Computers & Graphics 98 (2021) 322–343

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

The Stellar decomposition: A compact representation for simplicial complexes and beyond☆

Riccardo Fellegara[a,*], Kenneth Weiss[b], Leila De Floriani[c]

[a] German Aerospace Center (DLR), Braunschweig, Germany
[b] Lawrence Livermore National Laboratory, Livermore, CA, USA
[c] University of Maryland at College Park, College Park, MD, USA

ARTICLE INFO

Article history:
Received 24 August 2020
Revised 26 April 2021
Accepted 3 May 2021
Available online 8 May 2021

Keywords:
Mesh data structure
Scalable representations
Vietoris-Rips complex
High dimensional simplicial complexes

ABSTRACT

We introduce the *Stellar decomposition*, a model for efficient topological data structures over a broad range of simplicial and cell complexes. A Stellar decomposition of a complex is a collection of *regions* indexing the complex's vertices and cells such that each region has sufficient information to locally reconstruct the *star* of its vertices, i.e., the cells incident in the region's vertices. Stellar decompositions are general in that they can compactly represent and efficiently traverse arbitrary complexes with a manifold or non-manifold domain. They are scalable to complexes in high dimension and of large size, and they enable users to easily construct tailored application-dependent data structures using a fraction of the memory required by a corresponding global topological data structure on the complex.

As a concrete realization of this model for spatially embedded complexes, we introduce the *Stellar tree*, which combines a nested spatial tree with a simple tuning parameter to control the number of vertices in a region. Stellar trees exploit the complex's spatial locality by reordering vertex and cell indices according to the spatial decomposition and by compressing sequential ranges of indices. Stellar trees are competitive with state-of-the-art topological data structures for manifold simplicial complexes and offer significant improvements for cell complexes and non-manifold simplicial complexes. We conclude with a high-level description of several mesh processing and analysis applications that utilize Stellar trees to process large datasets.

1. Introduction

Efficient mesh data structures play a fundamental role in a broad range of mesh processing applications in computer graphics, geometric modeling, scientific visualization, geospatial data science and finite element analysis. Although simple problems can be easily modeled on small low dimensional meshes, phenomena of interest might occur only on much larger meshes and in higher dimensions. Thus, we often require flexibility to deal with increasingly complex meshes including those defined by irregularly connected heterogeneous and/or multidimensional cell types discretizing spaces with complicated topology. Moreover, as advances in computing capabilities continue to outpace those in memory, it becomes increasingly important to optimize and exploit locality within the mesh as we process and locally query it. Such queries are the primary means of interacting with the mesh and have traditionally been posed in terms of a few spatial and topological primitives. However, while there are simple, intuitive models for representing polygonal surfaces, there are numerous challenges in generalizing these structures to higher dimensions and in scaling to very large meshes.

In this paper, we introduce the *Stellar decomposition*, a model for topological data structures that supports efficient navigation of the topological connectivity of simplicial complexes and of certain classes of cell complexes, e.g., those composed of quadrilaterals, polygons, hexahedra, prisms and pyramids. We refer to this class of complexes as *Canonical Polytope complexes (CP complexes)*. The defining property of a Stellar decomposition is that the complex is broken up into *regions* indexing a collection of vertices such that each vertex within a region has sufficient information to locally reconstruct its *star*, i.e., the set of cells from the complex incident in that vertex.

A Stellar decomposition is *general*, in that it can easily represent arbitrary complexes with a manifold or non-manifold domain, it is *scalable* to complexes both in high dimensions and with a large number of cells, and it is *flexible*, in that it enables users to defer decisions about which topological connectivity relations to encode.

☆ This paper was recommended for publication by Rüdiger Westermann.

* Corresponding author.
E-mail addresses: riccardo.fellegara@dlr.de (R. Fellegara), kweiss@llnl.gov (K. Weiss), deflo@umiacs.umd.edu (L. De Floriani).

https://doi.org/10.1016/j.cag.2021.05.002
0097-8493/

It, therefore, supports the generation of optimal application-dependent local data structures at runtime. Due to the locality of successive queries in typical mesh processing applications, the construction costs of these local topological data structures are amortized over multiple mesh operations while processing a local region.

We also formally define and analyze the *Stellar tree* as a concrete instance of the Stellar decomposition model for spatially embedded complexes. Stellar trees utilize a hierarchical n-dimensional quadtree, or kD-tree, as vertex decomposition, and are easily *tunable* using a single parameter that defines the maximum number of vertices allowed in each local region.

While Stellar trees have been previously utilized in several mesh processing applications ranging from mesh simplification [90] to morphological feature extraction [38,68], they have not been formally defined and their performance has not yet been characterized in relation to existing topological data structures for simplicial and cell complexes. This paper presents a careful study of the storage requirements, generation algorithms and timings and query performance for Stellar trees over a wide range of CP complexes. As we demonstrate in Section 8, Stellar trees are competitive with dimension-specific state-of-the-art topological data structures for (pseudo)-manifold triangle and tetrahedral complexes and offer significant improvements for other CP complexes, especially over data structures for general simplicial complexes in 3D and higher dimensions. The source code for our Stellar tree implementation will be released in the public domain.

Contributions The contributions of this work include:

- The formal theoretical definition of a Stellar decomposition over *Canonical Polytope (CP) complexes*, a class of cell complexes that includes simplicial and cubical complexes of arbitrary dimension, as well as cells in the finite element 'zoo', such as polygons, pyramids and prisms.
- The definition of the *Stellar tree* as a concrete realization of the Stellar decomposition for spatially embedded complexes. The decomposition in a Stellar tree is based on a hierarchical spatial index with a simple tuning parameter to facilitate balancing storage and performance needs.
- The definition of *Sequential Range Encoding (SRE)*, a compact encoding for the entities indexed by each region of the decomposition. When applied to CP complexes reindexed by the spatial decomposition of a Stellar tree, SRE yields compressed Stellar trees with only a small overhead relative to the original CP complex's cells.

Outline The remainder of this paper is organized as follows. In Sections 2 and 3, we review background notions and related work, respectively. In Section 4, we define Stellar decompositions, describe our compact encoding, and provide a high-level description of the procedure for generating a Stellar decomposition. In Section 5, we define the Stellar tree, a spatio-topological realization of the Stellar decomposition. In Section 6, we describe a general mesh processing paradigm that can be followed by applications defined on a Stellar tree. In Section 7, we discuss our experimental setup and evaluate how our tuning parameter affects the quality of a Stellar tree's decomposition and its performance in extracting topological features. We then compare Stellar trees to several state-of-the-art topological data structures in Section 8. In Section 9, we describe how to extract local connectivity information from the Stellar tree and evaluate the performance of these algorithms. We provide a high-level overview of several mesh processing and analysis applications that have benefited from Stellar trees to process large datasets in Section 10 and conclude in Section 11 with some remarks and directions for future work.

2. Background notions

In this section, we review notions related to cell and simplicial complexes, which are the basic combinatorial structures for representing discretized shapes. Throughout the paper, we use n to denote the dimension of the ambient space in which the complex is embedded, d to represent the dimension of the complex and k to denote the dimension of a cell from the complex, where $0 \leq k \leq d$.

A k-dimensional *cell* in the n-dimensional Euclidean space $\mathbb{E}^n$ is a subset of $\mathbb{E}^n$ homeomorphic to a closed k-dimensional ball $B^k = \{x \in \mathbb{E}^k : \|x\| \leq 1\}$. A d-dimensional *cell complex* Γ in $\mathbb{E}^n$ is a finite set of cells with disjoint interiors and of dimension at most d such that the boundary of each k-cell γ in Γ consists of the union of other cells of Γ with dimension less than k. Such cells are referred to as the *faces* of γ. A cell which does not belong to the boundary of any other cell in Γ is called a *top cell*. Γ is a *pure* cell complex when all top cells have dimension d. The subset of $\mathbb{E}^n$ spanned by the cells of Γ is called the *domain* of Γ. An example of a pure cell 3-complex is shown in Fig. 1(a): all its top cells are 3-cells (tetrahedra).

Throughout this paper, we are concerned with a restricted class of cell complexes whose cells can be fully reconstructed by their set of vertices, e.g., via a canonical ordering [1–5]. We refer to this class of complexes as *Canonical Polytope complexes (CP complexes)*, and note that it includes simplicial complexes, cubical complexes, polygonal cell complexes and heterogeneous meshes with cells from the finite element 'zoo' (e.g., simplices, hexahedra, pyramids, and prisms). In what follows, we denote a CP complex as Σ. An example of a CP complex is shown in Fig. 1(b), which contains top edges, triangles, quadrilaterals, and tetrahedra.

