HAL
open science

The Topology ToolKit

Julien Tierny, Guillaume Favelier, Joshua A Levine, Charles Gueunet, Michael Michaux

To cite this version:

Julien Tierny, Guillaume Favelier, Joshua A Levine, Charles Gueunet, Michael Michaux. The Topology ToolKit. IEEE Transactions on Visualization and Computer Graphics, 2017. hal-01499905v2

HAL Id: hal-01499905

https://hal.science/hal-01499905v2

Submitted on 26 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

The Topology ToolKit

Julien Tierny, Guillaume Favelier, Joshua A. Levine, *Member, IEEE*, Charles Gueunet, and Michael Michaux

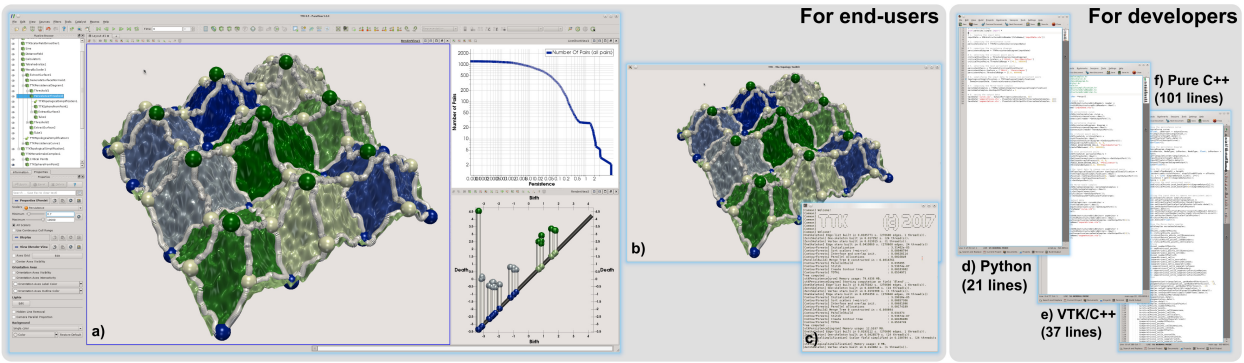

Fig. 1. TTK is a software platform for the topological analysis of scalar data in scientific visualization. It is both easily accessible to end users (with ParaView plugins (a), VTK-based generic GUIs (b) or command-line programs (c)) and flexible for developers (Python (d), VTK/C++ (e) or dependency-free C++ (f) bindings). TTK provides an efficient and unified approach to topological data representation and simplification, which enables in this example a discrete Morse-Smale complex (a) to comply to the level of simplification dictated by a piecewise linear persistence diagram (bottom-right linked view, a). Code snippets are provided (d-f) to reproduce this pipeline.

Abstract— This system paper presents the Topology ToolKit (TTK), a software platform designed for the topological analysis of scalar data in scientific visualization. While topological data analysis has gained in popularity over the last two decades, it has not yet been widely adopted as a standard data analysis tool for end users or developers. TTK aims at addressing this problem by providing a unified, generic, efficient, and robust implementation of key algorithms for the topological analysis of scalar data, including: critical points, integral lines, persistence diagrams, persistence curves, merge trees, contour trees, Morse-Smale complexes, fiber surfaces, continuous scatterplots, Jacobi sets, Reeb spaces, and more. TTK is easily accessible to end users due to a tight integration with ParaView. It is also easily accessible to developers through a variety of bindings (Python, VTK/C++) for fast prototyping or through direct, dependency-free, C++, to ease integration into pre-existing complex systems. While developing TTK, we faced several algorithmic and software engineering challenges, which we document in this paper. In particular, we present an algorithm for the construction of a discrete gradient that complies to the critical points extracted in the piecewise-linear setting. This algorithm guarantees a combinatorial consistency across the topological abstractions supported by TTK, and importantly, a unified implementation of topological data simplification for multi-scale exploration and analysis. We also present a cached triangulation data structure, that supports time efficient and generic traversals, which self-adjusts its memory usage on demand for input simplicial meshes and which implicitly emulates a triangulation for regular grids with no memory overhead. Finally, we describe an original software architecture, which guarantees memory efficient and direct accesses to TTK features, while still allowing for researchers powerful and easy bindings and extensions. TTK is open source (BSD license) and its code, online documentation and video tutorials are available on TTK's website [108].

Index Terms—Topological data analysis, scalar data, data segmentation, feature extraction, bivariate data, uncertain data.

1 Introduction

As scientific datasets become more intricate and larger in size, advanced data analysis algorithms are needed for their efficient visualization and exploration. For scalar field visualization, topological analysis techniques [69] have shown to be practical solutions in various contexts by enabling the concise and complete capture of the structure of the input data into high-level *topological abstractions* such as contour trees [29], Reeb graphs [18, 85, 109], or Morse-Smale complexes [36, 64]. Such topological abstractions are fundamental data structures that enable the development of advanced data analysis, exploration and visualization techniques, including for instance: small seed set extraction for fast isosurface traversal [30, 111], feature tracking [101], transfer function design for volume rendering [113], similarity estimation [105], or application-driven segmentation and analysis tasks [63, 67, 68, 75]. Successful applications in a variety of fields of science, including combustion [24, 63, 75], material sciences [67, 68], chemistry [57], or astrophysics [97, 102] to name a few, have even been documented, which further stresses the importance of this class of techniques. Despite this popularity and success in applications, topological data analysis (TDA) has not yet been widely adopted as a standard data analysis tool for end users and developers. While some open source implementations for specific algorithms are available [31, 39, 42, 43, 56, 96, 102, 106], we still identify three main issues preventing a wider adoption of TDA.

First, these implementations lack, in general, support for standard data file formats, genericity regarding the dimensionality of the input data, integration into graphical user front ends, or access through high-level scripting languages. These limitations challenge their adoption by end users and domain experts with little or no programming knowledge.

Second, regarding software developers, each implementation comes with its own internal data structures or its own list of third-party software dependencies, which challenges their integration into pre-existing, complex systems for visualization or data analysis.

- *Julien Tierny, Guillaume Favelier, Charles Gueunet and Michael Michaux are with Sorbonne Universites, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, France. E-mails: {julien.tierny, guillaume.favelier, charles.gueunet, michael.michaux}@lip6.fr.*
- *Joshua A. Levine is with the University of Arizona, USA. E-mail: josh@email.arizona.edu.*
- *Charles Gueunet is also with Kitware SAS, France. E-mail: charles.gueunet@kitware.com.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

Third, regarding researchers, despite the isolated open source implementations mentioned above, many TDA algorithms do not have publicly available implementations, which challenges reproducibility. While other research communities have excelled at providing software platforms that ease the implementation, benchmarking, and distribution of research code (such as the *Image Processing On Line* platform [4]), to the best of our knowledge, there has not been such a federating initiative for TDA codes in scientific visualization.

This system paper presents the Topology ToolKit (TTK) [108], a software platform for the topological analysis of scalar data in scientific visualization, which addresses the three core problems described above: *(i)* accessibility to end users, *(ii)* flexibility for developers and *(iii)* ease of extension and distribution of new algorithms for researchers. TTK provides a unified, generic, efficient, and robust implementation of key algorithms for the topological analysis of scalar data. It is easily accessible to end users thanks to a tight integration with ParaView (Fig. 1, left) and flexible for developers (Fig. 1, right) through a variety of bindings (Python, VTK/C++) or direct, third-party dependency-free, C++ access (to ease integration in pre-existing complex systems). Finally, it facilitates the implementation, integration, and distribution of TDA codes, by simply requiring the implementation of a handful of functions, while providing efficient data structures and, thanks to ParaView, advanced IO, rendering and interaction support for end users.

While developing TTK, we faced several algorithmic and software engineering challenges, which we document in this paper.

(i) Algorithmic consistency: For advanced analysis tasks, it can be desirable to combine several topological abstractions [57,75]. However, each abstraction comes with its own simplification mechanism, which challenges the development of a unified framework. More important, several competing formalisms exist to represent the input data, namely the piecewise-linear setting [13,45] and the Discrete Morse Theory setting [54]. The lack of compatibility between these two representations challenges even more the design of a unified framework.

(ii) Core data structures: Combinatorial algorithms for TDA mostly involve mesh traversal routines. Thus, generic and time efficient triangulation data structures must be derived.

(iii) Software engineering: Designing a software library which has no third-party dependency and which also seamlessly integrates into a complex visualization system such as ParaView is challenging. Related challenges include avoiding data copy within the visualization pipeline. Also, designing such a flexible library in a way that still enables easy extensions is an additional difficulty.

Contributions

This paper makes the following new contributions:

1. *An algorithm* (Sec. 4) to construct a discrete gradient which complies to the critical points extracted in the piecewise linear (PL) setting. Each critical simplex resulting from this algorithm is located in the star of a PL critical point. This relationship between the discrete and PL settings enables a combinatorial consistency across the different topological abstractions supported by TTK. As a byproduct, it allows for a unified and independent topological simplification procedure for multiscale exploration and analysis.
2. *A data structure* (Sec. 5) for time efficient traversals on 2D or 3D piecewise linear triangulations. In the case of input meshes, it self-adjusts its memory footprint depending on the traversal operations it is queried for. In the case of regular grids, it implicitly emulates a triangulation with no memory overhead.
3. *A software architecture* (Sec. 6) that eases the development and distribution of TDA code to end users. The creation of a new module only requires the implementation of a handful of functions, while TTK automatically generates a command-line program, a VTK-based GUI and a ParaView plugin connected to the module.
4. *A software collection* (Sec. 7) that implements in a unified and generic way a variety of TDA algorithms. It is accessible to end users as command line programs, VTK-based GUIs, or ParaView plugins. It is accessible to developers through a variety of bindings: Python, VTK/C++, or dependency-free C++.

2 Related work

In this section, we discuss three main categories of prior work related to this paper: existing visualization front ends, TDA software packages, and triangulation data structures for TDA.

2.1 Visualization front ends

In this subsection, we briefly mention existing efforts to enhance the accessibility of visualization to end users. Many of the successes in this area are either specific libraries and toolkits, such as the Visualization ToolKit (VTK) [91] and the Insight ToolKit (ITK) [6]. TTK is similar to these libraries albeit with a specific focus on topological data analysis. Related, largely turnkey tools often deliver features of such toolkits with richer interfaces. Examples include ParaView [8], VisIt [32], VisTrails [17], SCIRun [92], MeVisLab [3], and Amira [2]. Many of the above systems employ a dataflow network as a key component to their visualization pipeline [81]. Extensions of this model have been proposed for specific use-cases or application areas, for instance for higher order finite elements [90]. TTK also differs sharply from alternate forms of specializations, such as domain-specific languages for visualization that have been recently developed, including *Diderot* [73], *Scout* [78], *Vivaldi* [33], and *ViSlang* [87].

2.2 TDA software packages

Existing TDA software packages can be categorized into two groups. While their inputs and outputs greatly differ, they share the common goal of extracting features which capture topological invariants.

TDA on low-dimensional manifolds: The first category focuses on topological abstractions of scalar data on low-dimensional manifolds (typically in 2D or 3D) for the purpose of data analysis and visualization as discussed in Sec. 1. TTK primarily targets these applications. Such topological abstractions include for instance critical points [13], persistence diagrams [35,48], Reeb graphs [18,40,41,84,85,88,109] (and their loop-free variant, contour trees [23,29,103]), and Morse-Smale complexes [36,46,64,89,95]. Dillard implements *libtourte*, a library computing the contour tree [39], while Doraiswamy et al.'s *libRG* library [43] and *Recon* [42] as well as Tierny's *vtkReebGraph* [106] compute the Reeb graph. Shivashankar and Natarajan have focused on a scriptable implementation of the Morse-Smale complex [96] based on their parallel algorithms in 2D [94] and 3D [95]. Sousbie developed *DisPerSE*, an implementation of the Morse-Smale complex focused on cosmology data analysis [102]. Finally, Chen et al. provide an implementation of the Morse Decomposition for 2/2.5D vector fields [31].

