Parallel Computation of Piecewise Linear Morse-Smale Segmentations

Robin G. C. Maack, Jonas Lukasczyk, Julien Tierny, Hans Hagen, Ross Maciejewski, Christoph Garth

Parallel Computation of Piecewise Linear Morse-Smale Segmentations

Robin G. C. Maack, Jonas Lukasczyk, Julien Tierny, Hans Hagen, Ross Maciejewski, and Christoph Garth

Abstract—This paper presents a well-scaling parallel algorithm for the computation of Morse-Smale (MS) segmentations, including the region separators and region boundaries. The segmentation of the domain into ascending and descending manifolds, solely defined on the vertices, improves the computational time using path compression and fully segments the border region. Region boundaries and region separators are generated using a multi-label marching tetrahedra algorithm. This enables a fast and simple solution to find optimal parameter settings in preliminary exploration steps by generating an MS complex preview. It also poses a rapid option to generate a fast visual representation of the region geometries for immediate utilization. Two experiments demonstrate the performance of our approach with speedups of over an order of magnitude in comparison to two publicly available implementations. The example section shows the similarity to the MS complex, the useability of the approach, and the benefits of this method with respect to the presented datasets. We provide our implementation with the paper.

Index Terms—Topology, Visualization, Segmentation, Morse-Smale Complex, Watershed transformation.

1 Introduction

Topological Data Analysis (TDA) provides a family of effective feature characterizations, including the well-studied Morse-Smale (MS) complex. The MS complex is a central tool in TDA for feature-driven data analysis and visualization, as it segments the domain of a scalar field into regions with equivalent gradient flow behavior (see Fig. 1 and Sec. 3). This rather abstract feature characterization based on gradient flow has been applied successfully in several domains, including chemistry [1, 2], material science [3, 4], physics [5, 6], and cosmology [7].

However, due to the high computational complexity of MS complex generation, it often becomes a time-consuming bottleneck. Especially in the case of interactive analysis and visualization, users expect to quickly retrieve a visual output, where long wait times can interrupt their workflow. Furthermore, many applications do not require the computation of the full MS complex, but only two of its central features: 1) the MS segmentation of the domain that assigns extrema to each vertex by following the gradient along the steepest descend and ascend; and 2) the interfaces between different regions in the segmentation.

In this paper we describe a scalable implementation of these two tasks with high parallel efficiency, further referred to as the piecewise linear Morse-Smale segmentation (PLMSS) algorithm (Sec. 4). As the name suggests, the input of PLMSS is a scalar field defined on the vertices of a piecewise linear domain, i.e., a simplicial complex. PLMSS utilizes path compression to derive the MS segmentation of the domain and a multi-label marching tetrahedra procedure to derive the interfaces between different regions of the MS segmentation. We chose these two underlying algorithms since they are known to be embarrassingly parallelizable, and therefore expected to scale well. In Sec. 5 we demonstrate the benefit of using PLMSS for effective data analysis and visualization on four datasets.

We compare PLMSS against the corresponding subprocedures of two state-of-the-art Morse-Smale complex software libraries, i.e., the implementations available in the Topology ToolKit (TTK) [8] and MSCEER [9]. Specifically, we performed two strong scaling studies (Sec. 6) that show that MSCEER outperforms TTK, but PLMSS is still up to an order of magnitude faster than MSCEER, while also providing superior parallel efficiency. However, the qualitative and quantitative comparison between the region separators computed by PLMSS and their counterpart of MS complex 2-cells is more challenging. Although they both separate regions with different gradient flow behavior, they are defined and computed differently (Sec. 3). Yet, for many applications—including the scenarios presented in Sec. 5—the computation of region separators is sufficient without the need for expensive discrete gradient field and dual mesh computations.

In short, the contributions of our work are:

- A scalable algorithm with high parallel efficiency for the computation of piecewise linear Morse-Smale segmentations (PLMSS);
- A detailed performance benchmark that compares PLMSS with the corresponding subprocedures of two state-of-the-art Morse-Smale complex software libraries (TTK and MSCEER); and
- The integration of PLMSS in TTK to facilitate future benchmarks and reproducibility.

2 Related Work

The MS complex subdivides a given scalar field into regions of uniform gradient flow behavior, segmenting the domain

- *R. G. C. Maack, J. Lukasczyk, H. Hagen, and C. Garth are with RPTU Kaiserslautern-Landau. E-mails: {maack, lukasczyk, hagen, garth}@rptu.de*
- *J. Tierny is with the CNRS and Sorbonne Université. Email: julien.tierny@sorbonne-universite.fr*
- *R. Maciejewski is with Arizona State University. Email: rmacieje@asu.edu*

978-1-5386-5541-2/18/$31.00 ©2018 IEEE

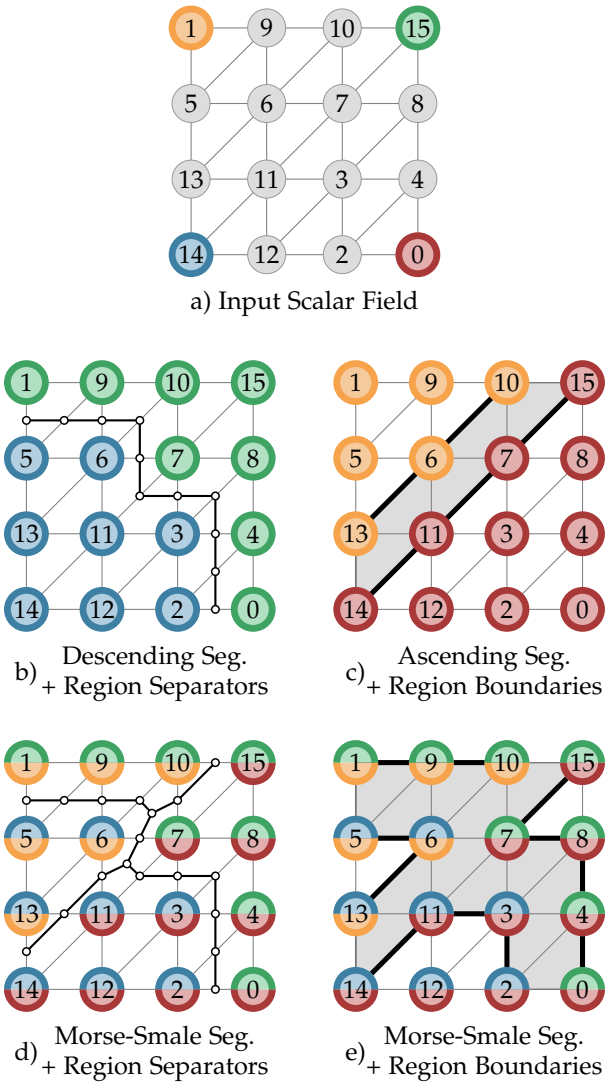


Fig. 1. A 4×4 example triangulation showing the input field (a) and available segmentations and region-separating geometries (b-e). Each color represents the influence area of a specific extremum (the areas of minima 0 and 1 are shown in red and yellow, while the area of maxima 14 and 15 are shown in blue and green). The descending segmentation (b) represents the influence area of maxima and the ascending segmentation (c) the ones of all minima. In the case of Morse-Smale segmentations (d-e) the nodes show the influence of minima-maxima combinations, leading to nodes being colored by the minima color at the bottom and the maximum color at the top. Subfigures (b) and (d) show region separators (thin black lines), while (c) and (e) show region boundaries (thick black lines).

such that each point in the same MS manifold will flow towards the same critical point pair considering forward and backward integration. Two approaches to computing the MS complex arose from Morse theory [10], where either a discrete gradient vector field is defined on the whole domain [11], or piecewise linear Morse theory is used to define a segmentation [12, 13]. For an in-depth comparison of both approaches, we refer to Lewiner's work [14].

Algorithms based on piecewise linear Morse theory are divided into boundary-based and region-growing algorithms. Boundary-based algorithms trace lines of steepest descent/ascent seeded at the saddles, such that every vertex on a line of steepest ascent/descent belongs to the same region. Region-growing algorithms grow sets of top-level cells, e.g. cubes or tetrahedra in 3D, located at the minima and maxima of a scalar function, iteratively enlarging regions. One representative of boundary-based algorithms is Edelsbrunner et al. [15] that first introduced the MS complex for piecewise linear 2-manifolds, recording paths of steepest ascent and descent. They also introduced the notion of the quasi MS complex that was extended to 3-manifolds [16] and later improved in geometric accuracy by Bremer et al. [17]. Concerning region-growing algorithms, Danovaro et al. [18] started growing regions by using triangles incident on maxima at vertices, adding edge incident triangles iteratively. They extended this approach by appending additional seeding points at initialization, while also enabling to process higher-dimensional scalar fields [19]. Gyulassy et al. [20] implemented a region-growing algorithm that labels the vertices to extract 3-, 2-, and 1-cells, also extendable from 3D to higher dimensional scalar fields.

Discrete Morse theory was developed by Forman [21], applying Morse theory to any type of simplicial cell complex. Many algorithms build upon the discrete gradient vector field to effectively compute the MS complex [22, 23, 24]. To efficiently compute the MS complex for large datasets, several parallel and distributed memory implementations were introduced. Gyulassy et al. [25] first proposed splitting a dataset into subsets called parcels, extracting the MS complex from each parcel to later merge them in a cancellation-based step. This allowed for the computation of datasets that do not fit into memory and gave rise to distributed memory approaches [26, 27]. Further parallel optimizations were achieved by merging gradient paths, enabling the computation of the gradient assignment and extrema traversals on the GPU [28, 29]. Subhash et al. [30] then accomplished computing all steps of the MS complex computation on the GPU. Even though some algorithms improved the steepest descent line tracing [17, 20] by allowing the traversal to use edges and triangles, still all presented algorithms often produce incorrect connectivity and inaccurate geometry due to the refinement of the underlying discrete domain [31]. Here, Gyulassy et al. [32] implemented a probabilistic algorithm to extract the correct geometry and connectivity. Morse-Smale complexes were already used in many applications such as material science [33], chemistry [34], and medicine [35], allowing for fast and consistent analysis of the data. Cancellation-based simplification [36, 37, 38] is often used in applications, where pairs of critical points are removed to simplify the MS complex and eliminate noise in the dataset. It counteracts over-segmentation of the domain and enables the extraction of persistent features.

