilized by, the user when the Reeb space is complicated [25], which is usual in real-world data. Interactions relying on the contour tree are not generalized in our approach (3 and 6) as we avoid obtaining the Reeb space.

Our contributions include (i) varying the input control polygon per fiber surface component, (ii) freeing the operations of the flexible fiber surface from precomputing the Reeb space. For contribution (i), we devise a new flexing algorithm. Contribution (ii) means that the flexible iso-surface also becomes Reeb-free as a special case. Our algorithm works for any 3-D tetrahedral grid, regardless of the homology of the domain. Our key idea is to follow the pre-image on-demand.

The remainder of this article is organized as follows. We introduce the related work in Section 2, and generalize the semantics of the original flexible isosurface to our Reeb-free flexible fiber surface in Section 3. We then revisit the existing algorithms of extracting a fiber surface in Section 4 with their implication to our computation. Section 4 explains how our algorithm identifies the connected components of fiber surfaces extracted with the algorithm by Klacansky et al., and further deform the fiber surface. Section 5 demonstrates the outcomes of our proof-of-concept implementation. Section 6 discusses the indication of generalizing flexible-isosurface to multi-fields in a Reeb-free manner including the limitations. Finally, Section 7 gives the conclusion and future work.

2 Related Work

The flexible fiber surface extends topological operations that are defined for isosurface. We thus introduce relevant work in the topological analysis for scalar fields and multi-fields.

2.1 Work in Scalar Field Analysis

Isosurface An isosurface can be visualized effectively with *marching cubes* [20]. Its basic idea is to rotate pre-defined cubes with triangular patches inside to reconstruct the isosurface. Since the original marching cubes algorithm had ambiguities of its topology, topological algorithms often tessellate the cubes into tetrahedra and apply *marching tetrahedra* [3]. In contrast to the marching cubes, the *continua-*

tion method [31, 14] starts from seeds and proceeds to adjacent cells. This enables component-wise tracking of a pre-image, on which our algorithm is based.

Topological Analysis The Reeb graph [24] is a quotient space defined by contracting each of the connected components of isosurfaces to a point. For simple domains, their Reeb graph is guaranteed to be a tree – hence it has the special name *contour tree*. Reeb graphs and contour trees are useful for displaying the global configuration of the connected components of isosurfaces [2], simplifying the data [10, 8], indexing isosurfaces [16], extracting the topological changes in an isosurface [22, 26] and designing transfer functions [26, 30]. The standard algorithm for the contour tree [7] uses as input the isosurface topology along the edges of the simplicial mesh. On the other hand, van Kreveld et al. [19] tracked the pre-image explicitly, which is in fact the approach of our pre-image tracking as well. More generally, the Reeb graph must be computed [23] instead.

Parallelization of these algorithms has been, and is still, a challenge [22, 11] especially for distributed systems [9]. The dependency to pre-computed topology thus restricts the scalability of component-wise manipulations we aim to achieve. The manipulation, however, is in fact independent of the pre-computation as we show with our Reeb-free approach.

Interface One of the attributes of the contour tree is its ability to provide seeds for continuation. This forms the basis for the flexible isosurface [8]. This assumes that features are represented with connected components of isosurfaces, and lets the users analyze them while avoiding irrelevant surface components.

The original flexible isosurface interface (Fig. 1) consists of two views, one for the domain and another for the range. The interface shows the connected components in the domain. The components are assigned distinct colors so that the user can visually identify the connected components. These colors are simultaneously projected in the range view as color labels in order to indicate the component's isovalue. The user can see the pre-computed contour tree optionally in the 2D view on the right. This view shows the 1D range with an auxiliary dimension so that one can see the tree structure. In fact, labels are overlayed at the contraction points in the tree to indicate the position of the components. Though the contour tree was mandatory in this particular work, this was not necessary for data exploration as the user still could explore the global context by seeing the cumulative distribution.

We summarize several characteristic operations of the flexible isosurface below:

- Initialization: the user can *initialize* a flexible isosurface, i.e. show the entire pre-image or the *largest contour segmentations* [21].
- Selection: the user *selects* the components of interest in order to apply further operations. The user clicks on the components or on their labels mapped to the contour tree. Components can also be *deselected* with the mouse.
- Evolution: the user varies the isovalue for a subset of visible components by dragging the corresponding projection in the contour tree or all at once by manipulating the isovalue slider.
- Deletion: the user can *delete* selected components. Those are components such as artifacts and objects irrelevant to the analysis.

