ISSN 1213-6972 (print)
ISSN 1213-6964 (online)
Journal of WSCG
http://www.wscg.eu
Vol.32, No-1-2, 2024

Exploitation of local adjacencies for parallel construction of a Reeb graph variant: cerebral vascular tree case

Charles LEPAIRE, Hakim BELHAOUARI
University of Poitiers
CNRS, XLIM,
I3M common lab,
Poitiers, France
charles.lepaire@univ-poitiers.fr
hakim.belhaouari@univ-poitiers.fr

Romain PASCUAL
Karlsruhe Institute of Technology,
Karlsruhe, Germany
romain.pascual@kit.edu

Philippe MESEURE
University of Poitiers,
CNRS, XLIM,
Poitiers, France
philippe.meseure@xlim.fr

ABSTRACT

Strokes concerned more than 795,000 individuals annually in the United States as of 2021[1]. Detecting thrombus (blood clot) is crucial for aiding surgeons in diagnosis, a process heavily reliant on 3D models reconstructed from medical imaging. While these models are very dense with information (many vertices, edges, faces in the mesh, and noise), extracting the critical data is essential to produce an accurate analysis to support the work of practitioners. Our research, conducted in collaboration with a consortium of surgeons, leverages generalized maps (g-maps) to compute quality criteria on the cerebral vascular tree. According to medical professionals, artifacts due to noise and thin topological changes are significant parameters among these criteria. These parameters can be determined via the Reeb graph, a topological descriptor commonly used in topological data analysis (TDA). In this article, we introduce a novel classification of saddle points, and a Reeb graph variant called the Local to Global Reeb graph (LGRG). We present parallel computation methods for critical points and LGRG, relying only on local information thanks to the homogeneity of the g-map formalism. We show that LGRG preserves the most subtle topological changes while simplifying the input into a graph formalism that respects the global structure of the mesh, allowing its use in future analyses.

Keywords

Topological data analysis ; Generalized maps ; Parallel computing ; Topology-based geometric modeling ; Reeb graph variant ; Critical points ; Cerebral vascular tree

1 INTRODUCTION

Topological data analysis (TDA) comes from mathematics, applying techniques from algebraic topology to analyze the shape and structure of data. It focuses on understanding the underlying geometric properties of data sets, such as their connectivity, holes, and loops, regardless of the specific metric used to represent the data. TDA can reveal essential features and patterns that may not be obvious through traditional statistical or geometric methods, proving particularly useful for analyzing complex, high-dimensional, or noisy data sets.

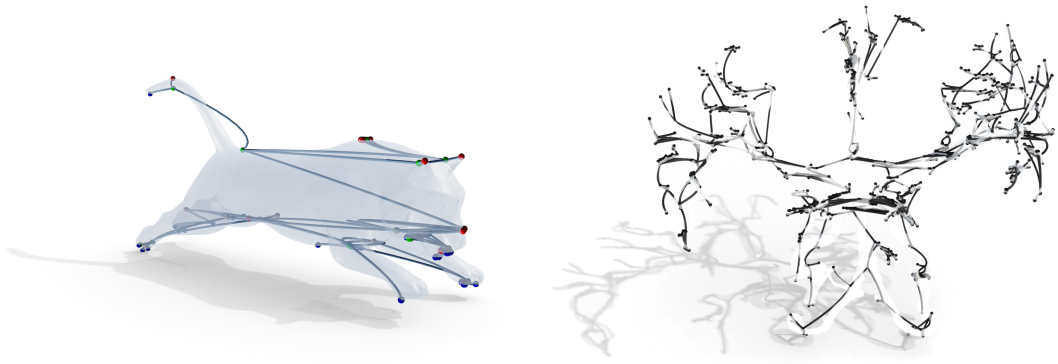
TDA leverages topological descriptors, retrieving specific information from the data. Most of these descriptors rely on the critical points of a function of interest over the data, i.e., the points where the function's derivative is null or undefined. Therefore, computing and classifying the critical points of a function is a crucial subroutine for many TDA tasks. For instance, a Reeb graph consists of nodes corresponding to the critical points and arcs describing the topological modifications between these points.

TDA being agnostic to the application domain, it has been successfully applied across various areas, ranging from medicine [16, 19, 26] to musicology [2] or chemistry [20]. Recently, TDA has been used to analyze 3D medical imaging for assisting practitioners [29, 27], e.g., for discovery in preclinical spinal cord injury and traumatic brain injury [19]. In particular, the Reeb graph has been used for 3D micro-vascular modeling [39] with non-invasive imaging modality to analyze important retina-associated diseases.

As an answer to the increasing data size, recent approaches propose parallel and massively parallel implementations of algorithms to build topological descriptors [28, 30]. Moreover, the complexity of data also presents challenges related to dimension and noise.

Objectives. We aim for a massively parallel implementation of a Reeb graph construction algorithm that preserves the fine-grained topological information of the data.

[1] cf. CDC facts: `https://www.cdc.gov/stroke/facts.htm`



(a) (b)

Figure 1: Local to Global Reeb Graph of a tiger (a), and a cerebral vascular tree (b).

Contributions. We propose to compute a Reeb graph variant called the Local to Global Reeb Graph (LGRG) via a local information diffusion algorithm (examples are given in Figure 1). Our LGRG construction algorithm results in three main contributions.

- We present a refined classification of critical points.
- We offer an extended Reeb graph, which can be computed using only local information provided by a topological description of the structure, without requiring a simplicial complex but only a mesh.
- We propose a massively parallel implementation of an algorithm computing LGRG relying on the generalized maps data structure [10].

A key idea of our approach is to exploit topological proximity rather than geometric one when building the graph's edge. We obtain a Reeb graph variant that also considers degenerate critical points (more precisely, degenerate saddle points), offering a robust and comprehensive solution for handling complex structures. To run in parallel, our algorithm for computing LGRG requires non-concurrent access to local topological information, namely, the edges and face corners around a vertex, which are natively available within generalized maps.

Paper organization. The paper is organized as follows. Section 2 reviews the various approaches to build (variants of) Reeb graphs and the existing parallelized algorithms. Section 3 recalls the generalized map data structure and the islet algorithm used for parallelization. Section 4 presents our refined classification of the critical points, while Section 5 describes our Local to Global Reeb graph. Section 6 is dedicated to our experimental results and comparison with other tools, with emphasis on cerebral vascular trees. Finally, Section 7 gives concluding remarks and discusses possible extensions of our work.