The Watershed transform, originally defined by Beucher and Lantuéjoul [39], is another approach to Morse theory, segmenting the domain, usually gray-scale images, into catchment basins that represent the zone of influence of minima and watershed lines, separating catchment basins from each other. Beucher [40] described catchment basins as areas where each drop of water ends up at the same minimum when flowing down the surface. In contrast to Morse theory, minima don't have to be distinct but can rather consist of multiple vertices of the same function value. Those definitions were further improved to be rigorous [41, 42] and extended to the discrete case [43, 44]. De Floriani [45] distinguishes three types of watershed

algorithms that are either based on topographic distance, simulated immersion, or rainfalling simulation. Algorithms based on topographic distance compute shortest paths to find corresponding catchment basins [43, 46]. Gabrielyan et al. [47] and Yeghiazaryan and Voiculescu [48] provide GPU implementations using an approach similar to path compression, speeding up the shortest path computation. Simulated immersion approaches seed catchment basins at minima, extending them by processing vertices in increasing function value order [44, 49]. Rainfalling simulation algorithms use an inverted logic, finding and labeling minima first, then decreasing in function value for each vertex by steepest descent until a labeled vertex is found [50, 51].

Marching tetrahedra [52] is an algorithm to extract boundary surfaces from a tetrahedralization separating differently labeled vertices from each other. It is a generalization of the marching cubes algorithm [53, 54] that initially allowed the creation of iso-surfaces at a selected isovalue by subdividing voxels such that areas with values above and below the isovalue were separated. Yet, in contrast to the original marching cubes algorithm, it allows multiple labels to be present at each tetrahedron and always extracts a distinct triangulation at each tetrahedron, eliminating ambiguous cases. The effect of various simplicial subdivisions on the quality of the resulting surfaces has been studied by Carr et al. [55]. Also, various applications use this approach for its simplicity and performance [56, 57, 58, 59].

3 PRELIMINARIES

This section describes the formal setting of our work. It contains definitions adapted from the Topology ToolKit (TTK) [8, 60]. We refer the reader to textbooks [61, 62] for comprehensive introductions to computational topology.

3.1 Input Data

The input is a piecewise linear (PL) scalar field $f : \mathcal{M} \to \mathbb{R}$ defined on a d-dimensional simplicial complex, with $d \leqslant 3$ in our applications. The *star* $St(\sigma)$ of a simplex σ is the set of simplices of $\mathcal{M}$ which contain σ as a face. The *link* $Lk(\sigma)$ is the set of faces of the simplices of $St(\sigma)$ which do not intersect σ. The input field f is provided on the vertices of $\mathcal{M}$ and interpolated on the simplices of higher dimensions. f is assumed to be injective, which is achieved in practice by substituting the f value of a vertex by its position in the non-ambiguous, global vertex order (by increasing f values).

3.2 Critical Points

The sub-level set $f^{-1}_{-\infty}(w)$ of an isovalue $w \in \mathbb{R}$ is defined as $f^{-1}_{-\infty}(w) = \{p \in \mathcal{M} \mid f(p) < w\}$. As w continuously increases, the topology of $f^{-1}_{-\infty}(w)$ changes at specific vertices of $\mathcal{M}$, called the *critical points* of f. Let $Lk^-(v)$ be the *lower link* of the vertex v: $Lk^-(v) = \{\sigma \in Lk(v) \mid \forall u \in \sigma : f(u) < f(v)\}$. The *upper link* of v is defined symmetrically: $Lk^+(v) = \{\sigma \in Lk(v) \mid \forall u \in \sigma : f(u) > f(v)\}$. A vertex v is *regular* if and only if both $Lk^-(v)$ and $Lk^+(v)$ are simply connected. Otherwise, v is a *critical vertex* of f [13]. A critical vertex v can be classified by its *index* $\mathcal{I}(v)$, which is 0 for minima, 1 for 1-saddles, $(d-1)$ for $(d-1)$-saddles and d for maxima. Vertices for which the number of connected components of $Lk^-(v)$ or $Lk^+(v)$ are greater than 2 are called *degenerate saddles*.

3.3 Integral Lines

Integral lines are piecewise linear curves on $\mathcal{M}$ which locally describe the gradient of f. They can be used to capture and visualize adjacency relations between critical points. Given a vertex v, its *forward* integral line, noted $\mathcal{L}^+(v)$, is a path along the edges of $\mathcal{M}$, initiated in v, such that each edge of $\mathcal{L}^+(v)$ connects a vertex v' to its highest neighbor v''. Then forward integrals are guaranteed to terminate in local maxima of f. A *backward* integral line, noted $\mathcal{L}^-(v)$, is defined symmetrically (i.e. integrating downwards towards minima).

Moreover, we define a *forward extremal integral line* as a forward integral line started at a connected component of upper link $Lk^+(s)$ of a saddle s. Backward extremal integral lines are defined symmetrically. We say that a saddle s is a *forward separating saddle* if there exist at least two forward extremal integral lines starting at s which terminate in distinct local maxima. Backward-separating saddles are defined symmetrically. In practice, extremal integral lines help capture adjacency relations between critical points.

3.4 Morse-Smale Segmentation

In this section, we formalize the notion of Morse-Smale segmentation computed by our approach.

For a given vertex v, let m and M be its *integration extremities*: m is the local minimum reached by the backward integral line started in v, while M is the local maximum reached by the forward integral line started in v. We now introduce an equivalence relation $v_1 \sim v_2$ between two vertices v_1 and v_2, which holds if their integration extremities are identical. The *Morse-Smale (MS) segmentation* is then a decomposition of the set of vertices of $\mathcal{M}$ into maximal subsets $\mathcal{M}_i$, called *MS regions*, such that for all pairs of vertices $(v_1, v_2) \in \mathcal{M}_i$, we have $v_1 \sim v_2$.

Let τ be a d'-simplex of $\mathcal{M}$ (with $0 \leq d' < d$), which only contains vertices belonging to a single MS region $\mathcal{M}_i$. If the link $Lk(\tau)$ includes vertices which do *not* belong to $\mathcal{M}_i$, we say that τ is a boundary simplex for $\mathcal{M}_i$. Then the *region boundary* of $\mathcal{M}_i$ is the simplicial complex formed by the union of all the boundary simplices of $\mathcal{M}_i$ (and their faces). Each *region boundary* separates $\mathcal{M}_i$ from the remaining dataset. The *region separators* separate all regions $\mathcal{M}_i \in \mathcal{M}$ from each other. To create them, every d-simplex of $\mathcal{M}$ that contains vertices belonging to at least two distinct MS regions spawns $(d-1)$-simplices inside its convex hull, as depicted in Fig. 4 and Fig. 5.

3.5 Discrete Morse Theory

We now conclude this section of preliminaries with notions (adapted from [63]) of discrete Morse theory [11], or DMT for short, as it has become a central component in modern implementations of the notion of Morse-Smale complex. We discuss the key differences between the Morse-Smale complex and the structures extracted by our approach (formalized in Secs. 3.3 and 3.4).

A *discrete vector* is a pair formed by a simplex $\sigma_i \in \mathcal{M}$ (of dimension i) and one of its co-facets σ_{i+1} (i.e. one of its co-faces of dimension $i+1$), noted $\{\sigma_i < \sigma_{i+1}\}$. σ_{i+1} is usually referred to as the *head* of the vector, while σ_i is its *tail*. Examples of discrete vectors include a pair between a vertex and one of its incident edges or a pair between an edge and a triangle containing it. A *discrete vector field* on $\mathcal{M}$ is then defined as a collection $\mathcal{V}$ of pairs $\{\sigma_i < \sigma_{i+1}\}$, such that each simplex of $\mathcal{M}$ is involved in at most one pair. A simplex σ_i which is involved in no discrete vector $\mathcal{V}$ is called a *critical simplex*.

A *v-path* is a sequence of discrete vectors $\{\{\sigma_i^0 < \sigma_{i+1}^0\}, \ldots, \{\sigma_i^k < \sigma_{i+1}^k\}\}$, such that *(i)* $\sigma_i^j \neq \sigma_i^{j+1}$ (i.e. the tails of two consecutive vectors are distinct) and *(ii)* $\sigma_i^{j+1} < \sigma_{i+1}^j$ (i.e. the tail of a vector in the sequence is a face of the head of the previous vector), for any $0 < j < k$. A *v-path* can be interpreted as the discrete analog to the notion of PL integral line introduced in Sec. 3.3. We say that a v-path *terminates* at a critical simplex σ_i if σ_i is a face of the head of its last vector $\{\sigma_i^k < \sigma_{i+1}^k\}$. Symmetrically, we say that a v-path *starts* at a critical simplex σ_{i+1} if σ_{i+1} is a co-facet of the tail of its first vector $\{\sigma_i^0 < \sigma_{i+1}^0\}$. Then, the collection of all the v-paths terminating in a given critical simplex σ_i is called the *discrete stable set* of σ_i and is noted $\mathcal{M}(\sigma_i)$. Symmetrically, the collection of all the v-path starting at a given critical simplex σ_i is called the *discrete unstable set* of σ_i and is noted $\mathcal{M}'(\sigma_i)$.

A *discrete gradient field* is then a discrete vector field such that all its possible *v-paths* are loop-free. Several algorithms have been proposed to compute such a discrete gradient field from an input PL scalar field (see [23] for instance). The *discrete Morse complex* is then defined as the complex formed by the discrete unstable sets of all the critical simplices. It is a cell complex made of d'-dimensional cells (with $d' \in \{0, 1, \ldots, d\}$), such that each d'-dimensional cell is the discrete unstable set of a critical d'-simplex. The *opposite discrete Morse complex* is defined symmetrically, i.e. it is the cell complex formed by the discrete stable sets of all the critical simplices. Finally, the *discrete Morse-Smale complex* is defined as the complex formed by the intersections of the cells of the discrete Morse complex and the opposite discrete Morse complex.

Several conceptual differences exist between the Morse-Smale complex and the Morse-Smale (MS) segmentations considered in our work. First, as their name suggests, MS segmentations only provide vertex-based decompositions of the input domain, not a cell complex that exhaustively and precisely captures all possible adjacency relations between integral lines (formally v-paths). Thus, MS segmentations target a subset of the applications enabled by the Morse-Smale complex (specifically, involving data segmentation). While the separatrices of the MS regions (Sec. 3.4) resemble the 2-dimensional cells of the Morse-Smale complex, they only correspond to the unstable sets of *separating* saddles (Sec. 3.3), which constitutes a subset of all the saddles (i.e. saddles where isosurfaces change their genus are not considered). Finally, note that in DMT, local maxima (critical d-simplices) cannot strictly occur on the boundary of $\mathcal{M}$, which only includes d'-simplices (with $d' < d$).