A pair of cells in a CP complex Σ are mutually *incident* if one is a face of the other. They are *h-adjacent* if they have the same dimension $k > h$ and are incident in a common h-face. We informally refer to vertices (0-cells) as *adjacent* if they are both incident in a common edge (1-cell) and, similarly, for k-cells that are incident in a common $(k-1)$-cell (i.e., they are $(k-1)$-adjacent). The *(combinatorial) boundary* of a CP cell σ is defined by the set of its faces. The *star* of a CP cell σ is the set of its *co-faces*, i.e., CP cells in Σ that have σ as a face. An example of star for a 0-cell (vertex) is shown in Fig. 2(a). In this example, the star of vertex v_0 is formed by five edges, four triangles, a quad, and a tetrahedron. Of these CP cells, tetrahedron σ_5, quad σ_1 and triangle σ_4 are *top* cells. The *link* of a CP cell σ is the set of all the faces of cells in the star that are not incident in σ. An example of link for a 0-cell (vertex) is shown in Fig. 2(b). In this example, the link of v_0 is composed of six vertices, six edges, and a triangle.

Two h-cells σ and σ' in Σ are $(h-1)$-*connected* if there is a sequence, called an *h-path*, of $(h-1)$-adjacent h-cells in Σ from σ to σ'. A complex Σ is *h-connected*, if for every pair of h-cells σ_1 and σ_2, there is an h-path in Σ joining σ_1 and σ_2.

We can now define a d-dimensional *CP complex* Σ as a set of CP-cells in $\mathbb{E}^n$ of dimension at most d such that: (1) Σ contains all CP-cells in the boundary of the CP-cells in Σ; (2) the intersection of any two CP-cells in Σ is *conforming*, i.e., it is either empty, or it consists of faces shared by both CP-cells. *Simplicial complexes* are an important subset of CP complexes whose cells are *simplices*. Let k be a non-negative integer. A k-simplex σ is the convex hull of $k+1$ independent points in $\mathbb{E}^n$ (with $k \leq n$), called vertices of σ. A *face* of a k-simplex σ is an h-simplex $(0 \leq h \leq k)$ generated by $h+1$ vertices of σ.

Other important notions are those of *manifolds* and *pseudo-manifolds*. A subset M of the Euclidean space $\mathbb{E}^n$ is called a *d-manifold*, with $d \leq n$, if and only if every point of M has a neighborhood homeomorphic to the open d-dimensional ball. A more practical concept for the purpose of representing CP complexes is that of pseudo-manifold. A pure d-dimensional CP complex Σ is

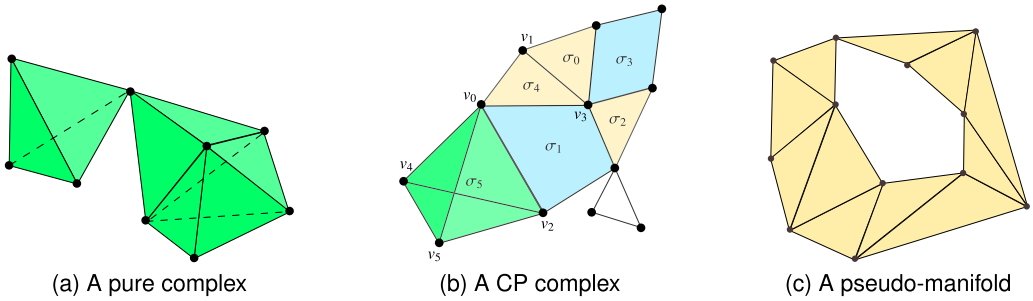

Fig. 1. Examples of CP complexes. (a) A pure simplicial 3-complex with four tetrahedra. (b) A CP complex with three top edges, three top triangles, two top quads and a top tetrahedron. (c) A 2-dimensional *pseudo-manifold* with eleven triangles.

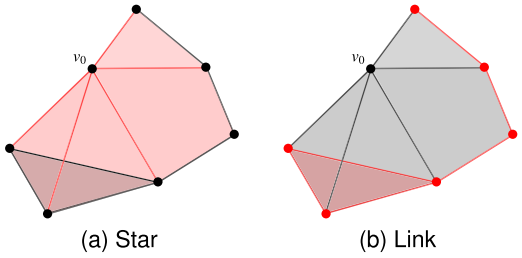

Fig. 2. The star and the link of 0-cell (vertex) v_0 from the complex in Fig. 1(b). Cells belonging to either the star (a) or link (b) of v_0 are highlighted in red. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

said to be a *pseudo-manifold* when it is $(d-1)$-connected and its $(d-1)$-cells are incident in at most two d-cells. Informally, we refer to the connected and compact subspace of $\mathbb{E}^n$ not satisfying the manifold conditions as *non-manifold*.

Queries on a cell complex are often posed in terms of *topological relations*, which are defined by the adjacencies and incidences of its cells. Let us consider a d-dimensional CP complex Σ and a k-cell $\sigma \in \Sigma$, with $0 \leq k \leq d$:

- a *boundary relation* $R_{k,p}(\sigma)$, with $0 \leq p < k$, consists of the p-cells of Σ in the boundary of σ;
- a *co-boundary relation* $R_{k,q}(\sigma)$, with $k < q \leq d$, consists of the q-cells of Σ in the star of σ;
- an *adjacency relation* $R_{k,k}(\sigma)$ consists of the set of k-cells of Σ that are $(k-1)$-adjacent to σ.

For some examples of topological relations, consider the CP complex in Fig. 1(b): Boundary relation $R_{3,0}$ for tetrahedron σ_5 is the list of its boundary vertices, i.e., $R_{3,0}(\sigma_5) = \{v_0, v_2, v_4, v_5\}$. Co-boundary relation $R_{0,2}$ for vertex v_3 is the list of its incident 2-cells (triangles and quads), i.e., $R_{0,2}(v_3) = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$. Adjacency relation $R_{0,0}$ for vertex v_0 is the list of its adjacent vertices, i.e., $R_{0,0}(v_0) = \{v_1, v_2, v_3, v_4, v_5\}$.

3. Related work

In this section, we review the state of the art on topological mesh data structures, hierarchical spatial indexes, data layouts and distributed mesh data structures.

3.1. Topological mesh data structures

There has been much research on efficient representations for manifold cell and simplicial complexes, especially for the 2D case. A comprehensive survey of topological data structures for manifold and non-manifold shapes can be found in [6].

A topological data structure over a cell complex encodes a subset of its topological relations and supports the efficient reconstruction of local topological connectivity over its cells. Topological data structures can be classified according to: (i) the *dimension* of the cell complex, (ii) the *domain* to be approximated, i.e., manifolds versus non-manifold shapes, (iii) the subset of *topological information* directly encoded, and (iv) the *organization* of topological information directly encoded, i.e., explicit or implicit data structures.

The explicit cells and topological relations can either be allocated on demand using small local data structures, such as linked lists, or contiguously, e.g. using arrays. In the former case, pointers are used to reference the elements, which can be useful when the data structure needs to support frequent updates to the underlying cells or their connectivity. In the latter case, indexes of the cells within the array can be used to efficiently reference the elements. Recently, an approach has been proposed in [7] to reconstruct topological relations on demand and to cache them for later reuse.

Broadly speaking, topological data structures can be categorized as *incidence-based* or *adjacency-based*. Whereas incidence-based data structures primarily encode their topological connectivity through incidence relations over all the complex's cells, adjacency-based data structures primarily encode their connectivity through adjacency relations over its top cells.

The *Incidence Graph* (*IG*) [8] is the prototypical incidence-based data structure for cell complexes in arbitrary dimension. The IG explicitly encodes all cells of a given cell complex Γ, and for each p-cell γ, its immediate boundary and co-boundary relations (i.e., $R_{p,p-1}$ and $R_{p,p+1}$). Several compact representations with the same expressive power as the IG have been developed for simplicial complexes [9,10], which typically require less than half the storage space as the IG [11].

Several incidence-based data structures have been developed for manifold 2-complexes, which encode the incidences among edges. The *half-edge* data structure [12] is the most widely data structure of this type [13,14]. Design tradeoffs for data structures based on half-edges are discussed in [15]. *Half-faces* [16] generalize the notion of a half-edge to polyhedral complexes, while *combinatorial maps* [17,18] generalize this notion to higher dimensions.

Indexed data structures [19] provide a more compact alternative by explicitly encoding only vertices, top cells and the boundary relations from top cells to their vertices. Since the cells of a CP complex are entirely determined by their ordered list of vertices, this provides sufficient information to efficiently extract all boundary relations among the cells, but not the co-boundary or adjacency relations. The *Indexed data structure with Adjacencies (IA)* [20,21] extends the indexed representation to manifold simplicial complexes of arbitrary dimension by explicitly encoding adjacency relation $R_{d,d}$, giving rise to an adjacency-based representation. All remaining topological relations can be efficiently

recovered if we also encode a top simplex in the star of each vertex (i.e., a subset of relation $R_{0,d}$).