- Addition: the user *adds* a hidden surface component to the domain by clicking on its point contraction in the contour tree.
- Simplification: if the abundance of surface components makes navigating in the contour tree difficult, the user can *simplify* (i.e. hide) or *unsimplify* (show) the isosurface by thresholding surface statistics. This is achieved by cutting away or putting back the arcs of the contour tree, respectively.

2.2 Work in Multi-Fields Analysis

In multi-fields, the isosurface in scalar fields we have seen above generalizes as the fiber surface.

Fiber Surface Carr et al. [5] approximated the fiber surface as the isosurface of a scalar field, where the scalar value was the distance to the control polygon in the range. Later, Klacansky et al. [18] proposed an algorithm to compute the fiber surface without this approximation. We use the latter algorithm when extracting the fiber surface.

Topological Analysis Edelsbrunner et al. [13] generalized the Reeb graph to multi-fields as the Reeb space by contracting each connected component of the fiber f^{-1} to a point. The connectivity of connected components was not computed in their algorithm. This was later achieved by quantizing the range [4] into rectangular regions and connecting their pre-image in the domain. This approach found application to visualizing nuclear scission [12] and fiber topology [25].

Eventually, Tierny and Carr [28] computed the Reeb space without the quantization. The algorithm partitions the domain with *singular fibers*, at which topological events (such as mergers, splits, birth and death of the pre-image) happen. They estimate the steps of the algorithm to be $O(n_j \times n_T)$. n_j is the number of tetrahedral edges E; n_T is the number of tetrahedra. Though some optimization is possible [28], the fact remains that irrelevant topological events in the data may heavily increase the running time. As with the topological analysis for scalar fields, this can restrict the scalability of the analysis. Our *Reeb-free* computation, on the other hand scales linearly with the size of the feature that is actually of interest to the user. Last but not least, Pareto optimality gives an alternative generalization of scalar topology [15].

3 Generalizing the Semantics

Our flexible fiber surface generalizes the iso-surface evolution to multi-fields. The advantages and disadvantages of our generalization will be discussed in Section 6.

3.1 Generalizing the Interface

Fig. 1 shows our fiber surfacing interface on the right. In the interface, the domain view on the left shows the fiber surface as the flexible isosurface interface does for the isosurface. In contrast, our range view on the right replaces the topological information with a scatterplot to navigate the user in the range. The prototype focuses on proving the concept.

3.2 Generalizing the Component-Wise Operations

Initialization When the user *initializes* the fiber surface of a control polygon, the domain view shows the user all the surface components to understand their distribution in the domain.

Selection and Deselection (De)selection can be achieved in the domain view. In contrast, the component-wise selection and deselection cannot be done in the range, since all the components of a fiber surface overlap in the view.

Evolution The user reshapes the control polygon for selected group of components by dragging the control points.

Deletion Selected components can be removed from the domain view.

Addition and Simplification As we free the flexible fiber surface from the Reeb space, these operations become infeasible. Indeed, addition requires displaying the components to the user, which is the Reeb space itself. Simplification, too, requires access to the global topology.

4 The Algorithm

We assume a tetrahedral grid and the barycentric interpolation. First we extract the fiber surface using the algorithm by Klacansky et al. [18]. In each segment of the control polygon, this algorithm first extracts a *base fiber surface*, which is the pre-image of the line that covers the segment (see Fig. 2). The surface is the set of triangular patches obtained by applying the marching tetrahedra to the scalar field defined as the signed distance to the line. The pre-image of the complement of the segment is then *clipped*, i.e. cut away.

Next, we identify components, and the user selects interesting ones. The user shall then move the control point to specify the *target control polygon*. During the process, the flexing algorithm proposed in this article tracks the movement of the fiber surface components individually by tracking the *active tetrahedra*. (They cover the surface components being deformed (Alg. 1).)