4 METHOD

In this section, the algorithms for the computation of the PLMSS are described in detail. First, necessary preprocessing steps and data structures are presented. Then the ascending and descending segmentation of the domain is described, followed by the computation of the MS segmentation.

4.1 Preprocessing

To prevent ambiguity during the computation of integral paths, we apply a variant of *Simulation of Simplicity* [64] on the input scalar field f. We first sort all vertices of the domain according to their scalar value, where we resolve ties based on the indices of the compared vertices. Then, we derive the so-called order field $\bar{f}$ that records for each vertex its index in this sorted array. Note, that each critical point of f is also a critical point of $\bar{f}$, but $\bar{f}$ might exhibit additional critical points that result from the disambiguation. These spurious critical points, however, can be removed via topological simplification, which we apply in order to remove non-persistent critical points from the scalar field. For a detailed discussion on topological simplification and its implementation in TTK, we refer the reader to the work of Lukasczyk et al. [36].

The advantage of processing an order field over the original input scalar field is that $\bar{f}$ is injective, i.e., every vertex has a distinct largest and smallest neighbor in the order field. It is only possible that a vertex has no neighbor with a larger or smaller order value, in which case the vertex is a maximum or minimum, respectively. Hence, there is always a distinct direction of steepest ascent and descent, which is essential for the computation of the ascending and descending manifolds, described next.

4.2 Segmentation and Extrema Retrieval

The segmentation of the domain is a two-step process. In the first step, the ascending and descending segmentations are created; representing areas of influence of minima and maxima, respectively. These segmentations are intersected to create the MS segmentation, representing the areas of influence of minimum-maximum pairs.

4.2.1 Ascending and Descending Segmentation

MS segmentations subdivide a domain into areas of similar flow behavior, meaning that forward and backward integration for any vertex in the same region leads to the same extremum pair. This means that each MS subset of the domain corresponds to all steepest descent/ascent paths that terminate in the same pair of extrema. To achieve this, first, every vertex has to be assigned to its minimum and maximum. Therefore, the ascending (asc) and descending (dsc) segmentations of the domain are computed. As the process is the same for both directions, without loss of generality, it will be described for the descending segmentation.

Maximum assignment for each vertex can be achieved by iteratively finding the largest neighbor's largest neighbor. As this process is lengthy, taking many steps to converge to the maximum, path compression [65] is used to double the step size in each iteration. Fig. 2 gives an example path compression run using 7 ordered vertices.

The segmentation computation starts by assigning each vertex v to its largest neighbor in the triangulation according to the order field function value $dsc(v) = argmax_{x \in N(v)} \bar{f}(x)$, allowing for a fast lookup later in the process. $N(v)$ is the set of all vertices that are connected to v via an edge in the triangulation.

At the same time, maxima can be extracted by recording cases where no larger neighbor is found. For further processing, each vertex that is not a maximum is written to a list of vertices L_0 that did not find their maximum yet.

In the second step, the maximum for each vertex in L_0 is found using path compression. Here, the value of each vertex gets assigned to the largest neighbor's largest neighbor $dsc(v) = dsc(dsc(v))$. This allows doubling the step size towards the maximum in each iteration. If the corresponding maximum is not found $dsc(v) \neq dsc(dsc(v))$, the vertex did not converge to its maximum and is written to a second list L_1. After fully iterating over L_0, the process starts again using L_1, i.e. $L_0 = L_1$. If $L_1 = \emptyset$ after iterating over L_0 the maximum $dsc(v)$ is found for every vertex v.

In parallel environments, each vertex can be evaluated independently with little communication in between iterations, as both steps iterate over a set of vertices. Here, the first step of finding the largest neighbor is equally distributed such that every thread executes the same amount of vertices. Still, every thread t keeps a local list of active vertices, i.e. L_{0t}, L_{1t}, executing the following iterations independently for each thread. It is also possible to compute the ascending and descending segmentations simultaneously, further improving performance. To do this, both the largest and smallest neighbors are found at the same time in the first step, and a vertex is added to L_1 if it did not converge in both directions in the second step.

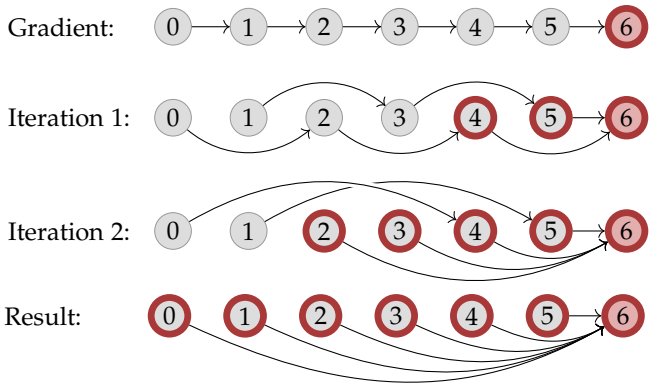


Fig. 2. Path compression example showing vertices as circles and current vertex assignment as arrows. The number attached to a vertex is the order field function value, and the outer ring shows if a vertex converged (red). The gradient step assigns the largest neighbor to each vertex and each following Iteration sets the neighbor's neighbor for active vertices. Please note that the order in which every iteration is executed matters. In this example, it starts with the smallest active vertex and continues in an increasing fashion. If iteration 1 would start with the largest vertex in decreasing order the assignment would already terminate after the first iteration.

4.2.2 Morse-Smale Segmentation

The ascending and descending segmentation obtained by the algorithm above can be combined into an MS segmentation. As the descending segmentation assigns a maximum to each vertex and the ascending segmentation assigns a minimum to each vertex, vertices with the same minimum and maximum are assigned to the same MS id. Therefore, the extremum pair combination is written to each vertex as a tuple, allowing access to the involved extrema. This process is trivial to parallelize, as each thread can independently write the MS ids for its vertices.

4.3 Multi-Label Marching Triangles/Tetrahedra

To visually divide MS regions from each other, region-separating geometries can be created between the regions. As the triangulation consists of triangles in the 2D case and tetrahedra in the 3D case, both cases have to be treated in slightly different ways. In 2D, region-separating geometry is created using edges that split triangles with multiple labels, whereas in 3D, triangles are utilized to separate the vertices of multi-label tetrahedra. Like marching tetrahedra [66], each tetrahedron or triangle is evaluated independently, considering the labels at its vertices for generating the bisecting geometry.

4.3.1 Triangles

In the 2D case, a triangle can either have 1, 2, or 3 unique labels at its vertices. In the case of 1 label, no edges have to be generated as the vertices belong to the same region. When 2 different labels are present, one vertex a has a different label than the other two vertices b, c. Here, as shown in Fig. 4, the centers of the edges connecting a to b and a to c are used as the endpoints for the edge that splits the labels. In the case of 3 unique labels, an edge is created from the triangle center to all three of its edges.

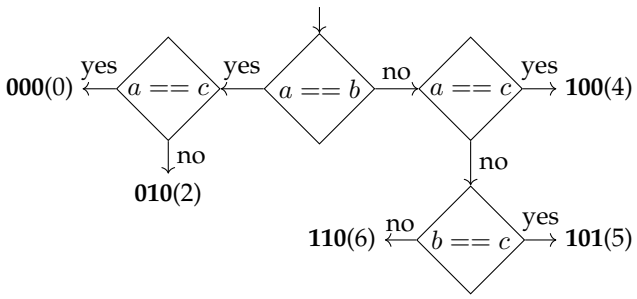


Fig. 3. Decision tree for the triangle binary code creation. Each rhombus represents a decision, each node shows the resulting code with the bit representation in bold and integer representation in brackets.

Computationally, this is achieved using a lookup table that describes every possible configuration in a triangle. Therefore, a 3-bit binary code with values in the range of $\{0, .., 6\}$ is created to describe the current triangle configuration, utilizing the labels at the three vertices a, b, c of the considered triangle, converted to a dense local representation such that $a = 0, b \in \{0, 1\}, c \in \{0, 1, 2\}$. The label a is always considered to be 0, b can either be 0 or 1 depending on the equality to label a, and c can either be 0, 1, or 2 depending on the equality to labels a and b. Therefore, $b \in \{0, 1\}$ determines the left bit and $c \in \{0, 1, 2\}$ determines the last two bits. Fig. 3 provides the decision tree. It should be noted that some binary codes cannot appear, as $c = 1$ can only be the case if $b = 1$. Also, codes like $011(3)$ are not possible in general, as there is no label 3.

Each of the five valid binary codes corresponds to a triangle configuration, as shown in Fig. 4. This allows retrieving the triangle edges that need to be connected. The whole procedure is well-scaling as it is executed per triangle.

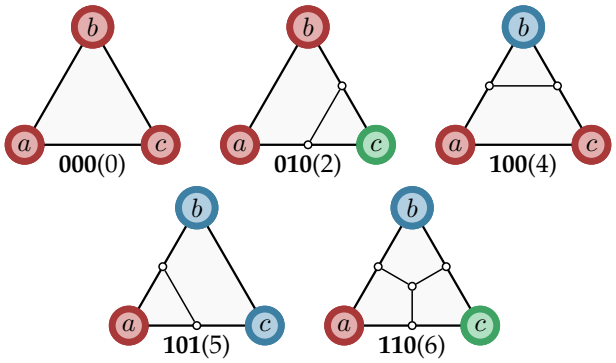


Fig. 4. All five valid cases for splitting a triangle. The label at the vertices of the triangles is drawn by color (red = 0, blue = 1, green = 2), showing binary code and integer representation at the bottom, and the resulting separating edge(s) in the middle. White circles mark the points of intersection on the edges of the triangle.

4.3.2 Tetrahedra

In 3D, the domain is subdivided into tetrahedra. Therefore, up to 4 unique labels a, b, c, d can be present at the vertices of a tetrahedron. Similar to the triangle case, a 5-bit binary code is created and translated into a tetrahedron configuration. This configuration is used to create a consistent triangulation that separates unique labels from each other.

The binary code for tetrahedra is created by an extended logic. a is considered to be 0 again, $b \in \{0, 1\}$ determines the left bit, $c \in \{0, 1, 2\}$ determines the next 2 bits and $d \in \{0, 1, 2, 3\}$ determines the last two bits. If the label of a vertex with lower index matches, its label is used as the resulting label, otherwise the index of the own vertex is used. All valid configurations are provided in Tab. 1. Some binary codes are invalid, as some labels might not exist and can not be assigned to a vertex of a higher index. E.g. $00010(2)$ is not possible as d would have label 2, but c has label 0, making label 2 non-existent in this configuration.