The *Corner-Table (CoT)* data structure [22] is also adjacency-based. It is defined only for triangle meshes, where it has the same representational power as the IA data structure. It uses *corners* as a conceptual abstraction to represent individual vertices of a triangle and encodes topological relations among corners and their incident vertices and triangles. Several efficient extensions of the Corner-Table data structure have been proposed that exploit properties of manifold triangle meshes [23,24]. The *Sorted Opposite Table (SOT)* data structure [25] extends the Corner-Table data structure to tetrahedral meshes and introduces several storage optimizations. Notably, it supports the reconstruction of boundary relation $R_{d,0}$ from co-boundary relations $R_{0,d}$ (implicitly encoded) and $R_{d,d}$ relations (explicitly encoded), reducing its topological overhead by nearly a factor of two. Since modifications to the mesh require non-local reconstructions of the associated data structures, this representation is suitable for applications on static meshes.

The *Generalized Indexed data structure with Adjacencies (IA* data structure)* [26] extends the representational domain of the IA data structure to arbitrary non-manifold and mixed dimensional simplicial complexes. The IA* data structure is compact, in the sense that it gracefully degrades to the IA data structure in locally manifold neighborhoods of the mesh, and has been shown to be more compact than incidence-based data structures, especially as the dimension increases [11]. A similar data structure for non-manifold complexes was presented in [27]. A detailed description can be found in Section 8.2.

The *Simplex tree* [28] also encodes general simplicial complexes of arbitrary dimension. It explicitly stores all simplices of the complex within a *trie* [29] whose nodes are in bijection with the simplices of the complex. A public domain implementation is available in the *GUDHI* library [30]. We provide a detailed description of this data structure in Section 8.2. Boissonnat et al. [31] also propose two top-based data structures targeting a compact Simplex tree representation. The *Maximal Simplex Tree* (*MST*) is an induced subgraph of the Simplex tree, in which only the paths corresponding to top simplices are encoded, but most operations require processing the entire complex. The *Simplex Array List* (*SAL*) is a hybrid data structure computed from the top simplices of a simplicial complex Σ that improves processing efficiency by increasing the storage overhead. Both the *MST* and the *SAL* are interesting structures from a theoretical point-of-view, but, as described in [31], the model does not currently scale to large datasets and results are limited to complexes with only a few thousand vertices. Moreover, to the best of our knowledge, there is no public domain implementation currently available.

The *Skeleton-Blocker* data structure [32] encodes simplicial complexes that are close to *flag complexes*, simplicial complexes whose top simplices are entirely determined from the structure of their 1-skeleton, i.e., the vertices and edges of the complex, and has been successfully employed for executing edge contractions on such complexes. It encodes the 1-skeleton and the *blockers*, simplices that are not in Σ, but whose faces are. Its generation procedure is computationally intensive for general simplicial complexes since identifying the *blockers* requires inserting simplices of all dimensions.

We compare the Stellar tree representation with the IA, CoT, and SOT data structures as well as with the Simplex tree, and IA* data structure in Section 8.2.

3.2. Hierarchical spatial indexes

A spatial index is a data structure used for indexing spatial information, such as points, lines or surfaces in the Euclidean space. Spatial indexes form a decomposition of the embedding space into *regions*. This can be driven by: (i) an *object-based* or a *space-based* criterion for generating the decomposition; and (ii) a *hierarchical* or a *non-hierarchical* (*flat*) organization of the regions. These properties are independent, and, thus, we can have hierarchical object-based decompositions as well as flat space-based ones.

We now consider how the regions of a decomposition can intersect. In an *overlapping* decomposition the intersection between the regions can be non-empty on both the interiors and on the boundary of their domain, while, in a *non-overlapping* decomposition, intersections can only occur on region boundaries. We say that a region is *nested* within another region if it is entirely contained within that region. In the remainder of this section, we focus primarily on *hierarchical spatial indexes*, which can be classified by the dimensionality of the underlying ambient space and by the types of entities indexed.

Hierarchical spatial indexes for point data are provided by *Point Region (PR)* quadtrees/octrees and kD-trees [33]. In these indexes, the shape of the tree is independent of the order in which the points are inserted, and the points are only indexed by leaf blocks. The storage requirements of these data structures can be reduced by allowing leaf blocks to index multiple points, as in the *bucket PR* quadtree/octree [33], whose *bucketing threshold* determines the number of points that a leaf block can index before it is refined.

Several data structures have been proposed for spatial indexing of *polygonal maps (PM)*, including graphs and planar triangle meshes. *PM quadtrees* [34] extend the PR quadtrees to represent polygonal maps considered as a structured collection of edges. While there are several variants (PM_1, PM_2, PM_3 and the randomized *PMR)*, which differ in the criterion used to refine leaf blocks, all maintain within the leaf blocks a list of intersecting edges from the mesh. The PM_2*-Triangle quadtree* [35] specializes PM quadtrees over triangle meshes and has been applied to terrain models. The PM index family has also been extended to *PM-octrees* encoding polyhedral objects in 3D [33,36,37], where the subdivision rules have been adjusted to handle edges and polygonal faces of the mesh elements. Another proposal for triangulated terrain models are *Terrain trees* [38], that are a spatial index family for the efficient representation and analysis of large-scale triangulated terrains generated from *LiDAR* (*Light Detection and Ranging*) point clouds. A collection of spatial indexes for tetrahedral meshes called *Tetrahedral trees* was developed in [39,40].

We note that data structures in the PM family are *spatial data structures* optimized for efficient spatial queries on a complex (e.g., point location, containment and proximity queries) and are not equipped to reconstruct the connectivity of the complex. In contrast, the *PR-star octree* [41] is a topological data structure for tetrahedral meshes embedded in 3D space. It augments the bucket PR octree with a list of tetrahedra incident in the vertices of its leaf blocks, i.e., those in the *star* of its vertices.

In this paper, we have generalized the PR-star data structure to handle a broader class of complexes (CP complexes) in arbitrary dimensions and with an arbitrary domain (i.e., non-manifold and non-pure complexes). At the same time, our new leaf block encoding exploits the spatial coherence of the mesh, yielding a significant storage saving compared to PR-star trees (see Section 8.1). As we discuss in Section 10, Stellar trees have been shown to be effective in several geometrical and topological applications including local curvature estimation, mesh validation and simplification [41], morphological feature extraction [68] and morphological simplification [91], among others.

3.3. Optimized data layouts

Considerable effort has been devoted to reindexing meshes to better exploit their underlying spatial locality, for example, to support streamed processing [42], better cache locality [43] or

compression [44]. Cignoni et al. [45] introduce an external memory spatial data structure for triangle meshes embedded in $\mathbb{E}^3$. Whereas our aim is to enable efficient topological operations on the elements of general simplicial and CP complexes, the objective of [45] is to support compact out-of-core processing of massive triangle meshes. Since the data structure in [45] is dimension-specific, by exploiting geometric and topological properties of triangle meshes in $\mathbb{E}^3$, it would be difficult to generalize to CP complexes or to higher dimensions. Dey et al. [46] use an octree to index a large triangle mesh for localized Delaunay remeshing. Due to the significant overhead associated with their computations, the octrees in [46] are typically shallow, containing very few octree blocks. In the context of interactive rendering and visualization of large triangulated terrains and polygonal models, Cignoni et al. [47,48] associate patches of triangles with the simplices of a multiresolution diamond hierarchy [49].

3.4. Distributed mesh data structures

Stellar decompositions and Stellar trees are also related to distributed mesh data structures [50,51], which partition large meshes across multiple processors for parallel processing e.g. in numerical simulations [52–54]. In the latter, each computational *domain* maintains a mapping between its boundary elements and their counterparts on neighboring domains. To reduce interprocess communication during computation, each domain might also include one or more layers of elements from other domains surrounding its elements, typically referred to as *ghost*, *rind* or *halo* layers [55–57]. Although each region of a Stellar decomposition (or tree) can be seen as a computational domain in a distributed data structure with a single ghost layer (i.e., the elements in the star of its boundary vertices), Stellar trees are aimed at providing efficient processing on coherent subsets of the mesh (regions), where users can generate optimized local topological data structures. In a distributed regime, we envision Stellar trees helping more with fine-grained (intra-domain) parallelism than with coarse-grained multi-domain partitions.

4. Stellar decomposition

The *Stellar decomposition* is a model for data structures representing *Canonical Polytope (CP) complexes*. We denote a CP complex as Σ, its ordered lists of vertices as Σ_V and of top CP cells as Σ_T. We provide a definition of the Stellar decomposition in Section 4.1, and describe its encoding in Section 4.2.

4.1. Definition

Given a CP complex Σ, a *decomposition* Δ of its vertices Σ_V is a collection of subsets of Σ_V such that every vertex $v \in \Sigma_V$ belongs to one of these subsets. We will refer to the elements of decomposition Δ as *regions*, which we will denote as r.

A Stellar decomposition $\mathbb{S}_\mathbb{D}$ defines a map from the regions of a decomposition Δ of its vertex set Σ_V to the vertices and top CP cells of complex Σ. Formally, a Stellar decomposition is defined by three components:

1. a *CP complex* Σ;
2. a *decomposition* Δ whose regions cover the vertices of Σ;
3. a *map* Φ from regions of Δ to entities of Σ.