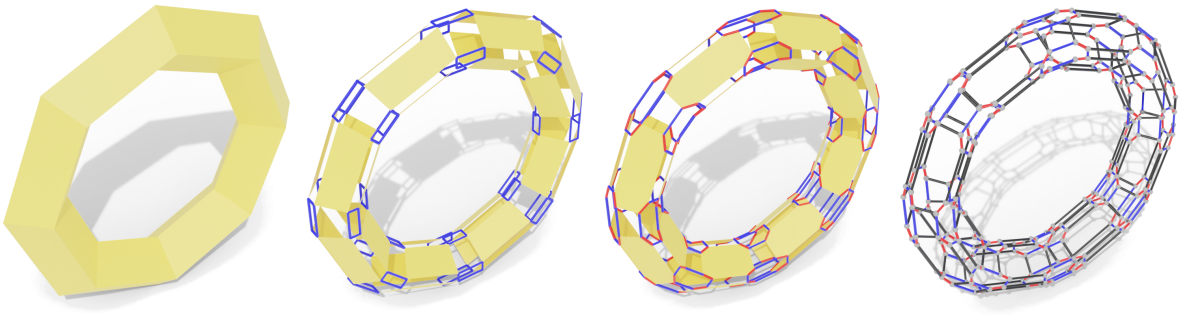
2 RELATED WORK

Given a Morse function [18] $f : M \to \mathbb{R}$ on a manifold M with distinct critical points, the Reeb graph [25] is made of nodes associated with the critical points of the function and arcs where the level sets ($f^{-1}(c)$, for some c in $\mathbb{R}$) retain the same connectivity. Using discrete Morse function in [5], Reeb graphs can be studied for discrete structures [23, 8, 33, 12, 24, 35].

Most approaches that compute Reeb graphs assume that data are represented as a simplicial complex [12, 33] since it simplifies the algorithmic formulation of several standard computations in algebraic topology, such as homology groups. To this end, standard approaches usually triangulate manifolds before computations. For instance, a polyhedron would be decomposed into tetrahedra as a preprocessing step. We propose to work directly on a manifold mesh representation without this preprocessing step.

Discrete Reeb graphs proved fruitful across several areas of scientific visualization [13], such as shape understanding [4], segmentation [38, 15], feature detection [32], or data surface simplification [8]. Within medical applications, Makram et al. [17] used Reeb graphs to automatically locate cephalometric landmarks, Sun et al. [31] applied Reeb graphs to capture the clustering structure of brain white matter nerve fibers. Here, we leverage the topological information provided by Reeb graphs to analyze surface meshes of cerebral vascular trees generated by MRI image reconstruction.

Various generalizations of Reeb graphs have been proposed. The Extended Reeb Graph (ERG) was introduced in [7] as a conceptual model for surface representation. The ERG generalizes Reeb graphs by also considering degenerate critical points, a recurrent problem in discrete surface models while respecting the semantic representation of the Reeb graph. The Augmented Reeb Graph (ARG) adds additional information to the basic structure, such as shape, size, color, or texture. It improves the Reeb graph's accuracy and relevance in understanding complex data [34, 14]. Our Reeb graph variant is similar to the ERG but only accepts degenerated saddle points. Compared to the ARG, we do not



(a) (b) (c) (d)

Figure 2: Recursive decomposition of an object to obtain its g-map representation: (a) geometric object, (b) face split (α_2), (c) edge split (α_1), (d) vertex split (α_0) and final graph.

store information on the Reeb graph but consider it built embedded into the mesh. Thus, information can be directly retrieved from the object.

Advances in medical imaging enabled the acquisition of higher quality, multimodal MRI images, resulting in larger, more complex yet more detailed and fine-grained images. A direct consequence is an increase in computation times needed to build Reeb graphs which resulted from parallel implementations for algorithms computing Reeb graphs. Hajij and Rosen [15] proposed one of the first parallel algorithms for Reeb graphs based on Doraiswamy and Natarajan's algorithm [11]. Their method partitions the manifold into submanifolds, computes Reeb graphs on the submanifolds, and glues the different Reeb graphs to obtain the complete graph. In particular, the partition stage assumes that the entire object is split into connected parts with approximately as many vertices in each part, without further consideration of the partitioning. Gueunet et al. [14] proposed a parallel algorithm for computing an ERG based on the optimal algorithm of Parsa [21], i.e., scanning the mesh along a given direction while maintaining the subcomponents of the level sets, incrementally building the Reeb graph. Guenet et al.'s algorithm scans the mesh in two directions but requires a post-processing step. These (variants of) Reeb graph constructions highly filter the input mesh, resulting in a graph where details are lost. This lost information can be essential for determining the object's structure with a more accurate representation. Our approach exploits local information of the vertices to categorize the critical points and then construct a graph describing the data's topology. Both parts run in parallel without filtering the input.

3 GENERALIZED MAP

This section presents the formalism of generalized maps (g-maps). In brief, a g-map is a compact view of the simplicial set formalism [1].

3.1 Topological model

Generalized maps [10] correspond to a generalization of edge-based models [37] in any dimension and belong to the class of combinatorial models used in topology-based geometric modeling. In this approach, the topology of an object is described as a cellular subdivision. Intuitively, an object's representation can be recovered by recursively splitting its topological cells by decreasing dimension. For instance, Figure 2 shows the decomposition of a low-poly torus in 2D. From the initial object (Fig. 2a), we separate the adjacent faces, which are linked by blue arcs depicting the α_2-links, i.e., adjacency relations along dimension 2 (Fig. 2b). This process is iterated by splitting edges within faces via α_1-links for dimension 1 (Fig. 2c) and finally the vertices via α_0-links for dimension 0 (Fig. 2d). This last subdivision yields the darts of the g-map which we view as nodes of a graph whose arcs are the links obtained in the decomposition.

Formally, g-maps can be defined as undirected graphs labeled on arcs by dimensions and subject to additional constraints [22]. In this article, we consider g-maps of dimension 2, or 2-g-maps that we will simply call g-maps. We summarize here the properties that suffice to appreciate the content of our contributions. (1) Any dart admits a unique incident α_i-link for each $i \in \{0,1,2\}$. (2) When two darts share an α_i-link (for $i \in \{0,1,2\}$), they belong to the same k-cells for $k \neq i$, but to distinct i-cells. (3) G-maps represent orientable and non-orientable quasi-manifolds.

