DOI: 10.1111/cgf.13764

COMPUTER GRAPHICS *forum*
Volume 39 (2020), number 1 pp. 244–259

Efficient Homology-Preserving Simplification of High-Dimensional Simplicial Shapes

Riccardo Fellegara[1], Federico Iuricich[2], Leila De Floriani[1] and Ulderico Fugacci[3]

[1]University of Maryland, College Park, MD, USA
felle@umd.edu, deflo@umiacs.umd.edu
[2]Clemson University, Clemson, SC, USA
fiurici@clemson.edu
[3]Graz University of Technology, Graz, Austria
fugacci@tugraz.at

Abstract

Simplicial complexes are widely used to discretize shapes. In low dimensions, a 3D shape is represented by discretizing its boundary surface, encoded as a triangle mesh, or by discretizing the enclosed volume, encoded as a tetrahedral mesh. High-dimensional simplicial complexes have recently found their application in topological data analysis. Topological data analysis aims at studying a point cloud P, possibly embedded in a high-dimensional metric space, by investigating the topological characteristics of the simplicial complexes built on P. Analysing such complexes is not feasible due to their size and dimensions. To this aim, the idea of simplifying a complex while preserving its topological features has been proposed in the literature. Here, we consider the problem of efficiently simplifying simplicial complexes in arbitrary dimensions. We provide a new definition for the edge contraction operator, based on a top-based data structure, with the objective of preserving structural aspects of a simplicial shape (i.e., its homology), and a new algorithm for verifying the link condition on a top-based representation. We implement the simplification algorithm obtained by coupling the new edge contraction and the link condition on a specific top-based data structure, that we use to demonstrate the scalability of our approach.

Keywords: simplicial shapes, edge contraction, geometric modelling, mesh processing, homology-preserving simplification, topological data analysis

ACM CCS: I.3.5 [Computer Graphics]: • Computational Geometry and Object Modelling–Hierarchy and geometric transformations

1. Introduction

Simplicial complexes are the most common tool for encoding a geometric shape. In particular, triangle and tetrahedral meshes are often used for discretizing 3D shapes and for studying their geometric properties. In topological data analysis, where the aim is to extract descriptors from data in the form of topological invariants such as homology or persistent homology [ELZ02], high-dimensional simplicial complexes are a key discretization tool.

Such complexes are computed from point data or complex networks, including sensor networks [dSG07, SHPW17], brain networks [DMFC12, CRS15], social networks [BG14] and others [DPS*13]. Examples of high-dimensional simplicial complexes that can be computed on these type of data are alpha shapes [EM94], tidy sets [Zom10b] or Vietoris-Rips (VR) complexes [Zom10a].

In such applications, simplicial complexes are not limited to triangle or tetrahedral meshes, but they are defined as collections of p-dimensional hyper-tetrahedra, simply called *p-simplices*. With the growing of the dimension p of the simplices, the number of simplices forming a simplicial complex grows exponentially. This, combined with the native huge size of real-world datasets, requires new scalable and efficient approaches capable of dealing with the challenge posed by big data.

Reducing the size of a simplicial complex, while preserving its geometric properties, is a standard approach in mesh processing.

Edge contraction has been the most common operator for simplifying simplicial shapes since its first application in computer graphics and visualization [PH97, PS97, GGK02, CDFM*04, DFKP05, WRK*10, ZQC*14]. More recently, a new class of algorithms appeared in topological data analysis that aims at simplifying a mesh via edge contraction, while preserving some structural properties, like its homology [ALS11] or its persistent homology [ID17, DS18].

Our aim here is to design and develop an efficient approach for simplifying a simplicial complex in arbitrary dimensions while preserving its homology. Roughly speaking, the homology of a simplicial shape counts the presence of n-dimensional 'holes', like connected components, loops, voids, and so on. Edge contraction alone does not preserve the homology of a shape. However, by means of the link condition [DEGN99, ALS11], one can check if a certain edge contraction would modify the homology and prevent it. We propose an efficient dimension-independent and homology-preserving simplification algorithm for simplicial complexes, based on edge contraction, that scales well with both the size and the dimension of the complex. Our approach makes use of a class of data structures for simplicial complexes, that we call *top-based* data structures. A *top-based* representation of a simplicial complex encodes only its vertices and top simplices, i.e., the simplices which do not bound any other simplex in the complex, thereby providing scalability with the dimension and the size of the complex. At the basis of our simplification approach, there is also a new link condition that we introduce for a top-based representation, and we prove to be equivalent to the classical one [DEGN99, ALS11].

The major contributions of this paper are:

(1) the definition of a dimension-independent homology-preserving edge contraction algorithm on a top-based representation, and a proof of its correctness;
(2) the definition of a new link condition for a top-based representation and a proof of its equivalence to the link condition in [ALS11];
(3) the design and implementation of an efficient algorithm for simplifying a simplicial complex based on a very compact top-based data structure, the *Stellar tree* [FWD17];
(4) an extensive comparison with respect to a data structure for simplicial complexes designed for performing edge contractions, the *Skeleton-Blocker* data structure [ALS11] for both high-dimensional complexes, and triangulated and tetrahedralized shapes.

The remainder of this paper is organized as follows. In Section 2, we review some background notions on simplicial complexes, on simplicial homology, and on homology-preserving edge contraction. In Section 3, we discuss related work on data structures for simplicial complexes and on edge contraction. In Section 4, we define *top-based edge contraction*, which is an edge contraction working on a top-based representation of a simplicial complex. In Section 5, we introduce a new way for checking the link condition, that we call the *top-based link condition*, and we describe an algorithm for performing homology-preserving edge contraction on a top-based representation. In Section 6, we briefly review the top-based data structure used in our work, the *Stellar tree*, and we describe the homology-preserving simplification algorithm that we have developed based on it. Experimental results and comparisons with a state-of-the-art data structure are presented in Section 7, and concluding remarks are drawn in Section 8.

2. Background Notions

In this section, we review the definition of simplicial complex, and some notions on simplicial homology. We then review the edge contraction operator, and the so-called *link condition*, a condition ensuring that an edge contraction is homology-preserving.

2.1. Simplicial complexes

A *p-simplex* σ is the convex hull of $p+1$ affinely independent points in the Euclidean space. For instance, a 0-simplex is a single point, a 1-simplex an edge, a 2-simplex a triangle, and a 3-simplex a tetrahedron. We denote the p-simplex spanned by the vertices $v_0, v_1, \ldots, v_p$ as $\sigma = \{v_0, v_1, \ldots, v_p\}$. In the following, with a small abuse of notation, we denote as v the simplex $\{v\}$ spanned by a single vertex v, whenever no ambiguity arises. In a p-simplex σ, p is called the *dimension* of σ. Any simplex σ, which is the convex hull of a non-empty subset of the points generating a simplex τ, is a *face* of τ. Conversely, τ is a *coface* of σ. A simplex σ, which is only a coface of itself, is a *top simplex* (or *facet*).

A *simplicial complex* Σ is a finite set of simplices, such that:

- each face of a simplex in Σ belongs to Σ;
- for each pair of simplices σ and τ, either $\sigma \cap \tau = \emptyset$ or $\sigma \cap \tau$ is a face of both.

If d is the maximum dimension of all the simplices in Σ, we say that Σ is a simplicial d-complex. Given a simplex $\sigma \in \Sigma$, the *boundary* of σ, denoted as $Bd(\sigma)$, is the set of faces of σ, while the *star*, or *coboundary*, of σ, denoted as $St(\sigma)$, is the set of cofaces of σ. The *immediate boundary* of a p-simplex σ is formed by its $(p-1)$-faces, while its *immediate coboundary* is formed by its $(p+1)$-cofaces. The *link* of a simplex $\sigma \in \Sigma$ is the set of faces of the simplices in $St(\sigma)$ that do not intersect σ and is denoted as $Lk(\sigma)$ (see Figure 1).

VR-complexes [Hau94, Zom10a] are simplicial complexes widely used for representing the shape of a point data embedded in the Euclidean space. They are defined in terms of a *neighbourhood graph* for a set V of points. Given a threshold value ω, the *neighbourhood graph* defined by set V is the graph with vertex set V, in which there exists an edge for each pair of vertices in V having a distance less or equal to ω. The VR-complex is the *flag complex* of the neighbourhood graph. Recall that the *flag complex* of a graph $G = (V, E)$ is the simplicial complex whose simplices coincide with the maximal cliques of G. Note that the definition of flag complex holds in general for abstract simplicial complexes, i.e., simplicial complexes not embedded in the Euclidean space.

2.2. Simplicial homology

Simplicial homology is a powerful tool for shape analysis, which provides invariants for shape description and characterization. Given a simplicial complex Σ, it is possible to define the *chain complex* associated with Σ, $C_*(\Sigma) := (C_p(\Sigma), \partial_p)_{p \in \mathbb{N}}$. $C_p(\Sigma)$ is the free

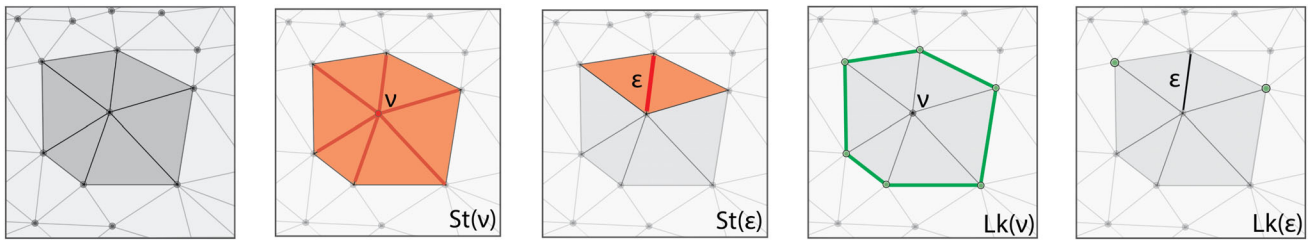


Figure 1: *An example of a manifold simplicial 2-complex (also called a triangle mesh). The star of a vertex $St(v)$ consists of the vertex v itself and of the 1-simplices (edges) and the 2-simplices (triangles) incident in it. The star of an edge $St(\epsilon)$ consists of the edge ϵ itself and its incident 2-simplices (triangles). The link of a vertex $Lk(v)$ consists of the faces of the triangles and of the edges in $St(v)$ which are not incident into v. Similarly, the link of an edge $Lk(\epsilon)$ consists two vertices, one for each incident triangle.*

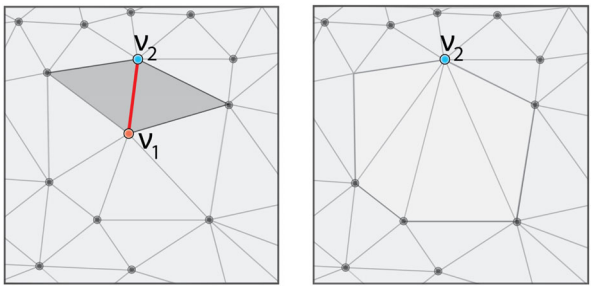
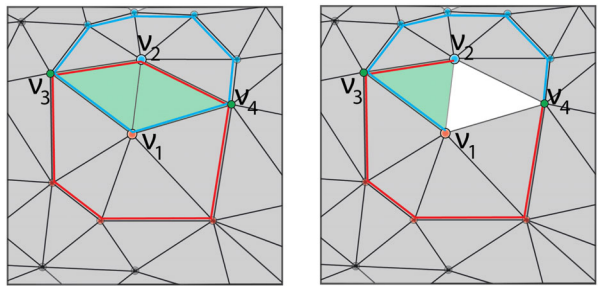