Thus, a Stellar decomposition is a triple $\mathbb{S}_\mathbb{D} = (\Sigma, \Delta, \Phi)$. Since Σ is entirely characterized by its vertices, and top CP cells, we define map Φ in terms of the two components: Φ_{VERT}, the mapping to vertices and Φ_{TOP}, the mapping to top CP cells.

For the vertices, we have a map from Δ to Σ_V based on an application-dependent *'belongs to'* property. Formally, $\Phi_{VERT} : \Delta \rightarrow \mathcal{P}(\Sigma_V)$ is a map from Δ to the powerset of Σ_V where

$\forall r \in \Delta, \Phi_{VERT}(r) = \{v \in \Sigma_V : v$ *'belongs to'* $r\}$.

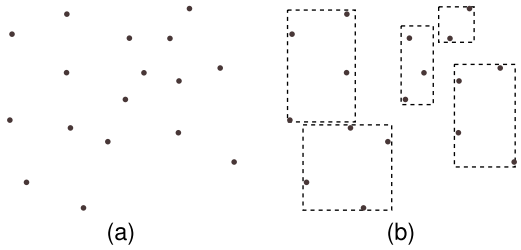

Fig. 3. Example mapping function Φ_{VERT} in 2D. An initial set of points (a) is associated with the regions of a decomposition Δ (b).

Fig. 3 illustrates an example decomposition Δ over a point set where mapping function Φ_{VERT} associates points with regions of Δ. In this paper, we will assume that each vertex in Σ_V is uniquely associated with a single region r in Δ.

The Stellar decomposition gets its name from the properties of its top cell map Φ_{TOP}. For each region r of Δ, $\Phi_{TOP}(r)$ is the set of all top CP cells of Σ_T incident in one or more vertices of $\Phi_{VERT}(r)$. In other words, $\Phi_{TOP}(r)$ is defined by the union of cells in the *star* of the vertices in $\Phi_{VERT}(r)$. Formally, $\Phi_{TOP} : \Delta \rightarrow \mathcal{P}(\Sigma_T)$ is a function from Δ to the powerset of Σ_T, where

$$\forall r \in \Delta, \Phi_{TOP}(r) = \{\sigma \in \Sigma_T | \exists v \in R_{k,0}(\sigma) : v \in \Phi_{VERT}(r)\}. \quad (1)$$

Fig. 4 illustrates mapping Φ_{TOP} for two regions of the decomposition of Fig. 3(b) on a triangle mesh defined over its vertices. Note that Φ_{TOP} is based on a topological rather than a spatial property: A top CP cell σ is only associated with a region r when one (or more) of its vertices is associated with r under Φ_{VERT}.

To characterize this representation, we define the *spanning number* χ_σ of a top CP cell in a Stellar decomposition as the number of regions to which it is associated.

Definition 4.1. Given Stellar decomposition $\mathbb{S}_\mathbb{D} = (\Sigma, \Delta, \Phi)$, the *spanning number* χ_σ of a top CP cell $\sigma \in \Sigma_T$ is the number of regions in Δ that map to σ. Formally,

$$\forall \sigma \in \Sigma_T, \ \chi_\sigma = |\{r \in \Delta | \sigma \in \Phi_{TOP}(r)\}|. \quad (2)$$

A consequence of the unique mapping of each vertex in Φ_{VERT} is that it provides an upper bound on the spanning number of a top CP cell in a Stellar decomposition. Specifically, the spanning number χ_σ of a top CP cell σ is bounded by the cardinality of its vertex incidence relation $R_{k,0}$: $1 \leq \chi_\sigma \leq |R_{k,0}(\sigma)|$.

It is also interesting to consider the *average spanning number* χ as a global characteristic of the efficiency of a Stellar decomposition over a complex, measuring the average number of times each top CP cell is represented.

Definition 4.2. The *average spanning number* χ of a Stellar decomposition $\mathbb{S}_\mathbb{D}$ is the average number of regions indexing its top cells. Formally,

$$\chi = \left(\sum_{\sigma \in \Sigma_T} \chi_\sigma\right) / |\Sigma_T| = \left(\sum_{r \in \Delta} |\Phi_{TOP}(r)|\right) / |\Sigma_T|. \quad (3)$$

4.2. Encoding

In this section, we describe how we represent the two components of a Stellar decomposition, providing a detailed description of the data structures for representing a CP complex (subsection 4.2.1), and a compressed encoding for the regions of the partitioning (subsection 4.2.2). We do not describe how the decomposition Δ is represented, as this is specific to each concrete realization of the Stellar decomposition model.

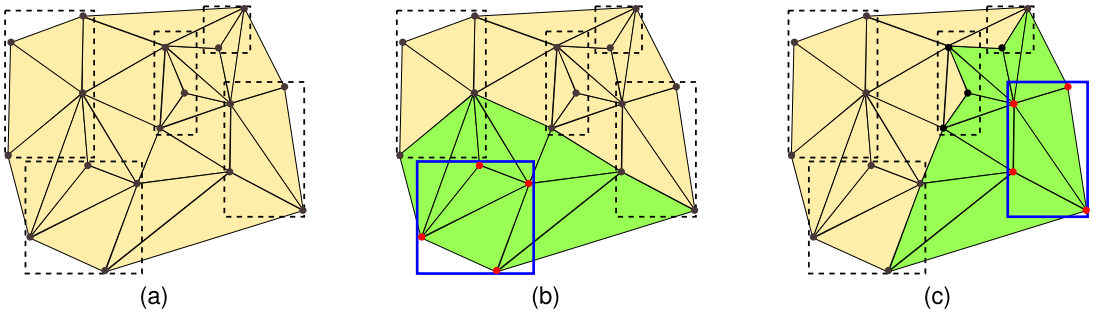

Fig. 4. Mapping function Φ_{TOP} for the decomposition Δ from Fig. 3. Given a triangle mesh (a) and a vertex map Φ_{VERT} on Δ, Φ_{TOP} associates the triangles in the star of the vertices in $\Phi_{VERT}(r)$ to $\Phi_{TOP}(r)$. (b) and (c) highlight the triangles (green) associated with two different regions (blue) of Δ. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

4.2.1. Indexed representation of the CP complex

We represent the underlying CP complex as an indexed complex [19], which encodes the vertices, top CP k-cells and the boundary relation $R_{k,0}$ of each top k-cell in Σ. In the following, we assume a d-dimensional CP complex Σ embedded in $\mathbb{E}^n$.

We use an array-based representation for the vertices and for the top cells of Σ. Since the arrays are stored contiguously, each vertex v has a unique position index i_v in the vertex array, that we denote as Σ_V. Similarly, each top CP cell σ has a unique position index i_σ. The top CP cells of Σ are encoded using separate arrays Σ_{T_k} for each dimension $k \leq d$ that has top CP cells in Σ. Σ_{T_k} encodes the boundary connectivity from the top CP k-cells of Σ to their vertices, i.e., relation $R_{k,0}$ in terms of the indices i_v of the vertices of its cells within Σ_V. This requires $|R_{k,0}(\sigma)|$ references for a top k-cell σ, e.g., k+1 vertex indices for a k-simplex and 2^k references for a k-cube. Thus, the total storage cost of Σ_T is:

$$\sum_{k=1}^{d} \sum_{\sigma \in \Sigma_{T_k}} |R_{k,0}(\sigma)|. \tag{4}$$

We note that when Σ is pure (i.e., its top CP cells all have the same dimension d), the encoding of Σ requires only two arrays: one for the vertices and one for the top cells. For simplicity, we refer to the top cell arrays collectively as Σ_T.

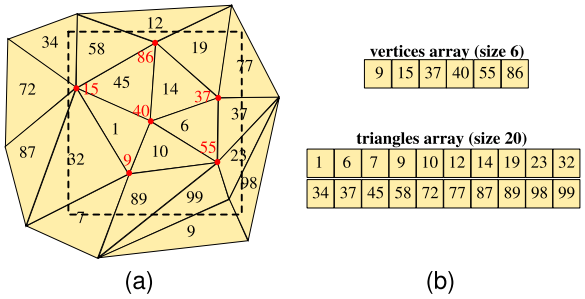

Fig. 5. EXPLICIT encoding for triangles within a region (dotted square). The arrays explicitly encode the 6 vertices and 20 triangles in the region.

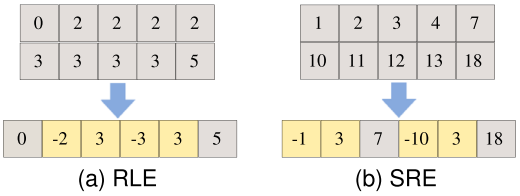

Fig. 6. Compressed arrays of non-negative integers using (a) *Run Length Encoding* (RLE) and (b) *Sequential Range Encoding* (SRE).