G-maps provide a combinatorial description of the object's decomposition into topological cells. These cells correspond to subgraphs induced by dimensions, called orbits. An orbit consists of all the darts reachable from an initial dart through a subset of all possible dimensions. For instance, the orbit on the dimensions 1 and 2, called a $\langle \alpha_1, \alpha_2 \rangle$-orbit, retrieves darts by following links between cells of dimensions 1 and 2, thus always within the same cell of dimension 0. In other words, an $\langle \alpha_1, \alpha_2 \rangle$-orbit encodes a vertex. Adding geometric positions to vertices then means that all darts within an

$\langle \alpha_1, \alpha_2 \rangle$-orbit share the same position value. Such information is called embedding [3], which provides geometric information to the topological cells. This information is stored on the vertices of the graph, extending the topological representation of g-maps. In this article, we will manipulate 3D positions attached to the vertices and orientation boolean attached to the empty orbit, i.e., directly to the darts.

Working with an orientation boolean attached to each dart means that the represented object is necessarily orientable. In this case, whenever a dart has the value `true`, its α_i-neighbors (for $i \in \{0,1,2\}$) have `false`, and vice-versa. This property can be seen in Figure 3a, where the `true` darts are drawn in pink and the `false` darts in black. This property can be used to prevent concurrent writing in parallel algorithms. Indeed, if only `true` darts write information, no dart can write at the same time as any of its neighbors. Additionally, we use the orientation boolean to orient vertices (see Section 4.1).

3.2 Refresher on the islet algorithm

The islet algorithm [9] looks like the leader election algorithm of distributed systems over networks by choosing a dart within an $\langle o \rangle$-orbit. It spreads information on all α_i-links given by $\langle o \rangle$, iterating once over each dimension before starting another step. It was used to partition orbits to speed up the computation time of specific topological transformations, namely the augment and update operations. As shown in [9], the algorithm can efficiently be parallelized.

In Figure 3, we illustrate the islet algorithm to find a representative dart within each vertex. We choose the representative dart as the one with the minimal ID (which is an integer). Since vertices correspond to $\langle \alpha_1, \alpha_2 \rangle$-orbits, the values are alternately propagated on α_1-links and α_2-links. The object used is a subdivided square, whose g-map representation is given in Figure 3a, along with the darts' ID. Figure 3b shows the propagation along the α_1-links and Figure 3c along the α_2-links. For example, the ID 4 in Figure 3a has been propagated to three new darts in Figure 3c, now covering the two adjacent faces. Some darts received new values at each propagation step, meaning that the process should still be iterated. The algorithm stops when no dart changes its value, which necessarily happens as each orbit contains only finitely many darts. The result of the algorithm is given in Figure 3d, where darts in each $\langle \alpha_1, \alpha_2 \rangle$-orbit share the minimum ID of a dart within the vertex.

4 CLASSIFICATION OF THE CRITICAL POINTS

This section presents our refined categorization of critical points, how to compute them, and how to exploit the topological structure to obtain a parallel computation. Note that our computation method is axis-dependent (like all standard methods).

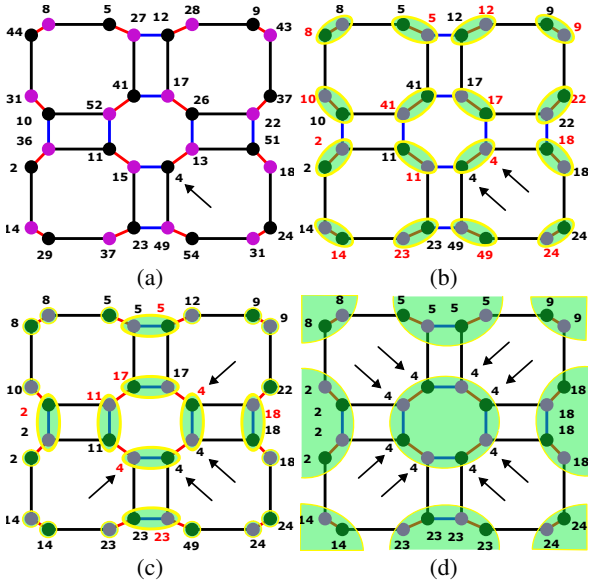


Figure 3: Steps of the islet algorithm 3.2 on the orbit $\langle \alpha_1, \alpha_2 \rangle$. (a) The initial g-map with values associated with the darts. (b) diffusion of minimal value inside the islet produced by α_1-links. (c) diffusion inside the islet from α_2-links. (d) Final state where the minimal value has been diffused for the whole orbit.

4.1 Local elevations

Assuming an oriented data structure to encode a mesh, we propose to classify the critical points of the mesh based on *elevation configurations* of the topological neighborhood of vertices. To compute the vertices elevation configurations, we first compute edge elevations, which yield face corner elevations along consecutive edges within faces, from which we deduce the vertex elevation configuration by aggregating the elevations of all its incident face corners. We now detail each step of the computation, assumed to be computed along a given axis **u** for a given vertex v.

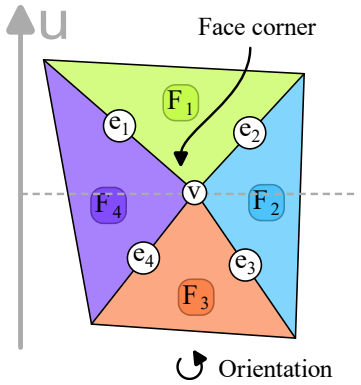


Figure 4: Computing the elevations.

Edge elevation Each edge e incident to v computes an *edge elevation* relative to v by comparing the position of v and that of its other endpoint v', depending on $\mathbf{u}$. Formally, the edge elevation is the sign of the dot product $vv' \cdot \mathbf{u}$. Intuitively, a positive edge elevation describes an edge in the same direction as the selected axis, i.e., v' is 'above' v along $\mathbf{u}$. In Figure 4, the edge e_1 has a positive edge relative to v. Note that the edge elevation might be null. Besides, the edge elevation is relative to the vertex v, and the edge elevation of e relative to v' is the opposite of the one relative to v.