Although powerful, the above tools often lack the level of integration required for end users' adoption. Most of them come with custom file formats [31,39,42,43,102]. This forces users to write data converters for each tool, which greatly impairs adoption by end users, who can be domain experts with no programming background. Also, these tools often come only as libraries or command-line programs, which can also discourage end users. In contrast, some tools [56,106] directly rely on established toolkits such as the Visualization ToolKit (VTK) [91] or the R statistical environment [1] and thus benefit from a rich support for most standard data file formats. However, the dependency on these complex environments has not been designed in these tools to be optional. This constitutes a serious limiting factor for developers of pre-existing complex systems, who often want to minimize the number of third-party dependencies their system rely on. In contrast, TTK's software architecture (Sec. 6) has been specifically designed such that TTK can be called by dependency-free C++ codes, by using primitive types only. This allows integrating TTK seamlessly in any system written in C++, without having to pull any extra third-party dependency. Optionally, TTK can wrap around both VTK and ParaView to leverage their rich IO support as well as their powerful user interface capabilities. Finally, existing tools often lack genericity in terms of input data representation or dimensionality. For instance, the two Morse-Smale complex implementations by Shivashankar and Natarajan [96] are either designed for triangulations in 2D or regular grids in 3D. In contrast, TTK supports in a unified way both representations in both dimensions. Also, TTK is based on a tighter integration with ParaView, which allows end users without any programming skill to easily interact with it, without even having to use scripting languages such as Python.

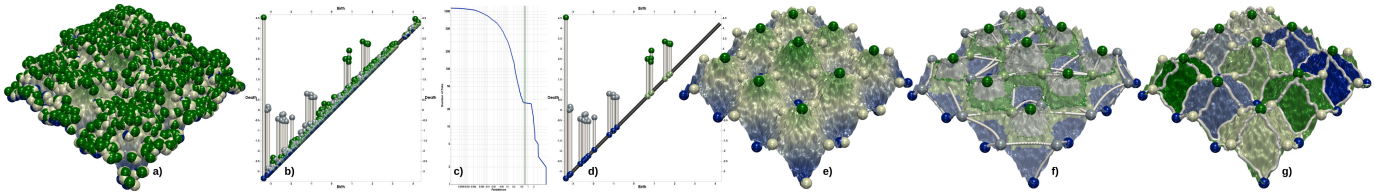

Fig. 2. Height function f (blue to green) on a noisy terrain. (a) Initially, f admits many critical points (spheres, blue: minima, white: saddles, green: maxima), resulting in a persistence diagram $\mathscr{D}(f)$ with many small bars near the diagonal (b). The persistence curve $\mathscr{C}(f)$ exhibits a clear plateau (c) separating noise from features (green vertical line). Our approach to topological simplification pre-simplifies f to the corresponding persistence threshold in a unified way, which is consistently interpreted by all the supported topological abstractions: persistence diagrams (d), critical points (e), contour trees (f) and even discrete Morse-Smale complexes (g). Note that any application-driven metric could be used in place of persistence.

TDA on high-dimensional point clouds: A second set of TDA software packages rather focus on estimating persistent homology [45, 48, 118] on point clouds, usually for topology inference applications in high dimensions. One of the earliest implementations is *Mapper* [99], which is used by the commercial product *Ayasdi* [11]. Both *Dionysus* [82] and *JavaPlex* [7] provide implementations of the standard algorithm by Zomorodian and Carlsson [118]. *Dionysus* also implements persistent cohomology [37] and zigzag persistent cohomology [27]. *Perseus* [83] implements a preprocessing procedure that reduces the number of filtered cells in the standard algorithm [80].

More recent implementations focus on variations, either in the domain or outputs. Gerber et al. implement *MSR* [56] for approximating Morse-Smale complexes on k-nearest neighbor graphs of high-dimensional data [55]. *Gudhi* [77] implements persistent homology on simplicial complexes relying on the *Simplex tree* [20] data structure. *Phat* [15] relies on matrix reduction operators for efficient computations. Fasy et al.'s TDA package provides an interface to *Gudhi*, *Dionysus*, and *Phat* in the popular analysis language R [1, 51]. *SimPers* computes persistent homology on simplicial maps [38]. Bubenik's persistence landscapes toolbox [26] computes a more descriptive statistical summary of homology than the typical persistence diagram. In contrast to these packages, TTK specifically targets low dimensional (2D or 3D) domains for applications in scientific data analysis and visualization.

2.3 Triangulation data structures for TDA

Combinatorial TDA algorithms mostly involve mesh traversal routines. Therefore, corresponding implementations must rely on data structures providing time efficient traversal queries. Data structures for triangulations is a well researched topic [53, 72] and a variety of approaches exist for storing them with various sorts of trade-offs between memory footprint and time efficiency. Some data structures, such as *OpenMesh* [22], *surface_mesh* [98] or *SimplexMesh* [14], employ variants of the half-edge structure [114]. In contrast, several other mesh libraries employ a cell-based representation, where only points and cells of highest dimension are stored. For instance, VTK [91] implements this strategy with its mesh data structures, whose design clearly trades efficiency for generality as these structures support arbitrary polyhedra and polygons. CGAL's [104] 2D and 3D triangulation data structures [71, 86], Mesquite [25], and VCGLib [5], the underlying library behind MeshLab [34], all employ cell-based data structures.

A major drawback of these cell-based implementations is that the connectivity of cells of intermediate dimensions (edges and triangles in tet-meshes) must be re-computed upon each query. However, such queries are common practice in TDA (Sec. 5.1). This drawback is attenuated by data structures which aim at balancing time and memory efficiency [20, 21, 115–117] but it is accentuated by data structures which further compress the adjacency information [19, 61, 62, 76]. In contrast, TTK implements a cached triangulation data structure (Sec. 5), which provides lookup-based time-efficient queries, while self-adjusting its memory footprint on demand, and implicitly emulating triangulations in the case of regular grids.

3 Preliminaries

This section briefly describes our formal setting. We refer the reader to reference books for introductions to Morse theory [79], computational topology [45] and Discrete Morse Theory [54].

3.1 Input data

Without loss of generality, we assume that the input data is a piecewise linear (PL) scalar field $f : \mathscr{M} \to \mathbb{R}$ defined on a PL d-manifold $\mathscr{M}$ with d equals 2 or 3. It has value at the set of vertices $\mathscr{M}^0$ of $\mathscr{M}$ and is linearly interpolated on the simplices of higher dimension. Adjacency relations on $\mathscr{M}$ can be described in a dimension independent way. The *star* $St(\sigma)$ of a simplex σ is the set of simplices of $\mathscr{M}$ 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. In the following, the topology of $\mathscr{M}$ will be mostly described in terms of its *Betti numbers* β_i (the ranks of its homology groups [45]), which correspond in 3D to the numbers of connected components (β_0), non collapsible cycles (β_1) and voids (β_2).

3.2 Geometric features

For visualization and data analysis purposes, several low-level geometric features can be defined given the input data. Given an isovalue $i \in \mathbb{R}$, the *level set*, noted $f^{-1}(i)$, is the pre-image of i onto $\mathscr{M}$ through f: $f^{-1}(i) = \{p \in \mathscr{M} \mid f(p) = i\}$. The *sub-level set*, noted $f^{-1}_{-\infty}(i)$, is defined as the pre-image of the open interval $(-\infty, i)$ onto $\mathscr{M}$ through f: $f^{-1}_{-\infty}(i) = \{p \in \mathscr{M} \mid f(p) < i\}$. Symmetrically, the *sur-level set* $f^{-1}_{+\infty}(i)$ is defined by $f^{-1}_{+\infty}(i) = \{p \in \mathscr{M} \mid f(p) > i\}$. An *integral line* is a path on $\mathscr{M}$ which is everywhere tangential to ∇f. Topological data analysis can be seen as the study of the topological transitions (in terms of Betti numbers) of these objects as one sweeps the range $\mathbb{R}$ [79].

3.3 Critical points

The points of $\mathscr{M}$ where the Betti numbers of $f^{-1}_{-\infty}(i)$ change are the *critical points* of f (Fig. 2(e)). 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* $Lk^+(v)$ is given by $Lk^+(v) = \{\sigma \in Lk(v) \mid \forall u \in \sigma : f(u) > f(v)\}$. To classify $Lk(v)$ without ambiguity into either lower or upper links, the restriction of f to the vertices of $\mathscr{M}$ is assumed to be injective. This is easily enforced in practice by a variant of simulation of simplicity [50]. This is achieved by considering an associated injective integer offset $\mathscr{O}(v)$, which initially typically corresponds to the vertex position offset in memory. Then, when comparing two vertices, if these share the same value f, their order is disambiguated by their offset $\mathscr{O}$. A vertex v is regular, if and only if both $Lk^-(v)$ and $Lk^+(v)$ are simply connected. Otherwise, v is a *critical point* of f [13]. Let d be the dimension of $\mathscr{M}$. Critical points can be classified with their *index* $\mathscr{I}$, which equals 0 for minima ($Lk^-(v) = \emptyset$), 1 for 1-saddles ($\beta_0(Lk^-(v)) = 2$), $(d-1)$ for $(d-1)$-saddles ($\beta_0(Lk^+(v)) = 2$) and d for maxima ($Lk^+(v) = \emptyset$). Vertices for which $\beta_0(Lk^-(v))$ or $\beta_0(Lk^+(v))$ are greater than 2 are called *degenerate saddles*. For bivariate functions $f : \mathscr{M} \to \mathbb{R}^2$, the notion of *Jacobi set* [44] extends that of critical points [107]. Bivariate analogs of level-sets (*fibers*) also change their topology in their vicinity.

3.4 Topological persistence

The distribution of critical points of f can be represented by a first topological abstraction, called the *persistence diagram* [35, 48] (Fig. 2(d)), which also provides a measure of topological noise on critical point pairs. By applying the Elder's rule [45], critical points can be arranged in a set of pairs, such that each critical point appears in only one pair (c_i, c_j) with $f(c_i) < f(c_j)$ and $\mathscr{I}(c_i) = \mathscr{I}(c_j) - 1$. The persistence diagram $\mathscr{D}(f)$ embeds each pair (c_i, c_j) in the plane such that its horizontal coordinate equals $f(c_i)$, and the vertical coordinate of c_i and c_j

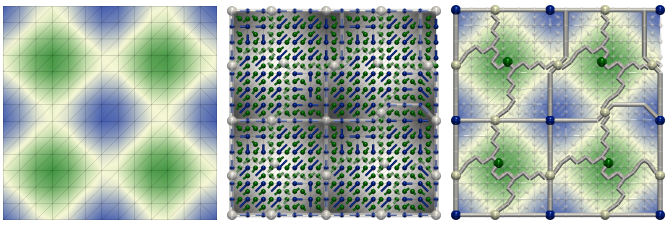

Fig. 3. Discrete Morse Theory on a 2D toy example. Left: input PL scalar field f. Center: discrete gradient (the origin of each pair is displayed with a sphere, vertex-edge and edge-triangle pairs are shown in blue and green). Right: discrete Morse-Smale complex $\mathscr{MS}(f)$ obtained by extracting the V-paths emanating from critical simplices.

are $f(c_i)$ and $f(c_j)$. The height of the pair $p = f(c_j) - f(c_i)$ is called the *persistence* and denotes the life-span of the topological feature created in c_i and destroyed in c_j. In low dimensions, the persistence of the pairs linking critical points of index $(0,1)$, $((d-1),d)$ and $(1,2)$ (in 3D) denotes the life-span of connected components, voids and non-collapsible cycles of $f^{-1}_{-\infty}(i)$. The *persistence curve* $\mathscr{C}(f)$ plots the number of critical pairs as a function of their persistence (Fig. 2(c)).

3.5 Reeb graphs

The *Reeb graph* [88] segments $\mathscr{M}$ into regions where the connectivity of $f^{-1}(i)$ does not change. Let $f^{-1}(f(p))_p$ be the connected component of $f^{-1}(f(p))$ containing p. The Reeb graph $\mathscr{R}(f)$ is a one-dimensional simplicial complex defined as the quotient space $\mathscr{R}(f) = \mathscr{M}/\sim$ by the equivalence relation $p_1 \sim p_2$, which holds if $p_2 \in f^{-1}(f(p_1))_{p_1}$. For bivariate data $f : \mathscr{M} \to \mathbb{R}^2$, Reeb graphs extend to *Reeb spaces* [47, 107], being this time 2D cell complexes.