4.2.2. A compressed region representation

In this subsection, we discuss two encoding strategies for the data maps in each region of the partition Δ. We begin with a simple strategy that explicitly encodes the arrays of vertices and top CP cells associated with each region and work our way to a compressed representation of these arrays. Coupling this compressed representation with a reorganization of the vertices and cells of the CP complex (as we will describe in Section 4.3) yields a significant reduction in storage requirements. We will demonstrate this claim in Section 8.1 on a data structure instantiating the Stellar decomposition.

Recall that under Φ, each region r in Δ maps to an array of vertices and an array of top CP cells from the complex Σ which we denote as r_V and r_T, respectively. A straightforward strategy would be to encode arrays of vertices and top CP cells that explicitly enumerate the associated elements for each region r. We refer to this as the EXPLICIT Stellar decomposition encoding. An example of this encoding for a single region with six vertices in r_V and twenty triangles in r_T is shown in Fig. 5.

It is apparent that the above encoding can be very expensive due to the redundant encoding of top CP cells with vertices in multiple regions. A less obvious redundancy is that it does not account for the ordering of the elements.

We now consider a COMPRESSED Stellar decomposition encoding that compacts the vertex and top CP cells arrays in each region r by exploiting the *locality* of the elements within r. The COMPRESSED encoding reduces the storage requirements within region arrays by replacing *runs* of incrementing consecutive sequences of indices using a generalization of *run-length encoding (RLE)* [58]. RLE is a form of data compression in which *runs* of consecutive identical values are encoded as pairs of integers representing the value and repetition count, rather than as multiple copies of the original value. For example, in Fig. 6(a), the four entries with value '2' are compacted into a pair of entries [-2, 3], where a negative first number indicates the start of a run and its value, while the second number indicates the remaining elements of the run in the range.

While we do not have such duplicated runs in our indexed representation, we often have increasing sequences of indexes, such as {40,41,42,43,44}, within a local vertex array r_V or top CP cells array r_T. We therefore use a generalized RLE scheme to compress such sequences, which we refer to as *Sequential Range Encoding (SRE)*. SRE encodes a run of *consecutive* non-negative indexes using a pair of integers, representing the starting index, and the number of remaining elements in the range. As with RLE, we can intersperse runs (sequences) with non-runs in the same array by negating the starting index of a run (e.g. [-40, 4] for the above example). Thus, it is easy to determine whether or not we are in a run while we iterate through a sequential range encoded array. A nice feature of this scheme is that it allows us to dynamically append individual

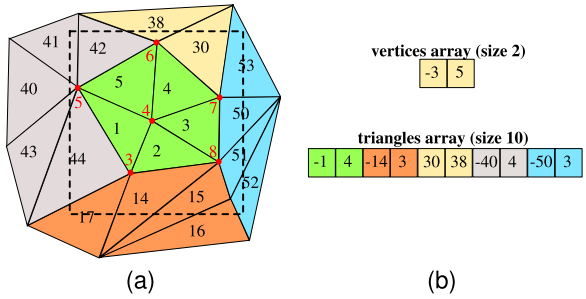

Fig. 7. COMPRESSED encoding within a region (dotted square) after reindexing the vertices and triangles of the mesh from Fig. 5.

elements or runs to an SRE array without any storage overhead (other than occasional array reallocations).

Furthermore, we can easily *expand* a compacted range by replacing its entries with the first two values of the range and appending the remaining values to the end of the array. After the updates are finished, we can sort the array and reapply SRE compaction to recover space. Fig. 6 (b) shows an example SRE array over an array, where, e.g., sequence {1,2,3,4} is represented as [-1, 3].

To facilitate comparisons between the EXPLICIT and COMPRESSED representations of a Stellar decomposition, we introduce a global characteristic that measures the average storage requirements to represent a top CP cell.

Definition 4.3. The *average reference number* μ of a Stellar decomposition is the average number of references required to encode a top CP cell in the r_T arrays of the regions in Δ. Formally:

$$\mu = \left(\sum_{r \in \Delta} |r_T| \right) / |\Sigma_T| \tag{5}$$

where $|r_T|$ is the size of the top CP cells array in a region r.

In contrast to the average spanning number χ, which is a property of the decomposition, the average reference number μ is a property of how the decomposition is encoded. An EXPLICIT representation is equivalent to a COMPRESSED representation without any compressed runs, and, thus, it is always the case that $\mu \leq \chi$. In the EXPLICIT representation (i.e., without any sequence-based compression), $\mu = \chi$, while in the COMPRESSED representation, μ decreases as the compression of the r_V and r_T arrays becomes more effective. Fig. 7 illustrates a COMPRESSED representation of the mesh from Fig. 5 after its vertex and triangle arrays have been reordered (in an external process) and highlights its sequential ranges, where r_V is encoded by a single run and r_T is encoded by four sequential runs as well as several non-run indices.

4.3. Generating a stellar decomposition

We now describe how to generate a COMPRESSED Stellar decomposition from an indexed CP complex Σ and a given partition Δ on its vertices Σ_V. This process consists of three phases:

1. reindex the vertices of Σ following a traversal of the regions of Δ and SRE-compress the r_V arrays;
2. insert the top CP cells of Σ into Δ;
3. reindex the top CP cells of Σ based on locality within common regions of Δ and SRE-compress the regions r_T arrays.

As it can be noted, the generation process ignores how the partitioning on the vertices is obtained, since this step is defined by the data structure instantiating a Stellar decomposition. The reindexing of the vertices follows a traversal of the regions of Δ in such a way that all vertices associated with a region have a contiguous range of indices in the reindexed global vertex array Σ_V (as detailed in the Supplementary material).

The second phase inserts each top CP k-cell σ, with index i_σ in Σ_{T_k}, into all the regions of Δ that index its vertices. This is done by iterating through the vertices of σ and inserting i_σ into the r_T array of each region r whose vertex map $\Phi_{VERT}(r)$ contains at least one of these vertices. As such, each top CP k-cell σ appears in at least one and at most $|R_{k,0}(\sigma)|$ regions of Δ. Due to the vertex reindexing of step 1, this operation is extremely efficient. Determining if a vertex of a given cell lies in a block requires only a range comparison on its index i_v.

Finally, we reindex the top CP cell arrays Σ_T to better exploit the locality induced by the vertex-based partitioning and compress the local r_T arrays using a sequential range encoding over this new index. The reindexing and the compression of the top CP cells is obtained following a traversal of the regions of Δ in such a way that all top CP cells associated with the same set of regions have a contiguous range of indices in the reindexed arrays Σ_T. This last step is detailed in the Supplementary material. As we demonstrate in Section 8, this compression yields significant storage savings in a wide range of mesh datasets.

5. Stellar trees

The Stellar decomposition is a general model that is agnostic about how the partitioning is attained and about its relationship with the underlying CP complex. Thus, for example, we can define a Stellar decomposition using Voronoi diagrams, or based on a nearest neighbor clustering of the vertices of a given CP complex. In this section, we introduce *Stellar trees* as a class of Stellar decompositions defined over nested spatial decompositions of the CP complex and discuss some of our design decisions. Before defining a Stellar tree (Section 5.1), its encoding (Section 5.2) and its generation procedure (Section 5.3), we review some underlying notions.

The *ambient space* $\mathbb{A}$ is the subset of $\mathbb{E}^n$ in which the data is embedded. We consider the region bounding the ambient space to be a hyper-rectangular *axis-aligned bounding block*, which we refer to simply as a *block*. A k-dimensional *closed* block r in $\mathbb{E}^n$, with $k \leq n$, is the Cartesian product of closed intervals $[l_i, u_i]$ where exactly k of them are non-degenerate, i.e., $r = \{(x_1, \ldots, x_n) \in \mathbb{E}^n \mid x_i \in [l_i, u_i]\}$ and $|\{i \mid l_i < u_i\}| = k$.

Given two blocks $r := [l_i, u_i]$ and $r' := [l'_i, u'_i]$, r' is a *face* of r if, for each dimension i, either their intervals overlap (i.e., $l'_i = l_i$ and $u'_i = u_i$) or the i-th interval of r' is degenerate (i.e., $l'_i = u'_i = l_i$, or $l'_i = u'_i = u_i$). Given a block r, we refer to its 0-dimensional face of degenerate intervals $x_i = l_i$ as its *lower corner* and to its 0-dimensional face where $x_i = u_i$ as its *upper corner*. The above block definition describes *closed* blocks. It can be useful to allow some faces of r to be *open*, especially on faces of neighboring blocks that overlap only on their boundaries. A k-dimensional *half-open* block r in $\mathbb{E}^n$ is defined as $r = \{(x_1, \ldots, x_n) \in \mathbb{E}^n \mid x_i \in [l_i, u_i)\}$ and $|\{i \mid l_i < u_i\}| = k$. Note that the faces of a half-open block r incident in its lower corner are *closed*, while all other faces of r are *open*.