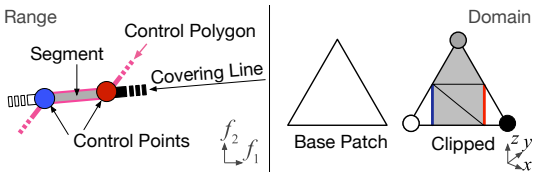

Fig. 2 Fiber surface extraction [18]. The white / grey / black color indicates the position in the covering line. A base fiber surface patch is clipped at the pre-image (blue & red) of control points.

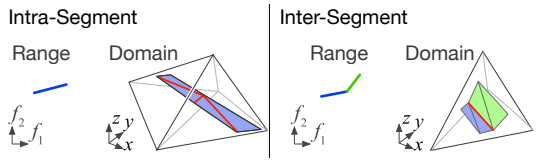

Fig. 3 Intra-segment connections and inter-segment connections (both in red) between clipped patches induced by the control polygons in the range.

4.1 Identifying the Connected Components

Once the surface has been approximated, we can check its connectivity with the union-find algorithm [27]. The elements of union-find are the points in the mesh, and we connect the pairs of such points if they lie in adjacent patches. In the traditional marching tetrahedra, a point shared by adjacent mesh triangles resides in the same tetrahedral edge. Identifying the points is solved by identifying the edge [3] since no two different points reside in a single edge. However, a vertex may not lie in the edge for our fiber surface computation since the patches are clipped (Fig. 2).

A naive approach is to glue the triangular patch corner points when they have close coordinate values. However, a fiber surface can have intersecting patches, and thus two points in disconnected patches can share close coordinates. We instead find the connection between a base fiber's patches and clipped patch corners separately.

4.1.1 Patch Corners in Base Fiber Surface

To compute the location of corners in the base fiber surface is to extract the isosurface of the signed distance to a control polygon segment. The connectivity between the points can be obtained by recording the point ID at the tetrahedral edge. As a tetrahedral edge intersects with a base fiber surface only once at most, we keep record of the point IDs for each control polygon segment separately. By using these point IDs as the elements of the union-find data structure, we connect the patch corners in a base fiber surface as long as they are actually connected to each other.

4.1.2 Patch Corners Due to Clipping

As we can see in Fig. 3, two connected patches can belong to the same segment (*intra-segment* connection) or two neighboring ones (*inter-segment* connection).

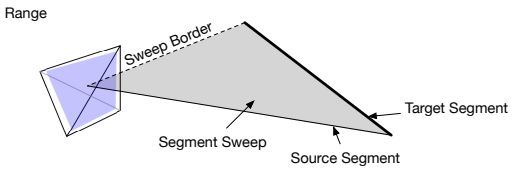

Fig. 4 When the user drags a control point, the segments sweep the range, starting from the source position and ending at the target position. Our algorithm walks through the *active tetrahedra*, which contain the pre-image of a moving segment.

Finding Intra-Segment Connections Fig. 3 shows that an intra-segment connection can have a connection of corner points inside a single tetrahedron and across two tetrahedra adjacent to each other. To connect the points in the former manner, our algorithm joins two points with the union-find if they are from the same base fiber surface and inside the same tetrahedron. In addition, two points that are from the same base fiber surface and clipped by the identical control point are given the same point ID when they touch each other at a tetrahedral face. To do so, we record the point ID at the face for each control point separately just as we did for base fiber surfaces in Section 4.1.1. Notice that clipped patch corners are given a unique ID only when touching a tetrahedral face.

Finding Inter-Segment Connections Two clipped patches which are pre-image of different but adjacent segments belong to the same connected component as long as they are inside the same tetrahedron and were clipped due to the same control point. We loop through the segments, clip the patches, and connect every neighboring pairs. If the control polygon is closed, i.e. $(x_1, y_1) = (x_{l+1}, y_{l+1})$, we clip the first segment's patches but defer connecting them until the end of the loop.

4.2 Following the Connected Components

While the user drags a control point, the control polygon sweeps the range. As illustrated in Fig. 4, this starts from the *source segments* and ends at the *target segments*. We model the control point to move along the *sweep border*, which is a line segment connecting the start and target positions. We follow the surface components being continuously deformed in the domain. If we observe a moving component from inside a tetrahedron, the component starts its motion from the original location and moves towards its destination. As the sweep proceeds, the pre-image penetrates into adjacent tetrahedra, and moves out from our example tetrahedron if the destination is outside.