Figure 2: *Edge contraction of $\epsilon = \{v_1, v_2\}$ on a triangle mesh. All edges and triangles incident in v_1 are redirected to v_2 with the exception of the two triangles incident in ϵ (dark grey triangles) which are removed from the mesh.*



(a) (b)

Figure 3: *The edge contraction applied to $\epsilon = \{v_1, v_2\}$ in (a) is homology-preserving: the intersection of $Lk(v_1)$ and $Lk(v_2)$, the two vertices v_3 and v_4, coincides with $Lk(\epsilon)$. The edge contraction in (b) is not homology-preserving and it does not satisfy the link condition: the intersection of $Lk(v_1)$ and $Lk(v_2)$ consists of the two vertices v_3 and v_4, but the link of ϵ contains only vertex v_3.*

Abelian group generated by the p-simplices of simplicial complex Σ, and $\partial_p : C_p(\Sigma) \to C_{p-1}(\Sigma)$ is a homomorphism, called a *boundary map*, which encodes the boundary relations between p-simplices and $(p-1)$-simplices of Σ such that $\partial^2 = 0$. We denote as $Z_p(\Sigma) := \ker \partial_p$ the group of the p-cycles of Σ, and as $B_p(\Sigma) := Im\partial_{p+1}$ the group of the p-boundaries of Σ. Then, the p-th *homology group of* Σ is defined as

$$H_p(\Sigma) := H_p(C_*(\Sigma)) = \frac{Z_p(\Sigma)}{B_p(\Sigma)}.$$

Intuitively, homology groups reveal the presence of *holes* in a shape. The non-null elements of each homology group are cycles, which do not represent the boundary of any collection of simplices of Σ. The rank β_p of the p-th homology group of a simplicial complex Σ is called the p-th *Betti number* of Σ. Specifically, β_0 counts the number of connected components of Σ, β_1 the number of its tunnels and holes, and β_2 the number of its voids.

2.3. Homology-preserving edge contraction

We consider a simplicial d-complex Σ, and we denote as Σ_V the set of its vertices. The contraction of an edge $\epsilon = \{v_1, v_2\}$ of Σ collapses vertex v_1 into vertex v_2, thus eliminating both edge ϵ and vertex v_1. We can define an edge contraction by defining first a map μ_V on the set of vertices Σ_V as:

$$\mu_V(v) := \begin{cases} v_2 & \text{if } v = v_1 \\ v & \text{otherwise} \end{cases}.$$

Thus, the *edge contraction* of $\epsilon = \{v_1, v_2\}$ is the map μ on Σ induced by vertex map μ_V: for each simplex $\sigma = \{u_0, u_1, \ldots, u_p\}$, where $u_i \in \Sigma_V$ for $i = 0, 1, \ldots, p$, $\mu(\sigma) = \{\mu_V(u_0), \mu_V(u_1), \ldots, \mu_V(u_p)\}$. The complex obtained from Σ by applying map μ is denoted as $\mu(\Sigma)$. Figure 2 shows an example of an edge contraction.

The simplification of a simplicial complex Σ through edge contraction can possibly modify the homology of Σ. The *link condition*, introduced in [DEGN99] for manifold simplicial 2- and 3-complexes, and extended in [ALS11] to arbitrary simplicial complexes, provides a test for verifying if an edge contraction preserves the homology of the complex.

An edge $\epsilon = \{v_1, v_2\} \in \Sigma$ is said to satisfy the *link condition* if and only if

$$Lk(v_1) \cap Lk(v_2) \subseteq Lk(\epsilon). \quad (1)$$

Since the opposite inclusion is always satisfied, the link condition is often expressed in the literature as $Lk(v_1) \cap Lk(v_2) = Lk(\epsilon)$. If an

edge ϵ satisfies the link condition, then the edge contraction applied to ϵ preserves the homology of Σ. Notice that the opposite is not true. Figure 3(a) shows an example of an edge that satisfies the link condition, and Figure 3(b) shows one that does not.

Since the size of a simplicial complex increases exponentially with its dimension [Ede87], checking the link condition is computationally expensive on high-dimensional complexes. For this reason, a weaker condition, called the *p-link condition* (or the *weak link condition*), has been introduced in [DHKS13]. An edge $\epsilon = \{v_1, v_2\}$ satisfies the *p-link condition* if and only if every $(p-1)$-simplex in $Lk(v_1) \cap Lk(v_2)$ is also in $Lk(\epsilon)$. Thus, the p-link condition provides an incremental test for the link condition. Note that an edge $\epsilon = \{v_1, v_2\}$ satisfies the link condition if and only if it satisfies the p-link condition for all $1 \leq p \leq d$.

3. Related Work

Several data structures have been proposed in the literature for encoding a simplicial complex in low dimensions, especially for manifold simplicial 2-complexes (*triangle meshes*), and manifold simplicial 3-complexes (*tetrahedral meshes*) [DFH05, GR09, GR10, GLLR11]. However, only few existing data structures are dimension-independent. We can classify dimension-independent data structures as *simplex-based* and *top-based*, depending on whether they encode all the simplices in the complex, or only its vertices and top simplices.

The most general simplex-based data structure is the *Incidence Graph (IG)* [Ede87], which encodes the *Hasse diagram* of a complex, and can represent general cell complexes. The *IG* can be viewed as a graph $G = (N, A)$, in which the nodes represent the simplices of Σ, and the (undirected) arcs connect pairs of simplices of Σ of consecutive dimension, which are in the immediate boundary or coboundary of each other. A less verbose data structure with the same representation power is the *Incidence Simplicial (IS)* data structure [DHPC10], which also encodes all simplices of Σ. For each p-simplex $\sigma \in \Sigma$, the *IS* data structure encodes its $(p-1)$-faces, as in the *IG*, but, instead of storing all the $(p+1)$-simplices in the star of σ, the *IS* stores only one $(p+1)$-simplex for each connected component in the link of σ. It has been shown in [DHPC10] that the *IS* is definitely more compact than the *IG* in low dimensions, but the gap is reduced when working in higher dimensions.

The *Simplex Tree* data structure [BM14] encodes all the simplices of a complex in a *trie*. It is a specific spanning tree of the Hasse diagram in which only the arcs following the lexicographic order of the vertices are encoded. In addition, a linked circular list connects nodes at the same depth, having the same last vertex. The Simplex Tree has been defined for efficient extraction of boundary relations, as required in performing homology computations [ELZ02]. In [FIDF19], it has been shown that the Simplex Tree is generally *five* times more compact than the *IG*. Still, since it encodes all the simplices of the simplicial complex, the Simplex Tree also suffers from scalability issues.

Top-based data structures are a more compact and scalable alternative. The size of a simplicial complex, i.e., the number of its simplices, increases with its dimension, while the number of its vertices and that of its top simplices do not. Top-based data structures take advantage of this fact by encoding only vertices and top simplices. These representations have been widely used in low dimensions as well, for triangle [PBCF93, RSS01, GR10, GLLR11] and tetrahedral meshes [Nie97, GR09, GR10], for simplicial 2-complexes [DFMPS04] and 3-complexes [DFH03].

The *Generalized Indexed data structure with Adjacencies* (IA^*) [CDFW11] is the first top-based data structure defined in a dimension-independent way, generalizing the *IA* data structure for triangle and tetrahedral meshes both in the domain and in the dimension. The IA^* data structure encodes all vertices and top simplices of a simplicial complex Σ, plus the boundary relation between each top simplex and its vertices, adjacency relations among top simplices, and partial information about the star of each vertex. Public-domain implementations are available [Can14, Iur16]. In [FIDF19, FWD17], it has been shown that the IA^* data structure is more compact than the *IG*, requiring, on average, 80% less storage, and it is always *more compact* than the Simplex Tree, requiring from 30% of the storage on lower dimensional datasets, and a small fraction of the storage on higher dimensional datasets, where it can be observed a degenerate behaviour of the Simplex Tree as the complex dimension increases. A more compact representation for simplicial complexes embedded in the Euclidean space is provided by the *Stellar tree* [FWD17], as described in Section 6.

In [BST17], two top-based data structures have been proposed. The *Maximal Simplex Tree* is an induced subgraph of the Simplex Tree, in which only the paths corresponding to top simplices are encoded. Although it is a compact representation, it does not provide a complete description of the complex, since it is not possible to extract from it all its simplices. The *Simplex Array List* is a directed acyclic graph storing all the edges of a simplicial complex Σ and representing the incidence relations among the remaining simplices in such a way that a simplex in Σ is represented in at least one path in such a graph. While the Simplex Array List could be a viable option for achieving compactness, performing edge contractions on such representation would be unfeasible, requiring to rebuild the entire data structure at each edge contraction [BST17]. Moreover, no implemented version is available in the public domain for both these data structures.

The *Skeleton-Blocker* data structure [ALS11] has been specifically designed for homology-preserving edge contractions on simplicial complexes. It encodes the 1-skeleton of a simplicial complex Σ, which is the 1-complex formed by its vertices and edges, plus, for each edge, the set of incident blockers. A *blocker* (also called an *empty* simplex) is a p-simplex σ that is not in Σ, but all its faces are. The *Skeleton-Blocker* uses a sort of dual approach for encoding a complex, and thus, it cannot be classified either as simplex-based or top-based. In Section 7, we describe the *Skeleton-Blocker* data structure in more details when comparing it to our approach based on the *Stellar tree* [FWD17].

Edge contraction has been a very common operation for simplifying triangle [PS97, GGK02] and tetrahedral meshes [CDFM*04, WRK*10, ZQC*14]. A very large body of literature exists dealing with geometric ways of preserving the quality of the simplified mesh and of the surface approximation, and with

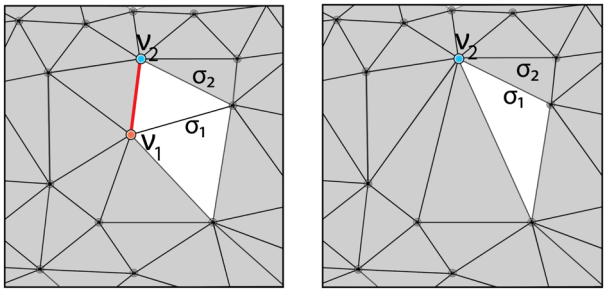


Figure 4: *Edge contraction of $\epsilon = \{v_1, v_2\}$ on a triangle mesh. Top simplex σ_1 becomes a face of triangle σ_2 after the contraction of edge ϵ.*

multi-resolution models based on such operation [DFKP05]. Edge contraction algorithms for general (non-manifold) simplicial 2- and 3-complexes have been presented in [DFMPS04, DFH03], where the complexes have been encoded in dimension-specific top-based data structures. Edge contraction algorithms for arbitrary-dimensional simplicial complexes are described in [PH97] based on an encoding of the complex as an *IG* and in [ALS11] based on the Skeleton-Blocker representation.

4. Edge Contraction on a Top-Based Representation of a Simplicial Complex

In this section, we provide the definition of the edge contraction operator for a top-based representation of a simplicial complex. Note that our definition is independent of the way the top-based representation is actually encoded. Given a simplicial complex Σ, its *top-based representation*, that we denote as Σ_{top}, consists only of the top simplices of Σ expressed in terms of their vertices.

A *top-based edge contraction* consists of applying map μ, defined in Section 2.3, to the simplices of Σ_{top} rather than to all the simplices of Σ. An edge contraction on Σ and the corresponding top-based edge contraction on Σ_{top} determine (up to closure) the same simplicial complex, as proven in Proposition 1. Recall that the *closure* of a set of simplices S, denoted as $Cl(S)$, is the smallest simplicial complex that contains each simplex in S.

Proposition 1. $Cl(\mu(\Sigma_{top})) = \mu(\Sigma)$.

The proof is reported in Appendix A.1.