We now focus on *nested decompositions*, hierarchical space-based decompositions whose overlapping blocks are nested and whose leaf blocks Δ_L (i.e., those without any nested blocks) form a non-overlapping cover of the ambient space $\mathbb{A}$. The nesting relationship defines a *containment hierarchy* $\mathbb{H}$, which can be described using a rooted *tree*. The tree's root $\mathbb{H}_{ROOT}$ covers the ambient space $\mathbb{A}$; the tree's leaves $\mathbb{H}_L$ encode the regions of the decomposition Δ; and its internal nodes $\mathbb{H}_I$ provide access to the regions of the decomposition.

Nested decompositions can adopt different hierarchical refinement strategies. Among the most popular are those based on *regular* refinement and *bisection* refinement of simple primitives (e.g.,

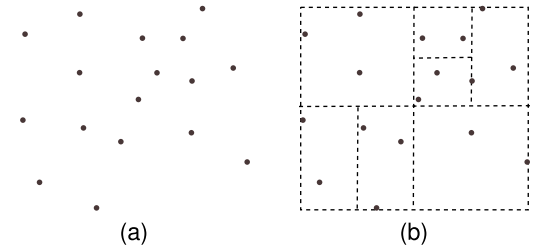

Fig. 8. A mapping function Φ_{VERT} over a nested spatial decomposition Δ. The vertices (a) are partitioned into regions by Δ's leaf blocks (b) using a bucketing threshold, $k_V = 4$, i.e. at most 4 vertices can be in a region.

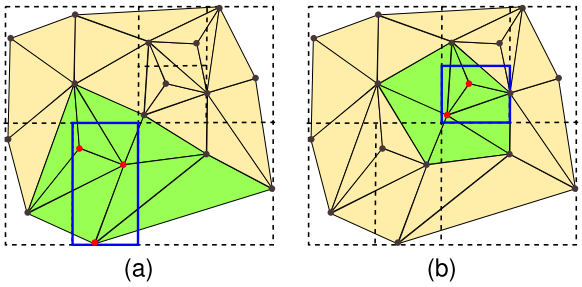

Fig. 9. Top cell mapping function Φ_{TOP} for two blocks (blue) of the nested decomposition from Fig. 8 on the triangle mesh from Fig. 4. $\Phi_{TOP}(r)$ maps the triangles in the star of the vertices in $\Phi_{VERT}(r)$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

simplices and cubes). An n-dimensional block r is regularly refined by adding vertices at all edge and face midpoints of r and replacing r with 2^n disjoint blocks covering r. This generates *quadtrees* in 2D, and *octrees* in 3D [33]. In bisection refinement, a block is bisected along an axis-aligned hyperplane into two blocks, generating *kD-trees* [59].

5.1. Definition

Since a Stellar tree $\mathbb{S}_\mathbb{T}$ is a type of Stellar decomposition, it consists of three components: (1) a *CP complex* Σ embedded in an *ambient space* $\mathbb{A}$; (2) a *nested decomposition* Δ covering the domain of Σ; and (3) a *map* Φ from blocks of Δ to entities of Σ. The nested decomposition is described by a containment hierarchy $\mathbb{H}$, represented by a *tree* whose blocks use the *half-open* boundary convention to ensure that every point in the domain is covered by exactly one leaf block.

Since Stellar trees are defined over nested spatial decompositions that cover the ambient space, we customize the vertex mapping function Φ_{VERT} to partition the vertices of Σ according to spatial containment: each vertex is associated with its single containing leaf block. Formally,

$$\forall r \in \Delta_L, \Phi_{VERT}(r) = \{v \in \Sigma_V : v \cap r \neq \emptyset\}. \quad (6)$$

A two-dimensional example is shown in Fig. 8, where a set of points are associated with the leaf blocks of Δ through Φ_{VERT}.

The top CP cells mapping function Φ_{TOP} for a Stellar tree has the same definition as for the Stellar decomposition (see Eq. 1). Fig. 9 shows the mapping Φ_{TOP} for two blocks of the nested kD-tree decomposition of Fig. 8(b) over the triangle mesh from Fig. 4.

Since the nested decomposition Δ, and, consequently, the tree $\mathbb{H}$ describing it, are determined by the number of vertices indexed by a block, we utilize a *bucket PR tree* [33] to drive our decomposition. This provides a single tuning parameter, the *bucketing threshold* k_V, that uniquely determines the decomposition for a given complex Σ.

Recall that a (leaf) block r in a bucket PR-tree is considered *full* when it indexes more than k_V vertices (in our case, when $|\Phi_{VERT}(r)| > k_V$). Insertion of a vertex into a full block causes the block to refine and to redistribute its indexed vertices among its children. As such, the domain decomposition of a Stellar tree depends only on the bucketing threshold k_V. Smaller values of k_V yield deeper hierarchies whose leaf blocks index relatively few vertices and top CP cells, while larger values of k_V yield shallower hierarchies with leaf blocks that index more vertices and top CP cells. Thus, k_V and the average spanning number χ of a Stellar tree are inversely correlated.

In practice, we use different spatial indexes to represent $\mathbb{H}$ based on the dimension n of the ambient space $\mathbb{A}$. In lower dimensions, we use a quadtree-like subdivision, i.e., a quadtree in 2D, and an octree in 3D, while in higher-dimensions, we switch to a kD-tree subdivision. As discussed in [33], while quadtree-like subdivisions are quite efficient in low dimensions, the data becomes sparser in higher dimensions (due to the *curse of dimensionality* [60]), and tends to be better encoded by kD-trees.

5.2. Encoding

We represent the containment hierarchy $\mathbb{H}$ using an explicit pointer-based data structure, in which the blocks of $\mathbb{H}$ use a type of `Node` structure that changes state from leaf to internal block during the generation process of a Stellar tree.

We use a *brood-based* encoding [61], where each block in $\mathbb{H}$ encodes a pointer to its parent block and a single pointer to its brood of children. This reduces the overall storage since leaves do not need to encode pointers to their children, and also allows us to use the same representation for n-dimensional quadtrees and kD-trees. We explicitly encode all internal blocks, but only represent leaf blocks r in $\mathbb{H}$ with non-empty maps $\Phi(r)$.

The mapped entities of the CP complex Σ are encoded in the leaf blocks $\mathbb{H}_L$ using the mapping arrays Φ. Note that each leaf block r encodes the arrays of vertices r_V and of top CP cells r_T in terms of the indices i_v and i_σ, respectively, that identify v and σ in the Σ_V and Σ_T arrays. For each block r, we have: (1) three pointers for the hierarchy: one to its parent, another to its list of children and it is pointed to by one parent; (2) a pointer to an array of vertices r_V and the size of this array; (3) a pointer to an array of top CP cells r_T and the size of this array. Thus, the hierarchy $\mathbb{H}$ of a Stellar tree requires $7|\mathbb{H}|$ storage.

By considering the encodings, defined in Section 4.2.2, for the CP complex Σ, and for the vertices and top cp-cells associated with the regions of $\mathbb{H}$, we can estimate the storage requirements for the EXPLICIT and COMPRESSED Stellar trees. An EXPLICIT Stellar tree requires a total of $|\Sigma_V|$ references for its vertex arrays, since each vertex is indexed by a single leaf block, and a total of $\chi|\Sigma_T|$ references for all top CP cells arrays. Thus, the total cost of the EXPLICIT Stellar tree, including the hierarchy (but excluding the cost of the indexed mesh) is: $7|\mathbb{H}| + |\Sigma_V| + \chi|\Sigma_T|$.

Conversely, in a COMPRESSED Stellar tree, we can reindex the vertex array Σ_V in such a way that all vertices associated with the same leaf block are indexed consecutively (see Section A.1 in the Supplementary materials for additional details). Thus, we can encode the r_V arrays using only two integers per leaf block for a total cost of $2|\mathbb{H}_L|$ rather than $|\Sigma_V|$. Moreover, since leaf blocks no longer need to reference an arbitrary array, these two references can be folded into the block's hierarchical representation for r_V: instead of a pointer to a array and a size of the array, we simply encode the range of vertices in the same space. As the cost of representing the r_T arrays is $\mu|\Sigma_T|$, the total cost for encoding a COMPRESSED Stellar tree (excluding the cost of the indexed mesh representation) is: $7|\mathbb{H}| + \mu|\Sigma_T|$.

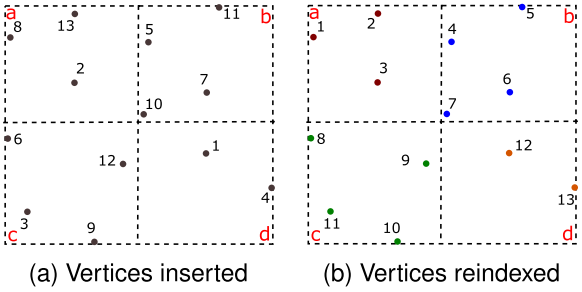

Fig. 10. Generating a nested hierarchy $\mathbb{H}$ with $k_V = 4$ over vertices. After inserting the vertices (a), we reindex Σ_V according to $\mathbb{H}_L$ (b).