Variants of the Reeb graph can be defined relative to the connected components of $f^{-1}_{-\infty}(i)$ and $f^{-1}_{+\infty}(i)$, yielding the notion of *merge tree* (specifically *join* and *split* trees for $f^{-1}_{-\infty}(i)$ and $f^{-1}_{+\infty}(i)$). In 2D, the persistence pairs of $\mathscr{D}(f)$ can be efficiently computed from the join (split) tree, by pairing each saddle, in increasing (decreasing) order of f values, with the highest (lowest) non-paired minimum (maximum) its contains in its sub-tree [45]. The contour tree (the loop-free variant of $\mathscr{R}(f)$ for simply connected domains) can be efficiently computed by combining the join and split trees [29, 103] (Fig. 2(f)).

3.6 Morse-Smale complexes and Discrete Morse Theory

The *Morse-Smale complex* [36] segments $\mathscr{M}$ into regions where integral lines share both their origin and destination (Fig. 2(g)). Given a critical point p, its *ascending* (resp. *descending*) *manifold* is defined as the set of points belonging to integral lines whose origin (resp. destination) is p. The *Morse complex* is the complex formed by all descending manifolds. f is said to be a *Morse-Smale function* if it admits no degenerate saddle and if its ascending and descending manifolds only intersect transversally (if the codimension of their intersection equals the sum of their codimensions [60]). Then, the *Morse-Smale complex* $\mathscr{MS}(f)$ is the complex formed by the intersection of the Morse complex of f and that of $-f$. Concretely, all the points of a given cell (of arbitrary dimension) of $\mathscr{MS}(f)$ belong to integral lines having the same origin and destination. In 3D, the persistence pairs corresponding to non-collapsible cycles can be efficiently extracted by visiting the 2-saddles not already present in $\mathscr{D}(f)$ (Sec. 3.5) in ascending order and pairing each one with the highest, non-paired, 1-saddle it is connected to through a 1-dimensional cell of $\mathscr{MS}(f)$ (called *saddle-connector*) and reverting the gradient along that cell. The robust computation of Morse-Smale complexes for PL scalar fields has been a long time challenge for the community, as existing algorithms [46] were highly complex and required many special cases to account for the transversal intersection condition in 3D. Fortunately, an alternate formalism, namely *Discrete Morse Theory* [54] (DMT), enabled the definition of elegant and robust algorithms [64, 89, 95], improving the applicability of Morse-Smale complexes. We briefly describe here DMT in the case of PL manifolds, but it remains valid for arbitrary CW complexes.

A *discrete Morse function* is a function that assigns a scalar value to every simplex in $\mathscr{M}$, such that each i-simplex $\sigma_i \in \mathscr{M}$ has at most one co-face σ_{i+1} (resp. face σ_{i-1}) with lower (resp. higher) function value: $|\{\sigma_{i+1} > \sigma_i \mid f(\sigma_{i+1}) \leq f(\sigma_i)\}| \leq 1$ and $|\{\sigma_{i-1} < \sigma_i \mid f(\sigma_{i-1}) \geq f(\sigma_i)\}| \leq 1$. Simplices for which these two numbers are zero are called *critical simplices* and their dimension matches their index $\mathscr{I}$. A *discrete vector* is a pair of simplices $\{\sigma_i < \sigma_{i+1}\}$. A *discrete vector field* V is a collection of such pairs such that each simplex appears in at most one pair. Then, a V-*path* is a sequence of pairs of V $\{\sigma^0_i < \sigma^0_{i+1}\}, \{\sigma^1_i < \sigma^1_{i+1}\}, \ldots, \{\sigma^r_i < \sigma^r_{i+1}\}$ such that $\sigma^j_i \neq \sigma^{j+1}_i < \sigma^j_{i+1}$ for each $j = 0, \ldots, r$. A *discrete gradient* is a discrete vector field for which all V-paths are monotonic and loop free. For these, V-paths are discrete analogs of integral lines, and simplices which are left unpaired in V are critical. Then, the 1-dimensional cells of $\mathscr{MS}(f)$ connected to maxima (resp. minima), called 1-*separatrices*, can be constructed by extracting ascending (resp. descending) V-path(s) from $(d-1)$-saddles (resp. 1-saddles). In 3D, its 2-dimensional cells, called 2-*separatrices* can be constructed with a descending (resp. ascending) breadth-first search traversal from 2-saddles (resp. 1-saddles) on the primal (resp. dual) of $\mathscr{M}$. Last, the 1-dimensional cells of $\mathscr{MS}(f)$ linking a 1-saddle s_1 to a 2-saddle s_2, called *saddle-connectors*, can be extracted by computing the V-path(s), restricted to a given 2-separatrix, linking s_1 to s_2. Fig. 3 illustrates these notions. Note that all saddles are non-degenerate and the transversal intersection is respected by construction.

4 Unified Topological Analysis and Simplification

Although it allows a robust and consistent computation of $\mathscr{MS}(f)$, the DMT formalism is not directly compatible with the PL setting. DMT critical points are typically much more numerous than PL critical points in practice. Moreover, in contrast to PL critical points which are located on vertices only, they are located on simplices of all dimensions. This incompatibility challenges the design of a unified framework such as TTK, for which the support of advanced analysis, robustly combining several topological abstractions, can be highly valuable in practice. Moreover, each topological abstraction traditionally comes with its own simplification mechanism. This further challenges the development of a unified framework, as multiple instances of simplification algorithms (not necessarily consistent with each other) need to be implemented and maintained. In this section, we describe an approach that addresses these two issues (unified representations and simplifications).

4.1 Initial discrete gradient

Given an input function f valued on the vertices of $\mathscr{M}$, several algorithms [64–66, 89, 95] have been proposed for the construction of a discrete gradient (Sec. 3.6). Among them, we focus on a variation of the elegant algorithm by Shivashankar and Natarajan [95] for our initial gradient computation, as its localized nature will ease the following discussion and allows for a trivial and efficient parallelization. For completeness, we briefly sketch its main steps in Alg. 1.

The algorithm takes as an input a scalar field f and an injective offset field $\mathscr{O}$ (Sec. 3.3). It visits the simplices of $\mathscr{M}$ dimension-by-dimension. For each dimension i, i-simplices are processed independently (and thus in parallel). The candidate set $C(\sigma_i)$ (line 12) is constructed as the set of co-faces of σ_i for which σ_i maximizes the i-dimensional faces. σ_i is then paired with the minimizer of this candidate set (line 14). Here, the notions of minimizer and maximizer require the definition of a comparator (lines 1 to 8), which iteratively compares two i-simplices σ_i and σ'_i by comparing their maximum vertices. If these two vertices are identical ($\mathscr{O}(v[j]) == \mathscr{O}(v'[j])$), the next couple of maximum vertices will be considered until all vertices are visited.

We now discuss a key observation regarding the above algorithm [95], which is useful for the development of a unified TDA framework.

Property 1 (PL Matching) *Let f be a PL scalar field defined on a closed, PL 2 or 3-manifold $\mathscr{M}$. Algorithm 1 will produce a discrete gradient field V such that each PL critical point p of index $\mathscr{I}(p)$ will admit at least one critical simplex of dimension $\mathscr{I}(p)$ in its star $St(p)$.*

This property can be justified based on two key observations (see [93] for an alternate discussion). First, *(i)* If a $(d-1)$-simplex σ_{d-1} maximizes both its d-co-faces, one of the two will be critical as σ_{d-1} can be paired only once. *(ii)* For any i-simplex σ_i, its maximizing face is the one which does not contain its minimizing vertex and no other

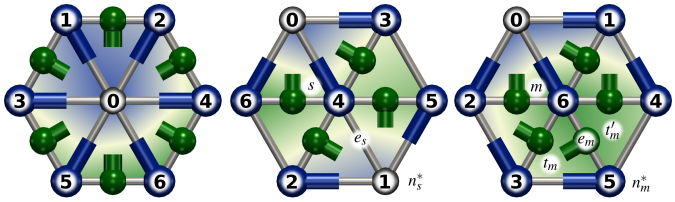

Fig. 4. Alg. 1 leaves at least one vertex (0, left), one edge (e_s, center) and one triangle (t'_m, right) unpaired in the star of a 2D PL minimum (left), saddle (center), and maximum (right). Vertex-edge and edge-triangle pairs are shown with blue and green balls-and-sticks respectively.

Algorithm 1: Initial discrete gradient construction [95]

input : PL scalar field $f : \mathscr{M} \to \mathbb{R}$ (with injective offset field $\mathscr{O} : \mathscr{M}^0 \to \mathbb{R}$);
output: Discrete gradient field V.

1 `Function`($\sigma_i < \sigma'_i$)
2 $v = \texttt{sort}_{f,\mathscr{O}}({\sigma_i}^0)$;
3 $v' = \texttt{sort}_{f,\mathscr{O}}({\sigma'}_i^0)$;
4 **for** $j \leftarrow i+1$ ***to*** 0 **do**
5 **if** $f(v[j]) < f(v'[j])$ **then return** true;
6 **if** $f(v[j]) == f(v'[j])$ && $\mathscr{O}(v[j]) < \mathscr{O}(v'[j])$ **then return** true;
7 **if** $\mathscr{O}(v[j]) \neq \mathscr{O}(v'[j])$ **then return** false;
8 **return** false;
9 **begin**
10 **for** $i \leftarrow 0$ ***to*** $(d-1)$ **do**
11 **foreach** $\sigma_i \in \mathscr{M}$ **do**
12 $C(\sigma_i) = \{\sigma_{i+1} > \sigma_i \mid \sigma_i = argmax_{\sigma'_i < \sigma_{i+1}} f(\sigma'_i)\}$;
13 **if** $C(\sigma_i) \neq \emptyset$ **then**
14 $V = V \cup \{\sigma_i < argmin_{C(\sigma_i)} f(\sigma_{i+1})\}$;
15 **end**

face of σ_i can be paired with it. These two observations allow a simple and non-ambiguous characterization of the pairing resulting from Alg. 1 in the star of each PL critical point.

In 2D, critical simplices of dimension 0 precisely coincide with PL minima (Fig. 4, left), since a local minimum is the maximizer of none of its incoming edges (observation *(ii)*), and will therefore never be paired. By definition, the lower link $Lk^-(s)$ of a PL saddle s is made of at least two connected components, each of which containing a local minimizing vertex. Let n_s^* be the second lowest vertex of this set of local minimizers (Fig. 4, center). s will be paired by Alg. 1 with the edge linking it to the lowest vertex of this set (line 14). Let e_s be the edge linking s to n_s^*. n_s^* cannot be paired with e_s as n_s^* is its minimizing vertex (observation *(ii)*). The direct neighbors of n_s^* on $Lk(s)$ are necessarily higher than n_s^* by definition. Thus, the co-faces of e_s cannot be paired with it, as e_s contains n_s^*, which is the minimizing vertex of both triangles (observation *(ii)*). Thus, the edge e_s will not be paired by Alg. 1 and will be critical. Let n_m^* be the global maximum of f restricted to the link $Lk(m)$ of a PL maximum m (Fig. 4, right). Let t_m and t'_m be the two triangles of $St(m)$ connected to the edge e_m linking m to n_m^*. e_m is by definition the maximizer of both triangles t_m and t'_m but can be paired with only one of them (observation *(i)*). Thus, the remaining triangle will be left unpaired by Alg. 1, and thus critical. A similar, yet slightly more involved, reasoning applies constructively in 3D, as described in Appendix A (supplemental material).

4.2 PL-compliant discrete gradient

The *PL matching* property (Sec. 4.1) indicates that we are guaranteed to find one DMT critical simplex for each PL critical point. This allows us to introduce an injective map $\xi : PL(f) \to DMT(f)$, from the set of PL critical points $PL(f)$ to that of DMT critical simplices $DMT(f)$, that maps each PL critical point p of index $\mathscr{I}(p)$ to a unique DMT critical simplex of dimension $\mathscr{I}(p)$ in its star. If multiple DMT critical $\mathscr{I}(p)$-simplices exist in $St(p)$, we select the highest as $\xi(p)$. We relax ξ in the presence of a degenerate saddle s and allow s to be matched with $m > 1$ DMT critical simplices, where m is the multiplicity of s.

However, in practice, the majority of the simplices of $DMT(f)$ will be left unmatched by ξ, requiring an additional cleanup procedure, which we introduce here. Let $DMT'(f)$ be this set: $DMT'(f) = \{\sigma \in DMT(f) \mid \xi^{-1}(\sigma) = \emptyset\}$. These simplices are extraneous singularities which can be considered as artifacts of the discrete gradient construction. Thus, we introduce a procedure to simplify the gradient field V, such that the simplices of $DMT'(f)$ are no longer critical.