Proposition 1 tells us that the top simplices of simplicial complex $\mu(\Sigma)$, obtained from Σ after the contraction of edge $\epsilon = \{v_1, v_2\}$, can be retrieved by applying map μ to the simplices in Σ_{top}. On the other hand, applying μ to a simplex τ in Σ_{top} may generate a simplex $\mu(\tau)$ that is no longer a top simplex, and thus, the top-based representation $\mu(\Sigma)_{top}$ of complex $\mu(\Sigma)$ might not be a proper subset of $\mu(\Sigma_{top})$. Figure 4 shows an example of a top simplex, edge σ_1, that becomes a face of σ_2 after the contraction of edge $\epsilon = \{v_1, v_2\}$.

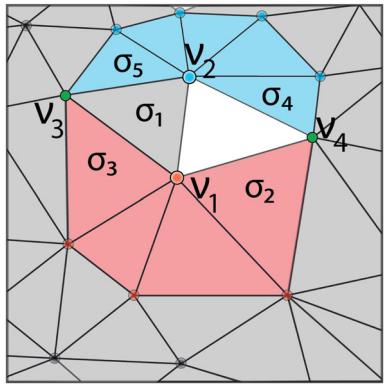


Figure 5: *Example of an edge contraction on $\epsilon = \{v_1, v_2\}$ not satisfying the link condition. The top simplices incident in v_1 (depicted in red) and those incident in v_2 (depicted in blue) are intersected. The simplices σ_4 and σ_2 are incident in vertex v_4, but, since there is no top simplex in $St_{top}(\epsilon)$ that contains v_4, the link condition is not verified.*

5. Homology-Preserving Edge Contraction

In this section, we present an algorithm for homology-preserving edge contraction on the top-based representation of a simplicial complex. Notice that it can be applied to any top-based representation, like an adjacency-based data structure [PBCF93, CDFW11, DFH03, DFMPS04], a hierarchical topological data structure [FWD17] and a corner-based data structure [RSS01, GR09, GR10, GLLR11]. Given the top-based representation Σ_{top} of a simplicial complex Σ, the objective is to obtain, through a homology-preserving edge contraction applied to Σ_{top}, the top-based representation of the resulting complex. To this aim, we define a *top-based link condition*, which we prove to be equivalent to the link condition.

5.1. Top-based link condition

The link condition provides a sufficient condition for declaring an edge contraction homology-preserving. This test is computationally expensive since it requires considering all the simplices of a complex. We introduce here a new link condition that works only on the top simplices, and thus, on the top-based representation of a simplicial complex.

We define, for each simplex σ in Σ, the *top star* of σ, $St_{top}(\sigma)$, as the set of top simplices in the star of σ. We define an edge $\epsilon = \{v_1, v_2\} \in \Sigma$ to satisfy the *top-based link condition* if and only if

$$\forall \tau_1, \tau_2 \text{ exists } \tau \in St_{top}(\epsilon) \text{ such that } \tau_1 \cap \tau_2 \subseteq \tau \qquad (2)$$

with $\tau_1 \in St_{top}(v_1) - St_{top}(\epsilon)$ and $\tau_2 \in St_{top}(v_2) - St_{top}(\epsilon)$.

Proposition 2. *The link condition* (1) *and the top-based link condition* (2) *are equivalent.*

The proof is reported in Appendix A.2.

The above proposition ensures that, if an edge ϵ in Σ satisfies the top-based link condition, then an edge contraction applied to ϵ preserves the homology of Σ.

Algorithm 1. LINK_CONDITION (ϵ, $St_{top}(v_1)$, $St_{top}(v_2)$, $St_{top}(\epsilon)$)

Input: $\epsilon = \{v_1, v_2\}$, edge to be checked
Input: $St_{top}(v_1)$, $St_{top}(v_2)$, $St_{top}(\epsilon)$, top stars of v_1, v_2 and ϵ // *encoded as sorted arrays*
Output: *true* if ϵ verifies the link condition, *false* otherwise

1: // T_i: *list of the top simplices incident in* v_i *but not in* ϵ
2: $T_1 \leftarrow St_{top}(v_1) - St_{top}(\epsilon)$
3: $T_2 \leftarrow St_{top}(v_2) - St_{top}(\epsilon)$
4: **for each** τ_1 in T_1 **do**
5: **for each** τ_2 in T_2 **do**
6: // ρ *is the face shared by* τ_1 *and* τ_2
7: $\rho \leftarrow$ COMMON_FACE(τ_1, τ_2)
8: **if** $\rho \neq \emptyset$ **then**
9: // *if* ρ *is not a face of any top simplex in* $St_{top}(\epsilon)$
10: **if** NOT_BOUNDARY_FACE(ρ, $St_{top}(\epsilon)$) **then**
11: return *false* // *the link condition is not verified*
12: **end if**
13: **end if**
14: **end for**
15: **end for**
16: return *true*

Algorithm 1 illustrates how to verify the top-based link condition. The first step consists of generating two lists T_1 and T_2, containing all top simplices in the star of v_1 and v_2, excluding those in the star of ϵ (rows 2 and 3). Then, a loop is performed on all the top simplices in T_1 and T_2 and, for each pair of top simplices τ_1 and τ_2, their shared face is extracted (row 7). If such face $\rho = \tau_1 \cap \tau_2$ exists (row 8) and ρ is not a face of any top simplex in the star of ϵ (row 10), the top-based link condition is not satisfied.

An example is shown in Figure 5. Here, by considering the top simplices incident in v_1 and v_2 and disregarding those incident in ϵ, we find the two vertices v_3 and v_4, depicted in green, contained, respectively, in T_1 and T_2. Since only v_3 is in the boundary of a top simplex σ_1 in the top star of ϵ, we conclude that the link condition is *not* satisfied.

We analyse the time complexity of Algorithm 1. We denote by m the maximum cardinality between $St_{top}(v_1)$ and $St_{top}(v_2)$ and by d the dimension of Σ. Generating lists T_1 and T_2 requires $\mathcal{O}(m\log(m))$ time, since $St_{top}(v_1)$, $St_{top}(v_2)$ and $St_{top}(\epsilon)$ are sorted arrays. Computing the face shared by two top simplices τ_1 and τ_2 (function NOT_BOUNDARY_FACE) is linear in the number of vertices of τ_1 and τ_2, and thus, $\mathcal{O}(d)$. For computing the boundary face ρ of a top simplex in $St_{top}(\epsilon)$, the algorithm iterates over all the top simplices in $St_{top}(\epsilon)$: for each top simplex τ, it checks if τ contains the vertices of ρ. The complexity of function NOT_BOUNDARY_FACE is thus $\mathcal{O}(dm)$. The time complexity of Algorithm 1 is dominated by that of the two nested loops, and thus, it is $\mathcal{O}(dm^3)$, which reduces to $\mathcal{O}(m^3)$, if we assume d to be a constant.

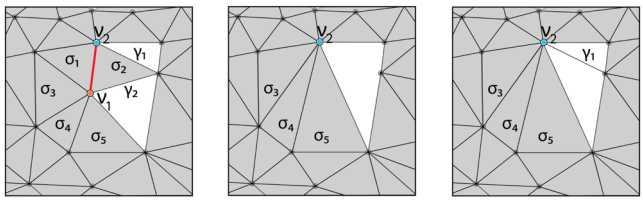


Figure 6: *Example of an edge contraction where* $St_{top}(\epsilon) = \{\sigma_1, \sigma_2\}$. *(b) The set of simplices* $R = \{\sigma_1, \sigma_2\}$ *is removed from* Σ_{top} *and the set of simplices* $I = \{\sigma_3, \sigma_4, \sigma_5\}$ *is updated changing their boundary vertex* v_1 *with* v_2. *(c) Since both faces* γ_1 *and* γ_2 *of* σ_2 *are not faces of any other simplex,* γ_1 *is introduced as a top simplex in* Σ_{top} *for preserving its homology.*

5.2. A homology-preserving top-based edge contraction algorithm

In Section 4, we have shown that a top-based edge contraction is equivalent to a classic edge contraction defined on all the simplices of a simplicial complex Σ. Thanks to Proposition 1, the top simplices of the simplicial complex $\mu(\Sigma)$ obtained after the contraction of edge $\epsilon = \{v_1, v_2\}$ can be retrieved by applying map μ to the simplices in Σ_{top}. It can happen that the application of function μ to a simplex τ in Σ_{top} produces a simplex $\mu(\tau)$ that is no longer a top simplex. Thus, $\mu(\Sigma_{top})$ may not be a top-based representation, since it might contain also non-top simplices, and thus, $\mu(\Sigma_{top}) \supsetneq \mu(\Sigma)_{top}$, as shown in the example of Figure 4. Our objective is to obtain $\mu(\Sigma)_{top}$ as result of a homology-preserving edge contraction applied to Σ_{top}.

We consider edge $\epsilon = \{v_1, v_2\}$ to be contracted into vertex v_2. Given a simplex τ in Σ_{top}, if τ belongs to $\Sigma_{top} - St_{top}(v_1)$, $\mu(\tau) = \tau$ is a top simplex in $\mu(\Sigma)$. Otherwise, we have two cases depending on whether τ belongs to $St_{top}(\epsilon)$ or $St_{top}(v_1) - St_{top}(\epsilon)$. The first case is discussed in Proposition 3, while the second case is discussed in Proposition 4.

Proposition 3. *Let* τ *be a simplex in* $St_{top}(\epsilon)$ *and, for* $i = 1, 2$, *let denote* γ_i *as the simplex* $\tau - v_i$. *We have that:*

$$\mu(\tau) \in \mu(\Sigma)_{top} \iff St_{top}(\gamma_1) \cup St_{top}(\gamma_2) = \{\tau\}. \quad (3)$$

The proof is reported in Appendix A.3.

Proposition 3 provides a simple condition for verifying whether a new top simplex is introduced during an edge contraction. In the example of Figure 6, we consider σ_2 and its boundary faces γ_1 and γ_2. Since both γ_1 and γ_2 are not faces of any other simplex, $\mu(\sigma_2) = \gamma_1$ is reintroduced as top simplex.

Proposition 4 considers the case in which simplex τ belongs to $St_{top}(v_1) - St_{top}(\epsilon)$. It shows that, if the top-based link condition is satisfied, all the top simplices in $St_{\text{top}}(v_1) - St_{top}(\epsilon)$ will be mapped into top simplices in $\mu(\Sigma)_{top}$. In other words, none of them will become a face of another top simplex. In general, this does not occur if the top-based link condition is not satisfied.

Proposition 4. *Let τ be a simplex in $St_{top}(v_1) - St_{top}(\epsilon)$. If the link condition* (1) *(or, equivalently, the top-based link condition* (2)*) is satisfied, then $\mu(\tau) \in \mu(\Sigma)_{top}$. Moreover, $\mu^{-1}(\mu(\tau)) \cap \Sigma_{top} = \{\tau\}$, i.e., τ is the only top simplex of Σ which is mapped into $\mu(\tau)$ via μ.*

The proof is reported in Appendix A.4.