5.3. Generating a stellar tree

In this section, we describe how to generate a COMPRESSED Stellar tree from an indexed CP complex Σ and a given bucketing threshold k_V. We can also deal with input complexes that are not already indexed. For example, if our input is a "soup" of CP cells in which each CP cell is specified by a list of coordinates, we can generate an indexed representation of the complex as we insert the vertices and generate the decomposition.

First, given a user-defined bucketing threshold k_V, we generate a bucket PR-tree over the vertices of Σ. The procedure for inserting a vertex v with index i_v in Σ_V into $\mathbb{H}$ is recursive. We use the geometric position of v to traverse the internal blocks to reach the unique leaf block r containing v. After adding v to r (i.e., appending i_v into the r_V array of r), we check if this causes an overflow in r. If it does, we refine r and reinsert its indexed vertices into its children. Once all the vertices in Σ have been inserted, the decomposition is fixed.

The rest of the Stellar tree generation process follows the strategy described in Section 4.3 and detailed in the supplementary materials. One key optimization between a generic partitioning on the vertices and a nested hierarchical decomposition relates to extracting the vertex index ranges. In a Stellar tree, this step is performed through a depth-first traversal of the tree, which, for each leaf block r, generates a contiguous range of indices for the vertices in r, and, for each internal block, provides a single contiguous index range for the vertices in all descendant blocks. For example, in Fig. 10, after executing this step on leaf block b, we have $v_s = 4$ and $v_e = 7$. Similarly, at the end of this step the root $\mathbb{H}_{ROOT}$ has $v_s = 1$ and $v_e = 13$.

We provide an experimental evaluation of the timings for generating a Stellar tree in Section 8.3.

6. Processing paradigm for stellar trees

Mesh processing applications rarely process individual mesh elements. Rather, they typically operate on the entire complex, or on large regions of interest within the complex. The structure of a Stellar tree naturally supports a *batched* processing strategy, i.e., a strategy in which portions of the complex are reconstructed and processed within each block of the tree. As these local blocks are processed, their representation and extracted topological relations can be customized to suit the needs of the application. This helps in amortizing the reconstruction costs and, thus, processing the entire complex efficiently.

The general paradigm for executing application algorithms on a Stellar tree is to iterate through the leaf blocks of hierarchy $\mathbb{H}$, locally processing the encoded complex in a streaming manner. For each leaf block r in $\mathbb{H}$, a local topological data structure catered to the application's needs is constructed and used to process the indexed subcomplex. We refer to this local data structure in a block

Algorithm 1 STELLAR_TREE_PROCESSING_PARADIGM(r, c) .

Input: r is a block in $\mathbb{H}$
Input: c is a fixed-size *LRU*-cache
1: **if** r is an internal block in $\mathbb{H}$ **then**
2: **for all** blocks r_C in CHILDREN(r) **do**
3: STELLAR_TREE_PROCESSING_PARADIGM(r_C,c)
4: **else** // r *is a leaf block in* $\mathbb{H}$
5: **if** r is in c **then**
6: $r_E \leftarrow$ GET(c,r)
7: **else**
8: $r_E \leftarrow$ EXPAND(r) // *expand* r *into* r_E
9: execute application algorithm using r_E
10: **if** MAX_SIZE$(c) > 0$ **then** // *we are using a cache*
11: save r_E in c
12: **else**
13: discard r_E

r as an *expanded leaf-block representation*, and we denote it as r_E. Once we finish processing leaf block r, we discard r_E and begin processing the next block.

For efficiency and at relatively low storage overhead, we can cache the expanded leaf block representation r_E, using a *Least-Recent-Used (LRU)* cache. This is especially advantageous in applications that require processing portions of the complex in neighboring leaf blocks. Adopting a fixed-size cache allows us to amortize the extraction costs of the local data structures, with a controllable storage overhead.

Algorithm 1 outlines the general strategy for processing a Stellar tree. The algorithm recursively visits all the blocks of the hierarchy $\mathbb{H}$. For each leaf block r, we either recover r_E from the *LRU* cache (rows 5–8), or construct the desired application-dependent local topological data structure r_E. After using this local data structure to process the local geometry in r (row 9), we either cache or discard r_E (rows 10–13).

Within this general processing paradigm, we can have two different approaches, that we call *local* and *global*, depending on how auxiliary data structures are encoded and maintained. In a *local* approach, the scope of these auxiliary data structures is limited to that of a single leaf block r, or to a restricted subset of its neighbors. In general, a local approach is preferred for applications that extract, or analyze local features, such as those that depend only on the link or star of cells. These includes, for instance, the extraction of geometric features, like the curvature at a vertex, or the extraction of morphological features, like critical points, when the complex is a discretization of the domain of a scalar field. In these examples, the auxiliary data structures are just needed within the scope of a leaf block r, and thus, immediately discarded after extracting the corresponding feature in r. Conversely, in a *global* approach, data structures are maintained over the entire complex and updated during the visit of the tree. A global approach can be preferable for applications that require the analysis or the processing of the entire complex, like geometric simplification, morphological segmentation, or validation of geometric and topological properties. In these examples, auxiliary data structures are used to represent partial results over the complex.

The decision between using a local and global approach can be driven by the needs of the application or as a tradeoff balancing memory usage and execution times. Due to the limited scope of auxiliary data structures in the local approach, the storage overhead is typically proportional to the complexity of the local complex but requires an increased number of memory allocations compared to a global approach since each leaf block expansion requires memory allocations. Conversely, while auxiliary data structures in the global approach are allocated only once, these structures can

Table 1
Overview of experimental datasets. For each CP complex Σ, we list the number of vertices $|\Sigma_V|$ and of top CP-cells $|\Sigma_T|$.

	TRIANGULAR			QUADRILATERAL			TETRAHEDRAL			HEXAHEDRAL			PROBABILISTIC			V-RIPS		
	NEPTUNE	STATUETTE	LUCY	NEPTUNE	STATUETTE	LUCY	BONSAI	VISMALE	FOOT	F16	SAN FERN	VISMALE	5D	7D	40D	VISMALE 7D	FOOT 10D	LUCY 34D
$\|\Sigma_V\|$	2.00M	5.00M	14.0M	12.0M	30.0M	84.1M	4.25M	4.65M	5.02M	27.9M	61.3M	136M	385K	239K	204K	4.65M	5.02M	14.0M
$\|\Sigma_T\|$	4.01M	10.0M	28.1M	12.0M	30.0M	84.2M	24.4M	26.5M	29.5M	25.4M	55.9M	125M	26.5M	258M	16.5M	6.39M	63.9M	41.1M

require significantly more storage space compared to the local approach.

In Section 10, we present applications on mesh processing and analysis, based on Stellar trees, on which these two paradigms have been extensively applied.

7. Experimental setup

In this section, we describe our experimental setup, including the datasets used in our evaluation (Section 7.1). We also evaluate how the bucketing threshold k_V affects the quality of a Stellar tree's decomposition and its performance in extracting topological queries (Section 7.2).

7.1. Experimental datasets

We have performed experiments on a range of CP complexes consisting of triangle, quadrilateral, tetrahedral and hexahedral meshes in $\mathbb{E}^3$ as well as pure non-manifold simplicial complexes in higher dimensions and higher dimensional non-manifold simplicial complexes (embedded in $\mathbb{E}^3$). Table 1 summarizes the datasets used in our experiments and their numbers of vertices and top cells.

Our triangle and tetrahedral meshes are *native* models ranging from 4 to 28 million triangles and from 24 to 29 million tetrahedra, where we use the term native to refer to models from public domain repositories discretizing objects in space. Since we only had access to relatively small native quadrilateral and hexahedral meshes (with tens to hundreds of thousand elements), we have generated some larger models ranging from 12 to 125 million elements from our triangle and tetrahedral models. The generation procedure refines each triangle into three quadrilaterals and each tetrahedron into four hexahedra by adding vertices at the face centroids.

To experiment with *pure* non-manifold models in higher dimensions, we have generated some models based on a process that we call *probabilistic Sierpinski filtering*, where we regularly refine all simplices in the complex and randomly remove a fixed proportion of the generated simplices in each iteration. For our experiments, we have created 5-, 7- and 40-dimensional models using different levels of refinement and a filtering threshold of 65%, yielding pure simplicial complexes with 16.5 million to 258 million top simplices.

Finally, to experiment with general simplicial complexes in higher dimensions, we have generated several (non-pure) *Vietoris-Rips* complexes, which we embed in a lower dimensional space. A Vietoris-Rips (V-Rips) complex is the *flag* complex defined by a neighborhood graph over a point cloud whose arcs connect pairs of points with distance less than a user-provided parameter ϵ. Given the neighborhood graph, the simplices of the V-Rips complexes are defined by its *cliques*, subsets of the graph vertices that form a complete subgraph. We refer to [62] for further details. For our experiments, we have generated V-Rips complexes over the vertices of a triangle model (LUCY) and of two tetrahedral models (VISMALE and FOOT) from our manifold datasets and set our distance threshold ϵ to $\{0.1\%, 0.5\%, 0.4\%\}$ of the bounding box diagonal, respectively. The range of top simplices in the generated complexes goes from 6.4 million to 64 million and their dimension from 7 to 34. Although the generated complexes are synthetic, they provide a good starting point to demonstrate the efficiency of the Stellar tree in higher dimensions.