Saddle-maximum pair removal: Discrete Morse Theory [54] indicates that two critical simplices σ_i and σ_{i+1} linked by a unique V-path P can be simplified by reversing the gradient precisely along P. Such an operation guarantees that V indeed remains loop-free, that σ_i and σ_{i+1} are no longer critical and that no other simplex becomes critical. We make use of this property in the following (see Fig. 5). First, we construct the graph G_0, whose nodes correspond to the critical d and $(d-1)$-simplices of $DMT'(f)$ and whose arcs correspond to V-paths linking them. G_0 is the subset of the 1-separatrices of $\mathscr{MS}(f)$ ending in maxima to be removed. Next, we visit each arc $a = (\sigma_i, \sigma_{i+1}) \in G_0$ in increasing order of function value difference. If a is the only V-path connecting σ_i to σ_{i+1}, we remove a from G_0, reverse its V-path, and update the connectivity of G_0: each arc $a' \in G_0$ that was connected to σ_{i+1} gets reconnected to the $(i+1)$-simplices of G_0 that were connected to σ_i and the function value difference for a' and the corresponding V-path are updated. All other connections to σ_i are removed. This process continues iteratively to remove all the d-simplices of G_0.

Saddle-saddle pair removal: For PL 3-manifolds, an extra step is required to remove saddles of $DMT'(f)$ that are connected in pairs by V-paths. Similarly to G_0, we construct the graph G_1 whose nodes correspond to the remaining 1 and 2-saddles of $DMT'(f)$ and whose arcs correspond to V-paths linking them. G_1 is the subset of the saddle-saddle connectors of $\mathscr{MS}(f)$ linking 1 and 2-saddles which remain to remove. Next, G_1 is iteratively processed as described above for G_0.

For closed PL manifolds, the *PL matching* property is guaranteed and our algorithm will remove all critical simplices of $DMT'(f)$. Thus, each remaining critical simplex of V will be located in the star of a PL critical point, and we say that V is *PL-compliant*. Note that this drastically differs from the *conforming* algorithm [66], which constrains the separatrices of $\mathscr{MS}(f)$ to match an input segmentation. This also differs from the *assignGradient2* procedure [95], which only simplifies critical simplices if they are adjacent to each other, although they can be arbitrarily far from each other in practice, hence requiring our algorithm for the complete removal of the simplices of $DMT'(f)$. The impact of the boundary of $\mathscr{M}$ on our algorithm is discussed in Sec. 8.

4.3 Unified topological simplification

Now that V is PL-compliant, it is possible to robustly combine multiple PL topological abstractions with the discrete Morse-Smale complex in a single pipeline (Figs 1 and 2), the exact correspondence between $DMT(f)$ and $PL(f)$ being given by ξ. As a byproduct, one can now mutualize the topological simplification procedure traditionally used for multi-scale analysis and exploration, by pre-simplifying the data itself, prior to the computation of the topological abstraction under consideration. Several combinatorial algorithms [10, 16, 49, 110] have been proposed for this purpose. Here, we focus on the approach by Tierny and Pascucci [110] as it supports arbitrary simplification heuristics. In particular, given a list of extrema E to maintain, this algorithm will minimally modify both f and $\mathscr{O}$ such that $PL(f)$ only admits the critical points of E as extrema. This is achieved by an iterative flattening of the sub-level set components corresponding to the critical points to remove [110]. The set of maintained extrema E can be selected according to persistence (Sec. 3.4, in which case the output topological abstractions will be guaranteed to be consistent with post-process simplification schemes [64, 85]) or any application-driven metric. Once f and $\mathscr{O}$ have been pre-simplified, the topological abstractions under consideration can be constructed for this simplified data (Fig. 2). Our PL-compliant discrete gradient algorithm will guarantee that the discrete Morse-Smale complex complies to this simplification.

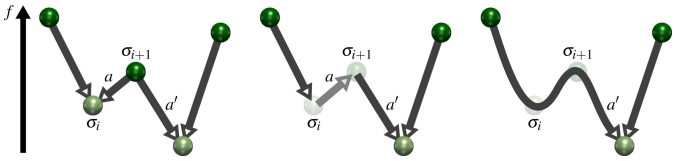

Fig. 5. Removing a critical simplex pair from $DMT'(f)$. The arc $a \in G_0$ which minimizes its function difference is selected (left). Its corresponding V-path is reversed (center). The connectivity of G_0 is updated (right).

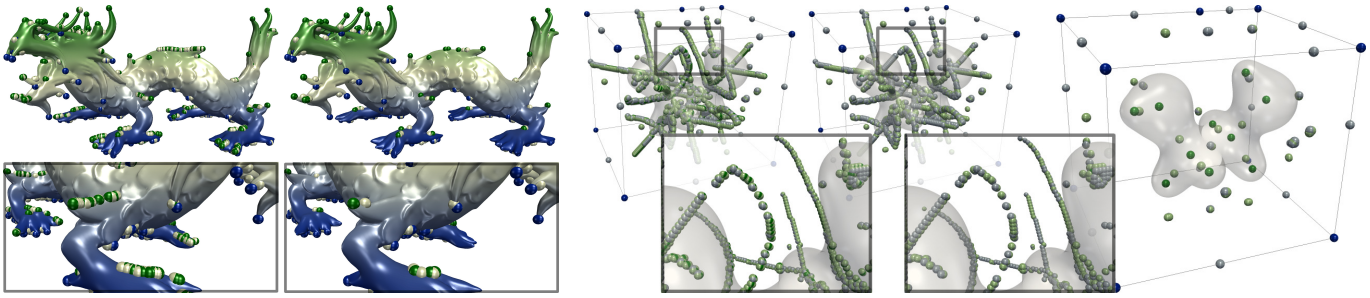

Fig. 6. Critical simplices (spheres) before and after PL-compliance on a 2D (left, 140k triangles) and 3D example (right, 10.5M tetrahedra). Each critical simplex is colored according to its dimension, from dark blue (minima) to dark green (maxima). In 3D, $|DMT(f)|$ decreases from 6,109 (left) to 2,997 (center) after the removal of non PL saddle-maximum pairs, and to 93 (right) after the removal of the non PL saddle-saddle pairs.

4.4 Performance

The initial gradient computation (Alg. 1) and the PL critical point extraction (Sec. 3.3) are both linear-time algorithms, which we implemented in parallel with OpenMP. The simplification of G_0 and G_1 is implemented sequentially, in a similar way to previous Morse-Smale complex simplification approaches [64, 95]. The optional pre-simplification of the data [110] is implemented sequentially and each separatrix of the Morse-Smale complex (Sec. 3.6) is extracted in parallel. Tab. 1 reports performance numbers, obtained on a Xeon CPU (2.6 GHz, 2x6 cores), for the examples shown in Fig. 6. This table shows that, when fully constructing $\mathscr{MS}(f)$ (with all separatrices), the PL-compliance step results in a small overhead overall. Note that in practice, the extraction of the geometry of the saddle-saddle connectors (used for the removal of the corresponding pairs by path reversal) is more demanding than that of saddle-maximum separatrices (column *S-M* and *S-S* in Tab. 1), as it requires computing the corresponding 2-dimensional separatrices.

5 Cached Triangulation Data Structure

Combinatorial TDA algorithms mostly involve mesh traversal routines. Thus, they must build on top of time efficient data structures. This section describes a new triangulation data structure designed to optimize time efficiency, while adjusting its memory footprint on demand.

5.1 Traversal specifications

Two core types of traversal queries are typically found in TDA algorithms, which consist of either accessing the faces or the co-faces of a simplex, as illustrated in the following traversal examples.

(i) Boundary: A frequent test in TDA algorithms consists of checking if an i-simplex σ_i is located on the boundary of $\mathscr{M}$. This can be achieved by querying the $(d-1)$-co-faces of σ_i (where d is the dimension of $\mathscr{M}$) and verifying if some admit only one d-co-face.

(ii) Skeleton: It is often necessary in TDA to access the k-skeleton of $\mathscr{M}$ (all i-simplices of $\mathscr{M}$, such that $i \leq k$). For instance, the construction of the merge tree (Sec. 3.5) requires accessing the 1-skeleton of $\mathscr{M}$.

(iii) Link: The star and the link are key notions to characterize the neighborhood of a simplex (Sec. 3.1). In particular, the extraction of the critical points of f (Sec. 3.3) requires constructing the link $Lk(v)$ of each vertex v, and to classify it into its lower $Lk^-(v)$ and upper $Lk^+(v)$ links. This construction can be achieved by querying the d-co-faces of v, then querying their $(d-1)$-faces (which do not intersect v, to obtain $Lk(v)$) and then querying their 0-faces to examine if a vertex of $Lk(v)$ is lower or higher than v. A similar classification needs to be performed on the link $Lk(e)$ of each edge e for Jacobi set extraction [107].

(iv) Face / co-face: Another typical traversal example consists of querying for each i-simplex, its list of $(i-1)$-faces and $(i+1)$-co-faces. This query is typical for the computation of the discrete gradient (Alg. 1), where maximizing faces are first identified to construct a candidate set of co-faces (line 12). Since this algorithm performs such queries for each dimension, the $(i-1)$-faces and $(i+1)$-co-faces of each i-simplex must be efficiently accessed.

As shown in the above examples, TDA algorithms may potentially need to access the k-faces and l-co-faces of each i-simplex, for *arbitrary* k and l such that $0 \leq k < i < l \leq d$. Thus, we designed our triangulation data structure to support *all* the following types of traversal operations:

1. Given a vertex v, access its 1-, 2-, and 3-co-faces;
2. Given an edge e, access its 0-faces and its 2- and 3-co-faces;
3. Given a triangle t, access its 0- and 1-faces and its 3-co-faces;
4. Given a tetrahedron T, access its 0-, 1-, and 2-faces.

Table 1. Running time of the different steps of the PL-compliance algorithm (in seconds, with 12 cores) for the examples of Fig. 6. $PL(f)$, S-M, S-S, $\mathscr{MS}(f)$ and $\mathscr{MS}'(f)$ respectively stand for the computation times for the PL critical points, the saddle-maximum pair removal, the saddle-saddle pair removal, the total Morse-Smale complex construction (including gradient processing) with and without the PL-compliance.

Dataset	$\lvert DMT(f)\rvert$	$\lvert PL(f)\rvert$	Alg. 1	$PL(f)$	S-M	S-S	$\mathscr{MS}(f)$	$\mathscr{MS}'(f)$
Dragon	1,118	318	0.016	0.018	0.004	0	**0.074**	0.072
EthaneDiol	6,109	93	4.943	1.525	0.144	3.864	**13.829**	11.804

5.2 Explicit triangulation

To optimize time efficiency, we designed our data structure to support the above traversal queries through constant time lookups. Given an input cell-based representation (Sec. 2.3) providing a list of 3D points $\mathscr{L}_P$ and a list $\mathscr{L}_S$ of d-simplices (representing each d-simplex by a list of vertex identifiers), we construct our explicit triangulation data structure as follows. The exhaustive list of edges can be constructed by enumerating all the unique vertex identifier pairs present in $\mathscr{L}_S$. This can be done in $O(|\sigma_d|)$ steps, where $|\sigma_d|$ is the number of d-simplices in $\mathscr{M}$, by using a vertex-driven lookup table, which progressively stores the constructed vertex pairs on a per vertex basis, to avoid potential edge duplicates. Similarly, if d equals 3, the exhaustive list of triangles can be constructed by enumerating all the unique vertex triplets present in $\mathscr{L}_S$, which can also be done in $O(|\sigma_d|)$ steps by using a vertex driven lookup table to avoid potential triangle duplicates. Given these list of 1-, 2-, and 3-simplices, one can simply iterate over the list of k-simplices ($k \in \{1,2,3\}$) to store for each vertex its list of k-co-faces. This is done in $O(|\sigma_k|)$ steps. The construction of such lists enables supporting the first type of traversal operation (Sec. 5.1) with constant time lookups. Given the list of k-co-faces for each vertex, one can construct the list of co-faces for each edge. Given an edge $e = (v_0, v_1)$, the list of k-co-faces of e can be obtained as the intersection of the sets of k-co-faces of v_0 and v_1. This can be done in $O(|\sigma_1|)$ steps. The list of co-faces for each triangle can be constructed similarly in $O(|\sigma_2|)$ steps. Last, given the list of l-co-faces of each k-simplex, the list of k-faces of each l-simplex is built in $O(|\sigma_l|)$ steps. The pre-construction of all the lookup tables mentioned above allows for time efficient queries based on constant time lookups for all the traversal types described in Sec. 5.1, at the expense of an excessive memory footprint.