Algorithm 2. CONTRACT (ϵ, $St_{top}(v_1)$, $St_{top}(v_2)$, $St_{top}(\epsilon)$, Σ_{top})

Input: Σ_{top}, top-based representation of simplicial complex Σ
Input: $\epsilon = \{v_1, v_2\}$, edge to be contracted
Input: $St_{top}(v_1)$, $St_{top}(v_2)$, $St_{top}(\epsilon)$, top stars of v_1, v_2 and ϵ
1: // *For each top simplex in the star of ϵ*
2: **for each** τ in $St_{top}(\epsilon)$ **do**
3: // *Consider the simplices γ_i such that $\tau \leftarrow \gamma_i \cup v_i$*
4: $\gamma_1 \leftarrow \tau - v_1$
5: $\gamma_2 \leftarrow \tau - v_2$
6: // *$St_{top}(\gamma_i)$ is formed only by the top simplices in $St_{top}(v_j)$ that have γ_i as face*
7: $St_{top}(\gamma_1) \leftarrow$ TOP_STAR(γ_1, $St_{top}(v_2)$)
8: $St_{top}(\gamma_2) \leftarrow$ TOP_STAR(γ_2, $St_{top}(v_1)$)
9: **if** $St_{top}(\gamma_1) \cup St_{top}(\gamma_2) = \{\tau\}$ **then**
10: $\Sigma_{top} \leftarrow \Sigma_{top} \cup \{\gamma_1\}$ // *Add γ_1 to Σ_{top}*
11: **end if**
12: $\Sigma_{top} \leftarrow \Sigma_{top} - \{\tau\}$ // *Remove τ from Σ_{top}*
13: **end for**
14: // *For each top simplex τ in the star of v_1*
15: **for each** τ in $(St_{top}(v_1) - St_{top}(\epsilon))$ **do**
16: $\tau \leftarrow (\tau - v_1) \cup v_2$ // *Replace v_1 with v_2 in τ*
17: **end for**

Algorithm 2 describes the procedure for performing the edge contraction μ of ϵ on a top-based representation. Since the validity of the link condition is required, the edge contraction described by the following algorithm is homology-preserving.

The algorithm takes as input the edge ϵ to be contracted, the top-based representation Σ_{top}, and the top stars of v_1, v_2 and ϵ. For each top simplex τ in $St_{top}(\epsilon)$, the algorithm checks if a top simplex γ_1 needs to be added in its place in order to preserve the homology of the complex. More precisely, as from Proposition 3, a simplex $\tau \in St_{top}(\epsilon)$ has to be replaced in Σ_{top} with $\mu(\tau) = \gamma_1$ if $St_{top}(\gamma_1) \cup St_{top}(\gamma_2) = \{\tau\}$. For checking this condition, Algorithm 2 executes the following steps. Two-candidate simplices γ_1 and γ_2 are extracted by removing v_1 and v_2, respectively, from τ (rows 4 and 5). The top stars of γ_1 and γ_2 are extracted by starting from $St_{top}(v_2)$ and $St_{top}(v_1)$, respectively (rows 7 and 8). Specifically, function TOP_STAR iterates over the top simplices in $St_{top}(v_j)$, and add to $St_{top}(\gamma_i)$ only those top simplices that have γ_i in their boundary. Finally (in rows 9 and 10), as established by Proposition 3, if the union of $St_{top}(\gamma_1)$ and $St_{top}(\gamma_2)$ consists just of τ, γ_1 is added to Σ_{top}. Then, τ is removed from Σ_{top} (row 12). The final step of Algorithm 2 updates the top simplices incident in v_1, but not in both v_1 and v_2 (rows 15–17). Thanks to Proposition 4, we can safely update Σ_{top} by replacing v_1 with v_2 in τ (rows 16).

We analyse the time complexity of Algorithm 2 in terms of the maximum cardinality m of $St_{top}(v_1)$ and $St_{top}(v_2)$ and of the dimension d of Σ. Function TOP_STAR iterates over the top simplices in $St_{top}(v_1)$ (or $St_{top}(v_2)$): for each top simplex σ, it checks if σ shares the same vertices with γ_2 (or γ_1). Thus, it requires $\mathcal{O}(dm)$ time. Note that the first loop is executed exactly $|St_{top}(\epsilon)|$ times, while the second one a number of times linear in the number of elements in $St_{top}(v_1) - St_{top}(\epsilon)$. Thus, the overall complexity of Algorithm 2 is $\mathcal{O}(dm^2)$, which is quadratic in m, if we assume d to be a constant.

6. Homology-Preserving Simplification on the Stellar Tree

In this section, we briefly review the *Stellar tree*, the top-based data structure we use for encoding simplicial complexes. We then describe an algorithm for homology-preserving simplification based on it.

6.1. The Stellar Tree

A *Stellar tree* is a spatio-topological data structure for simplicial complexes of arbitrary dimension, highly scalable and well suited for the analysis of both low- and high-dimensional simplicial complexes [FWD17]. It is based on a clustering of the vertices of a simplicial complex according to a nested decomposition of an n-dimensional hyper-cubical domain containing such vertices. A nested decomposition is a hierarchical space-based decomposition into hyper-cubical blocks in which overlapping blocks are nested and *leaf* blocks, i.e., blocks not containing other blocks inside, form a non-overlapping cover of the domain. The Stellar tree is based on an n-dimensional generalization of a bucket *Point Region* (PR) quadtree [Sam06], where the vertices of the complex are clustered into buckets with a pre-defined capacity. Only the top simplices of the complex are encoded in a Stellar tree. A top simplex is associated with all leaf blocks containing its vertices.

Figure 7 shows an example of a Stellar tree for the simplicial 2-complex in Figure 7(a). A bucket capacity equal to 3 is used, i.e., at most three vertices can belong to the same leaf block (see Figure 7b). Each triangle (top simplex) σ is introduced into a leaf block if at least one of its vertices is therein contained. For example, as shown in Figure 7(c), block b_3 contains three vertices and three triangles. The final Stellar tree is shown in Figure 7(d).

In our implementation of the Stellar tree, the vertex coordinates and the connectivity of the top simplices in the complex (i.e., for each top simplex, its bounding vertices) are maintained in two global arrays, that we call *vertices* and *top-simplices* arrays. To encode the vertices and top simplices associated with the leaf blocks (i.e., the indexes of the vertices and top simplices in the two global arrays), a *compressed* encoding [FWD17] is used, which is based on a variant of the *run-length encoding* [HM91]. It has been shown in [FWD17] that the Stellar tree requires from 40% to 50% less storage than the most compact dimension-independent data structure for simplicial complexes, the *IA** data structure [CDFW11], and from 65% to 99% less storage than the Simplex Tree [BM14].

6.2. Simplification through edge contraction

The simplification algorithm developed for the Stellar tree applies a homology-preserving edge contraction (Algorithm 2), after verifying the top-based link condition (Algorithm 1), to the candidate edges in each of its leaf blocks. The tree is visited according to a

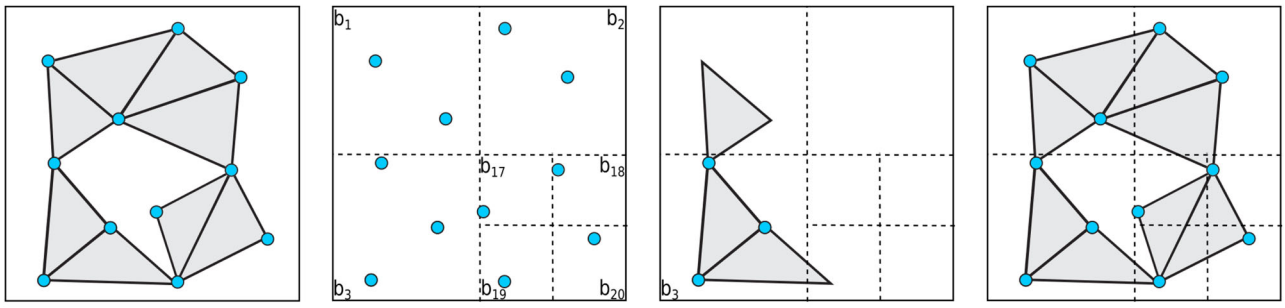


Figure 7: *A simplicial 2-complex (a). The nested decomposition obtained by subdividing the space based on its points through a bucket PR-quadtree (b). Each leaf block of a Stellar tree encodes the top simplices incident in at least one of the vertices contained in it. For example, in (c), leaf block b_3 contains three vertices and three triangles. In (d), the Stellar tree for the complex in (a) is shown.*

depth-first traversal. As a consequence, the indexes of the vertices inside a leaf block b are always greater than those in the leaf blocks preceding b in the traversal. In the following, an edge $\epsilon = \{v_1, v_2\}$ is considered as a pair of vertex indexes in the vertices array of Σ. For simplicity, we still denote the two vertex indexes as v_1 and v_2, and we assume that $v_1 < v_2$.

Each leaf block b is considered separately. The homology-preserving simplification algorithm performs the following steps on b:

(1) *Extract the top stars of all the vertices in leaf block b*. The top stars of the extreme vertices of the edges are needed both for verifying the top-based link condition and performing edge contraction.
(2) *Identify candidate edges*. An edge $\epsilon = \{v_1, v_2\}$ is a *candidate* edge if at least vertex v_2 is indexed by the current block b. Thus, the top star of ϵ can be reconstructed inside b from $St_{top}(v_2)$. Note that there can be two types of candidate edges in b, depending on whether they have both their extreme vertices in b or just one of them. We call the former *internal edges*, and the latter *cross edges*. In the example of Figure 8(a), internal edges are depicted in black and cross edges in red.
(3) *Simplify candidate edges.* Candidate edges are processed without following any priority, as our objective is just to preserve the homology of the simplicial complex, which can be obtained through any simplification order. The simplification procedure can be easily extended to handle a priority queue, in which the edges are sorted following a specific criterion (like, for example, edge length). On each candidate edge ϵ, the top-based link condition is checked by applying Algorithm 1. If verified, the edge contraction on ϵ is performed by Algorithm 2, which also updates the global *vertices* and *top-simplices* arrays. Finally, the indexing structure of the Stellar tree is updated by locally modifying leaf block b, and, if ϵ is a cross edge, also the leaf block containing vertex v_1.
(4) *Maintain and update the cache.* We use an auxiliary *Least Recent Used (LRU)* cache for increasing runtime efficiency by keeping in cache the top stars of the vertices indexed by a subset of those leaf blocks which are in the proximity of block b. At the end of the processing of a leaf block b, the top stars of the vertices in b are stored in cache.

Figure 8(b) shows an example of a contraction of a cross edge ϵ. The triangles involved in the top-based link condition are those incident in v_2 (i.e., σ_0 and σ_1) and those incident in v_1 (i.e., σ_1, σ_2, σ_3 and σ_4). Edge ϵ is contracted since the top-based link condition is verified. After edge contraction (Figure 8(c)), triangle σ_1 is removed from b_1 and b_3, as it was incident in ϵ, and triangles σ_3 and σ_4 are removed as well as v_1 was the only vertex indexed by b_1 in which they were incident. Dually, triangles σ_2, σ_3 and σ_4 are added to b_3.