TABLE 1
All valid binary codes, the codes converted to an integer, the number of unique labels of the tetrahedron, and the value of each label.

Binary code	Case	#Labels	a	b	c	d
00000	0	1	0	0	0	0
00011	3	2	0	0	0	3
01000	8	2	0	0	2	0
01010	10	2	0	0	2	2
01011	11	3	0	0	2	3
10000	16	2	0	1	0	0
10001	17	2	0	1	0	1
10011	19	3	0	1	0	3
10100	20	2	0	1	1	0
10101	21	2	0	1	1	1
10111	23	3	0	1	1	3
11000	24	3	0	1	2	0
11001	25	3	0	1	2	1
11010	26	3	0	1	2	2
11011	27	4	0	1	2	3

After the binary code is determined, a lookup table is used to retrieve the resulting separating triangles utilizing the edge, triangle, and tetrahedra centers to be connected. This allows for a fast triangulation of the tetrahedra, as the resulting triangle vertices are directly retrieved. Fig. 5 shows the resulting triangulation for all cases ignoring permutations and rotations. The triangulation across tetrahedra is always consistent, as the triangle labels of the tetrahedron mimic the 2D case, i.e. triangles are either split by connecting their edge centers to their triangle center, or two triangle edges are connected. Therefore, the triangle connecting two incident tetrahedra is always split in the same way and the resulting triangulation separates labels from each other without any holes in the geometry.

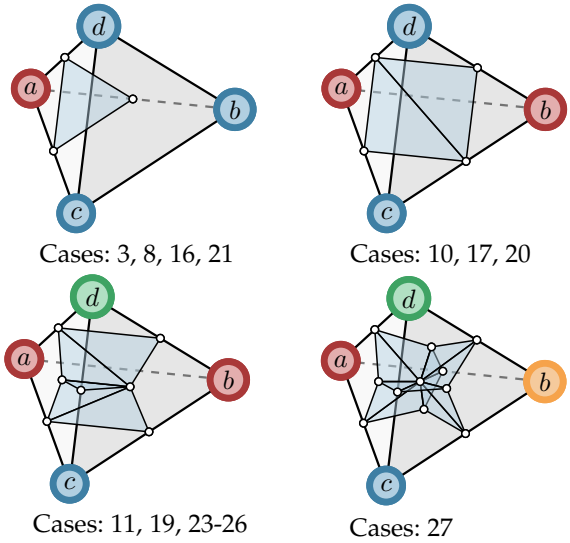


Fig. 5. Triangulation for tetrahedra with more than 1 unique label ignoring permutations and rotations. The case number below the tetrahedra refers to Tab. 1.

4.3.3 Triangulation Options

As the information requested slightly varies from user to user, two region-separating geometries are available. The user can utilize any segmentation (ascending, descending, and Morse-Smale) together with either region boundaries or region separators. Fig. 1 showcases a closeup view of the segmentations (b-e), where (b) and (d) show region separators, and (c) and (e) show region boundaries.

Segmentation Selection: For some applications, a certain segmentation will be of great interest. Here, either the MS, ascending, or descending segmentation can be chosen to deliver the labels for the marching tetrahedra algorithm. Instead of using the MS segmentation, it is also possible to show the union of the ascending and descending segmentation, producing intersecting geometry at the meeting points of both region-separating geometries.

Region Separating Geometries: Both the region separators and the region boundaries have slightly different use cases and allow to map different information to its geometry. The region separators divide all regions, or areas of influence of minima-maxima pairs, from each other by building geometry between them. Therefore, information about the minima and maxima involved in each separating triangle can be displayed to the user. Each subset of the surface that separates the same two regions can be extracted here. Still, some overhead is involved in computing the region

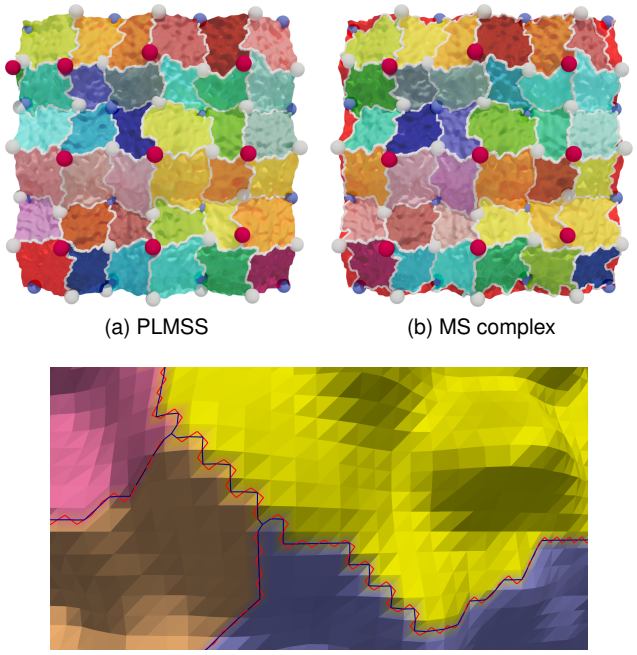
separators, as many triangles have to be used to separate the regions from each other, depicted in Fig. 5.

Region boundaries allow extracting the hull of regions or areas of influence of minima-maxima pairs. They use all tetrahedra with 3 vertices of the same label. Those 3 vertices form a triangle in the input simplicial complex that can be directly used as a separating geometry. This option will result in faster computation times and allows the extraction of geometry for each region.

5 Results

In this section, three example datasets and their region-separating geometry outputs for both algorithms are provided. The Noisy Terrain dataset in Sec. 5.1 is used to highlight differences between both algorithms in the 2D case. Sec. 5.2 shows that both algorithms produce a very similar output when no saddle-saddle 2-cells are present. The last two datasets indicate that the MS complex is providing more geometry than necessary to effectively extract useful information in many cases, while the PLMSS can extract the areas of influence without additional preprocessing.

5.1 Noisy Terrain



(a) PLMSS (b) MS complex

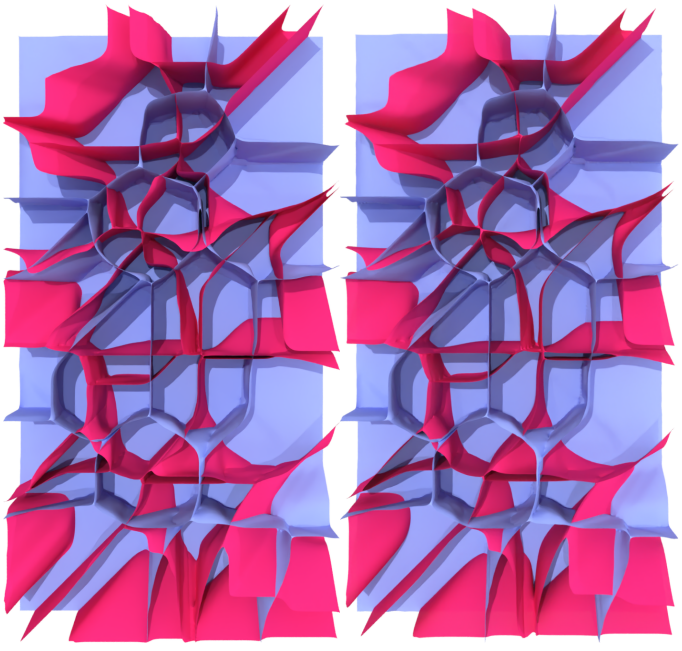
(c) Region separators

Fig. 6. Noisy Terrain dataset [67] showing the PLMSS and MS complex next to each other. Both show critical points (blue = Minima, white = Saddles, red = Maxima), separating geometries in white, and the surface colored by MS complex ids. The border region of the MS complex is not fully segmented as a vanilla implementation of the expansion-based discrete gradient computation algorithm [23] may miss maxima on the boundary, whereas the boundary of the PLMSS is fully segmented. (c) visually compares the region-separating geometry of the PLMSS (blue) and the MS complex (red). The MS complex produces geometries with a characteristic step-function shape (red), whereas the PLMSS produces separating geometries with a linear-slope shape (blue).

The Noisy Terrain dataset is a triangulated surface with elevation scalars attached, chosen as an illustrative example. It has a 300x300 resolution and can be found in the TTK data repository [67]. Generally, the dataset consists of hills and valleys on a regular grid, where the hills are getting smaller the closer they are to the border. Additionally, noise was added to the terrain to showcase topological simplification.

Fig. 6 compares the PLMSS with the MS complex, using the same color coding for critical points, separating geometries, and segmentation in both versions. Slight differences can be detected regarding the separating geometries, where the MS complex separating geometries are defined on the dual graph and are connecting triangle centers in the 2D case. Concerning the PLMSS, triangles are split according to the labels at the triangle vertices. Another difference can be spotted at the borders of the MS complex, where great sections of the border are not labeled, as a vanilla implementation of the expansion-based discrete gradient computation algorithm of Robins et al. [23] (implemented in TTK) may miss PL maxima on the domain boundary (local post-processing of the discrete gradient would be required to enforce the detection of discrete maxima in the star of boundary PL maxima). As the PLMSS assigns a maximum-minimum pair to each vertex, every vertex is properly labeled without skipping the boundary region.

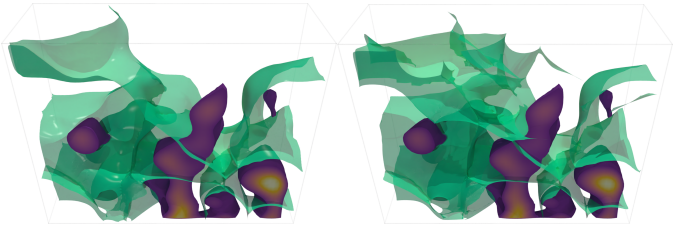
5.2 AT Molecule



(a) PLMSS (b) MS complex

Fig. 7. Comparison between the region separators computed by PLMSS (a), and the MS complex 2-cells computed by TTK (b) using the AT dataset. The resulting surfaces are colored by the segmentation type, i.e., red for the descending and blue for the ascending segmentation. To ease visual comparison, both surfaces were smoothed 20 times using TTK's geometry smoother. Both surfaces are almost visually identical.