Face corner elevation A face corner is a region delimited by two consecutive edges incident to the same vertex in a given face. Note that the orientation of the object naturally orients the face corners. In Figure 4, the face F_1 together with the edges e_1 and e_2 define one of the four face corners of the vertex v. The *face corner elevation* is the oriented pair of edge elevations for the two edges defining the face corner. If both edges have a positive or null edge elevation, the face corner has a `HIGH` elevation. Similarly, if both edges have a negative or null edge elevation, the face corner has a `LOW` elevation. Note that, if both edges have a null edge elevation, the face corner is `FLAT`. For instance, (F_3, e_3, e_4) has a `LOW` face corner elevation in Figure 4, while (F_1, e_1, e_2) has a `HIGH` one. The two remaining cases correspond to non-null edge elevations with opposite signs. The face corner elevation then depends on the orientation. If the first edge has a negative elevation and the second a positive one, the face corner elevation is `UP`. Reversely, a positive elevation followed by a negative one yields a `DOWN` face corner elevation. In Figure 4, (F_4, e_4, e_1) has a `UP` face corner elevation and (F_2, e_2, e_3) a `DOWN` one.

Vertex elevation sequence Finally, we derive a *vertex elevation sequence* for v as an ordered sequence of face corner elevations around a vertex. The sequence is obtained from the natural ordering induced by the object's orientation. This sequence is defined as up to one circular permutation. For instance, one vertex elevation sequence for v in Figure 4 is given by the sequence of face corner elevations of (F_1, e_1, e_2), (F_2, e_2, e_3), (F_3, e_3, e_4), (F_4, e_4, e_1), i.e., `HIGH`, `DOWN`, `LOW`, `UP`. The vertex elevation sequence is then used to classify the vertex and find the critical points. Note that, if one of the face corner elevations is `FLAT`, it is not considered when calculating the elevation sequence.

4.2 Classification

We propose to retrieve and classify a mesh's critical points based on the elevation sequence of its vertices. We distinguish four types of critical points. In contrast with the usual maxima, minima, and saddle points, we further differentiate between saddles of birth and death. A (local) *minimum* is a vertex with only `HIGH`s in its elevation sequence, e.g., Figure 5a. A (local) *maximum* is a vertex with only `LOW`s in its elevation sequence, e.g., Figure 5b. A *saddle* is a vertex with at least two, by definition non-consecutive, `UP`s or `DOWN`s. The difference between saddles of birth and saddles of death comes from the direction of the vertex normal. The saddle is a *saddle of birth* if its normal aligns with $\mathbf{u}$ (positive dot product) and a *saddle of death* otherwise. Concretely, a saddle of birth separates the inside of an object below and the outside above along the axis $\mathbf{u}$. Examples of saddles of birth and death are respectively given in Figure 5c and in Figure 5d. Any other elevation sequence corresponds to a *regular*, i.e., non-critical, point.

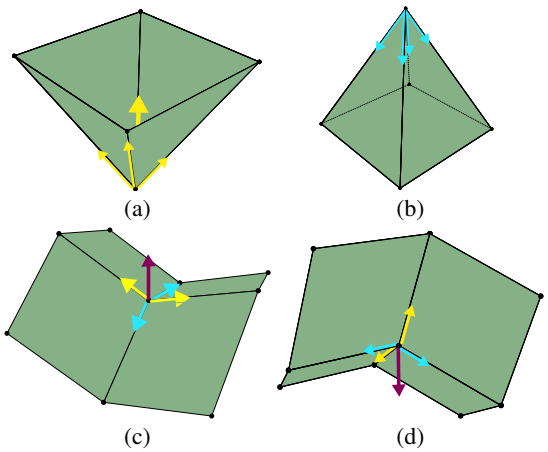


Figure 5: Classification of critical points (a) Local minimum. (b) Local maximum. (c) Saddle of birth. (d) Saddle of death.

Our classification represents two minor contributions. First, the classification only relies on topologically local information around a vertex and can be computed without first triangulating the manifold. Second, we further distinguish between the saddle points using only local information. A watchful reader will notice that standard approaches to Reeb graph computation build the classification of critical points essentially from all edge elevations around a vertex. We introduced the intermediate step of face corner elevations as an additional speed-up when extracting critical points in parallel. We now detail this parallel computation.

4.3 Parallelization

The three steps from Section 4.1 for the detection and classification of critical points only require local information around the vertices and can therefore easily be parallelized.

Edge elevation Within the g-map data structure, an edge e and a reference vertex v are both encoded within a single dart, while the α_0-link allows accessing the other endpoint of the edge (in constant time). While two α_2-linked darts encode a given edge e and a given vertex

v (they correspond to the two faces sharing the edge), only one dart has `true` as the orientation boolean. Since its α_0-neighbor also has `false` for orientation boolean, the edge elevation can be computed by comparing the position of each `true`-oriented dart with its α_0-neighbor. The subroutine can be realized simultaneously on all `true`-oriented darts without concurrent read or write, i.e., in parallel.

Face corner elevation A face corner corresponds to a $\langle\alpha_1\rangle$-orbit in a g-map and thus to two darts with different orientation boolean values. Thus, the face corner elevation is obtained by running a parallel for loop on all darts oriented `true` and comparing the dart's edge elevation with that of its α_1-neighbor. The four cases, `HIGH`, `LOW`, `UP`, and `DOWN` are respectively illustrated in Figures 6a, 6b, 6c, and 6d.

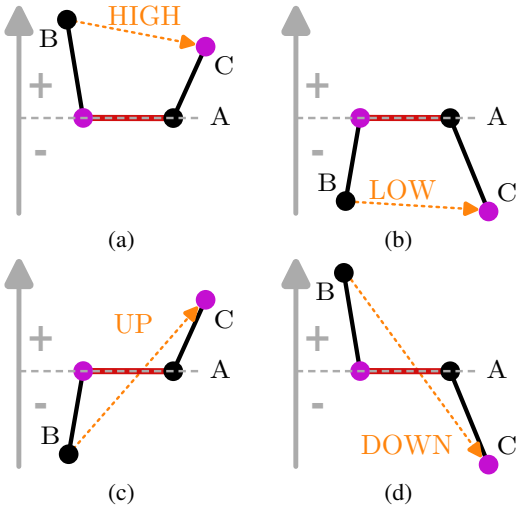


Figure 6: The different face corner elevations. (a) `HIGH`: Both edges have a positive elevation (B and C are "higher" than A). (b) `LOW`: Both edges have a negative elevation (B and C are "lower" than A). (c) `UP`: The dart from the orbit of A with a `true` orientation belongs to an edge with a negative elevation, and the other edge has a positive elevation. B is lower than A, which is lower than C. (d) `DOWN`: The dart from the orbit of A with a `true` orientation value belongs to an edge with a positive elevation, and the other edge has a negative elevation. (B is higher than A, which is higher than C).