Algorithm 3. LEAF_SIMPLIFICATION(b, Σ_{top}, C, b_R)

Input: b, current leaf block
Input: Σ_{top}, top-based representation of simplicial complex Σ
Input: C, LRU cache
Input: b_R, root block of the hierarchy
1: // *Extract the local top stars for the vertices in* b
2: $top\_stars \leftarrow$ LOCAL_TOP_STARS(b, Σ_{top})
3: // *Create a queue of candidate edges*
4: $Q \leftarrow$ CANDIDATE_EDGES(b, Σ_{top})
5: **while** $Q \neq \emptyset$ **do**
6: $\epsilon \leftarrow$ DEQUEUE(Q) // $\epsilon = \{v_1, v_2\}$
7: $St_{top}(v_1) \leftarrow$ TOP_STAR(v_1, top_stars, C, b_R, Σ_{top})
8: $St_{top}(v_2) \leftarrow$ TOP_STAR(v_2, top_stars, C, b_R, Σ_{top})
9: $St_{top}(\epsilon) \leftarrow St_{top}(v_1) \cap St_{top}(v_2)$
10: // *Check the link condition for* ϵ *(see Algorithm 1)*
11: **if** LINK_CONDITION(ϵ, $St_{top}(v_1)$, $St_{top}(v_2)$, $St_{top}(\epsilon)$) **then**
12: // *If it is verified contract the edge (see Algorithm 2)*
13: CONTRACT(ϵ, $St_{top}(v_1)$, $St_{top}(v_2)$, $St_{top}(\epsilon)$, Σ_{top})
14: // *Update the hierarchy*
15: UPDATE_INDEX(ϵ, $St_{top}(v_1)$, b, b_R)
16: **end if**
17: **end while**
18: $C \leftarrow C \cup top\_stars$ // *Add top_stars to the LRU-cache*

Algorithm 3 provides a pseudo-code description of the homology-preserving edge contraction on a Stellar tree. At the beginning, the local top stars for all the vertices indexed by leaf block b (row 2) are extracted as well as the candidate edges (row 4). Extracting the top stars St_{top} of the vertices requires knowing the set of vertices contained in b. An array, denoted as *top_stars*, is used for encoding the local top stars: the i-th position in the array corresponds to the

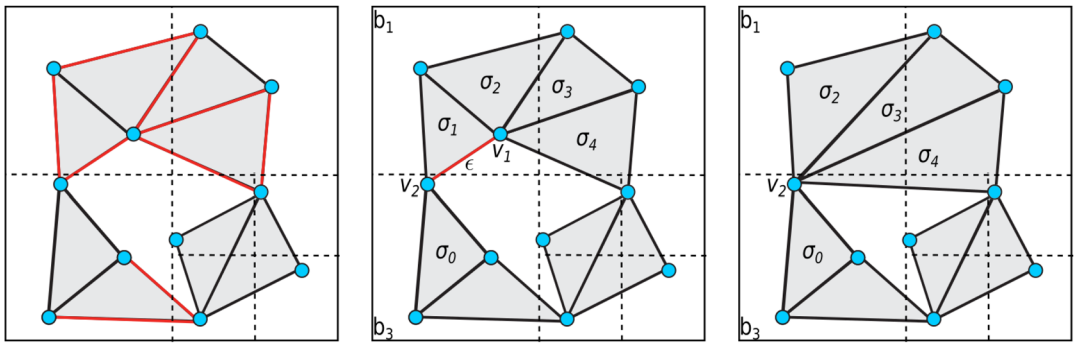


Figure 8: *(a) The set of* internal edges*, i.e., edges contained in a single-leaf block, are depicted in black, while the set of* cross edges*, i.e., edges with vertices contained in two different blocks, are depicted in red. (b) Example of a contraction operation over edge $\epsilon = \{v_1, v_2\}$, with the triangles and leaf blocks involved in the simplification/update phase. (c) The updated Stellar tree.*

i-th vertex indexed in b. The top stars are extracted by cycling on the set of top simplices in b. For each top simplex σ in b, the algorithm iterates through the vertices of σ, and, for each vertex v of σ indexed by b, σ is added to the list of top simplices incident in v (row 2, function LOCAL_TOP_STARS). A queue Q is initialized with the candidate edges $\epsilon = \{v_1, v_2\}$ (row 4, function CANDIDATE_EDGES)which are identified by iterating over the top simplices of b.

During the simplification step, the candidate edges in Q are processed (rows from 5 to 17). For the current edge ϵ, the top stars of v_1 and v_2 are considered (rows 6 and 7, function TOP_STAR). By definition of candidate edge, v_2 is indexed in b, and thus, its top star is encoded in the top_stars array. Conversely, v_1 can be either indexed in b, or in a leaf block b' that has been already processed, since $v_1 < v_2$. Thus, $St_{top}(v_1)$ can be found in cache, or it has to be re-computed. In a worst-case scenario, the hierarchy needs to be traversed in order to identify b', which corresponds to execute a point location for v_1, and to compute the local top stars of b'. The top star of ϵ is obtained by intersecting the top stars of v_1 and v_2 (row 9). The top-based link condition for ϵ is tested by using Algorithm 1 (row 11) and, if the condition is verified, Algorithm 2 is applied to contract edge ϵ (row 13). Procedure UPDATE_INDEX modifies the Stellar tree (row 15). If ϵ is an *internal edge*, the update is local to leaf block b. If ϵ is a *cross edge* between two blocks b and b', the top simplices that were incident in v_1, but not in any other vertex of b, are added to b. Dually, the top simplices in b' that were incident only in v_1 are removed from b'. Also, the top simplices in $St_{top}(\epsilon)$ are removed from b, and from b', if ϵ is a cross edge. After processing all the candidates edges, the local top stars encoded in the top_stars array are added to the cache (row 18).

We analyse here the running time of Algorithm 3. The time complexity of LOCAL_TOP_STAR and of CANDIDATE_EDGES is linear in the number t_b of top simplices in b, and in the number e_b of edges in b, respectively. Since $St_{top}(v_1)$ and $St_{top}(v_2)$ are sorted arrays, computing $St_{top}(\epsilon)$ requires $\mathcal{O}(m\log(m))$, where m is the maximum of the cardinalities of $St_{top}(v_1)$ and $St_{top}(v_2)$. As discussed in Sections 5.1 and 5.2, LINK_CONDITION and CONTRACT require $\mathcal{O}(m^3)$ and $\mathcal{O}(m^2)$, respectively. All steps of Algorithm 3 perform local operations on b, except when the top star of v_1 is not in cache, which may occur only in TOP_STAR or in UPDATE_INDEX. This leads to a worst-case cost linear in the number of tree blocks, which is not a significant performance measure, as it is generally the case with hierarchical spatial indexes [Sam06]. This behaviour is also confirmed by our experimental evaluation (see Section 7), which shows that the running time of Algorithm 3 is dominated by checking the top-based link condition and performing edge contractions, while navigating the hierarchy has a negligible computational impact.

7. Experimental Analysis and Comparison

In this section, we evaluate the performances of the top-based homology-preserving edge contraction operator we have developed on the Stellar tree. We perform comparisons between the top-based link condition proposed here and the weak link condition in [DHKS13], that we have also implemented on the Stellar tree. We extensively compare our results with those obtained by performing edge contraction based on the Skeleton-Blocker data structure [ALS11]. This latter is considered the best state-of-the-art data structure for performing edge contraction [SLA15]. The edge contraction and link condition used by the Skeleton-Blocker data structure are those implemented in the Gudhi library [GUD18]. The hardware configuration used in our experiments is a single workstation equipped with an Intel Xeon E5-2630 v4 CPU at 2.20 GHz and 64GB of RAM. The source code is available at [Fel19].

7.1. The Skeleton-Blocker data structure

The Skeleton-Blocker [ALS11] is a data structure designed specifically for performing homology-preserving edge contractions on simplicial complexes close to flag complexes. This data structure encodes a simplicial complex Σ as its 1-skeleton plus a set of *blockers*. A blocker is a 'missing' simplex, i.e., a p-simplex that is not in Σ, but all its faces are. In the example of Figure 3(b), $\{v_1, v_2, v_4\}$ identifies a blocker. The Skeleton-Blocker data structure can be generated efficiently when considering flag complexes. Recall that the top simplices of a flag complex are the maximal cliques of the 1-skeleton of the complex. Since flag complexes have no blockers, a Skeleton-Blocker representation of a flag complex is just its 1-skeleton. Conversely, for arbitrary simplicial complexes, the blockers need to be computed when the data structure is

generated, thus leading to high computation times (as shown in our experiments).

The Skeleton-Blocker data structure supports an efficient implementation of the link condition. As proved in [ALS11], the link condition for an edge $\epsilon = \{v_1, v_2\}$ is verified if there are no blockers incident into ϵ. Thus, let B_1 denote the set of blockers incident into vertex v_1. The link condition is verified if v_2 does not appear in any blocker $b \in B_1$. We recall that a *blocker* of dimension k is a collection of $k+1$ vertices. Thus, searching if $v_2 \in b$ can be done in $\mathcal{O}(\log(k))$ time. If d is the dimension of the complex, the link condition can be verified in $\mathcal{O}(|B_1|\log(d))$ in the worst case [ALS11]. Updating the data structure after an edge contraction can be quite time-consuming, depending on the number of blockers. After contracting $\epsilon = \{v_1, v_2\}$, the two sets of blockers B_1 and B_2 associated with v_1 and v_2, respectively, need to be updated. The faces of each blocker b are extracted by considering all the subsets of vertices b. We call S_1 and S_2 the set of simplices in B_1 and B_2, respectively. For each pair of simplices (σ_1, σ_2), with $\sigma_1 \in S_1$ and $\sigma_2 \in S_2$, it is checked if simplex $\sigma = \sigma_1 \cup \sigma_2$ belongs to Σ, and if the simplices on its boundary belong to $Lk(v_1) \cup Lk(v_2)$. We refer to [ALS11] for a precise analysis of the complexity of these operations.

A direct comparison of the complexity of our approach implemented in the Stellar tree and the one defined in the Skeleton-Blocker is not straightforward as the Skeleton-Blocker is mainly dependent on the number of blockers in the star of each vertex, while our approach depends on the number of top simplices. As shown by the experiments, the Skeleton-Blocker is faster at checking the link condition, but its performance at updating the structure after a simplification is heavily affected by the number of blockers in Σ. Conversely, the Stellar tree is slower at checking the link condition, but it is faster at updating the top-based representation of the complex and the hierarchy indexing it.

7.2. Simplifying a simplicial complex

In this section, we evaluate the performances of the homology-preserving edge contraction as implemented on the Stellar tree and on the Skeleton-Blocker data structure when simplifying arbitrary-dimensional simplicial complexes.

The datasets used in these experiments are listed in Table 1. Two point clouds are originated by GPS data (CHICAGO and ATHENS) embedded in a 2D space. Three datasets are extracted from point clouds embedded in 3D space, two datasets are obtained from volumetric data (VISMALE and FOOT) and one from a 3D surface (LUCY). Two point clouds are sampled from a 4D sphere (S3) and from a 5D Klein bottle (KL). Two point clouds HUMAN GENOME and HIV represent real data in 10 and 20 dimensions, respectively [OPT*17]. Overall, the datasets used contain between 1000 and 14 millions points.

From the point clouds, we generate *VR-complexes* by using an algorithm we have developed for the Stellar tree. The algorithm computes first the neighbourhood graph from the point cloud by connecting pairs of points closer than a user-defined distance ω, and, for this, it exploits the hierarchical spatial index defining the Stellar tree. Then, it computes the maximal cliques of such graph by extending the *Bron-Kerbosch* clique computation algorithm with pivoting [TTT06]. The dimension of the VR-complexes generated span from 7 to 68. In Table 1, we also show the compression achieved while simplifying the generated VR-complexes by applying edge contraction. Note that all possible contractions that maintain homology have been applied. The percentage of simplices removed is, on average, around 95%–99%, and also the dimension of the complex is usually drastically reduced.