To limit this footprint, we introduce a preconditioning mechanism. This mechanism exploits the fact that many of the lookup tables described above can be constructed independently. Our preconditioning mechanism enforces third-party algorithms to clearly specify in advance (typically in an initialization step) the type of traversal queries they are going to perform. For instance, regarding the skeleton traversal (Sec. 5.1), to access the 1-cofaces of each vertex v, a developer needs to call at initialization the `preprocessVertexEdges()` precondition function of our data structure, which will build, once for all and only if they have not been constructed yet, the list of 1-simplices of $\mathscr{M}$ and the list of 1-co-faces for each vertex. Thus, each traversal query

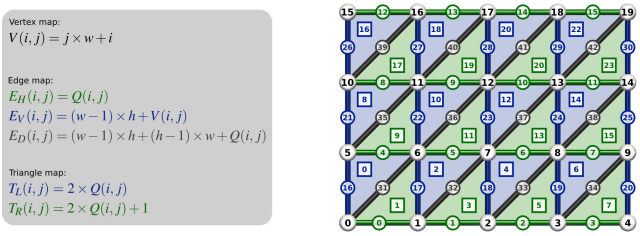

Fig. 7. Analytic expressions for the vertex, edge and triangle maps in a 2D regular grid of width w and height h. The identifier of each simplex is expressed as a function of the (i,j)-coordinates of its bottom left vertex.

supported by our triangulation data structure is associated with its own precondition function to be called beforehand, which will only trigger the computation of the necessary lookup tables, if they have not been constructed yet. This preconditioning mechanism guarantees a memory footprint limited to the only required lookup tables. This strategy is particularly useful when a single instance of triangulation is used by multiple algorithms, as it is typically the case in a dataflow model such as VTK's pipeline. There, our data structure will be progressively enhanced upon the precondition calls triggered by the algorithms present in the pipeline (see Appendix B for implementation details).

5.3 Implicit triangulation

In scientific visualization, the input scalar data is often provided as a regular grid. However, the typical size of these grids, especially in 3D, make the explicit storage of their triangulation impractical. To address this issue, we introduce an implicit triangulation data structure which emulates the above lookup tables in the case of regular grids.

In 2D, quads are considered to be pairs of triangles, whose common edge always follows the same diagonal. Given this regular structure, one can introduce analytic expressions to associate each i-simplex to a unique identifier. For brevity, we only discuss interior simplices here. Let $V : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ be an injective function mapping a vertex to its identifier given its (i,j)-coordinates in a grid of width w and height h (in terms of number of vertices): $V(i,j) = j \times w + i$. A similar function can be derived for each quad, given the (i,j)-coordinates of its bottom left vertex: $Q(i,j) = j \times (w-1) + i$. One can introduce a similar convention, to uniquely enumerate edges, given the (i,j)-coordinate of their bottom left vertex, into horizontal ($E_H(i,j)$), vertical ($E_V(i,j)$) and diagonal ($E_D(i,j)$) edges (Fig. 7). Triangles can be enumerated similarly into left ($T_L(i,j)$) and right ($T_R(i,j)$) triangles (Fig. 7).

Once such a convention is established, the traversal queries of Sec. 5.1 can be efficiently implemented by considering the pre-images of these functions. For instance, to perform traversal queries given an edge with identifier e, the edge must first be classified as horizontal ($e < (w-1) \times h$) or vertical ($e < (w-1) \times h + (h-1) \times w$) or diagonal. Then, one can easily retrieve the (i,j)-coordinate of its bottom left vertex from the analytic expressions E_H, E_V or E_D. Then, the 0-faces of e are given by $V(i,j)$ for its bottom left vertex and $V(i+1,j)$, $V(i,j+1)$ and $V(i+1,j+1)$ for its second vertex for horizontal, vertical and diagonal edges respectively. A very similar, yet more involved, strategy is derived in 3D, by considering that each voxel is composed of 6 tetrahedra. This regular structure also allows introducing simplex maps analytically. Then, given the identifier of an i-simplex σ_i, its k-faces and l-co-faces can be easily evaluated for arbitrary k and l, by properly classifying σ_i and considering the pre-image of the corresponding identifier function, as described above in 2D. Since all identifiers are computed on the fly, this strategy effectively emulates our explicit data structure with no memory overhead. Note that in practice, we implemented both our explicit and implicit strategies with a common programming interface, which allows developers to manipulate our data structure generically, irrespective of its implicit or explicit nature.

5.4 Performance

Tab. 2 reports performance numbers obtained on a Xeon CPU (2.6 GHz, 2x6 cores) with the explicit (1.25M tetrahedra, 71 MB in binary `VTU` file format) and implicit triangulations of a 64^3 grid. Each test loops sequentially on the entire set of i-simplices of $\mathscr{M}$ (i is reported in parentheses) and stores the result of the corresponding query. This table shows that our explicit data structure indeed adapts its memory footprint depending on the type of traversal it undergoes. In particular, the most memory-demanding traversals involve the list of edge co-faces ($i = 1$) and the overhead required by our data structure ranges from 32% to 218%. Once it is preconditioned, our explicit structure provides query times which are on par with its implicit counterpart. Tab. 2 also reports the speedup of our explicit data structure, as the total time of a first run of each test (including preconditioning), divided by the time of a second run. Overall, this indicates that, if developers consent to multiply the memory footprint of their triangulation by a factor of up to 3.18 in the worst case, typical time performance gains of one order of magnitude (and up to 3) can be expected for their TDA algorithms.

6 Software Architecture

In this section, we describe our design decisions as well as the software engineering challenges that we faced when developing TTK.

6.1 Design motivations

The flexibility for software developers (challenge *(ii)*, Sec. 1) required to write TTK with an efficient, low-level and portable programming language, such as C++. Moreover, to ease the integration of TTK into pre-existing, complex visualization systems, we designed its software architecture such that its implementation of TDA algorithms relies on no third party library. We achieved this by designing a first core layer of dependency-free, templated *functor* classes (Sec. 6.2).

To make TTK easily accessible to end users (challenge *(i)*), we tightly integrated it into an advanced visualization programming environment, in particular VTK [91]. Several alternatives to VTK could have been considered and we briefly motivate our choice here. In practice we found only a few downsides in using VTK, the most notable being the modest time performance of their mesh data structures, which trade speed for generality. However, our cached triangulation data structure (Sec. 5) has been specifically designed to address this time efficiency issue. Apart from this downside, an integration into VTK comes with numerous advantages. First, VTK is easily accessible to the masses due to its open source and portable nature. Second, it provides a rich support for various standard file formats. We believe this aspect is instrumental for end users' adoption. Third, it provides a rich, object-oriented rendering pipeline, which eases the development of interactive applications. Forth, implementations following VTK's API can easily be integrated into ParaView. This integration does not only increase exposure to end users with a powerful visualization front end, but it also automatically provides, without additional effort, software bindings for Python, which becomes more and more popular among engineers and scientists. Such a support is provided by ParaView's `pvpython` Python wrapper. Finally, a key argument in our choice towards VTK was its implementation of raw data pointers, providing direct accesses to the primitive-typed (`int`, `float`, `double`, etc.), internal buffers of its data structures, in reading and writing mode. This capability is necessary to design a dependency-free functor layer (Sec. 6.2) meant to interact with VTK, without data copy and without the inclusion of any VTK header. Our last target, ease of extension for researchers (challenge *(iii)*), is partly achieved due to our integration with VTK, as TDA developers only need to focus on the write-up of their core algorithm, without caring about IO, rendering, or interaction, but still benefiting from the

Table 2. Running times (in seconds), memory footprint (MB) and overhead of our cached triangulation data structure for the traversal examples of Sec. 5.1 on the explicit (*e-Time*, 1.25M tetrahedra, 71 MB in binary VTU) and the implicit (*i-Time*) triangulation of a 64^3 grid.

Traversal Example	Precondition Time	Memory Footprint	Memory Overhead	e-Time	Cache Speedup	i-Time
Boundary Vertices (0)	1.082	48	68 %	0.001	997	0.003
Boundary Edges (1)	1.568	101	142 %	0.008	200	0.027
Boundary Triangles (2)	1.099	50	70 %	0.011	100	0.038
1-skeleton (0)	0.200	23	32 %	0.025	9	0.107
Vertex Link (0)	1.391	105	148 %	0.035	41	0.135
Edge Link (1)	0.512	128	180 %	0.129	5	1.857
Triangle Link (2)	1.135	67	94 %	0.551	3	0.810
Edge FaceCoFace (1)	1.800	155	218 %	0.280	7	0.477
Triangle FaceCoFace (2)	1.310	119	168 %	0.447	4	0.451

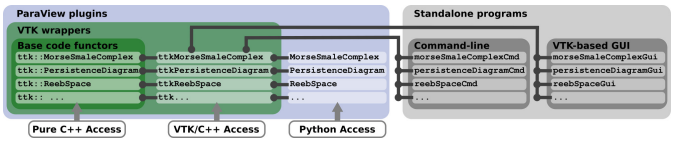

Fig. 8. TTK's software architecture is divided into 3 parts: the main library (green), ParaView plugins (blue) and automatically generated standalone programs (grey). Developers can access TTK through each of its layer, either from dependency-free C++, VTK/C++ or Python codes.

above advanced features offered by VTK and ParaView. We further improve the extension possibilities of TTK as described in Sec. 6.3.

6.2 Overview

TTK's software architecture is presented in Fig. 8. In the following, we will refer to the running example of TTK's scalar field smoother, which iteratively smooths data by averaging neighbor values.

Base code functors living in the `ttk` C++ namespace implement the TDA algorithms of TTK. They do not store data, but are given pointers to input and output buffers. Their processing routines are template functions, where the template parameters are the primitive data types used by the input and output buffers. This template structure enables writing generic code, irrespective of the representation of the scalar data (`float`, `char`, etc.). These functors are also typically passed a pointer to a cached triangulation data structure (Sec. 5), which they precondition at initialization time, depending on the type of traversal they are going to perform. Note that these classes include no VTK header. For instance, the `ttk::ScalarFieldSmoother` functor is given a pointer to the input and output data buffers and a pointer to a triangulation instance. It preconditions it at initialization with the function `preprocessVertexNeighbors()` and perform the smoothing based on the triangulation adjacency in a distinct template function.

VTK wrappers implemented as `vtkDataSetAlgorithm` filters [91] connect each base code functor to VTK. In particular, these filters typically query a pointer to the internal buffers of the input and output objects and call the processing function of the corresponding functor with the appropriate template argument. For example, the `vtkScalarFieldSmoother` selects the input field to smooth, allocates an output field and passes their pointers to its functor. Note that TTK functors can be used without VTK. An example is given in Fig. 1.

ParaView plugins are automatically created from the VTK wrappers. The specification of each plugin is documented in an XML file, which is interpreted at build time. Such a file describes the options of the VTK wrappers which will be exposed to ParaView's GUI and Python binding. In our smoothing example, a developer would declare in the XML file the VTK wrapper function which controls the number of smoothing iterations. Then, the Python class automatically generated from the resulting ParaView plugin would provide a variable that enables tuning this number. Fig. 1 illustrates some TTK plugins interacting together within a ParaView pipeline, as well as the corresponding Python script.

Standalone programs In certain situations, it may be desirable to run TDA algorithms in batch mode. Thus, we accompany each VTK wrapper with a generic command line program which reads all the datasets given as command line arguments, sends them to its VTK wrapper and writes all of its outputs to disk after execution. Developers only need to edit the main source file of this program to declare options (i.e. the number of smoothing iterations) to TTK's command line parser, similarly to the ParaView XML specification. TTK also automatically accompanies each VTK wrapper with a VTK-based GUI, which behaves similarly to the command line program. Once opened, this GUI lets users interact with the outputs of the VTK wrapper and, similarly to ParaView, let them toggle the display of each output with keystrokes.