The AT dataset from the TTK Tutorial Data [68] shows the simulation of the electron density of a molecule restricted to a plane but embedded in 3D space. This example, provided in Fig. 7, shows that the PLMSS and MS complex extract the same underlying geometry at heart if no saddle-saddle 2-cells are present in the dataset. To allow an easier



(a) PLMSS (b) MS complex

Fig. 8. Comparison of the PLMSS region boundaries and the MS complex using the Viscous Fingering dataset [67] simplified with an absolute persistence threshold of 0.1. Both images show the boundary interface of viscous fingers as contours of the salt concentration density scalar field, colored by the density from yellow (high concentration) to purple (low concentration). (a) shows that the PLMSS region boundaries can extract the region-separating geometries that separate single viscous fingers effectively without cluttering the visualization. (b) provides the original MS complex, where more geometry is extracted due to the saddle-saddle 2-cells, cluttering the image. Those saddle-saddle walls would have to be removed by additional postprocessing.

comparison of the separating geometries, all geometries were colored by the underlying segmentation (ascending in red and descending in blue) and smoothed. Most of the separating geometry coincides with at most 1 tetrahedron space in between both representations. The largest difference can be seen in the two narrow red geometries on the middle right of the images, as their distance is smaller when using the PLMSS. The difference between both representations themselves is a result of the dual graph definition of MS complex, compared to the per-vertex definition of the PLMSS.

5.3 Viscous Fingering

The Viscous Fingering dataset [67] represents the result of finite pointset method simulations that simulate the mixing of salt solutions inside water. Regions of high salt concentration form structures called viscous fingers. The analysis of such structures usually involves Reeb graphs or iso-surfaces to identify single fingers in the dataset [69]. The MS complex suffers from additional saddle-saddle 2-cells in such scenarios that stem from discrete Morse theory, complicating effective analysis. Filtering these saddle-saddle 2-cells out of the set of 2-cells is usually expensive as further postprocessing is required to identify and simplify the saddles responsible for such 2-cells. The PLMSS, on the other hand, segments the data into regions of influence of maxima-minima pairs, providing a region for each partial viscous finger. The granularity of these separating geometries can be controlled by topological simplification, allowing users to achieve the desired level of detail of the segmentation.

Fig. 8 shows the Viscous Fingering dataset from the TTK data repository [67] with an applied persistence threshold of 0.1. The colored isosurfaces provide the finger surfaces that correspond to regions of high salt concentration. To allow a deeper look into the dataset, it was clipped in the middle after all geometries were created. Figure 8a shows that the PLMSS effectively separates fingers from each other, providing the area of influence of the maxima and hence, for the viscous fingers. The region separators are used to extract the area of influence of single fingers to analyze the area they are growing into with increasing salt concentration. Fig. 8b provides the MS complex. As for additional saddle-saddle 2-cells, the image is less clear and more cluttered, which can be problematic with noisy or large datasets. Those saddle-saddle walls would have to be removed by additional postprocessing.

5.4 Rayleigh-Taylor instability (Miranda)

A concept of controlled fusion using hydrogen isotopes in a laser-lighted fuel capsule lead to the discovery of Rayleigh-Taylor instability [70] at the boundary of the capsule. The simulation models the heating process of two hydrogen isotopes for fusion burn. The energy from the laser is non-uniformly distributed and causes small perturbations that quickly grow. One time step of a simulation is analyzed using the MS complex of TTK and the PLMSS.

To filter noise from the data, an absolute persistence threshold of 0.1 was applied by utilizing localized topological simplification [36]. By solely extracting the border surfaces of the area of influence created by maximum-minimum pairs, the PLMSS removes clutter from the separating geometries of the MS complex due to the missing saddle-saddle 2-cells.

Fig. 9 compares the visual results of the PLMSS with the MS complex, using one timestep of a Rayleigh-Taylor instability simulation. Here, the PLMSS manages to extract the area of influence of minima and maxima in the dataset. The region-separating geometry is very structured and the boundary between regions of interest can be identified. In contrast, the MS complex introduces a lot of noise due to the remaining saddle-saddle 2-cells that clutter the resulting image. Additionally, the maxima on the boundary are missing, as a vanilla implementation of the expansion-based discrete gradient computation algorithm of Robins et al. [23] (implemented in TTK) may miss PL maxima on the domain boundary.

6 Performance

In this section, the computational performance of our implementation is analyzed and compared to the MS complex implementation of TTK [60] and MSCEER [9]. As hinted by the authors of MSCEER, we use the "steepest_lstar" and "extractms" packages, as they supply the fastest implementation without accurate geometry. Both strong scaling studies show that the PLMSS is scaling well with core count due to the mentioned improvements. We utilize the Rayleigh-Taylor instability (Miranda) dataset [70] at a resolution of 512^3 and the Foot dataset [67] at a resolution of 256^3 for the computation speed comparisons of all three algorithms.

6.1 Algorithmic improvements

In general, three main aspects of the PLMSS result in a strongly reduced computation time as compared to the MS complex. First, the segmentation of the domain is improved by path compression, which is much faster than computing a discrete gradient field. Second, the multi-label marching tetrahedra algorithm supports computing the separating geometries in a well-scaling way. Third, splitting the marching tetrahedra algorithms into indexing and geometry creation steps allows the allocation of the resources needed in the geometry creation step without additional computations.

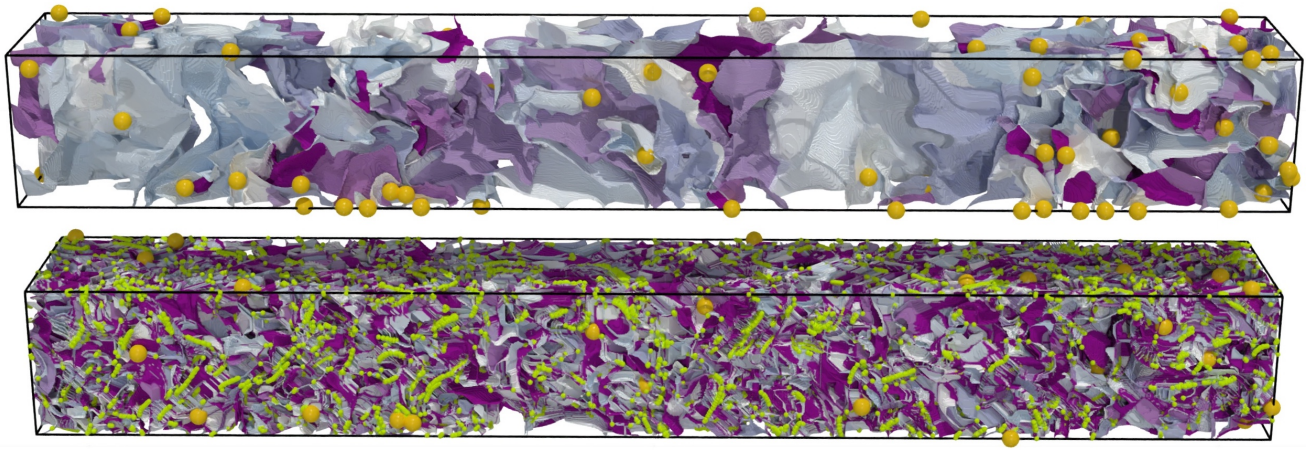


Fig. 9. Illustration showing the results of the piecewise linear Morse-Smale segmentation (PLMSS) (top) and the Morse-Smale (MS) complex (bottom) utilizing a Rayleigh-Taylor instability simulation [70] dataset. The image shows extrema as large orange spheres, additionally providing saddles as green small spheres in the MS complex, as saddle-saddle separatrices are the actual reason for the cluttered MS complex visualization. Even though filtering out saddle-saddle separatrices is possible, the computational overhead will favor the PLMSS in situations where saddle-saddle separatrices hide important features.

TABLE 2
Raw timing data of the MSCEER, TTK, and PLMSS algorithms in seconds. For each timing, the test was executed 10 times, removing the best and worst time regarding the time listed in the last row, and averaging the remaining runs. It should be noted that the top-level cell count is 6 times higher with TTK and PLMSS than with MSCEER. Similarly, the total number of cells in the input simplicial complex are differing by a factor of roughly 3.24 (Miranda MSCEER: 1,070,599,167 / TTK, PLMSS: 3,473,956,851) (Foot MSCEER: 133,432,831 / TTK, PLMSS: 432,287,731).

		Miranda 512^3						Foot 256^3					
Task	Algo.	1T	2T	4T	8T	16T	24T	1T	2T	4T	8T	16T	24T
DGF	TTK	1,326.76	685.52	365.00	200.15	112.10	84.55	152.65	78.00	42.45	23.32	13.32	10.08
	MSCEER	116.51	67.33	34.43	20.25	15.11	12.80	20.54	12.25	7.78	5.67	4.94	4.86
	PLMSS	-	-	-	-	-	-	-	-	-	-	-	-
Asc/Desc	TTK	442.15	269.68	197.04	167.68	158.19	152.42	35.12	28.23	24.54	22.92	22.38	22.63
	MSCEER	167.60	99.48	53.12	28.71	15.89	11.63	34.77	24.07	21.35	19.88	18.97	19.25
	PLMSS	39.61	20.08	10.64	5.73	3.07	2.31	4.40	2.23	1.16	0.68	0.43	0.36
MS	TTK	4.78	2.44	1.40	1.02	0.73	0.66	0.46	0.23	0.18	0.16	0.11	0.10
	MSCEER	-	-	-	-	-	-	-	-	-	-	-	-
	PLMSS	0.32	0.16	0.09	0.05	0.03	0.02	0.04	0.02	0.01	0.01	0.00	0.00
Index	TTK	152.24	90.30	46.73	24.59	12.80	9.43	26.73	16.49	8.83	4.58	2.56	2.07
	MSCEER	117.00	58.92	30.87	16.20	8.68	6.24	15.70	8.37	4.90	3.00	2.07	1.77
	PLMSS	39.00	19.62	10.24	5.28	2.72	1.89	4.81	2.42	1.27	0.67	0.41	0.29
Geometry	TTK	491.21	263.75	143.22	83.25	50.67	41.04	94.11	49.84	27.69	16.05	10.02	8.25
	MSCEER	-	-	-	-	-	-	-	-	-	-	-	-
	PLMSS	8.99	6.68	4.12	2.81	2.03	1.79	0.59	0.42	0.27	0.18	0.13	0.14
DGF + Index + Asc/Des	TTK	1,921.16	1,045.49	608.77	392.42	283.09	**246.39**	214.50	122.71	75.82	50.81	38.26	**34.78**
	MSCEER	401.12	225.73	118.43	65.16	39.68	**30.67**	71.02	44.70	34.03	28.56	25.98	**25.87**
	PLMSS	78.61	39.70	20.89	11.01	5.79	**4.20**	9.21	4.65	2.43	1.35	0.84	**0.65**

6.1.1 Segmentation

The segmentation of the domain into the ascending and descending manifold and the intersection of both manifolds, called MS manifold, are computed differently for the MS complex and the PLMSS. Both MS complex implementations require a discrete gradient field to be computed first, then following the v-paths along the gradient field to assign labels to each vertex. However, the PLMSS segmentation utilizes path compression to assign each vertex to its designated minimum or maximum, without the need for any additional structure other than the order field.