Vertex elevation sequence Retrieving the vertex elevation sequence requires to elect one representative dart per vertex, i.e., per $\langle\alpha_1, \alpha_2\rangle$-orbit. Choosing such a dart can be performed in parallel via the islet algorithm. By starting with the representative dart of a vertex, its elevation sequence is obtained by cycling through the $\alpha_2 \circ \alpha_1$-links around it and retrieving the associated face corner elevation. This task can again be performed in parallel within each vertex. In Figure 7, starting from any pink dart, we cycle through the other three pink darts via $\alpha_2 \circ \alpha_1$-links to obtain either `UP`, `DOWN`, `UP`, `DOWN` or `DOWN`, `UP`, `DOWN`, `UP`. Finally, each aggregated sequence is analyzed to classify the vertex.

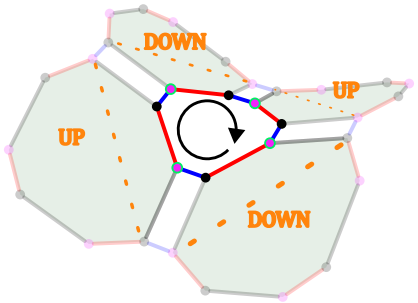


Figure 7: Elevation sequence around a vertex.

5 LOCAL TO GLOBAL REEB GRAPH (LGRG)

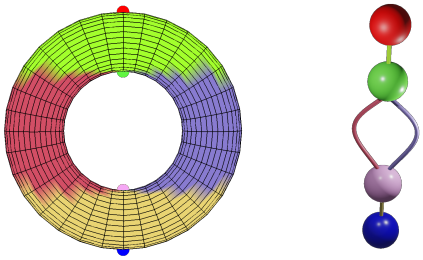
Once the critical points have been identified, we use a propagation algorithm that spreads information, similar to a gossip algorithm for distributed systems. The motivation is to obtain a partition of the mesh into sub-components created and ended by critical points. We then build our LGRG by connecting critical points (which are the nodes of the graph) with arcs encoding how the sub-components link these critical points. Note that a sub-component may correspond to several arcs in our Reeb graph variant.

Before detailing the propagation algorithm and the construction of the LGRG, we provide some insights via the example of Figure 8. The shown torus contains four critical points: a maximum in red, a minimum in blue, a saddle of birth in light pink, and a saddle of death in green (the elevations being computed along the vertical axis). These critical points split the torus vertices into four sub-components: green, red, purple, and yellow. The yellow sub-component corresponds to the propagation area of the blue minimum, ended by the pink saddle of birth, giving rise to the arc between the bottom two nodes in the graph of Figure 8b. For this specific object, each sub-component in Figure 8a results in an arc in Figure 8b. Note that each node is associated with a critical point drawn with the same color for ease of visualization.

In the sequel, we use the terms "above", "below", and "topmost" to be understood along $\mathbf{u}$, i.e., as the sign of the dot product with $\mathbf{u}$.

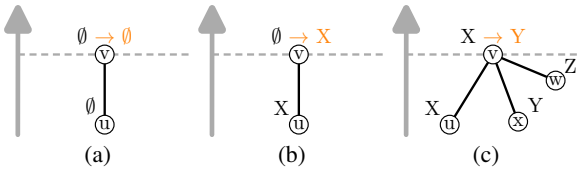
5.1 Propagation and sub-components

The motivation for a propagation algorithm is to obtain a construction fully given by the description of a behavior for the mesh vertices. Each vertex propagates some information related to a given critical point. Intuitively, the vertex spreads information describing where the



(a) (b)

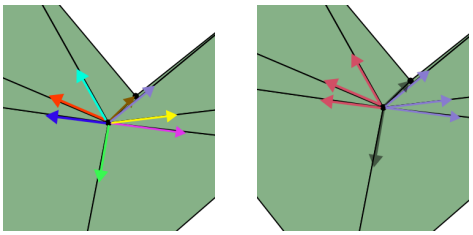
Figure 8: Area propagation on a mesh of a torus (a) the mesh and (b) the resulting graph, our Reeb graph variant, based on area adjacency.



(a) (b) (c)

Figure 10: Diffusion pattern of a vertex v. (a) v is unlabelled and does not receive any label: nothing occurs. (b) v is unlabelled and receives some label X: v updates its label to X. (c) v is already labeled by X and receives the labels Y and Z from neighbors: it compares X, Y and Z to find that Y is the highest, so v updates its label to Y.

last change in the connectivity of the level sets comes from. In practice, this corresponds to the topmost critical points located below the vertex propagating this information. Since the computation on vertices exploits locally retrieved information, it runs in parallel.



(a) (b)

Figure 9: On the left (a), a saddle of birth and its incident edges elevation generates two sub-components. On the right (b), the same saddle of birth with the elevation of its incident edges grouped by equivalent edges (using the same color) shows the beginning of red and purple sub-components.

The information propagated and the propagation used naturally follows from the semantics associated with the (non-)critical vertices of the mesh. Let us recall the meaning of vertex classes concerning the changes to the local connectivity of the level sets to clarify the propagated labels. A regular point does not change the local connectivity; therefore, it initially does not own any label. A maximum ends a level set, meaning it initially owns no label and does not propagate any. A minimum creates a new level set, thus initially owning a unique label spread upwards. Similarly, a saddle of death merges several level sets into one, so it initially owns a unique label spread upwards. The more complex case concerns the saddles of birth. Such critical points split a level set into several. Therefore, it initially needs to own one label per created level set. For instance, the saddle of birth given in Figure 9a is incident to several edges with positive elevation. We cluster them into connected sequences of identical edge elevation around the saddle, as shown in Figure 9b. Thus, we create one label per group of edges with positive elevation. All edges within a group initially propagate its label.