6.3 Implementing a new module for TTK

To ease the development of new TDA algorithms, TTK ships with Bash scripts which automate the creation and release of new TTK modules. In particular, the creation script generates a templated base code functor and its matching VTK wrapper, ParaView plugin, standalone command-line and GUI programs. At this point, each of these can be built and run. To implement their TDA algorithms, developers then only need to focus on the base code functor. The input and output specification should be enriched if needed from the default one in the VTK wrapper layer. Finally, options should be declared within the ParaView XML file and the standalone main source files. Also, another Bash script packages these components into a tarball for release purposes, with optional Doxygen online documentation and code anonymization.

7 Software Collection

Each TTK module comes with its own base-code functor, VTK wrapper, ParaView plugin and command-line and VTK-GUI programs.

Scalar data: Critical points (Sec. 3.3) often correspond directly to features of interest in scalar data. TTK implements a combinatorial algorithm for their extraction in the PL setting [13]. Merge trees and contour trees (Fig. 9*(i)*) are instrumental TDA abstractions for data segmentation tasks [24, 30, 57]. TTK implements their computation based on a multi-threaded approach [59]. The discrete Morse-smale complex (Sec. 3.6), which is a key abstraction for data segmentation and for the extraction of filament structures [52, 57, 68, 97, 102] (Fig. 9*(ii)*), is implemented with the PL-compliant approach described in Sec. 4. Persistence diagram and curves (Sec. 4) help users appreciate the distribution of critical points and tune simplification thresholds. The extraction of the extremum-saddle and saddle-saddle pairs has been implemented as described in Secs. 3.5 and 3.6. Topological simplification is implemented in an independent, unified manner (Sec. 4) for all the above abstractions by pre-simplifying the data with a combinatorial approach [110]. TTK also implements a few supporting features, including integral lines or data smoothing and normalization.

Bivariate scalar data: Jacobi sets (Sec. 3.3) correspond to points where the volume folds onto itself when projected to the plane by a bivariate function. TTK implements their combinatorial extraction [107]. Reeb space (Sec. 3.5) based segmentation capabilities help users peel scatterplots views of the data (Fig. 9*(v)*). TTK implements a recent combinatorial approach [107]. TTK also implements a few supporting features, including planar projections, continuous scatterplots [12], fiber and fiber surfaces [28] based on user strokes [74] (Fig. 9 *(iv)*).

Uncertain scalar data: Uncertain scalar fields are becoming more and more prominent in applications. TTK implements a combinatorial approach for extracting *mandatory critical points* [58], which predicts appearance regions for critical points, despite the uncertainty (Fig. 9*(vi)*). To support this, TTK also provides a module converting an ensemble dataset into a histogram-based representation of the data.

Miscellaneous: TTK also provides a number of support features, for data conversion, connected component size evaluation, geodesic distances, mesh subdivision, geometry smoothing, identifier fields, texture map computation from scalar data, or simplified sphere glyphs.

User experience: End users typically leverage ParaView's advanced interface capabilities to organize the linked views of their visualization (2D, 3D, line charts, etc.). TTK features are accessed through its plugins, used in conjunction with standard ParaView filters. For instance, in Fig. 1(a), the user selected critical point pairs in the persistence diagram (bottom right 2D linked view) by thresholding their persistence. Then, TTK's topological simplification was called to pre-simplify the data according to this selection of critical points. Note that any other, application-driven, user selection can be used instead of persistence. At this point, any topological abstraction can be computed on this pre-simplified data, like the Morse-Smale complex. Next, the user can interactively modify the persistence threshold and all the linked views are updated accordingly, as shown in our companion video. We refer the reader to TTK's website [108] for more video tutorials. Note that TTK's usage in ParaView requires no programming or scripting skill.

8 Limitations and Discussion

The PL matching property (Sec. 4.1) is only guaranteed for interior critical points. For non-closed domains, boundary PL critical points may admit no DMT critical simplex in their star. Thus, for simplicity, our implementation omits DMT critical simplices located on the boundary. This omission is not problematic in practice (Fig. 6), although it may prevent the removal of certain pairs of DMT critical simplices, for which one or both simplices are located on the boundary. More

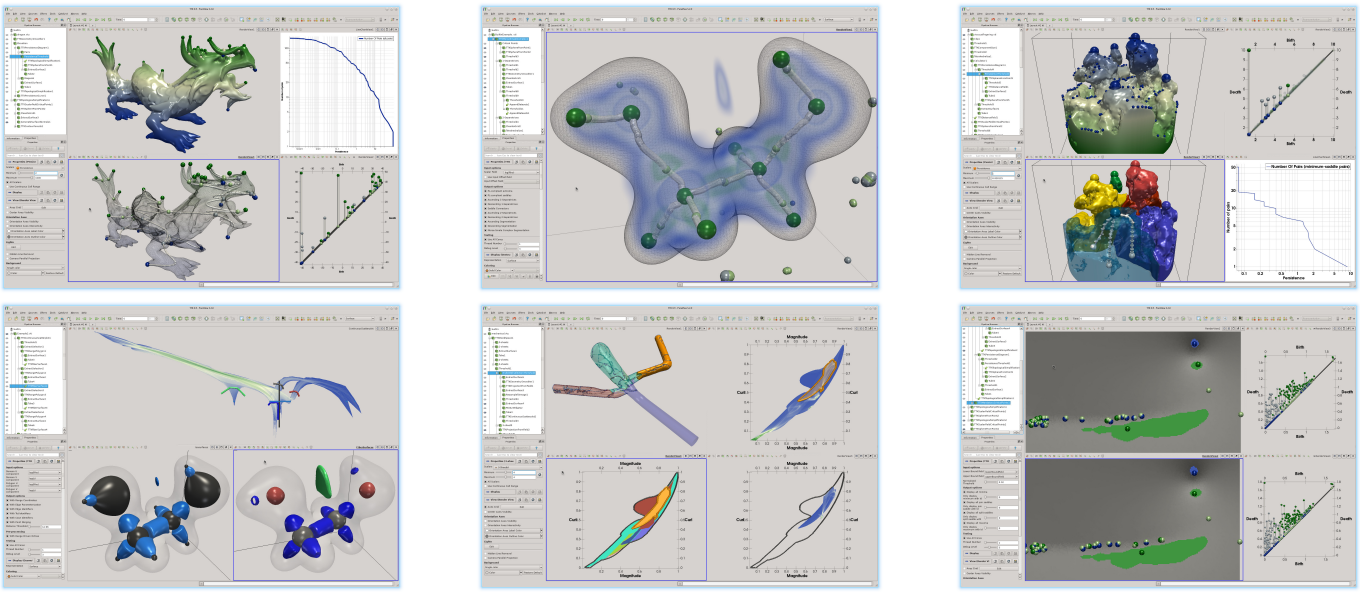

Fig. 9. Gallery of TTK pipelines executed in ParaView. From top left to bottom right: *(i)* The contour tree can be used for shape skeletonization. Skeleton noise is removed by imposing the level of simplification dictated by the persistence diagram in the bottom linked view. *(ii)* The interior 1-separatrices of the Morse-Smale complex emanating from 2-saddles directly capture the atomic and covalent structure of molecules (electron density). *(iii)* The cells of the Morse-complex (colored regions) enable segmenting and tracking features during viscous fingering [52]. *(iv)* Fiber surfaces (bottom right) from user strokes in the continuous scatter plot (top) enable an easy classification of the features in molecular systems [74]. *(v)* The Reeb space (top left) enables peeling continuous scatterplots (bottom left) into layers where the fibers are made of only one connected component in the volume [107]. This enables localized inspections of the scatterplots (bottom right). *(vi)* Mandatory critical points (colored regions) provide predictions on the location and function values of critical points for uncertain scalar fields with non-uniform error [58]. These regions always admit at least one critical point for any function randomly generated from this error (top and bottom, vortices in computational fluid dynamics).

generally, although our cached triangulation data structure can support fairly general domains (such as non-manifold simplicial complexes), more research still needs to be done at the topological analysis level towards generality with regard to the input domain representation.

The combinatorial procedure [110] that TTK employs to pre-simplify scalar data only supports the removal of $(0,1)$ and $((d-1),d)$ critical point pairs. More research needs to be done to find a practical algorithm for the removal of $(1,2)$ critical point pairs, as homological simplification and reconstruction in $\mathbb{R}^3$ has been shown to be NP-hard [9].

Regular grids are treated by TTK as implicit triangulations (which is key to guarantee the PL matching property) by considering the 6-tet subdivision of each voxel. Thus, TDA algorithms which already natively support arbitrary CW-complexes [64, 89, 95] may run more slowly with this representation as more cells will need to be considered.

We have been using preliminary versions of TTK internally for two years and published several research papers using it [52, 59, 107, 112]. Five master-level students, with a C++ background but no knowledge in rendering or user interfaces, have been using it on a daily basis. Typically, we found that TTK helped them shorten the prototyping phase of their research and access experimental validation faster, some of these students starting implementations after just a week of training. The pre-implemented IO and rendering support of ParaView was instrumental to provide an important time gain during prototyping. Moreover, the ease offered by ParaView to combine several plugins within a single pipeline also helped students easily leverage existing implementations in their prototypes to empower their own algorithms.

9 Conclusion

This system paper presented the Topology ToolKit (TTK) [108], a software platform designed for topological data analysis (TDA) in scientific visualization. TTK has been designed to be easily accessible to end users (with ParaView plugins and standalone programs) and flexible for software developers with a variety of bindings. For researchers, TTK eases the prototyping phase of TDA algorithms, without compromise on genericity or time efficiency, as developers only need to focus on the core routines of their algorithm; the IO, the rendering pipeline and the user interface capabilities being automatically generated. Although it focuses on TDA, we believe TTK also provides more generally an appealing infrastructure for any geometry-based visualization technique.

TTK builds on top of two main contributions: *(i)* a unified topological data representation and simplification and *(ii)* a time-efficient triangulation data structure. Our PL-compliant discrete gradient algorithm allows to robustly and consistently combine multiple topological abstractions, defined in the Discrete or PL settings, within a single coherent analysis, as showcased with the ParaView pipelines illustrated in this paper (Figs. 1, 2, 9). The genericity and time efficiency of TTK is in great part due to our novel cached triangulation data structure, which handles in a consistent way 2D or 3D explicit meshes or regular grids implicitly. In explicit mode, a preconditioning mechanism enables our data structure to deliver time-efficient traversal queries, while self-adjusting its memory footprint on demand. For typical TDA traversals, we showed that if developers consent to reasonably increase the memory footprint of their triangulation, significant speedups can be expected in practice for their algorithms. This aspect is particularly important when a single triangulation instance is accessed by multiple algorithms, as typically found in complex analysis pipelines.

TTK constitutes an invitation to a community-wide initiative to disseminate and benchmark TDA codes. Although we did our best to implement in TTK a substantial collection of TDA algorithms for scientific visualization, many more could be integrated in the future, including for instance Pareto set computation [70] or robustness evaluation [100]. Thus, we hope TTK will rapidly grow a developer community to help integrate more algorithms, as we specifically designed it to be easily extensible. We also hope that future extensions of TTK will form the basis of an established software platform for TDA research code, to improve the reproducibility and usability of TDA in applications.

Acknowledgments

This work is partially supported by the Bpifrance grant "AVIDO" (Programme d'Investissements d'Avenir FSN2, reference P112017-2661376/DOS0021427) and by the National Science Foundation IIS-1654221. We would like to thank the reviewers for their thoughtful remarks and suggestions. We would also like to thank Attila Gyulassy, Julien Jomier and Joachim Pouderoux for insightful discussions and Will Schroeder, who encouraged us to write this manuscript.

REFERENCES

[1] The R project. http://www.r-project.org/.

[2] Amira advanced 3d visualization. http://www.amiravis.com/, 2006.

[3] Mevislab: A development environment for medical image processing and visualization. http://www.mevislab.de/, 2006.

[4] Image processing on line. http://www.ipol.im/, 2009.

[5] Vcglib: Visualization and computer graphics library. http://vcg.isti.cnr.it/vcglib/, 2011. Accessed: 2017-01-01.

[6] *The ITK Software Guide*. Kitware, Inc., 2015.