Table 1: *Statistics on initial and simplified VR-complexes: number of vertices, number of top simplices, dimension of the complex in the initial and in the final simplicial complex and percentage of simplices removed (%). ω is the threshold used for generating the VR-complex.*

		Initial Σ			Final Σ			
Data	ω	$\lvert\Sigma_V\rvert$	$\lvert\Sigma_{top}\rvert$	d	$\lvert\Sigma_V\rvert$	$\lvert\Sigma_{top}\rvert$	d	%
CHICAGO	28	9.43K	4.66K	27	3.05K	0.23K	3	95.1
	56		6.59K	62	1.44K	0.27K	3	95.9
ATHENS	63	32.2K	21.5K	37	4.30K	1.58K	3	92.6
	126		39.1K	68	1.03K	1.91K	3	95.1
VISMALE	3.5	4.65M	6.39M	7	0.42M	1.91M	7	70.2
FOOT	4.5	5.02M	63.9M	10	0.33M	2.19M	10	96.6
LUCY	1.5	14.0M	41.1M	34	0.04M	0.14M	5	99.7
S3	0.5	2.00K	44.4K	13	0.07K	0.48K	8	98.9
	0.6		144K	20	0.08K	0.75K	12	99.5
KL	0.25	10.0K	49.2K	29	0.03K	0.08K	3	99.8
	0.35		112K	61	0.02K	0.06K	4	99.9
HIV	150	1.09K	2.81K	20	0.41K	0.12K	8	95.9
	160		21.3K	51	0.35K	0.09K	4	99.9
GENOME	9.0	1.40K	5.62K	19	0.48K	0.68K	10	87.9
	9.6		10.85K	21	0.42K	0.51K	10	95.3

We compare the storage requirements of the Stellar tree and of the Skeleton-Blocker data structure for encoding each simplicial complex. In Table 2, we show the static storage costs considering, for the Stellar tree, the encoding of the hierarchy and of the arrays of top simplices and vertices, and, for the Skeleton-Blocker data structure, the cost for encoding the 1-skeleton and the blockers. We can notice that the Stellar tree uses on average about 40% of the memory required by the Skeleton-Blocker. Only on some smaller datasets, the Skeleton-Blocker uses on average 10% less storage than a Stellar tree, and this happens since, in these cases, the compression produced by the spatial index is less effective due to the distribution of top simplices across the hierarchy.

In Table 2, we also present the comparison of the three simplification methods, i.e., by using the Stellar tree with the top-based link condition, and with the weak link condition, and by using the Skeleton-Blocker data structure. We present the time required for checking the link conditions (column *check*) and the time required for contracting the edges (column *contr.*) separately. Note that the total computation time (column tot) for the approach based on the Stellar tree does not correspond to the sum of the values in the $check$ and $contr.$ columns, as it takes into account also the time required for visiting the hierarchy, computing the local top stars in each leaf block b and inserting them in cache.

Table 2: *Number of contracted edges, experimental timings, and memory peak (before and after the simplification procedure) obtained by the two Stellar tree strategies and by the Skeleton-Blocker. The timings shown are the time required for checking the link condition (*check *column), executing the edge contractions (*contr. *column) and the overall timings of the whole simplification procedure (*tot *column), and are expressed in terms of seconds (s), minutes (min) and hours (h). The memory peaks consider the memory used for generating the data structure (*gen. *column) and those for executing the simplification procedure (*simpl. *column) and are expressed in terms of megabytes.*

Data	ω		Contr. Edges	Timings			Memory peak	
				check	contr.	tot	gen.	simpl.
CHICAGO	28	weak	6.38K	9.15 h	2.27 m	9.19 h	5.6	57.2K
		top		0.01 s	0.02 s	0.09 s		7.6
		Skel.		0.00 s	0.15 s	0.15 s	7.8	7.8
	56	weak	7.99K		out-of-memory		6.2	–
		top		0.04 s	0.06 s	0.23 s		10.8
		Skel.		0.00 s	0.71 s	0.71 s	14.1	14.1
ATHENS	63	weak	27.9K		out-of-memory		11.6	–
		top		0.08 s	0.11 s	0.38 s		14.9
		Skel.		0.00 s	0.74 s	0.75 s	26.4	26.8
	126	weak	31.2K		out-of-memory		10.0	–
		top		0.40 s	0.49 s	1.36 s		25.9
		Skel.		0.01 s	7.73 s	7.74 s	66.1	66.7
VISMALE	3.5	weak	4.23M	34.3 m	1.28 m	40.4 m	1.0K	2.0K
		top		4.34 m	0.89 m	7.20 m		2.0K
		Skel.		0.76 m	3.34 h	3.35 h	8.0K	8.0K
FOOT	4.5	weak	4.69M		killed after 25 h		7.5K	–
		top		2.89 h	26.0 m	3.32 h		10.7K
		Skel.			killed after 25 h		19.4K	–
LUCY	1.5	weak	14.0M		killed after 25 h		7.5K	–
		top		11.9 m	14.8 m	32.0 m		15.4K
		Skel.		23.19 s	14.6 h	14.6 h	50.9K	52.1K
S3	0.5	weak	1.93K	5.58 m	4.12 s	5.89 m	10.7	27.8
		top		9.44 s	1.38 s	10.89 s		13.1
		Skel.		0.04 s	1.26 m	1.26 m	12.1	13.1
	0.6	weak	1.93K		killed after 25 h		29.3	–
		top		5.91 m	0.79 m	6.72 m		46.0
		Skel.		0.11 s	19.4 m	19.4 m	18.7	22.5
KL	0.25	weak	9.97K		killed after 25 h		11.1	–
		top		1.25 s	0.67 s	2.19 s		17.8
		Skel.		0.01 s	14.2 s	14.2 s	32.2	32.7
	0.35	weak	9.98K		killed after 25 h		23.8	–
		top		1.21 m	17.9 s	1.54 m		61.0
		Skel.		0.02 s	1.67 m	1.67 m	61.6	63.6
HIV	150	weak	0.68K	12.1 m	5.9 s	12.2 m	4.2	0.9K
		top		1.41 s	0.10 s	1.54 s		5.2
		Skel.		0.00 s	1.59 s	1.60 s	3.4	4.3
	160	weak	0.74K		killed after 25 h		4.5	–
		top		15.3 s	0.69 s	16.0 s		6.3
		Skel.		0.00 s	0.90 s	0.90 s	3.8	4.9
GENOME	9.0	weak	0.92K	3.12 m	0.90 s	3.18 m	4.6	0.2K
		top		2.11 s	0.12 s	2.29 s		5.6
		Skel.		0.01 s	14.0 s	14.0 s	4.3	5.5
	9.6	weak	0.97K	47.0 m	23.0 s	47.6 m	5.3	0.8K
		top		19.3 s	0.75 s	20.0 s		7.1
		Skel.		0.01 s	26.3 s	26.3 s	4.9	7.0

The limitation of the approach based on the weak link condition is clear. The simplification process ends in very few cases and, when the dimension of the complex increases, the process exceeds the 64GB of available memory. In the few cases when the simplification ends, processing time mainly accounts for checking the link condition (as shown by a comparison of the results in columns *check* and *contr.*).

When comparing the memory usage (column *memory peak*, *simpl.*), we see that the use of the weak link condition is generally unfeasible. By looking at the few datasets for which the weak link condition ends the computation (CHICAGO, VISMALE, S3, HIV and GENOME), we can see that the approaches based on the top-based and on the weak link conditions use the same amount of memory only in VISMALE, where the dimension of the complex is limited to 7. When considering the memory peak, on S3, the top-based approach requires half of the memory used by the weak link condition and, on CHICAGO, HIV and GENOME datasets, only a small fraction of the memory is used by the top-based approach with respect to the other.

These results confirm that the weak link condition scales only when the dimension of the simplicial complex is low, while the performances easily degrade when the dimension of the complex increases. It cannot be used on VR-complexes of dimension larger than 30, because of memory requirements and of high increase in simplification time.

When comparing the performance of the approach based on the Stellar tree and on the top-based link condition with the one based on the Skeleton-Blocker, we can see that the former one is generally faster. The Skeleton-Blocker data structure requires more efforts for updating the structure during edge contraction, while it supports a fast verification of the link condition. On the VR-complexes built on road networks, the approach based on the Stellar tree requires from 20% to 60% of the time needed by the one based on the Skeleton-Blocker representation. Considering the memory peak, the Stellar tree is more compact on tracking datasets, using from 40% to 95% of the memory required by the Skeleton-Blocker data structure.

On the VR-complex built on VISMALE and LUCY datasets, the Stellar tree uses 5% of the time and requires from 70% to 75% less memory with respect to the Skeleton-Blocker data structure. On FOOT dataset, the generation of the Skeleton-Blocker does not end in 25 hours, and its memory peak, during this phase, reaches nearly 48GB, which is more than six times higher than the one of the Stellar tree.

On VR-complexes built from synthetic point clouds (S3 and KL), the approach based on the Stellar tree and on the top-based link condition is generally faster than the one based on the Skeleton-Blocker. The former provides a speedup that goes from 5% to 15% on S3, and from 15% to 90% on KL.

On KL, the Stellar tree is more compact, since it uses from 55% to 95% of the memory required by the Skeleton-Blocker. Conversely, on S3 with $\omega = 0.6$, the Stellar tree uses up to twice the memory required by the Skeleton-Blocker. In this case, just the indexed mesh representation (inside the Stellar tree) has the same space requirements as the Skeleton-Blocker (approximately 18MB), since the majority of the top simplices in this dataset are in high dimensions (between 13 and 17). Moreover, the average number of leaf blocks to which each top simplex belongs is higher for this dataset than for the other ones (six rather than between two and four blocks).

Table 3: *Number of vertices ($|\Sigma_V|$) and of triangles/tetrahedra ($|\Sigma_{top}|$), before and after the simplification process, of the 3D meshes used in the experiments.*

	Initial Σ		Final Σ	
Data	$\lvert\Sigma_V\rvert$	$\lvert\Sigma_{top}\rvert$	$\lvert\Sigma_V\rvert$	$\lvert\Sigma_{top}\rvert$
FERTILITY	242K	483K	0.02K	0.05K
HAND	351K	702K	0.01K	0.02K
BUDDHA	544K	1.09M	0.48K	1.37K
NEPTUNE	2.0M	4.01M	0.01K	0.04K
STATUETTE	5.0M	10.0M	0.02K	0.04K
LUCY	14.0M	28.1M	0.06K	0.12K
RBL	730K	3.89M	8.27K	65.0K
MITO	972K	5.54M	9.91K	96.2K

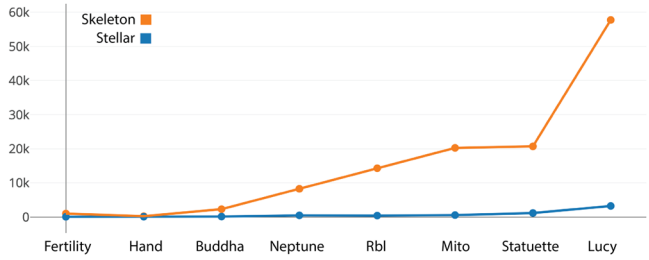


Figure 9: *Memory consumption (in megabytes) of the Skeleton-Blocker and the Stellar tree for loading a mesh and for generating the corresponding data structure.*

On the VR-complexes built from real-world point clouds (HIV and GENOME), the approach based on the Stellar tree has better, or comparable, performances with respect to the Skeleton-Blocker, except in one case. The Stellar tree is up to 7 times faster than the Skeleton-Blocker, in the best case, still having similar memory peaks. The only exception is HIV with $\omega = 160$, where the Skeleton-Blocker performs significantly better, since the number of blockers generated during the simplification procedure is low (approximately 100), leading to a fast update step.

7.3. Simplifying a 3D mesh

In this section, we evaluate the performances of the approaches based on the Stellar tree and on the Skeleton-Blocker data structure when simplifying triangle and tetrahedral meshes. In the following, we refer to these complexes as 3D meshes. In Table 3, we present the six triangle meshes used, which consist of a number of triangles varying from 483K to 28M, and the two tetrahedral meshes used, consisting of a number of tetrahedra varying from 3.89M to 5.54M.