For path compression to work, all neighbors of each vertex must be visited once to get the largest and smallest neighbor of that vertex. With this information, the maximum can be found iteratively by assigning its largest neighbor's largest neighbor to the vertex. This process is executed in multiple iterations, where each iteration finds the designated maximum of a vertex or a vertex closer to the designated maximum. Here, each time the step length is doubled, yielding extremum assignment in $log(s)$ steps for each vertex, where s is the number of vertices on the integral line of the vertex to the extremum. Also, the iterations get smaller every time, as more and more vertices are assigned to their extremum.

Tab. 2 shows the timings of those steps in the first three rows, where "DGF" refers to the discrete gradient field computation, "Asc/Desc" refers to the ascending and descending segmentation, and "MS" refers to the MS segmentation. Please note that the MSCEER algorithm does not compute an MS segmentation in the provided implementation.

Even in a single-threaded environment, the performance gains of retrieving the ascending and descending segmentations already show strong improvements in the computation

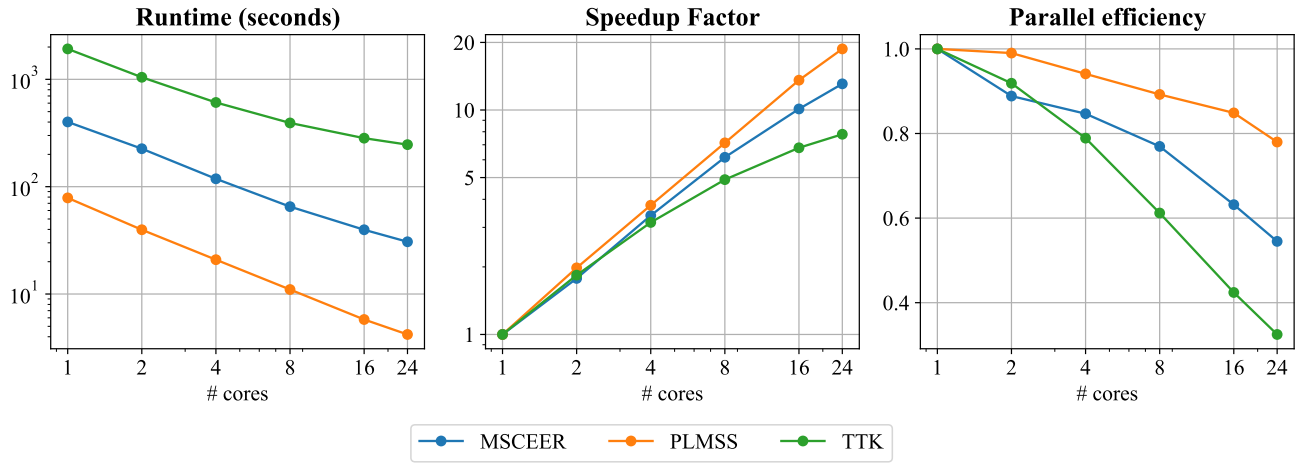


Fig. 10. Rayleigh-Taylor instability dataset [70] performance of all three algorithms at a resolution of 512^3. The left graph shows the runtime sum of the DGF, ascending and descending segmentation, and indexing as a log-log plot regarding the number of cores used for the experiment. In the middle, the speedup factor, i.e. the runtime of 1 thread divided by the runtime of x threads, is also shown as a log-log plot regarding core counts. On the right, the parallel efficiency, i.e. the speedup factor divided by the number of cores, is represented in a semi-log plot.

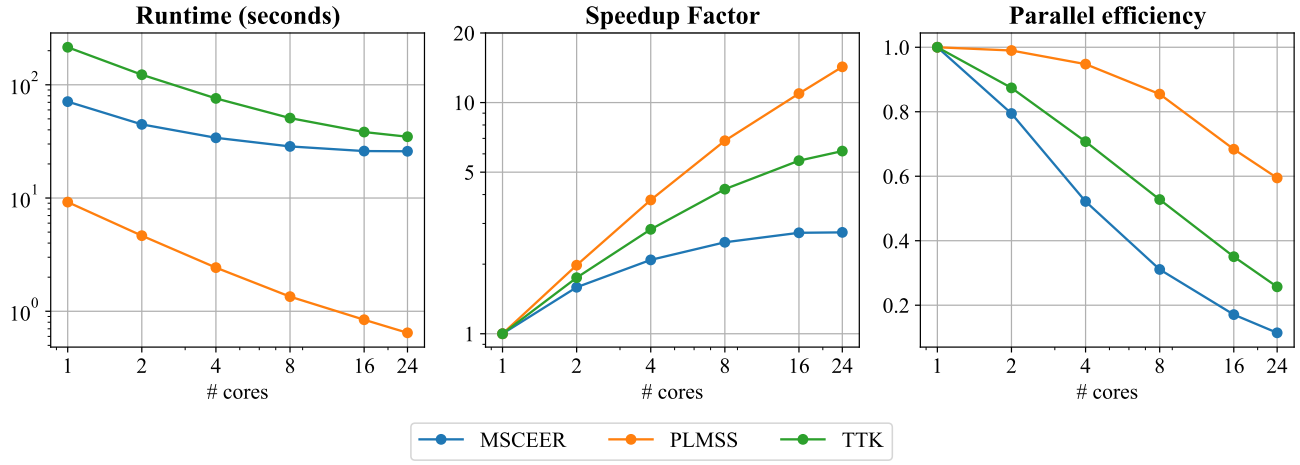


Fig. 11. Foot dataset [67] performance of all three algorithms at a resolution of 256^3. The left graph shows the runtime sum of the DGF, ascending and descending segmentation, and indexing as a log-log plot regarding the number of cores used for the experiment. In the middle, the speedup factor, i.e. the runtime of 1 thread divided by the runtime of x threads, is also shown as a log-log plot of core counts. On the right, the parallel efficiency, i.e. the speedup factor divided by the number of cores, is represented in a semi-log plot.

time of the PLMSS compared to the MS complex implementations. For the Miranda dataset, PLMSS only needs a total of 39.61s for the computation of the ascending and descending segmentation. Comparing this to the MS complex implementations (MSCEER: 284.11s / TTK: 1768.91s) leaves us at a speedup of 7x and 44x respectively.

6.1.2 *Multi-Label Marching Tetrahedra*

After segmenting the domain into various regions, region-separating geometries can be created that help to visualize the segmentation effectively. Again, the PLMSS and the MS complex implementations differ in the realization of this step. Regarding the MS complex implementations, paths between critical point pairs have to be traced on the DGF, whereas the PLMSS uses a marching tetrahedra algorithm.

Marching tetrahedra algorithms scale very well as they are executed per vertex. In our implementation, the binary code, as described in Sec. 4.3, and the number of triangles created per thread are computed in a preliminary step. This allows the allocation of the exact amount of memory needed for the triangles. In a follow-up step, this enables direct writing of triangles to memory.

Tab. 2 shows the timings of those steps in the 4th and 5th row, where ”Index” refers to the computation of simplex indices that will spawn triangles, and ”Geometry” represents writing the triangles to memory. Please note that the MSCEER algorithm does not compute the geometry itself in the provided implementation, but only gathers the indices of relevant simplices.

A comparison of the single thread timings with the Miranda dataset shows that the indexing is almost four times faster compared to TTK and three times faster than MSCEER. These speedups come from the excessive use of lookup tables and a per tetrahedra execution that does not require any tracing of paths. With the additional memory counting in the indexing step and the lower number of trian-

gles created, the geometry-creating part strongly improved regarding computation times.

6.2 TTK vs. MSCEER

Both MS complex solutions are slightly different in the specifics of their implementation. First of all, TTK uses a simplicial complex that uses tetrahedra and triangles as top-level simplices, whereas MSCEER uses cubes and squares. This already leads to roughly 324% more total cells and six times more top-level cells in the case of the TTK-based implementation, such as the MS complex or PLMSS. This will have an effect on the runtime of the algorithms, as well as the resolution of the extracted region-separating geometries. Still, this is only a limitation of the current TTK version and might change in the future to trade accuracy for runtime efficiency.

Another issue, when comparing both algorithms with each other, is the output generated by each implementation. The version provided by Gyulassy et al. [9] only provides an ascending and descending segmentation of the domain and the indices of top-level cells that would spawn region-separating geometries. The MS complex representation and the surface geometries would be created by a different software at runtime that was not supplied with the library.

6.3 Strong Scaling Setup

Both strong scaling studies were carried out on a dual Intel XEON SP 6126 node with 24 CPU cores and 384GB of RAM. For each algorithm, various timings were created, starting with the computation of the discrete gradient field, ascending and descending segmentation, MS segmentation, indexing of tetrahedra or cubes that generate region-separating geometries, and writing the region-separating geometries to memory. The results for both studies are shown in Tab. 2, where the last row shows the total time to get an ascending and descending segmentation and mark all top-level cells that generate region-separating geometries. To mimic the layout of the 2-cells of the MS complex, the region separators of the PLMSS were computed for those results.

For each combination of the dataset, the number of threads, and the algorithm, the experiment was executed 10 times. From these 10 runs, the best and the worst ones, regarding total computation time, were discarded. The remaining 8 runs were averaged to achieve more stable results.

6.4 Strong Scaling: Rayleigh-Taylor Instability

Utilizing the Rayleigh-Taylor instability dataset [70], a strong scaling analysis was carried out for a 512^3 subset of the data, simplified with a persistence threshold of 0.1. Computation times, speedup factor (i.e. the time a single thread takes divided by the time the current number of threads take), and parallel efficiency (i.e. speedup factor divided by the number of threads) are plotted against the number of cores, shown in Fig. 10.

The total execution time in the last row of Tab. 2 shows that the PLMSS is more than an order of magnitude faster than TTK and 5 to 7 times faster than MSCEER. As the PLMSS still has to be executed on roughly three times the cells, this is still an improvement of an order of magnitude regarding the time per cell. The speedup factor and parallel efficiency also show a clear trend that PLMSS is scaling very well with more cores. In this regard, MSCEER beats TTK in terms of scalability but clearly performs worse against PLMSS. The speedup factor graph also hints that more cores would be beneficial in future experiments as PLMSS still scales well at 24 cores. Due to insufficient computational resources, we were unable to offer a larger strong scaling analysis.