The partitioning algorithm alternates the propagation of labels along the edges with positive elevation and the update of vertex labels. When receiving labels, a vertex always keeps the topmost received label (note that only regular vertices follow this process). The various possible patterns are given in Figure 10. Note that the values of interest correspond to critical points. The algorithm stops when no vertex updates its label. A *subcomponent* corresponds to the set of vertices sharing the same label when the algorithm stops. It corresponds to all regular vertices immediately higher than the critical point that gave them its label. By construction, a subcomponent is delimited by one critical point below and at least one but possibly several critical points above.

The propagation in denser parts of the mesh is naturally slower than in sparser parts. Thus, some vertices may be first labeled by a critical point, which does not correspond to their final sub-component. Indeed, the algorithm later overwrites the label of these vertices (and thus of the whole area it propagated to) with one from a higher critical point, maybe even several times, until the highest one is found. In practice, an area propagates from a critical point upwards over a mesh until it can no longer propagate because the vertices of its boundary are either connected by edges with a negative elevation or connected by edges with a positive elevation to vertices labeled by a higher critical point (including critical points themselves).

The g-map data structure simplifies the algorithm as labels can be stored on darts and propagated along α_0-arcs. Reasoning on darts essentially means that (a) vertices corresponding to saddles of birth store several labels among the darts of their $\langle \alpha_1, \alpha_2 \rangle$-orbit, and (b) vertex update means choosing the topmost label within the $\langle \alpha_1, \alpha_2 \rangle$-orbit of regular vertices. The vertex label is updated via the islet algorithm (see Section 3.2), propagating labels instead of dart IDs.

5.2 Local to Global Reeb graph construction

The propagation algorithm terminates when no vertex updates its label, constructing a mesh partition into subcomponents. We then build the LGRG from the adjacency relationships between the subcomponents. If an edge links two vertices in the mesh belonging to distinct sub-components, we add an arc to the LGRG between the associated critical points. Note that only one arc is added to the LGRG even if several edges in the mesh link vertices between the same two sub-components. In the propagation algorithm, saddles of birth create several sub-components (identified by a group of consecutive edges with positive elevation). Since these subcomponents arise from the same critical point, the induced arcs in the LGRG are all linked to the same node corresponding to the saddle of birth. This construction also justifies propagating several identifiers from a saddle of birth. Indeed, had we propagated only one identifier, some arcs would have been missing in the LGRG. For instance, the two arcs between the saddle of birth and the saddle of death in the torus of Figure 8b would have been merged whereas they should remain distinct.

Once again, this step can be parallelized by checking all edges simultaneously and eliminating redundant pairs of subcomponents. The islet algorithm directly enables this computation by identifying a dart within each edge ($\langle\alpha_0, \alpha_2\rangle$-orbit) and comparing its label with the one of its α_0-neighbor.

6 RESULTS

Experiments were conducted on an 11th Gen Intel(R) Core(TM) i7-11850H @2.50GHz with 8 hyper-threaded cores (16 simultaneous threads) and 32GB of RAM. We used the implementation of g-maps provided by Jerboa [6] and Java streams (with JDK 17) for parallelization. We tested our method using the height function as a scalar field on standard geometric processing objects, e.g., a tiger, a dragon, etc. (see Figure 12), and medical vascular trees (see Figure 13) obtained via the marching cubes algorithm on the MRI images provided by the MICCAI CROWN challenge [36].

Table 1 provides statistics about the objects in our dataset and the critical points found by the three methods. Our approach finds a distribution of critical points on the standard objects similar to TTK, while ReCon produces many additional critical points. The example of the pegasus highlights that TTK filters the object and misses a lot of details. We find nearly 5 times more critical points on this object. These additional critical points come from the base and the wings, as shown in Figure 12e. Although it may not be relevant for encoding the topology of the pegasus, such a fine-grained description of the topological changes corresponds to the data of interest for the medical staff in our consortium. In the second half of the table, statistics are given on twelve cerebrovascular trees taken from various patients. Recon still finds many more saddle points than our method and TTK. However, we detect more minima and maxima, highlighting that we better capture subtle topological variations filtered out by the other tools.

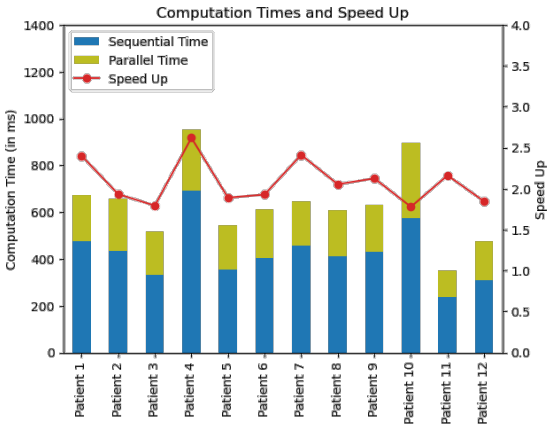
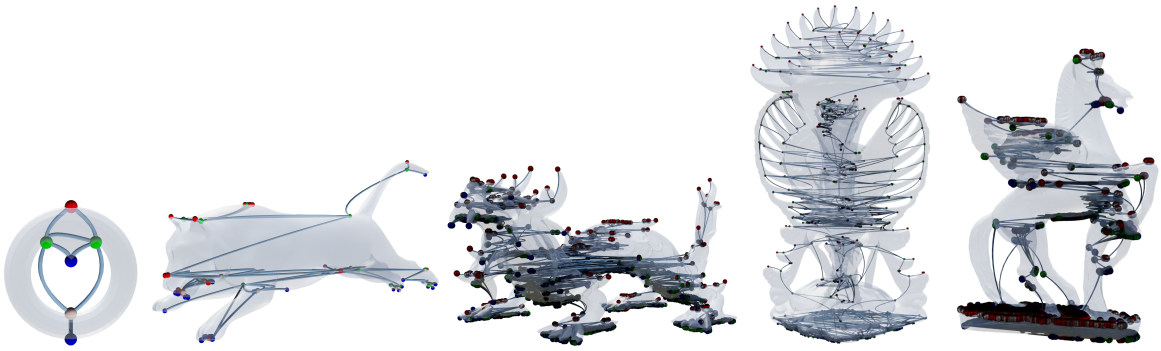


Figure 11: Comparison between sequential and parallel runtimes for LGRG on cerebral trees.