We detail this procedure in Alg. 1. We start by gathering the *active tetrahedra*, i.e. the tetrahedra holding the user-selected fiber surface patches. We pair each tetrahedron with the segments that define the component, and put all such pairs in a *queue*. If a tetrahedron overlaps with multiple segments, every segment gets its own pair.

Algorithm 1 Follow the deformation of surface components

Input: Source control polygon P_s, target control polygon P_t, Pairs (segment, active tetrahedron) $tets_s$ of P_s
Output: Pairs (segment, active tetrahedron) $tets_t$ of P_t

```
 1: queue ← tets_s
 2: while queue is not empty do
 3:     pop (seg, tet) from queue
 4:     if (seg,tet) is visited then
 5:         continue
 6:     end if
 7:     mark (seg, tet) as visited
 8:     if seg is not dragged then
 9:         continue
10:     end if
11:     if (seg, tet) has base fiber surface of target segment then
12:         put (seg, tet) in tets_t
13:     end if
14:     for tetrahedron tet_a adjacent to tet do
15:         if (seg, tet_a) is not visited and shared_face(tet,tet_a) intersects segment sweep in range
            then
16:             put (seg, tet_a) in queue
17:         end if
18:     end for
19:     for segment seg_n neighboring seg do
20:         if control point between seg and seg_n moved and (seg_n, tet) is not visited and tet inter-
            sects sweep border in range then
21:             put (seg_n, tet) in queue
22:         end if
23:     end for
24: end while
```

We pop a pair (seg, tet) from *queue* and operate on it (lines 3–13). In order to avoid processing the same pair twice, we mark the pair as visited. This visit flag is implemented as an array of tetrahedron IDs for each segment. If the points of tet have both positive and negative distance to the *seg*, tet may intersect with a surface component of *seg*. The pair then joins the output pairs P_t.

As we have checked the evolution inside tet for *seg*, we push adjacent tetrahedra tet_a's in *queue* for visiting it later as long as the component continues tet_a (lines 14–18). This continuation happens when tet and tet_a's touching face intersects *seg* in the range.

Finally, we pass tet to its neighbors seg_n's (lines 19–23). If a control point between *seg* and seg_n does not move during a drag, the evolution of component is independent of seg_n. We do not process such seg_n (line 20). Otherwise, we check whether the component of (seg,tet) continues to seg_n (line 20). If so, we put (seg_n,tet) in *queue*.

After we visited all the (seg,tet) pairs, we extract the fiber surface components in them using the method by Klacakanski et al.

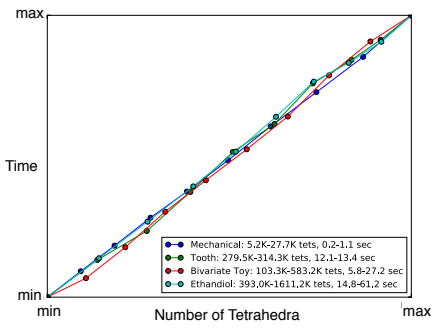

Fig. 5 The duration of dragging scale linearly with the number of tetrahedra we process. The examples share the same control polygon: it is defined as the diagonal line from the point $(0,0)$ to $(1,1)$ in the normalized range, and is dragged towards $(0.6,0.4)$ to produce 10 samples evenly along the trace.

5 Outcomes

We build a proof-of-concept interface with C++. We use VTK for the data structure and Qt for the GUI. The interface lets us display a pre-computed scatterplot or continuous scatterplot [1]. We selected a few typical, simple, datasets to evaluate our Reeb-free approach. We show that one can in fact achieve component-wise flexing of fiber surfaces. Our serial implementation is run on a PC with Intel Xeon CPU (3.20 GHz, 20 MB Cache) and 64 GB RAM. Each dragging completes in several seconds.

5.1 The Analysis of the Algorithm

As expected, the computation time scales linearly with the number of active tetrahedra the algorithm, Alg. 1, visits (Fig. 5). This demonstrates the fact that the size of the features the user is interested determines the response time. If the tetrahedra are distributed evenly in the range, the number of tetrahedra being swept shall scale linearly with the distance the dragged control point moves away. We can see this in the plot as the near-constant distance between two neighboring points of each line.