6.5 Strong Scaling: CT Scan of a Human Foot

The foot dataset [67] was chosen to represent smaller data sizes with a resolution of 256^3, simplified with a persistence threshold of 110. It consists of a CT scan of the tip of a foot, where the threshold of 110 was chosen to represent each bone with its own region.

Regarding the total execution time at 24 Threads in Tab. 2, a speedup of over 50x and almost 40x (TTK and MSCEER) can be achieved using PLMSS. The resulting graphs in Fig. 11 also show clear improvements regarding parallel efficiency, as PLMSS $(0, 59)$ still achieves good results that hint towards using even more cores, where TTK $(0, 26)$ and MSCEER $(0, 11)$ are already in a range where more cores do not strongly improve runtime performance and communication overhead takes over.

6.6 Discussion

For both datasets, the PLMSS showed good improvements over the MSCEER and TTK implementation, yielding a great parallel efficiency. Especially, for the foot dataset, great runtime performance improvements of roughly 40x were achieved. Even with the 5-7x runtime improvement regarding the Rayleigh-Taylor instability dataset, 6x more top-level cells had to be traversed compared to MSCEER.

7 Limitations

Our entire approach aims at efficiently computing ascending and descending segmentations of the input scalar field. Its output is not a complete Morse-Smale complex. First, it does not capture saddle-saddle connectors, which may be useful in certain applications. Second, it does not output an explicit CW complex modeling the Morse-Smale complex (i.e. where vertices encode critical points, 1-dimensional cells encode separatrices, 2-dimensional cells encode separating geometries, and 3-dimensional cells encode regions with identical integration extremities), only the domain segmentation is provided. This can be detrimental in applications involving post-processing of the Morse-Smale complex, such as regular remeshing [71] or hierarchical simplification [72]. For these applications, a standard algorithm based on discrete Morse theory (such as the one available in TTK [60] or MSCEER [9]) should be preferred.

8 Conclusion and Future Work

The presented algorithm describes a well-scaling approach to computing MS segmentations, allowing for speedups of more than an order of magnitude compared to two MS

complex implementations. Utilizing path compression to create the segmentations allows us to quickly extract the labels for the multi-label marching tetrahedra algorithm that powers the generation of region-separating geometries. The algorithm is not only faster but also has a lower memory footprint, as no discrete gradient vector field and little preprocessing of the triangulation is needed. Only a scalar field saving the 5-bit index representation for each vertex has to be created. The utilization of only top-level cells and vertices simplifies triangulation generation. Additionally, maxima at the border of the MS complex are not allowed by DMT design, often leading to missing border regions in each segmentation. This issue is fixed by segmenting the whole domain, also retrieving all border maxima in the process. Regarding the generated separating geometries, several use cases have been presented that show the applicability of our approach where the MS complex failed to deliver without expensive post-processing using saddle-saddle 2-cell cancellation. Simply speaking, our algorithm allows us to extract areas of influence of minima, maxima, and minima-maxima pairs by separating their boundaries with two available separating geometries that can be triggered by three segmentation options. Still, this does not invalidate the Morse-Smale complex as we only compute a segmentation and not the complex itself. Features like the saddle-saddle connectors are not computed. Therefore, we conclude that the PLMSS is a versatile tool to generate MS segmentations in a well-scaling parallel way, allowing users to explore their data much faster, while still being able to fall back to the MS complex on demand.

For future work, we are planning to improve the PLMSS in various ways. The load on each of the threads can be imbalanced when a particular thread receives a lot of triangles to generate. Here, a workload balance system could be introduced. To scale to even larger datasets, an MPI implementation will be provided to enable distributed memory execution. The knowledge gained from creating the segmentations will be implemented into the TTK MS complex implementation to improve its performance and useability. As shown in Sec. 6, a thorough comparison between MS complex implementations is out of the scope of this paper. Comparing all publicly available implementations and characterizing them in terms of input simplicial complex, handling of functions that are not Morse, available simplification models (pre- vs. post-simplification), output options, and memory footprint would be beneficial. Additionally, a version of the PLMSS using voxels as top-level cells in 3D could potentially speed up the computation considerably and would downsize the memory footprint even more. Also, the effect of path compression might be applicable to the computation of the MS complex, so additional research in integrating it might be of interest to achieve better runtime efficiency of MS complex implementations.

ACKNOWLEDGMENTS

This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 252408385 – IRTG 2057. This work is also partially supported by the European Commission grant ERC-2019-COG *"TORI"* (ref. 863464, https://erc-tori.github.io/).

REFERENCES

[1] M. Olejniczak, A. Severo Pereira Gomes, and J. Tierny, "A topological data analysis perspective on noncovalent interactions in relativistic calculations," *International Journal of Quantum Chemistry*, vol. 120, no. 8, p. e26133, 2020.

[2] H. Bhatia, A. G. Gyulassy, V. Lordi, J. E. Pask, V. Pascucci, and P.-T. Bremer, "Topoms: Comprehensive topological exploration for molecular and condensed-matter systems," *Journal of computational chemistry*, vol. 39, no. 16, pp. 936–952, 2018.

[3] A. Venkat, A. Gyulassy, G. Kosiba, A. Maiti, H. Reinstein, R. Gee, P.-T. Bremer, and V. Pascucci, "Towards replacing physical testing of granular materials with a topology-based model," *IEEE Transactions on Visualization and Computer Graphics*, no. 1, pp. 76–85, 2021.

[4] U. Homberg, D. Baum, A. Wiebel, S. Prohaska, and H.-C. Hege, "Definition, extraction, and validation of pore structures in porous materials," in *Topological methods in data analysis and visualization III*. Springer, 2014, pp. 235–248.

[5] D. Laney, A. Mascarenhas, P. Miller, and V. Pascucci, "Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1053–1060, 2006.

[6] A. Gyulassy, A. Knoll, K. C. Lau, B. Wang, P.-T. Bremer, M. E. Papka, L. A. Curtiss, and V. Pascucci, "Interstitial and interlayer ion diffusion geometry extraction in graphitic nanosphere battery materials," *IEEE transactions on visualization and computer graphics*, vol. 22, no. 1, pp. 916–925, 2015.

[7] T. Sousbie, "The persistent cosmic web and its filamentary structure–i. theory and implementation," *Monthly Notices of the Royal Astronomical Society*, vol. 414, no. 1, pp. 350–383, 2011.

[8] T. Bin Masood, J. Budin, M. Falk, G. Favelier, C. Garth, C. Gueunet, P. Guillou, L. Hofmann, P. Hristov, A. Kamakshidasan, C. Kappe, P. Klacansky, P. Laurin, J. Levine, J. Lukasczyk, D. Sakurai, M. Soler, P. Steneteg, J. Tierny, W. Usher, J. Vidal, and M. Wozniak, "An Overview of the Topology ToolKit," in *TopoInVis*, 2019.

[9] A. Gyulassy, P. Bremer, and V. Pascucci, "Shared-memory parallel computation of morse-smale complexes with improved accuracy," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 1183–1192, 2018, https://github.com/sci-visus/MSCEER.

[10] J. Milnor, *Morse Theory*. Princeton university press, 1963.

[11] R. Forman, "Morse theory for cell complexes," *Advances in mathematics*, vol. 134, no. 1, pp. 90–145, 1998.

[12] T. Banchoff, "Critical points and curvature for embedded polyhedra," *Journal of Differential Geometry*, vol. 1, no. 3-4, pp. 245–256, 1967.

[13] T. F. Banchoff, "Critical points and curvature for embedded polyhedral surfaces," *The American Mathematical Monthly*, vol. 77, no. 5, pp. 475–485, 1970.

[14] T. Lewiner, "Critical sets in discrete morse theories: Relating forman and piecewise-linear approaches," *Computer Aided Geometric Design*, vol. 30, no. 6, pp. 609–621, 2013.

[15] H. Edelsbrunner, J. Harer, and A. Zomorodian, "Hierarchical morse complexes for piecewise linear 2-manifolds," in *Proceedings of the seventeenth annual symposium on Computational geometry*, 2001, pp. 70–79.

[16] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci, "Morse-smale complexes for piecewise linear 3-manifolds," in *Proceedings of the nineteenth annual symposium on Computational geometry*, 2003, pp. 361–370.

[17] P.-T. Bremer, B. Hamann, H. Edelsbrunner, and V. Pascucci, "A topological hierarchy for functions on triangulated surfaces," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 4, pp. 385–396, 2004.

[18] E. Danovaro, L. D. Floriani, and M. M. Mesmoudi, "Topological analysis and characterization of discrete scalar fields," in *Geometry, Morphology, and Computational Imaging*. Springer, 2003, pp. 386–402.

[19] E. Danovaro, L. De Floriani, P. Magillo, M. M. Mesmoudi, and E. Puppo, "Morphology-driven simplification and multiresolution modeling of terrains," in *Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, 2003, pp. 63–70.

[20] A. Gyulassy, V. Natarajan, V. Pascucci, and B. Hamann, "Efficient computation of morse-smale complexes for three-dimensional scalar functions," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1440–1447, 2007.

[21] R. Forman, "A user's guide to discrete morse theory." *Séminaire Lotharingien de Combinatoire [electronic only]*, vol. 48, pp. B48c–35, 2002.

[22] D. Günther, J. Reininghaus, H. Wagner, and I. Hotz, "Efficient computation of 3d morse–smale complexes and persistent homology using discrete morse theory," *The Visual Computer*, vol. 28, no. 10, pp. 959–969, 2012.

[23] V. Robins, P. J. Wood, and A. P. Sheppard, "Theory and algorithms for constructing discrete morse complexes from grayscale digital images," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 33, no. 8, pp. 1646–1658, 2011.

[24] H. King, K. Knudson, and N. Mramor, "Generating discrete morse functions from point data," *Experimental Mathematics*, vol. 14, no. 4, pp. 435–444, 2005.

[25] A. Gyulassy, P.-T. Bremer, B. Hamann, and V. Pascucci, "A practical approach to morse-smale complex computation: Scalability and generality," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1619–1626, 2008.

[26] A. Gyulassy, V. Pascucci, T. Peterka, and R. Ross, "The parallel computation of morse-smale complexes," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 2012, pp. 484–495.