We also compared sequential and parallel executions of our method on the cerebral trees, i.e., where both the critical points classification and LGRG construction are performed as discussed in Sections 4 and 5. Computation times are given in Figure 11 and show an average speed up by a factor 2.08.

7 CONCLUSION

We proposed a novel approach for categorizing critical points by exploiting local topological information at each vertex and a variant of the Reeb graph built from a topology-induced information propagation from the critical points over the mesh. We then leveraged this local approach to provide a parallel algorithm for characterizing critical points and the computation of LGRG on the g-map data structure, which we implemented in Jerboa. In particular, using topological information reduces the dependency on geometric features, thus avoiding excessive filtering and maintaining data integrity. We plan to apply our LGRG construction for medical analysis to assist in the diagnosis of strokes. As of now, we also hope to understand better the topological features that need to be detected for the diagnosis with the help of the medical staff on the one hand and by exploiting machine learning approaches on our LGRG. For instance, apart from the circle of Willis, the existence of a topological cycle in the cerebral vascular tree is an indicator of a stroke. Still, whether this criterion is sufficient or even necessary remains unclear. Out of the medical field, our new LGRG also opens the way for



(a) (b) (c) (d) (e)

Figure 12: LGRG computed on a torus ((a), a tiger (b), a dragon called "XYZ RGB" (c), Garuda and Vishnu (d)), and a pegasus (e).

(a) (b)

Figure 13: LGRG computed on the cerebral vascular trees of patient 1 (a), and patient 2 (b).

Object	#faces	#vert	Minimum			Maximum			Saddles		
	(∗1000)	(∗1000)	TTK	ReCon	Ours	TTK	ReCon	Ours	TTK	ReCon	Ours
Torus	1.1	0.6	2	2	2	1	1	1	3	3	3
Tiger	116.7	58.4	26	191	29	16	230	23	40	419	30
XYZ RGB	199.8	99.9	648	7,768	659	519	14,330	508	1,102	20,875	1,168
G. & V.	222.9	111.4	120	3,903	120	120	7,814	121	307	11,576	308
Pegasus	667.5	333.7	287	3,169	1,164	229	6,605	3,556	517	9,723	584
Patient 1	129.4	65.7	198	385	509	210	685	1246	261	926	167
Patient 2	123.9	62.7	239	-	367	249	-	1,086	263	-	166
Patient 3	100.8	50.9	122	307	272	126	475	624	154	695	114
Patient 4	152.3	77.1	308	469	653	307	807	1,667	458	1127	310
Patient 5	106.3	54.0	155	418	430	163	733	990	194	1023	133
Patient 6	125.7	63.7	166	400	396	192	609	949	238	895	171
Patient 7	137.2	69.5	271	470	448	298	722	1,367	329	971	230
Patient 8	117.0	59.5	235	376	500	256	778	1,386	272	927	170
Patient 9	133.8	67.8	204	477	511	227	767	1,089	280	1097	189
Patient 10	160.0	80.9	212	490	462	231	778	908	279	1118	200
Patient 11	82.1	41.5	130	264	273	127	439	692	159	604	101
Patient 12	101.8	51.6	177	340	419	190	582	1,023	241	803	173

Table 1: Comparison of the critical points computed with TTK, ReCon, and our method.

locally induced mesh segmentation and data compression. Furthermore, exploring a C++/CUDA implementation holds promise for accelerating our parallelization efforts, potentially leading to significant runtime improvements.

8 ACKNOWLEDGMENT

This research work is funded by Region Nouvelle-Aquitaine, France (AAP ESR 2021 Program - ARTERIA Project, AAPR2021A-2021-12189710) and SIEMENS Healhineers.

REFERENCES

[1] S. Alayrangues, S. Peltier, G. Damiand, and P. Lienhardt. "Border operator for generalized maps". In: *DGCI*. Vol. 5810. Montreal, Canada: Springer, 2009, pp. 300–312. DOI: 10.1007/978-3-642-04397-0_26.

[2] M. Andreatta. "Méthodes algébriques en musique et musicologie du XXe siècle : aspects théoriques, analytiques et compositionnels". PhD Dissertation. Paris, EHESS, 2003.

[3] A. Arnould, H. Belhaouari, T. Bellet, P. Le Gall, and R. Pascual. "Preserving consistency in geometric modeling with graph transformations". In: *MSCS* 32.3 (2022), pp. 300–347. DOI: 10.1017/S0960129522000226.

[4] M. Attene, S. Biasotti, and M. Spagnuolo. "Shape understanding by contour-driven retiling". In: *Vis. Comput.* 19.2 (2003), pp. 127–138. DOI: 10.1007/s00371-002-0182-y.

[5] T. F. Banchoff. "Critical Points and Curvature for Embedded Polyhedral Surfaces". In: *Am. Math. Mon.* 77.5 (1970), p. 475. DOI: 10.2307/2317380.

[6] H. Belhaouari, A. Arnould, P. Le Gall, and T. Bellet. "Jerboa: A Graph Transformation Library for Topology-Based Geometric Modeling". In: *ICGT*. LNCS. 2014, pp. 269–284. DOI: 10.1007/978-3-319-09108-2_18.

[7] S. Biasotti, B. Falcidieno, and M. Spagnuolo. "Extended Reeb Graphs for Surface Understanding and Description". In: *DGCI*. LNCS. 2000, pp. 185–197. DOI: 10.1007/3-540-44438-6_16.

[8] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. "Reeb graphs for shape analysis and applications". In: *TCS*. Computational Algebraic Geometry and Applications 392.1 (2008), pp. 5–22. DOI: 10.1016/j.tcs.2007.10.018.

[9] P. Bourquat, H. Belhaouari, P. Meseure, V. Gauthier, and A. Arnould. "Transparent Parallelization of Enrichment Operations in Geometric Modeling". In: *VISIGRAPP/GRAPP*. 2020, pp. 125–136. DOI: 10.5220/0008965701250136.

[10] G. Damiand and P. Lienhardt. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing*. CRC Press, 2014. ISBN: 978-1-4822-0652-4.

[11] H. Doraiswamy and V. Natarajan. "Efficient algorithms for computing Reeb graphs". In: *Comput. Geom* 42.6 (2009), pp. 606–616. DOI: 10.1016/j.comgeo.2008.12.003.