5.2 A Simple Proof of Concept: the Tooth Dataset

The tooth dataset (Fig. 6) gives a simple proof of concept for our flexible fiber surface. We subdivided each cube of the input regular grid into 6 tetrahedra with the Freudenthal tessellation (see [6]), so that the tetrahedral faces are consistent across neighboring cubes.

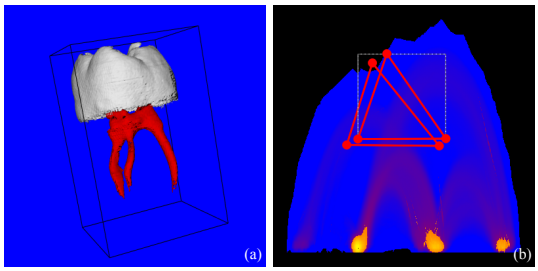

Fig. 6 Tooth dataset. The two fields are CT value and its gradient magnitude. (a) The domain and (b) the range. We start by drawing two control polygons that contain either only the crown or root. We then move them into a region that contains both. The evolution of the crown in white (or root in red) is restricted to the crown (root).

We report that the features are simple to identify without the Reeb space since we can identify the boundary of objects as hyperbolic curves [17], and their overlaps in the range resolve in the high gradient regions.

5.3 Comparison with the Flexible Isosurface

We take some small head CT dataset with the resolution of $50 \times 50 \times 50$ for comparing our results with the flexible isosurface. In Fig. 7, we have visualized the dataset with the flexible isosurface and with our Reeb-free interface. Due to the overlap of features in the range, the interaction with the domain is essential for our Reeb-free interface. The simplified contour tree is a significant advantage of the original flexible isosurface interface since it gives hints to an experienced user about the surface component evolution.

6 Discussion

We now discuss the consequence of our generalization of flexible isosurface to Reeb-free multi-field analysis.

Analysis of the Algorithm The number of steps required for tracking a component is $O(n_T)$, where n_T is the number of tetrahedra to be visited in our method. n_T shall be close to the number of tetrahedra necessary to extract the fiber surface partitioning the domain [25] with the Reeb space extraction [28]. Though our implementation is serial, the approach can apparently extend itself to distributed systems by locally running Alg. 1 in each node with occasional communications between different nodes. Though this requires further research, it should be more feasible than computing the Reeb graph of such systems.

Evolution The evolution of fiber surface components lets us understand how multivariate values distribute, and especially how the features continue in the domain.