[7] H. Adams, A. Tausz, and M. Vejdemo-Johansson. Javaplex: A research software package for persistent (co)homology. In *ICMS*, 2014. https://github.com/appliedtopology/javaplex.

[8] J. Ahrens, B. Geveci, and C. Law. Paraview: An end-user tool for large-data visualization. *The Visualization Handbook*, pp. 717–731, 2005.

[9] D. Attali, U. Bauer, O. Devillers, M. Glisse, and A. Lieutier. Homological reconstruction and simplification in R3. In *Symp. on Comp. Geom.*, 2013.

[10] D. Attali, M. Glisse, S. Hornus, F. Lazarus, and D. Morozov. Persistence-sensitive simplification of functions on surfaces in linear time. In *TopoInVis Workshop*, 2009.

[11] Ayasdi. https://www.ayasdi.com/.

[12] S. Bachthaler and D. Weiskopf. Continuous scatterplots. *IEEE Trans. on Vis. and Comp. Graph.*, 2008.

[13] T. F. Banchoff. Critical points and curvature for embedded polyhedral surfaces. *The American Mathematical Monthly*, 1970.

[14] C. Batty. Simplexmesh: A simplicial mesh structure that supports general non-manifold meshes and associated data. https://github.com/christopherbatty/SimplexMesh, 2013. Accessed: 2017-01-01.

[15] U. Bauer, M. Kerber, J. Reininghaus, and H. Wagner. PHAT - persistent homology algorithms toolbox. In *ICMS*, 2014. https://github.com/blazs/phat.

[16] U. Bauer, C. Lange, and M. Wardetzky. Optimal topological simplification of discrete functions on surfaces. *Disc. Compu. Geom.*, 2012.

[17] L. Bavoil, S. P. Callahan, C. E. Scheidegger, H. T. Vo, P. Crossno, C. T. Silva, and J. Freire. Vistrails: Enabling interactive multiple-view visualizations. In *IEEE VIS*, 2005.

[18] S. Biasotti, D. Giorgio, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theoretical Computer Science*, 2008.

[19] D. K. Blandford, G. E. Blelloch, D. E. Cardoze, and C. Kadow. Compact representations of simplicial meshes in two and three dimensions. *Int. J. Comput. Geometry Appl.*, 15(1):3–24, 2005.

[20] J. Boissonnat and C. Maria. The simplex tree: An efficient data structure for general simplicial complexes. *Algorithmica*, 70(3):406–427, 2014.

[21] J.-D. Boissonnat, C. S. Karthik, and S. Tavenas. Building efficient and compact data structures for simplicial complexes. In *Symp. on Comp. Geom.*, 2015.

[22] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. OpenMesh: A Generic and Efficient Polygon Mesh Data Structure. In *OpenSG*, 2002.

[23] R. L. Boyell and H. Ruston. Hybrid techniques for real-time radar simulation. In *IEEE Fall Joint Computer Conference*, 1963.

[24] P. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell. Interactive exploration and analysis of large scale simulations using topology-based data segmentation. *IEEE Trans. on Vis. and Comp. Graph.*, 2011.

[25] M. L. Brewer, L. F. Diachin, P. M. Knupp, T. Leurent, and D. J. Melander. The mesquite mesh quality improvement toolkit. In *IMR*, 2003.

[26] P. Bubenik and P. Dłotko. A persistence landscapes toolbox for topological statistics. *Symb. Comp.*, 2017. https://www.math.upenn.edu/~dlotko/persistenceLandscape.html.

[27] G. Carlsson, V. De Silva, and D. Morozov. Zigzag persistent homology and real-valued functions. In *Symp. on Comp. Geom.*, 2009.

[28] H. Carr, Z. Geng, J. Tierny, A. Chattopadhyay, and A. Knoll. Fiber surfaces: Generalizing isosurfaces to bivariate data. *Comp. Graph. For.*, 2015.

[29] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *Symp. on Dis. Alg.*, pp. 918–926, 2000.

[30] H. Carr, J. Snoeyink, and M. van de Panne. Simplifying flexible isosurfaces using local geometric measures. In *IEEE VIS*, 2004.

[31] G. Chen, K. Mischaikow, R. S. Laramee, P. Pilarczyk, and E. Zhang. Vector field editing and periodic orbit extraction using morse decomposition. *IEEE Trans. on Vis. and Comp. Graph.*, 2007. http://www.sci.utah.edu/~chengu/vfanalysis.HTM.

[32] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. M. Favre, and P. Navrátil. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*, pp. 357–372. Oct 2012.

[33] H. Choi, W. Choi, T. M. Quan, D. G. Hildebrand, H. Pfister, and W.-K. Jeong. Vivaldi: A domain-specific language for volume processing and visualization on distributed heterogeneous systems. *IEEE Trans. on Vis. and Comp. Graph.*, 20(12):2407–2416, 2014.

[34] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In V. Scarano, R. D. Chiara, and U. Erra, eds., *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.

[35] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. In *Symp. on Comp. Geom.*, 2005.

[36] L. De Floriani, U. Fugacci, F. Iuricich, and P. Magillo. Morse complexes for shape segmentation and homological analysis: discrete models and algorithms. *Comp. Grap. For.*, 2015.

[37] V. De Silva, D. Morozov, and M. Vejdemo-Johansson. Persistent cohomology and circular coordinates. *Disc. Compu. Geom.*, 2011.

[38] T. K. Dey, F. Fan, and Y. Wang. Computing topological persistence for simplicial maps. In *Symp. on Comp. Geom.*, 2014. http://web.cse.ohio-state.edu/~tamaldey/SimPers/Simpers.html.

[39] S. Dillard. A contour tree library. http://graphics.cs.ucdavis.edu/~sdillard/libtourtre/doc/html/, 2007.

[40] H. Doraiswamy and V. Natarajan. Output-sensitive construction of reeb graphs. *IEEE Trans. on Vis. and Comp. Graph.*, 2012.

[41] H. Doraiswamy and V. Natarajan. Computing reeb graphs as a union of contour trees. *IEEE Trans. on Vis. and Comp. Graph.*, 2013.

[42] H. Doraiswamy and V. Natarajan. Recon (Reeb graph computation). http://vgl.csa.iisc.ac.in/software/software.php?pid=003, 2014.

[43] H. Doraiswamy, A. Sood, and V. Natarajan. LibRG (Reeb graph computation). http://vgl.csa.iisc.ac.in/software/software.php?pid=001, 2012.

[44] H. Edelsbrunner and J. Harer. Jacobi sets of multiple morse functions. In *Foundations of Computational Mathematics*, 2004.

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

[46] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Morse-smale complexes for piecewise linear 3-manifolds. In *SoCG*, 2003.

[47] H. Edelsbrunner, J. Harer, and A. K. Patel. Reeb spaces of piecewise linear mappings. In *Symp. on Comp. Geom.*, 2008.

[48] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Disc. Compu. Geom.*, 2002.

[49] H. Edelsbrunner, D. Morozov, and V. Pascucci. Persistence-sensitive simplification of functions on 2-manifolds. In *SoCG*, 2006.

[50] H. Edelsbrunner and E. P. Mucke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. on Graph.*, 1990.

[51] B. T. Fasy, J. Kim, F. Lecci, and C. Maria. Introduction to the R package TDA. *CoRR*, abs/1411.1830, 2014. https://cran.r-project.org/web/packages/TDA/index.html.

[52] G. Favelier, C. Gueunet, and J. Tierny. Visualizing ensembles of viscous fingers. In *IEEE SciVis Contest*, 2016.

[53] L. D. Floriani and A. Hui. Data structures for simplicial complexes: An analysis and A comparison. In *EG Symp. on Geom. Proc.*, 2005.

[54] R. Forman. A user's guide to discrete Morse theory. *Adv. in Math.*, 1998.

[55] S. Gerber, P.-T. Bremer, V. Pascucci, and R. Whitaker. Visual exploration of high dimensional scalar functions. *IEEE Trans. on Vis. and Comp. Graph.*, 2010.

[56] S. Gerber, K. Potter, and O. Ruebel. msr: Morse-smale approximation, regression and visualization. https://cran.r-project.org/web/packages/msr/, 2015.

[57] D. Guenther, R. Alvarez-Boto, J. Contreras-Garcia, J.-P. Piquemal, and J. Tierny. Characterizing molecular interactions in chemical systems. *IEEE Trans. on Vis. and Comp. Graph.*, 2014.

[58] D. Guenther, J. Salmon, and J. Tierny. Mandatory critical points of 2D uncertain scalar fields. *Comp. Graph. For.*, 2014.

[59] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny. Contour forests: Fast multi-threaded augmented contour trees. In *IEEE LDAV*, 2016.

[60] V. Guillemin and A. Pollack. *Differential Topology*. Prentice-Hall, 1974.

[61] T. Gurung, D. E. Laney, P. Lindstrom, and J. Rossignac. Squad: Compact representation for triangle meshes. *Comp. Grap. For.*, 2011.

[62] T. Gurung, M. Luffel, P. Lindstrom, and J. Rossignac. Zipper: A compact connectivity data structure for triangle meshes. *Comp.-Aid. Des.*, 2013.

[63] A. Gyulassy, P. Bremer, R. Grout, H. Kolla, J. Chen, and V. Pascucci. Stability of dissipation elements: A case study in combustion. *Comp. Graph. For.*, 2014.

[64] A. Gyulassy, P. T. Bremer, B. Hamann, and V. Pascucci. A practical approach to morse-smale complex computation: Scalability and generality. *IEEE Trans. on Vis. and Comp. Graph.*, 2008.

[65] A. Gyulassy, P. T. Bremer, and V. Pascucci. Computing morse-smale complexes with accurate geometry. *IEEE Trans. on Vis. and Comp. Graph.*, 2012.

[66] A. Gyulassy, D. Guenther, J. A. Levine, J. Tierny, and V. Pascucci. Conforming morse-smale complexes. *IEEE Trans. on Vis. and Comp. Graph.*, 2014.

[67] A. Gyulassy, A. Knoll, K. Lau, B. Wang, P. Bremer, M. Papka, L. A. Curtiss, and V. Pascucci. Interstitial and interlayer ion diffusion geometry extraction in graphitic nanosphere battery materials. *IEEE Trans. on Vis. and Comp. Graph.*, 2015.

[68] A. Gyulassy, V. Natarajan, M. Duchaineau, V. Pascucci, E. Bringa, A. Higginbotham, and B. Hamann. Topologically Clean Distance Fields. *IEEE Trans. on Vis. and Comp. Graph.*, 2007.

[69] 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. *Comp. Grap. For.*, 2016.

[70] L. Hüttenberger, C. Heine, H. Carr, G. Scheuermann, and C. Garth. Towards multifield scalar topology based on pareto optimality. *Comp. Graph. For.*, 32(3):341–350, 2013.

[71] C. Jamin, S. Pion, and M. Teillaud. 3D triangulation data structure. In *CGAL User and Reference Manual*. 2016.

[72] K. I. Joy, J. Legakis, and R. MacCracken. Data structures for multiresolution representation of unstructured meshes. In *Hierarchical and Geometrical Methods in Scientific Visualization*. 2003.

[73] G. Kindlmann, C. Chiw, N. Seltzer, L. Samuels, and J. Reppy. Diderot: a domain-specific language for portable parallel scientific visualization and image analysis. *IEEE Trans. on Vis. and Comp. Graph.*, 2016.

[74] P. Klacansky, J. Tierny, H. Carr, and Z. Geng. Fast and exact fiber surfaces for tetrahedral meshes. *IEEE Trans. on Vis. and Comp. Graph.*, 2016.

[75] D. E. Laney, P. Bremer, A. Mascarenhas, P. Miller, and V. Pascucci. Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities. *IEEE Trans. on Vis. and Comp. Graph.*, 2006.

[76] M. Luffel, T. Gurung, P. Lindstrom, and J. Rossignac. Grouper: A compact, streamable triangle mesh data structure. *IEEE Trans. Vis. Comput. Graph.*, 20(1):84–98, 2014.

[77] C. Maria, J. Boissonnat, M. Glisse, and M. Yvinec. The gudhi library: Simplicial complexes and persistent homology. In *ICMS*, 2014. `http://gudhi.gforge.inria.fr/`.

[78] P. McCormick, J. Inman, J. Ahrens, J. Mohd-Yusof, G. Roth, and S. Cummins. Scout: a data-parallel programming language for graphics processors. *Parallel Computing*, 33(10):648–662, 2007.