Differently from the point cloud datasets used in the previous section, 3D meshes are commonly provided as a set of top simplices (either triangles or tetrahedra) and vertices. From this input, the Stellar decomposition is efficiently initialized, as the algorithm first generates the spatial decomposition from the vertices of the mesh, and then inserts the top simplices in those leaf blocks indexing their vertices. Conversely, the Skeleton-Blocker data structure is generally initialized from the set of blockers. When these latter are unknown, the edges and top simplices are extracted and

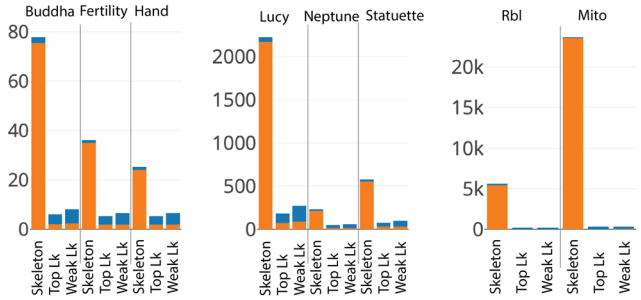


Figure 10: *Timings (in seconds) for performing a homology-preserving simplification on eight meshes. For each dataset, we indicate the time required for performing all the edge contractions (in orange) and the time required for verifying the link conditions (in blue) separately.*

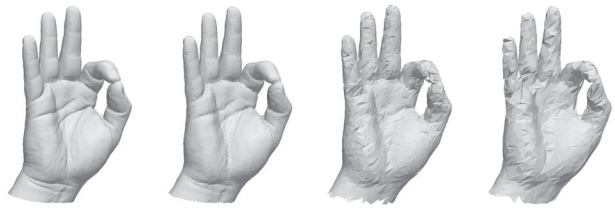
$[242K, 483K]$ (a) $[83K, 167K]$ (b) $[3K, 6K]$ (c) $[0.3K, 0.6K]$ (d)

Figure 11: *Simplification progression on* FERTILITY. *(a) shows the shape at full resolution, while (b), (c) and (d) show three intermediate simplifications. Between square brackets, we indicate the number of vertices and triangles as* $[|\Sigma_V|, |\Sigma_{top}|]$.

inserted one by one. Figure 9 shows the memory consumption for generating the two data structures with 3D meshes of increasing size. We see that the Stellar tree scales very well when the size of the mesh increases. Conversely, the Skeleton-Blocker data structure requires up to 15 times more memory, on triangle meshes, and 35 times more memory, on tetrahedral meshes, than the Stellar tree to be initialized.

Figure 10 shows the timings required for simplifying the 3D meshes by comparing the approaches based on the Skeleton-Blocker data structure, and on the Stellar tree using the top-based link condition and weak link condition. For each mesh, we distinguish between the time required for performing edge contraction (depicted in orange) and the time required for verifying the link condition (depicted in blue). When using the top-based link condition, the Stellar tree is from five to nine times faster, than the Skeleton-Blocker data structure in simplifying triangle meshes, and from 35 to 70 times faster in simplifying tetrahedral meshes. As expected, the link condition is efficiently verified when using the Skeleton-Blocker data structure, while most of the computational time is spent updating the data structure after applying an edge contraction (orange bar).

Figures 11 and 12 show the progressive simplification of two triangle meshes, FERTILITY and HAND datasets, respectively. Figures 11(a) and 12(a) show the 3D mesh at full resolution. Figures 11(b)–(d) and 12(b)–(d) show intermediate simplified shapes with preserved homology.



$[351K, 702K]$ (a) $[70K, 140K]$ (b) $[6K, 12K]$ (c) $[2K, 5K]$ (d)

Figure 12: *Simplification progression on* HAND. *(a) shows the shape at full resolution, while (b), (c) and (d) show three intermediate simplifications. Between square brackets, we indicate the number of vertices and triangles as* $[|\Sigma_V|, |\Sigma_{top}|]$.

8. Concluding Remarks

We have presented an efficient and scalable tool for simplifying simplicial complexes in arbitrary dimensions. The simplification algorithm we have defined and implemented takes advantage of a definition of the edge contraction operator on a *top-based* representation of the complex, resulting in a scalable simplification procedure. Using a top-based representation, we have also defined a new way for checking the *link condition*, suitable for the analysis of data in any dimension. The simplification algorithm has been implemented using a general-purpose top-based data structure for simplicial complexes, the Stellar tree [FWD17]. The source code of the simplification tool based on the Stellar tree is available in the public domain [Fel19].

In comparison with a data structure specifically designed for executing homology-preserving edge contractions, the Skeleton-Blocker [ALS11], our approach, based on the Stellar tree, exhibits always better performances on simplicial complexes that are not VR-complexes. On VR-complexes, it has similar performances as the Skeleton-Blocker, when the complexes are of small size, and better performances on complexes of larger size.

Currently, we are working on a multi-resolution model of a simplicial complex generated through a sequence of homology-preserving edge contractions. By navigating the model, we will be able to extract refined representations of the simplicial complex at different levels of resolution. Moreover, we can compute homology and its generators on the coarse representation, extracting refined homology generators alongside with the simplicial complex during the refinement process. To this aim, we are developing an instance of the refinement operator, inverse with respect to edge contraction, called vertex split, capable of refining a simplicial complex and its homology generators at the same time. The resulting tool would allow an interactive exploration of a simplicial shape, combining both the combinatorial information provided by the simplicial complex and the qualitative information provided by its homology.

Acknowledgements

The first and second authors contributed equally to this paper. This work has been partially supported by the US National Science Foundation under grant number IIS-1116747, and by the University

of Maryland under the 2017-2018 BSOS DRI Program. The authors wish to thank Kenneth Weiss for several helpful discussions.

Point clouds originated from tracking data are courtesy of *Map Construction Portal* (ATHENS, CHICAGO), volumetric datasets point clouds are courtesy of *Volvis* repository (FOOT) and the *Volume Library* (*volib*) (VISMALE). The triangulated surface point cloud is courtesy of *Aim@Shape* (FERTILITY, HAND, LUCY). We thank Clement Maria for providing the S3 and KL datasets.

Appendix

Here, we present the proofs of the propositions enunciated in Section 4 and Section 5.

A.1. Proof of Proposition 1

Proposition. $Cl(\mu(\Sigma_{top})) = \mu(\Sigma)$.

Proof. "$\subseteq$". Since $\Sigma_{top} \subseteq \Sigma$, then $\mu(\Sigma_{top}) \subseteq \mu(\Sigma)$. Moreover, since the closure operator respects the inclusions, $Cl(\mu(\Sigma_{top})) \subseteq Cl(\mu(\Sigma)) = \mu(\Sigma)$.

"$\supseteq$". Since the top simplices of a simplicial complex completely characterize the simplicial complex itself, it is enough to prove that $\mu(\Sigma_{top}) \supseteq \mu(\Sigma)_{top}$, where $\mu(\Sigma)_{top}$ is the set of top simplices in $\mu(\Sigma)$. Given a simplex τ in $\mu(\Sigma)_{top}$, let σ be a simplex in $\mu^{-1}(\tau)$ such that there exists no $\sigma' \in \mu^{-1}(\tau)$ for which $\sigma \subsetneq \sigma'$. Let us assume that $\sigma \notin \Sigma_{top}$. So, there exists a simplex σ'' in Σ_{top} such that $\sigma \subsetneq \sigma''$. This implies that $\mu(\sigma) \subseteq \mu(\sigma'')$. Moreover, thanks to the maximality condition satisfied by σ, $\tau = \mu(\sigma) \subsetneq \mu(\sigma'')$. This implies that τ is not in $\mu(\Sigma)_{top}$ leading to a contradiction. In conclusion, σ is a simplex in Σ_{top} such that $\mu(\sigma) = \tau$. Then, $\tau \in \mu(\Sigma_{top})$. ■

A.2. Proof of Proposition 2

Proposition. The link condition (1) and the top-based link condition (2) are equivalent.

Proof. "(1) $\Rightarrow$ (2)". Let τ_1, τ_2 be two simplices such that, for $i = 1, 2, \tau_i \in St_{top}(\nu_i) - St_{top}(\epsilon)$. By definition, $\tau_1 \cap \tau_2 \in Lk(\nu_1) \cap Lk(\nu_2)$. By applying condition (1), $\tau_1 \cap \tau_2 \in Lk(\epsilon)$. By definition of link, there exists τ in $St_{top}(\epsilon)$ such that $\tau_1 \cap \tau_2 \subseteq \tau$.

"(1) $\Leftarrow$ (2)". Let σ be a simplex in $Lk(\nu_1) \cap Lk(\nu_2)$. Hence, there exist τ_1, τ_2 such that, for $i = 1, 2$, $\tau_i \in St_{top}(\nu_i)$ and $\sigma \subseteq \tau_i$. If at least one among τ_1 and τ_2 belongs to $St_{top}(\epsilon)$, then $\sigma \in Lk(\epsilon)$. Otherwise, by applying condition (2), there exists $\tau \in St_{top}(\epsilon)$ such that $\sigma \subseteq \tau_1 \cap \tau_2 \subseteq \tau$. So, $\sigma \in Lk(\epsilon)$. ■

A.3. Proof of Proposition 3

Proposition. Let τ be a simplex in $St_{top}(\epsilon)$ and, for $i = 1, 2$, let denote γ_i as the simplex $\tau - \nu_i$. We have that:

$$\mu(\tau) \in \mu(\Sigma)_{top} \iff St_{top}(\gamma_1) \cup St_{top}(\gamma_2) = \{\tau\}.$$

Proof. "$\Rightarrow$". Let us suppose, by absurd, there exists $\tau' \in St_{top}(\gamma_1) \cup St_{top}(\gamma_2)$ different from τ. If $\tau' \in St_{top}(\gamma_1)$, then $\mu(\tau') = \tau'$. Otherwise, if $\tau' \in St_{top}(\gamma_2)$, then $\tau' = \gamma_2 \cup \sigma$ for a certain simplex σ disjoint from γ_2 and $\mu(\tau') = \mu(\gamma_2 \cup \sigma) = \mu(\gamma_2) \cup \mu(\sigma) = \gamma_1 \cup \sigma$. In both cases, $\mu(\tau') \supsetneq \gamma_1 = \mu(\tau)$ contradicting the fact that $\mu(\tau)$ is a top simplex of $\mu(\Sigma)$.

"$\Leftarrow$". Let us suppose, by absurd, there exists τ' in $\mu(\Sigma)$ such that $\tau' \supsetneq \mu(\tau)$. Let σ be a simplex in $\mu^{-1}(\tau')$. By definition of μ and τ', $\sigma \supseteq \gamma_1$ or $\sigma \supseteq \gamma_2$. So, by hypothesis, $\sigma \subseteq \tau$. Then, $\tau' = \mu(\sigma) \subseteq \mu(\tau)$ and this contradicts that $\tau' \supsetneq \mu(\tau)$. ■

A.4. Proof of Proposition 4

Proposition. Let τ be a simplex in $St_{top}(\nu_1) - St_{top}(\epsilon)$. If the link condition (1) (or, equivalently, the top-based link condition (2)) is satisfied, then $\mu(\tau) \in \mu(\Sigma)_{top}$. Moreover, $\mu^{-1}(\mu(\tau)) \cap \Sigma_{top} = \{\tau\}$, i.e., τ is the only top simplex of Σ which is mapped into $\mu(\tau)$ via μ.

Proof. Since $\tau \in St_{top}(\nu_1) - St_{top}(\epsilon)$, we can write it as $\tau = \nu_1 \cup \gamma$ such that, for $i = 1, 2$, $\nu_i \notin \gamma$. Let us suppose, by absurd, that there exists a simplex $\tau' \in \mu(\Sigma)$ such that $\mu(\tau) \subsetneq \tau'$. Let us consider σ a simplex in $\mu^{-1}(\tau')$ and let σ' be a top simplex of Σ containing σ. Since, by definition, $\nu_2 \cup \gamma = \mu(\tau) \subsetneq \tau'$, we have that for at least one value of i, $\nu_i \cup \gamma \subseteq \sigma'$. Since for $i = 1$, this contradicts the fact that τ is a top simplex in Σ, then $\nu_2 \cup \gamma \subseteq \sigma'$. So, $\sigma' \in St_{top}(\nu_2) - St_{top}(\epsilon)$. By applying the top-based link condition (2), there exists $\rho \in St_{top}(\epsilon)$ such that $\tau \cap \sigma' \subseteq \rho$. Since $\rho \in St_{top}(\epsilon)$ and $\tau \cap \sigma' = \gamma$, then $\tau = \nu_1 \cup \gamma \subsetneq \nu_1 \cup \nu_2 \cup \gamma \subseteq \rho$. This contradicts the fact that $\tau \in \Sigma_{top}$. So, $\mu(\tau) \in \mu(\Sigma)_{top}$.