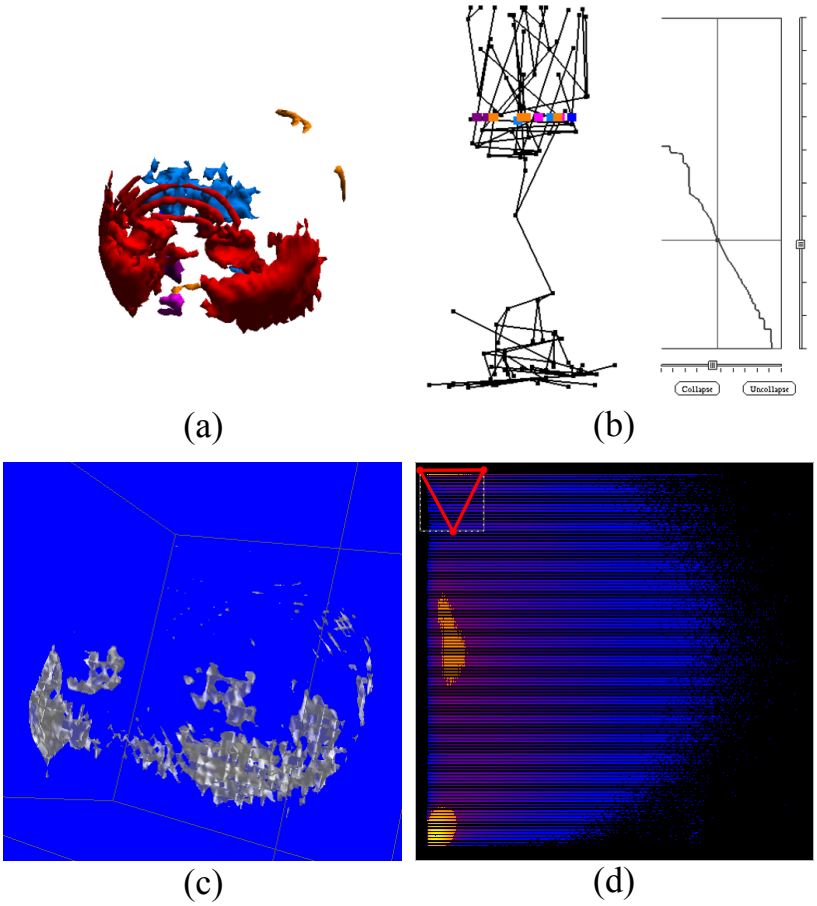

Fig. 7 Comparing the original flexible isosurface (a) (b) and our Reeb-free flexible fiber surface (c) (d) under severe overlaps of features in the range.

Simplification We did not simplify the topology of connected components although this was available in the flexible isosurface concept by Carr et al. thanks to the contour tree.

Global Exploration We show a scatterplot to provide the user with a global context to the analysis. The cruciality of this lack depends on the dataset to be analyzed. Datasets with similar objects tend to suffer because their image overlap in the range. The scatterplot can be peeled [28] for an effective exploration, though such an operation assumes pre-computing the Reeb space. Even if the Reeb space were available, navigating the user in the abundance of features is a challenge. This is because visualizing the Reeb space becomes rapidly challenging as it grows complex [25].

7 Conclusion and Future Work

We extended the flexible isosurface to multi-field without requiring Reeb space analysis. In particular, we generalized the semantics of component-wise pre-image evolution to multi-fields. Our approach does not require computing the pre-image topology explicitly. The algorithm identifies the connected components of fiber surfaces, and sweeps the range to track them. The lack of global pre-image topology and simplification is a downside of this approach (although rendering the Reeb space is itself an unsolved challenge). Through experiments for rather simple datasets, we demonstrated that the interaction in the domain does not necessarily require the Reeb space.

We see a few future directions: the global navigation and simplification of data that are affordable for non-experts of topological analysis; extension to different cell types and interpolants.

Acknowledgement

We thank Julien Tierny at Sorbonne Universities UPMC for offering some of the datasets [29].

This work was supported by the German Federal Ministry of Education and Research (HD(CP)^2 project, grant number 01LK1501C) and the Engineering and Physical Sciences Research Council (EPSRC) project EP/J013072/1.

References

1. Bachthaler, S., Weiskopf, D.: Continuous scatterplots. IEEE Transactions on Visualization and Computer Graphics **14**(6), 1428–1435 (2008)
2. Bajaj, C.L., Pascucci, V., Schikore, D.R.: The contour spectrum. In: Proceedings of IEEE Visualization '97, pp. 167–173 (1997)
3. Bloomenthal, J.: Polygonization of implicit surfaces. Computer Aided Geometric Design **5**(4), 341–355 (1988)
4. Carr, H., Duke, D.: Joint contour nets. IEEE Transactions on Visualization and Computer Graphics **20**(8), 1100–1113 (2014)
5. Carr, H., Geng, Z., Tierny, J., Chattopadhyay, A., Knoll, A.: Fiber surfaces: Generalizing isosurfaces to bivariate data. Computer Graphics Forum **34**(3), 241–250 (2015)
6. Carr, H., Moller, T., Snoeyink, J.: Artifacts caused by simplicial subdivision. IEEE Transactions on Visualization and Computer Graphics **12**(2), 231–242 (2006)
7. Carr, H., Snoeyink, J., Axen, U.: Computing contour trees in all dimensions. Computational Geometry **24**(2), 75–94 (2003)
8. Carr, H., Snoeyink, J., van de Panne, M.: Flexible isosurfaces: Simplifying and displaying scalar topology using the contour tree. Computational Geometry: Theory and Applications **43**(1), 42–58 (2010)
9. Carr, H.A., Weber, G.H., Sewell, C.M., Ahrens, J.P.: Parallel peak pruning for scalable SMP contour tree computation. In: Proceedings of 2016 IEEE 6th Symposium on Large Data Analysis and Visualization (LDAV), pp. 75–84 (2016)