[12] H. Doraiswamy and V. Natarajan. "Computing Reeb Graphs as a Union of Contour Trees". In: *TVCG* 19.2 (2013), pp. 249–262. DOI: 10.1109/TVCG.2012.115.

[13] F. Escolano, E. R. Hancock, and S. Biasotti. "Complexity Fusion for Indexing Reeb Digraphs". In: *Computer Analysis of Images and Patterns*. LNCS. 2013, pp. 120–127. DOI: 10.1007/978-3-642-40261-6_14.

[14] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny. "Task-based Augmented Reeb Graphs with Dynamic ST-Trees". In: *EGPGV19*. 2019. DOI: 10.2312/pgv.20191107.

[15] M. Hajij and P. Rosen. *An Efficient Data Retrieval Parallel Reeb Graph Algorithm*. 2020. DOI: 10.48550/arXiv.1810.08310.

[16] C. F. Loughrey, P. Fitzpatrick, N. Orr, and A. Jurek-Loughrey. "The topology of data: opportunities for cancer research". In: *Bioinformatics* 37.19 (2021), pp. 3091–3098. DOI: 10.1093/bioinformatics/btab553.

[17] M. Makram and H. Kamel. "Reeb Graph for Automatic 3 D Cephalometry". In: *Int. J. of Image Processing* 8.2 (2014).

[18] J. W. Milnor, M. Spivak, and R. Wells. *Morse theory*. Ann. Math. Stud. 51. PUP, 1963.

[19] J. L. Nielson et al. "Topological data analysis for discovery in preclinical spinal cord injury and traumatic brain injury". In: *Nat. Commun.* 6.1 (2015), p. 8581. DOI: 10.1038/ncomms9581.

[20] M. Olejniczak, P. Severo, A. Gomes, and J. Tierny. "A Topological Data Analysis Perspective on Non-Covalent Interactions in Relativistic Calculations". In: *Int. J. of Quantum Chemistry* e26133 (2019). DOI: 10.1002/qua.26133.

[21] S. Parsa. "A deterministic O(m log m) time algorithm for the Reeb graph". In: *Discrete Comput. Geom.* 49 (2013), pp. 864–878. DOI: 10.1145/2261250.2261289.

[22] R. Pascual, P. Le Gall, A. Arnould, and H. Belhaouari. "Topological consistency preservation with graph transformation schemes". In: *SCP* 214 (2022), p. 102728. DOI: 10.1016/j.scico.2021.102728.

[23] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas. "Robust on-line computation of Reeb graphs". In: *TOG* 26 (2007), p. 58. DOI: 10.1145/1276377.1276449.

[24] A. Polette, J. Meunier, and J.-L. Mari. "Feature extraction using a shape descriptor graph based on discrete curvature patches". In: *CGI*. 2015.

[25] G. Reeb. "Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique". In: *Comptes Rendus Acad. Sciences Paris* 222 (1946), pp. 847–849.

[26] M. Rucco, E. Merelli, D. Herman, D. Ramanan, T. Petrossian, L. Falsetti, C. Nitti, and A. Salvi. "Using topological data analysis for diagnosis pulmonary embolism". In: *J. of Theo. and Appl. Comp. Sci.* 9.1 (2015), pp. 41–55.

[27] A. Salch, A. Regalski, H. Abdallah, R. Suryadevara, M. J. Catanzaro, and V. A. Diwadkar. "From mathematics to medicine: A practical primer on topological data analysis (TDA) and the development of related analytic tools for the functional discovery of latent structure in fMRI data". In: *PLOS One* 16.8 (2021). DOI: 10.1371/journal.pone.0255859.

[28] N. Shivashankar and V. Natarajan. "Parallel Computation of 3D Morse-Smale Complexes". In: *CGF* 31 (2012), pp. 965–974. DOI: 10.1111/j.1467-8659.2012.03089.x.

[29] Y. Skaf and R. Laubenbacher. "Topological data analysis in biomedicine: A review". In: *J. Biomed. Inform* 130 (2022), p. 104082. DOI: 10.1016/j.jbi.2022.104082.

[30] V. Subhash, K. Pandey, and V. Natarajan. "A GPU Parallel Algorithm for Computing Morse-Smale Complexes". In: *TVCG* 29.9 (2023), pp. 3873–3887. DOI: 10.1109/TVCG.2022.3174769.

[31] J. Sun, M. Cieslak, S. T. Grafton, and S. Suri. "A Reeb graph approach to tractography". In: *SIGSPATIAL* (2015).

[32] S. Takahashi, Y. Takeshima, and I. Fujishiro. "Topological volume skeletonization and its application to transfer function design". In: *GMOD* 66.1 (2004), pp. 24–49. DOI: 10.1016/j.gmod.2003.08.002.

[33] J. Tierny, J.-P. Vandeborre, and M. Daoudi. "Partial 3D Shape Retrieval by Reeb Pattern Unfolding". In: *CGF* 28 (2009), pp. 41–55. DOI: 10.1111/j.1467-8659.2008.01190.x.

[34] T. Tung and F. Schmitt. "Augmented Reeb graphs for content-based retrieval of 3D mesh models". In: *SMI*. 2004, pp. 157–166. DOI: 10.1109/SMI.2004.1314503.

[35] J. Vidal, P. Guillou, and J. Tierny. "A Progressive Approach to Scalar Field Topology". In: *TVCG* 27.6 (2021), pp. 2833–2850. DOI: 10.1109/TVCG.2021.3060500.

[36] I. Vos, Y. Ruigrok, E. Bennink, M. Buser, B. Velthuis, and H. Kuijf. *Data of the Circle of Willis Intracranial Artery Classification and Quantification (CROWN) Challenge*. 2023. DOI: 10.34894/R05G1L.

[37] K. Weiler. "Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments". In: *IEEE CG&A* 5.1 (1985), pp. 21–40. DOI: 10.1109/MCG.1985.276271.

[38] N. Werghi. "Segmentation and modelling of full human body shape from 3D scan data: A survey". In: *VISAPP*. 2006, pp. 189–197.

[39] J. Zhang, A. H. Kashani, and Y. Shi. "3D Surface-Based Geometric and Topological Quantification of Retinal Microvasculature in OCT-Angiography via Reeb Analysis". In: *MICCAI*. Vol. 11764. LNCS. 2019, pp. 57–65. DOI: 10.1007/978-3-030-32239-7_7.