Let us prove now that $\mu^{-1}(\mu(\tau)) \cap \Sigma_{top} = \{\tau\}$. By definition of τ, $\mu^{-1}(\mu(\tau)) \subseteq \{\tau, \nu_2 \cup \gamma, \nu_1 \cup \nu_2 \cup \gamma\}$. Since $\tau \in \Sigma_{top}$, $\nu_1 \cup \nu_2 \cup \gamma$ cannot be a simplex of Σ. Let us suppose, by absurd, that $\nu_2 \cup \gamma \in \mu^{-1}(\mu(\tau)) \cap \Sigma_{top}$. So, since $\nu_2 \cup \gamma \in St_{top}(\nu_2) - St_{top}(\epsilon)$, by applying the top-based link condition (2), there exists $\rho \in St_{top}(\epsilon)$ such that $\tau \cap (\nu_2 \cup \gamma) \subseteq \rho$. Since $\rho \in St_{top}(\epsilon)$ and $\tau \cap (\nu_2 \cup \gamma) = \gamma$, then $\tau = \nu_1 \cup \gamma \subsetneq \nu_1 \cup \nu_2 \cup \gamma \subseteq \rho$. This contradicts the fact that $\tau \in \Sigma_{top}$. Therefore, $\mu^{-1}(\mu(\tau)) \cap \Sigma_{top} = \{\tau\}$. ■

References

[ALS11] ATTALI D., LIEUTIER A., SALINAS D.: Efficient data structure for representing and simplifying simplicial complexes in high dimensions. In *Proceedings of 27th Annual Symposium on Computational Geometry (SoCG2011)* (Paris, France, 2011).

[BG14] BAMPASIDOU M., GENTIMIS T.: Modeling collaborations with persistent homology. ArXiv e-prints:1403.5346, March 2014.

[BM14] BOISSONNAT J.-D., MARIA C.: The Simplex Tree: An efficient data structure for general simplicial complexes. *Algorithmica 70*, 3 (2014), 406–427.

[BST17] BOISSONNAT J.-D., KARTHIK C. S., TAVENAS S.: Building efficient and compact data structures for simplicial complexes. *Algorithmica 79*, 2 (October 2017), 530–567.

[Can14] Canino D.: *Mangrove topological data structure library*, 2014. https://sourceforge.net/projects/mangrovetds/. Online Accessed June 2019.

[CDFM*04] Cignoni P., De Floriani L., Magillo P., Puppo E., Scopigno R.: Selective refinement queries for volume visualization of unstructured tetrahedral meshes. *IEEE Transactions on Visualization and Computer Graphics 10*, 1 (January 2004), 29–45.

[CDFW11] Canino D., De Floriani L., Weiss K.: IA*: An adjacency-based representation for non-manifold simplicial shapes in arbitrary dimensions. *Computers & Graphics 35*, 3 (2011), 747–753.

[CRS15] Cassidy B., Rae C., Solo V.: Brain activity: Conditional dissimilarity and persistent homology. In *2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI)* (2015), pp. 1356–1359.

[DEGN99] Dey T. K., Edelsbrunner H., Guha S., Nekhayev D. V.: Topology preserving edge contraction. *Publications de l'Institut Mathématique 66* (1999), 23–45.

[DFH03] De Floriani L., Hui A.: A scalable data structure for three-dimensional non-manifold objects. In *SGP'03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 72–82.

[DFH05] De Floriani L., Hui A.: Data structures for simplicial complexes: An analysis and a comparison. In *Proceedings of Symposium on Geometry Processing* (2005), pp. 119–128.

[DFKP05] De Floriani L., Kobbelt L., Puppo E.: A survey on data structures for level-of-detail models. In *Advances in Multiresolution for Geometric Modelling*. M. Sabin and N. Dodgson (Eds.). Springer, Berlin, Germany (2005), pp. 49–74.

[DFMPS04] De Floriani L., Magillo P., Puppo E., Sobrero D.: A multi-resolution topological representation for non-manifold meshes. *Computer-Aided Design 36*, 2 (2004), 141–159. Solid Modeling and Applications.

[DHKS13] Dey T. K., Hirani A. N., Krishnamoorthy B., Smith G.: Edge contractions and simplicial homology. ArXiv e-prints:1304.0664, April 2013.

[DHPC10] De Floriani L., Hui A., Panozzo D., Canino D.: A dimension-independent data structure for simplicial complexes. In *Proceedings of the 19th International Meshing Roundtable, IMR 2010* (2010), S. M. Shontz (Ed.), Springer, pp. 403–420.

[DMFC12] Dabaghian Y., Mémoli F., Frank L., Carlsson G.: A topological paradigm for hippocampal spatial map formation using persistent homology. *PLoS Computational Biology 8*, 8 (August 2012), e1002581.

[DPS*13] Donato I., Petri G., Scolamiero M., Rondoni L., Vaccarino F.: Decimation of fast states and weak nodes: Topological variation via persistent homology. In *Proceedings of the European Conference on Complex Systems 2012* (Cham, 2013), T. Gilbert, M. Kirkilionis and G. Nicolis (Eds.), Springer International Publishing, pp. 295–301.

[DS18] Dey T. K., Slechta R.: Edge contraction in persistence-generated discrete Morse vector fields. *Computers & Graphics 74* (2018), 33–43.

[dSG07] De Silva V., Ghrist R.: Homological sensor networks. *Notices of the American Mathematical Society 54* (2007), 10–19.

[Ede87] Edelsbrunner H.: *Algorithms in Combinatorial Geometry*. Springer, Berlin, Heidelberg, 1987.

[ELZ02] Edelsbrunner H., Letscher D., Zomorodian A.: Topological persistence and simplification. *Discrete & Computational Geometry 28*, 4 (2002), 511–533.

[EM94] Edelsbrunner H., Mücke E. P.: Three-dimensional alpha shapes. *ACM Transactions on Graphics 13*, 1 (January 1994), 43–72.

[Fel19] Fellegara R.: Stellar tree: A framework for the representation and analysis of high dimensional unstructured data. https://github.com/UMDGeoVis/Stellar_tree, 2019. Online Accessed June 2019.

[FIDF19] Fugacci U., Iuricich F., De Floriani L.: Computing discrete Morse complexes from simplicial complexes. *Submitted to Graphical Models 103* (2019), 101023.

[FWD17] Fellegara R., Weiss K., De Floriani L.: The Stellar tree: A compact representation for simplicial complexes and beyond. ArXiv e-prints:1707.02211, 2017.

[GGK02] Gotsman C., Gumhold S., Kobbelt L.: Simplification and compression of 3D meshes. In *Tutorials on Multiresolution in Geometric Modelling*. G. Farin, H.-C. Hege, D. Hoffman, C. R. Johnson, K. Polthier, A. Iske, E. Quak and M. S. Floater. Springer, Berlin, Germany (2002), pp. 319–361.

[GLLR11] Gurung T., Laney D., Lindstrom P., Rossignac J.: SQuad: Compact representation for triangle meshes. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 355–364.

[GR09] Gurung T., Rossignac J.: SOT: A compact representation for tetrahedral meshes. In *SPM '09: Proceedings SIAM/ACM Geometric and Physical Modeling* (San Francisco, USA, 2009), pp. 79–88.

[GR10] Gurung T., Rossignac J.: *SOT: Compact Representation for Triangle and Tetrahedral Meshes*. Tech. Rep. GT-IC-10-01, College of Computing, Georgia Institute of Technology, Atlanta, GA, USA, 2010.

[GUD18] Geometric Understanding in Higher Dimensions (GUDHI), 2018. http://gudhi.gforge.inria.fr/. Online Accessed June 2019.

[Hau94] Hausmann J.: *On the Vietoris-Rips Complexes and a Cohomology Theory for Metric Spaces*. Publications internes de la

Section de mathématiques de l'Université de Genève. Université de Genève-Section de mathématiques, 1994.

[HM91] Held G., Marshall T.: *Data Compression; Techniques and Applications: Hardware and Software Considerations*. John Wiley & Sons, Chichester, 1991.

[ID17] Iuricich F., De Floriani L.: Hierarchical Forman triangulation: A multiscale model for scalar field analysis. *Computers & Graphics 66* (2017), 113–123. Shape Modeling International 2017.

[Iur16] Iuricich F.: *IA**: An indexed-based data structure with adjacencies for encoding simplicial complexes, 2016. https://github.com/UMDGeoVis/IAstar. Online Accessed June 2019.

[Nie97] Nielson G. M.: Tools for triangulations and tetrahedralizations and constructing functions defined over them. In *Scientific Visualization: Overviews, Methodologies and Techniques*. G. M. Nielson, H. Hagen and H. Müller (Eds.). IEEE Computer Society, Silver Spring, MD (1997), Chapter 20, pp. 429–525.

[OPT*17] Otter N., Porter M. A., Tillmann U., Grindrod P., Harrington H. A.: A roadmap for the computation of persistent homology. *EPJ Data Science 6*, 1 (2017), 1–38.

[PBCF93] Paoluzzi A., Bernardini F., Cattani C., Ferrucci V.: Dimension-independent modeling with simplicial complexes. *ACM Transactions on Graphics (TOG) 12*, 1 (1993), 56–102.

[PH97] Popović J., Hoppe H.: Progressive simplicial complexes. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 217–224.

[PS97] Puppo E., Scopigno R.: Simplification, LOD and multiresolution: Principles and applications. In *European Association for Computer Graphics Conference* (1997).

[RSS01] Rossignac J., Safonova A., Szymczak A.: 3D compression made simple: Edge-breaker on a corner table. In *Proceedings Shape Modeling International 2001* (Genova, Italy, May 2001), IEEE Computer Society.

[Sam06] Samet H.: *Foundations of Multidimensional and Metric Data Structures*. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann, Erscheinungsort nicht ermittelbar, 2006.

[SHPW17] Sodergren T., Hair J., Phillips J. M., Wang B.: Visualizing sensor network coverage with location uncertainty. ArXiv e-prints:1710.06925, October 2017.

[SLA15] Salinas D., Lafarge F., Alliez P.: Structure-aware mesh decimation. *Computer Graphics Forum 34* (2015), 211–227.

[TTT06] Tomita E., Tanaka A., Takahashi H.: The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science 363*, 1 (2006), 28–42.

[WRK*10] Wicke M., Ritchie D., Klingner B. M., Burke S., Shewchuk J. R., O'Brien J. F.: Dynamic local remeshing for elastoplastic simulation. *ACM Transactions on Graphics 29*, 4 (July 2010), 1.

[Zom10a] Zomorodian A.: Fast construction of the Vietoris-Rips complex. *Computers & Graphics 34*, 3 (2010), 263–271.

[Zom10b] Zomorodian A.: The tidy set. In *SoCG '10: Proceedings of the 2010 Annual Symposium on Computational Geometry* (New York, New York, USA, June 2010), ACM Press, p. 257.

[ZQC*14] Zhu B., Quigley E., Cong M., Solomon J., Fedkiw R.: Codimensional surface tension flow on simplicial complexes. *ACM Transactions on Graphics 33*, 4 (July 2014), 1–11.