10. Chiang, Y.J., Lu, X.: Progressive simplification of tetrahedral meshes preserving all isosurface topologies. Computer Graphics Forum **22**(3), 493–504 (2003)
11. Doraiswamy, H., Natarajan, V.: Computing Reeb graphs as a union of contour trees. IEEE Transactions on Visualization and Computer Graphics **19**(2), 249–262 (2013)
12. Duke, D., Carr, H., Knoll, A., Schunck, N., Nam, H.A., Staszczak, A.: Visualizing nuclear scission through a multifield extension of topological analysis. IEEE Transactions on Visualization and Computer Graphics **18**(12), 2033–2040 (2012)
13. Edelsbrunner, H., Harer, J., Patel, A.K.: Reeb spaces of piecewise linear mappings. In: Proceedings of the Twenty-fourth Annual Symposium on Computational Geometry, SCG '08, pp. 242–250. New York, NY, USA (2008)
14. Howic, C., Blake, E.: The mesh propagation algorithm for isosurface construction. Computer Graphics Forum **13**(3), 65–74 (1994)
15. Huettenberger, L., Heine, C., Carr, H., Scheuermann, G., Garth, C.: Towards multifield scalar topology based on pareto optimality. Computer Graphics Forum **32**(3pt3), 341 – 350 (2013)
16. Kettner, L., Rossignac, J., Snoeyink, J.: The Safari interface for visualizing time-dependent volume data using isosurfaces and contour spectra. Computational Geometry **25**(1), 97–116 (2003)
17. Kindlmann, G., Durkin, J.W.: Semi-automatic generation of transfer functions for direct volume rendering. In: Proceedings of the 1998 IEEE Symposium on Volume Visualization, VVS '98, pp. 79–86. New York, NY, USA (1998)
18. Klacansky, P., Tierny, J., Carr, H., Geng, Z.: Fast and exact fiber surfaces for tetrahedral meshes. IEEE Transactions on Visualization and Computer Graphics **23**(7), 1782–1795 (2017)
19. van Kreveld, M., van Oostrum, R., Bajaj, C., Pascucci, V., Schikore, D.: Contour trees and small seed sets for isosurface traversal. In: Proceedings of the Thirteenth Annual Symposium on Computational Geometry, SCG '97, pp. 212–220. New York, NY, USA (1997)
20. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3D surface construction algorithm. ACM SIGGRAPH Computer Graphics **21**(4), 163–169 (1987)
21. Manders, E.M.M., Hoebe, R., Strackee, J., Vossepoel, A.M., Aten, J.A.: Largest contour segmentation: A tool for the localization of spots in confocal images. Cytometry **23**(1), 15–21 (1996)
22. Pascucci, V., Cole-McLaughlin, K.: Parallel computation of the topology of level sets. Algorithmica **38**(1), 249–268 (2003)
23. Pascucci, V., Scorzelli, G., Bremer, P.T., Mascarenhas, A.: Robust on-line computation of Reeb graphs: Simplicity and speed. ACM Transactions on Graphics **26**(3), 58 (2007)
24. Reeb, G.: Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique. Comptes Rendus l'Acadèmie des Sciences de Paris **222**, 847–849 (1946)
25. Sakurai, D., Saeki, O., Carr, H., Wu, H.Y., Yamamoto, T., Duke, D., Takahashi, S.: Interactive visualization for singular fibers of functions $f : \mathbb{R}^3 \to \mathbb{R}^2$. IEEE Transactions on Visualization and Computer Graphics **22**(1), 945–954 (2016)
26. Takahashi, S., Takeshima, Y., Fujishiro, I.: Topological volume skeletonization and its application to transfer function design. Graphical Models **66**(1), 24–49 (2004)
27. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. Journal of the ACM **22**(2), 215–225 (1975)
28. Tierny, J., Carr, H.: Jacobi fiber surfaces for bivariate Reeb space computation. IEEE Transactions on Visualization and Computer Graphics **23**(1), 960–969 (2017)
29. Tierny, J., Favelier, G., Levine, J.A., Gueunet, C., Michaux, M.: The Topology ToolKit. Tech. rep., CNRS/UPMC, https://topology-tool-kit.github.io/
30. Weber, G.H., Dillard, S.E., Carr, H., Pascucci, V., Hamann, B.: Topology-controlled volume rendering. IEEE Transactions on Visualization and Computer Graphics **13**(2), 330–341 (2007)
31. Wyvill, B., McPheeters, C., Wyvill, G.: Animating soft objects. The Visual Computer **2**(4), 235–242 (1986)