[27] T. Peterka, R. Ross, A. Gyulassy, V. Pascucci, W. Kendall, H.-W. Shen, T.-Y. Lee, and A. Chaudhuri, "Scalable parallel building blocks for custom data analysis," in *2011 IEEE Symposium on Large Data Analysis and Visualization*. IEEE, 2011, pp. 105–112.

[28] N. Shivashankar, M. Senthilnathan, and V. Natarajan, "Parallel computation of 2d morse-smale complexes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 10, pp. 1757–1770, 2012.

[29] N. Shivashankar and V. Natarajan, "Parallel computation of 3d morse-smale complexes," *Computer Graphics Forum*, vol. 31, pp. 965–974, 2012.

[30] V. Subhash, K. Pandey, and V. Natarajan, "Gpu parallel computation of morse-smale complexes," in *2020 IEEE Visualization Conference (VIS)*. IEEE, 2020, pp. 36–40.

[31] C. Heine, H. Leitte, M. Hlawitschka, F. Iuricich, L. De Floriani, G. Scheuermann, H. Hagen, and C. Garth, "A survey of topology-based methods in visualization," in *Computer Graphics Forum*, vol. 35. Wiley Online Library, 2016, pp. 643–667.

[32] A. Gyulassy, P.-T. Bremer, and V. Pascucci, "Computing morse-smale complexes with accurate geometry," *IEEE transactions on visualization and computer graphics*, vol. 18, no. 12, pp. 2014–2022, 2012.

[33] A. Gyulassy, M. Duchaineau, V. Natarajan, V. Pascucci, E. Bringa, A. Higginbotham, and B. Hamann, "Topologically clean distance fields," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1432–1439, 2007.

[34] D. Günther, R. A. Boto, J. Contreras-Garcia, J.-P. Piquemal, and J. Tierny, "Characterizing molecular interactions in chemical systems," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 2476–2485, 2014.

[35] A. Gyulassy, D. Günther, J. A. Levine, J. Tierny, and V. Pascucci, "Conforming morse-smale complexes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2595–2603, 2014.

[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] R. Fellegara, F. Luricich, L. De Floriani, and K. Weiss, "Efficient computation and simplification of discrete morse decompositions on triangulated terrains," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2014, pp. 223–232.

[38] T. Weinkauf, Y. Gingold, and O. Sorkine, "Topology-based smoothing of 2d scalar fields with c1-continuity," in *Computer Graphics Forum*, vol. 29. Wiley Online Library, 2010, pp. 1221–1230.

[39] S. Beucher and C. Lantuejoul, "Use of watersheds in contour detection. int," in *Workshop on Image Processing, CCETT/IRISA, Rennes, France*, 1979.

[40] S. Beucher, "Watersheds of functions and picture segmentation," in *ICASSP'82. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 7. IEEE, 1982, pp. 1928–1931.

[41] F. Meyer, "Integrals, gradients and watershed lines," in *Proc. Mathematical morphology and its applications to signal processing*, 1993, pp. 70–75.

[42] L. Najman and M. Schmitt, "Definition and some properties of the watershed of a continuous function," in *Mathematical Morphology and its applications to Signal Processing*, 1993, pp. 76–81.

[43] F. Meyer, "Topographic distance and watershed lines," *Signal processing*, vol. 38, no. 1, pp. 113–125, 1994.

[44] L. Vincent and P. Soille, "Watersheds in digital spaces: an efficient algorithm based on immersion simulations," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 13, no. 06, pp. 583–598, 1991.

[45] L. De Floriani, U. Fugacci, F. Iuricich, and P. Magillo, "Morse complexes for shape segmentation and homological analysis: discrete models and algorithms," in *Computer Graphics Forum*, vol. 34. Wiley Online Library, 2015, pp. 761–785.

[46] F. Meyer and S. Beucher, "Morphological segmentation," *Journal of visual communication and image representation*, vol. 1, no. 1, pp. 21–46, 1990.

[47] Y. Gabrielyan, V. Yeghiazaryan, and I. Voiculescu, "Parallel partitioning: Path reducing and union–find based watershed for the gpu," in *2022 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2022, pp. 1501–1505.

[48] V. Yeghiazaryan and I. Voiculescu, "Path reducing watershed for the gpu," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 577–585.

[49] P. Soille, *Morphological Image Analysis Principles and Applications*. Springer, 2004.

[50] A. P. Mangan and R. T. Whitaker, "Partitioning 3d surface meshes using watershed segmentation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 308–321, 1999.

[51] S. L. Stoev and W. Straßer, "Extracting regions of interest applying a local watershed transformation," in *Proceedings Visualization 2000*. IEEE, 2000, pp. 21–28.

[52] G. Nielson and R. Franke, "Computing the separating surface for segmented data," in *Proceedings. Visualization '97 (Cat. No. 97CB36155)*, 1997, pp. 229–233.

[53] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM siggraph computer graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[54] G. M. Nielson and B. Hamann, *The asymptotic decider: Rosolving the ambiguity in marching cubes*. eScholarship, University of California, 1991.

[55] H. Carr, T. Moller, and J. Snoeyink, "Artifacts caused by simplicial subdivision," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 2, pp. 231–242, 2006.

[56] M. Chouchane, A. Rucci, and A. A. Franco, "A versatile and efficient voxelization-based meshing algorithm of multiple phases," *ACS Omega*, vol. 4, no. 6, pp. 11 141–11 144, 2019.

[57] T. Müller and F. Raether, "3d modelling of ceramic composites and simulation of their electrical, thermal and elastic properties," *Computational Materials Science*, vol. 81, pp. 205–211, 2014.

[58] D. Weinstein, "Scanline surfacing: building separating surfaces from planar contours," in *Proceedings Visualization 2000*, 2000, pp. 283–289.

[59] N. Zhang, X. Zhou, D. Sha, X. Yuan, K. K. Tamma, and B. Chen, "Integrating mesh and meshfree methods for physics-based fracture and debris cloud simulation." in *PBG@ SIGGRAPH*, 2006, pp. 145–154.

[60] 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, 2017, https://topology-tool-kit.github.io/.

[61] H. Edelsbrunner and J. Harer, *Computational Topology: An Introduction*. American Mathematical Society, 2009.

[62] A. J. Zomorodian, "Topology for computing," in *Algorithms and Theory of Computation Handbook (Second Edition)*, M. J. Atallah and M. Blanton, Eds. CRC Press, 2010, ch. 3, pp. 82–112.

[63] P. Guillou, J. Vidal, and J. Tierny, "Discrete Morse Sandwich: Fast Computation of Persistence Diagrams for Scalar Data – An Algorithm and A Benchmark," *ArXiv e-prints*, 2022.

[64] H. Edelsbrunner and E. P. Mücke, "Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms," *ACM Transactions on Graphics (tog)*, vol. 9, no. 1, pp. 66–104, 1990.

[65] R. Seidel and M. Sharir, "Top-down analysis of path compression," *SIAM Journal on Computing*, vol. 34, no. 3, pp. 515–525, 2005.

[66] A. Doi and A. Koide, "An efficient method of triangulating equivalued surfaces by using tetrahedral cells," *IEICE TRANSACTIONS on Information and Systems*, vol. 74, no. 1, pp. 214–224, 1991.

[67] TTK Contributers, "*TTK Data Repository*, 2020," https://github.com/topology-tool-kit/ttk-data.
[68] ——, "*TTK Tutorial Data*, 2021," https://topology-tool-kit.github.io/stuff/ttk_tutorial_data.zip.
[69] J. Lukasczyk, G. Aldrich, M. Steptoe, G. Favelier, C. Gueunet, J. Tierny, R. Maciejewski, B. Hamann, and H. Leitte, "Viscous fingering: A topological visual analytic approach," in *Applied Mechanics and Materials*, vol. 869. Trans Tech Publ, 2017, pp. 9–19.
[70] A. W. Cook, W. Cabot, and P. L. Miller, "The mixing transition in Rayleigh-Taylor instability," *Journal of Fluid Mechanics*, vol. 511, pp. 333–362, 2004.
[71] S. Dong, P. Bremer, M. Garland, V. Pascucci, and J. C. Hart, "Spectral surface quadrangulation," *ACM Trans. Graph.*, vol. 25, no. 3, 2006.
[72] A. Gyulassy, V. Natarajan, V. Pascucci, P. Bremer, and B. Hamann, "A topological approach to simplification of three-dimensional scalar functions," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 4, 2006.

Hans Hagen is a computer science professor at University of Kaiserslautern and an adjunct professor at University of California, Davis. He received a Bachelor's degree in computer science, a Master degree in mathematics from the University of Freiburg and a PhD in mathematics (geometry) from the University of Dortmund. His main research interests are scientific visualization and geometric modeling. He is a member of the IEEE Visualization Academy of Science, and he got the IEEE Visualization Career Award, the ACM Solid Modeling Pioneer Award and the John Gregory Memorial Award among others.

Robin G. C. Maack received the Master's degree in computer science in October 2020 from the University of Kaiserslautern. He started as a student assistant in January 2017 and now further develops his projects as a PhD student. His research interests include topological data analysis, medical image analysis and visualization, biochemical visualization, and uncertainty visualization.

Ross Maciejewski is a professor with the School of Computing and Augmented Intelligence at Arizona State University and director of the Center for Accelerating Operational Efficiciency - a Department of Homeland Security Center of Excellence. His primary research interests include the areas of geographical visualization and visual analytics.

Jonas Lukasczyk received his Ph.D. degree from the Visual Information Analysis Group, Technische Universitat Kaiserslautern, Germany, where he also studied applied computer science and mathematics. His recent work focuses on topology-based characterization of features and their evolution in large-scale simulations.

Christoph Garth received the PhD degree in computer science from Technische Universität (TU) Kaiserslautern in 2007. After four years as a postdoctoral researcher with the University of California, Davis, he rejoined TU Kaiserslautern where he is currently a full professor of computer science. His research interests include largescale data analysis and visualization, in situ visualization, topology-based methods in visualization, and interdisciplinary applications of visualization.

Julien Tierny received the PhD degree in computer science from the University of Lille, in 2008. He is currently a CNRS senior scientist, affiliated with Sorbonne University. Prior to his CNRS tenure, he held a Fulbright fellowship (U.S. Department of State) and was a postdoctoral researcher at the Scientific Computing and Imaging Institute at the University of Utah. His research expertise lies in topological methods for data analysis and visualization. He is the founder and lead developer of the Topology ToolKit (TTK), an open source library for topological data analysis.