[79] J. Milnor. *Morse Theory*. Princeton U. Press, 1963.

[80] K. Mischaikow and V. Nanda. Morse theory for filtrations and efficient computation of persistent homology. *Disc. Compu. Geom.*, 2013.

[81] K. Moreland. A survey of visualization pipelines. *IEEE Trans. on Vis. and Comp. Graph.*, 2013.

[82] D. Morozov. Dionysus. `http://www.mrzv.org/software/dionysus`, 2010. Accessed: 2016-09-15.

[83] V. Nanda. Perseus, the persistent homology software. `http://www.sas.upenn.edu/~vnanda/perseus`, 2013. Accessed: 2016-09-15.

[84] S. Parsa. A deterministic o(m log m) time algorithm for the reeb graph. In *Symp. on Comp. Geom.*, 2012.

[85] V. Pascucci, G. Scorzelli, P. T. Bremer, and A. Mascarenhas. Robust on-line computation of Reeb graphs: simplicity and speed. *ACM Trans. on Graph.*, 2007.

[86] S. Pion and M. Yvinec. 2D triangulation data structure. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.9 ed., 2016.

[87] P. Rautek, S. Bruckner, M. E. Gröller, and M. Hadwiger. Vislang: A system for interpreted domain-specific languages for scientific visualization. *IEEE Trans. on Vis. and Comp. Graph.*, 2014.

[88] G. Reeb. Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique. *Acad. des Sci.*, 1946.

[89] V. Robins, P. Wood, and A. Sheppard. Theory and algorithms for constructing discrete morse complexes from grayscale digital images. *IEEE Trans. on Pat. Ana. and Mach. Int.*, 2011.

[90] W. J. Schroeder, F. Bertel, M. Malaterre, D. Thompson, P. P. Pebay, R. O'Bara, and S. Tendulkar. Methods and framework for visualizing higher-order finite elements. *IEEE TVCG*, 2006.

[91] W. J. Schroeder, K. Martin, and W. E. Lorensen. *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics (3rd edition)*. Kitware, Inc., 2004.

[92] SCI Institute, 2016. SCIRun: A Scientific Computing Problem Solving Environment, Scientific Computing and Imaging Institute (SCI), Download from: `http://www.scirun.org`.

[93] N. Shivashankar. *Morse-Smale Complexes: Computation and Applications*. PhD thesis, Indian Institute of Science, 2014.

[94] N. Shivashankar, S. Maadasamy, and V. Natarajan. Parallel computation of 2d morse-smale complexes. *IEEE Trans. on Vis. and Comp. Graph.*, 2012.

[95] N. Shivashankar and V. Natarajan. Parallel computation of 3d morse-smale complexes. *Comp. Graph. For.*, 2012.

[96] N. Shivashankar and V. Natarajan. Efficient software for programmable visual analysis using morse-smale complexes. In *TopoInVis*, 2015. `http://vgl.csa.iisc.ac.in/mscomplex/software.html`.

[97] N. Shivashankar, P. Pranav, V. Natarajan, R. van de Weygaert, E. P. Bos, and S. Rieder. Felix: A topology based framework for visual exploration of cosmic filaments. *IEEE Trans. on Vis. and Comp. Graph.*, 2016. `http://vgl.serc.iisc.ernet.in/felix/index.html`.

[98] D. Sieger and M. Botsch. Design, implementation, and evaluation of the surface_mesh data structure. In *IMR*, 2011.

[99] G. Singh, F. Mémoli, and G. E. Carlsson. Topological methods for the analysis of high dimensional data sets and 3d object recognition. In *SPBG*, pp. 91–100, 2007. `http://danifold.net/mapper/`.

[100] P. Skraba, P. Rosen, B. Wang, G. Chen, H. Bhatia, and V. Pascucci. Critical point cancellation in 3d vector fields: Robustness and discussion. *IEEE Trans. on Vis. and Comp. Graph.*, 2016.

[101] B. S. Sohn and C. L. Bajaj. Time varying contour topology. *IEEE Trans. on Vis. and Comp. Graph.*, 2006.

[102] T. Sousbie. The persistent cosmic web and its filamentary structure: Theory and implementations. *Royal Astronomical Society*, 2011. `http://www2.iap.fr/users/sousbie/web/html/indexd41d.html`.

[103] S. Tarasov and M. Vyali. Construction of contour trees in 3d in o(n log n) steps. In *Symp. on Comp. Geom.*, 1998.

[104] The CGAL Project. *CGAL User and Reference Manual*. 2016.

[105] D. M. Thomas and V. Natarajan. Multiscale symmetry detection in scalar fields by clustering contours. *IEEE TVCG*, 2014.

[106] J. Tierny. vtkReebGraph class collection. `http://www.vtk.org/doc/nightly/html/classvtkReebGraph.html`, 2009.

[107] J. Tierny and H. Carr. Jacobi fiber surfaces for bivariate Reeb space computation. *IEEE Trans. on Vis. and Comp. Graph.*, 2016.

[108] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux. The Topology ToolKit. `https://topology-tool-kit.github.io/`.

[109] J. Tierny, A. Gyulassy, E. Simon, and V. Pascucci. Loop surgery for volumetric meshes: Reeb graphs reduced to contour trees. *IEEE Trans. on Vis. and Comp. Graph.*, 2009.

[110] J. Tierny and V. Pascucci. Generalized topological simplification of scalar fields on surfaces. *IEEE Trans. on Vis. and Comp. Graph.*, 2012.

[111] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pasucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Symp. on Comp. Geom.*, 1997.

[112] A. Vintescu, F. Dupont, G. Lavoué, P. Memari, and J. Tierny. Conformal factor persistence for fast hierarchical cone extraction. In *Eurographics (short papers)*, 2017.

[113] G. Weber, S. E. Dillard, H. Carr, V. Pascucci, and B. Hamann. Topology-controlled volume rendering. *IEEE TVCG*, 2007.

[114] K. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications*, 1985.

[115] K. Weiss, L. D. Floriani, R. Fellegara, and M. Velloso. The pr-star octree: a spatio-topological data structure for tetrahedral meshes. In *ACM Symposium on Advances in GIS*, 2011.

[116] K. Weiss, F. Iuricich, R. Fellegara, and L. D. Floriani. A primal/dual representation for discrete morse complexes on tetrahedral meshes. *Comp. Graph. For.*, 2013.

[117] A. Zomorodian. The tidy set: a minimal simplicial set for computing homology of clique complexes. In *Symp. on Comp. Geom.*, 2010.

[118] A. Zomorodian and G. Carlsson. Computing persistent homology. *Disc. Compu. Geom.*, 2005.

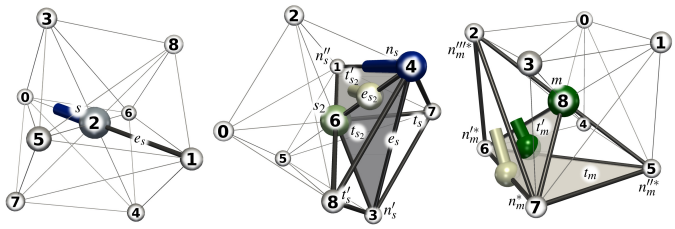

Fig. 10. Discrete gradient pairing obtained with Alg. 1 in 3D in the star of a PL 1-saddle (left), 2-saddle (center) and maximum (right). Vertex-edge, edge-triangle and triangle-tetrahedron pairs are shown with blue, white and green balls-and-sticks. Only a few pairs is shown to avoid occlusion.

APPENDIX A: PL-MATCHING PROPERTY IN 3D

Critical simplices of dimension 0 still precisely coincide with PL minima (same argument as in 2D). Moreover, PL 1-saddles will admit at least one critical edge in their star (same argument as in 2D). By definition, the upper link $Lk^+(s_2)$ of a PL 2-saddle s_2 is made of at least two connected components, each of which containing a local maximizing vertex. Thus, the restriction of f on $Lk(s_2)$ will admit at least two PL maxima, and therefore at least one PL saddle n_s (to satisfy the Morse inequalities), precisely separating $Lk^+(s_2)$ from $Lk^-(s_2)$. Due to the 2D argument, n_s will admit a critical edge e_s in its star on $Lk(s_2)$, linking it to a lower vertex $n'_s \in Lk^-(s_2)$. Let t_{s_2} be the triangle containing e_s and s_2. Since e_s is not paired with its co-faces t_s and t'_s, it means that n'_s is the minimizing vertex of both triangles. Thus, n'_s is also the minimizing vertex of the co-faces of t_{s_2} in $St(s_2)$. Thus, t_{s_2} cannot be paired with its co-faces (observation *(ii)*). Since $n'_s \in Lk^-(s_2)$, the edge e_{s_2} linking s_2 to n_s maximizes t_{s_2}. As discussed in the 2D case, n_s has to be paired with the edge connecting it to its minimum neighbor on $Lk(s_2)$, n''_s. Let t'_{s_2} be the triangle containing s_2, n_s and n''_s. e_{s_2} maximizes both t_{s_2} and t'_{s_2} but can be paired only once, with the one containing the minimizing vertex, t'_{s_2}. Thus, the triangle t_{s_2} will be left unpaired by Alg. 1, and thus critical.

Similarly, let n^*_m be the highest vertex of the link $Lk(m)$ of a 3D PL maximum m. Then, n^*_m is itself a 2D PL maximum on $Lk(m)$. Given the 2D argument, there must exist a triangle t_m on $Lk(m)$ which contains n^*_m and which is paired with no simplex of $Lk(m)$. Moreover, t_m also contains the vertex n'^*_m, which is the maximizer of the link of n^*_m on that of m (see the 2D argument). Let n''^*_m be the remaining vertex of t_m. Let t'_m be the triangle of $St(m)$ which contains both n^*_m and n'^*_m and let T_m and T'_m be its two adjacent tetrahedra in $St(m)$. We will consider that T_m is the tetrahedron containing n''^*_m and T'_m that containing a fourth vertex n'''^*_m. Note that n'''^*_m is lower than n''^*_m (otherwise t_m would not be unpaired on $Lk(m)$) and we have: $f(n'''^*_m) < f(n''^*_m) < f(n'^*_m) < f(n^*_m) < f(m)$. The triangle t'_m is the maximizer of both T_m and T'_m, however, it can be paired with only one of them (observation *(i)*), its minimizing co-face T'_m, as it contains the lowest vertex, n'''^*_m, of the two tetrahedra. Therefore, the tetrahedron T_m will be left unpaired by Alg. 1, and thus critical.

APPENDIX B: VTK PIPELINE INTEGRATION

A notable software engineering challenge was the seamless integration of our cached triangulation data structure (`ttk::Triangulation`) into VTK's pipeline. A naive strategy consists in storing one instance within each VTK wrapper (Sec. 6.2). However, this would duplicate the data structures in the frequent situation where multiple TTK modules are lined up within a single VTK pipeline. Instead, we implemented a strategy which makes each `ttk::Triangulation` object travel through each pipeline branch without data copy. In particular, for each VTK class supported by TTK (`vtkUnstructuredGrid`, `vtkPolyData`, `vtkImageData`, etc.), we derived by inheritance a TTK specialization (`ttkUnstructuredGrid`, `ttkPolyData`, `ttkImageData`, etc.) which holds a pointer to a `ttk::Triangulation` object. This pointer is copied upon VTK's `ShallowCopy()` operation and a new object is actually allocated upon a VTK `DeepCopy()` operation. On the latter operation, primitive-type pointers to the point and simplex lists ($\mathscr{L}_P$ and $\mathscr{L}_S$, Sec. 5.2) are passed to the triangulation data structure in the case of meshes, and dimensions are passed for that of regular grids. Within each VTK wrapper, if the input is a pure VTK object (and not a derived TTK object), it is first converted into its matching TTK derived class. Then the pointer to the `ttk::Triangulation` is extracted and passed to the base code functor (Sec. 6.2). Note that this mechanism is automatically handled by TTK and is completely hidden to developers, who only see more general `vtkDataSet` objects passed as arguments of their VTK wrappers. As a consequence, `ttk::Triangulation` objects are allocated only once per pipeline branch, and travel by pointers down this branch, without data copy, possibly progressively extending their lists of internal lookup tables upon the precondition calls triggered by the successive TTK modules present in the pipeline branch.