All tests have been performed on a PC equipped with a 3.2 gigahertz Intel i7-3930K CPU with 64 gigabytes of RAM. The source code will be made available at [87].

7.2. Calibrating stellar tree bucket thresholds

Spatial indexes typically involve a careful balance among index generation times, storage costs and query performances. Stellar trees provide users with a single tuning parameter k_V to control the maximum number of vertices indexed by each block of the tree. In the following, we calibrate k_V on a characteristic subset of three of our experimental datasets: NEPTUNE triangle mesh, BONSAI tetrahedral mesh, and VISMALE Vietoris-Rips complex. For each dataset, we generated 195 Stellar trees using k_V values ranging from 1 to 1500 and compared Stellar tree generation and query times as well as the number of blocks as a proxy for the complexity of the generated tree. Within this range, we increment k_V by 1 for values between 1 and 50, and by 10 for values between 60 and 1500. This allows us to evaluate the decomposition quality and the extraction performance for a fundamental topological query at different scales. For the latter, we use the vertex co-boundary extraction, i.e., the top cells incident in each vertex (which we describe in Section 9.2).

The results are summarized in the charts of Figs. 11 and 12, which compare the complexity of the generated Stellar tree (in terms of number of blocks), its generation and query times, and the average spanning (χ) and reference (μ) numbers as a function of the threshold value k_V.

To better compare different units (i.e., number of blocks and timings), each chart in Fig. 11 has two logarithmic y-axes, showing the time scale (blue curves using the left y-axes) and the number of blocks (red curves using the right y-axes), respectively. In this way, we can directly compare, for each dataset, how the k_V value influences the decomposition and the timing performances. After an initial rapid decrease in the generation time and block number, the curves begin to level off for increasingly large k_V values. While there are more than a million blocks when k_V is less than 10, the number of blocks rapidly decreases to hundreds of thousands for k_V's between 50 and 200, and grows even smaller for large k_V values (e.g., above 500), where the number of blocks remains steadily in the thousands to tens of thousands. This trend appears to be related to the point distribution within each dataset, which induces finer decompositions for k_V values between 1 and 50, and coarser decompositions for larger k_V values. This trend can also be observed for the generation times, which reduce by a factor of two for k_V values between 1 and 100, and by another factor of two for large bucketing thresholds. While the topological extraction query is largely unaffected by k_V size, it gets slightly faster for larger k_V values. When comparing the influence of k_V on χ and μ (shown in Fig. 12), we observe that the behavior of these two variables is very similar to that of the number of blocks. This is expected, since the top cells distribution is directly linked to the number of blocks in the tree. As mentioned in Section 4.1, the number of leaf blocks

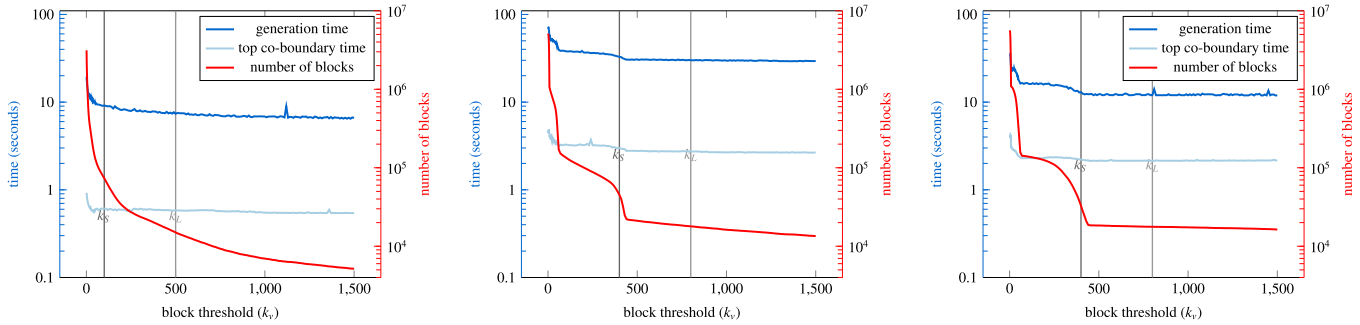

Fig. 11. Bucketing threshold calibration experiments comparing the number of Stellar tree blocks (red, right y-axis) and generation and top-coboundary extraction times (blue, left y-axis) against bucket threshold values (k_V). The vertical bars (gray) represent the k_V values selected for our experiments. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

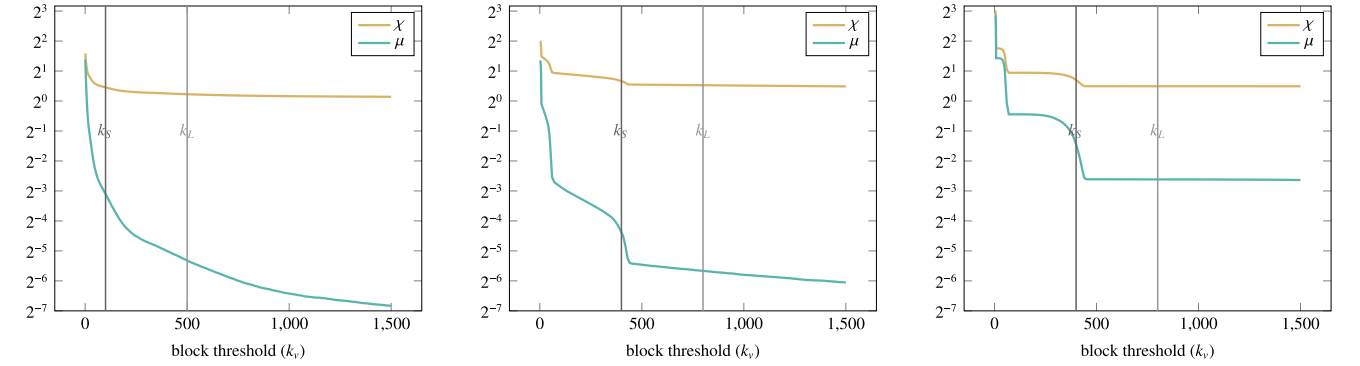

Fig. 12. Bucketing threshold calibration experiments comparing the evolution of the average spanning number χ and of the average reference number μ against bucket threshold values (k_V) for three datasets. The vertical bars (gray) represent the k_V values we selected for our experiments on these datasets.

indexing a top cell is bounded from above by the number of its vertices, and this defines a topological upper bound that reduces the overall storage requirements. We note that the SRE compression is able to reduce the number of references per top cell (μ), even for very small k_V values.

Our calibration experiments indicate that, while there are slight differences in timing and storage costs, Stellar tree performance is relatively stable over a wide range of k_V values. However, threshold values that are either too small or too large should be avoided, since in the first case the storage requirements and time performances are heavily affected, while in the latter case the benefit of having a hierarchical decomposition is limited, as both storage requirements and time performance are not clearly influenced by it. In the rest of this paper, for every model, we build two Stellar trees to compare how their performances depend on parameter k_V. These two k_V values are chosen in order to: (i) have a hierarchical decomposition that still plays a critical role in the storage requirements and time performances; and (ii) obtain trees with different characteristics: one deeper and another relatively shallower. In the following, we use k_S to refer to the smaller k_V value and k_L to the larger one. Since there is a direct correlation between the decomposition quality and χ, these calibration choices are also reflected in the χ values across our experimental datasets. Table 2 summarizes statistics on the Stellar trees obtained from each input data set by considering two values of the vertex threshold k_V, namely k_S and k_L. Fig. 13 illustrates the k_S octree decomposition for the 4M triangle NEPTUNE dataset.

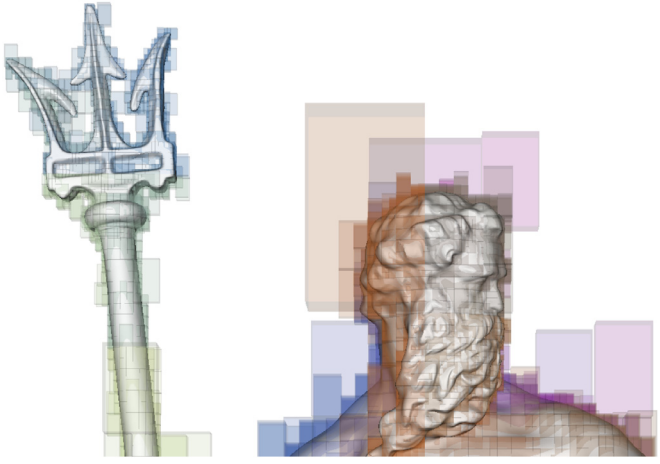

Fig. 13. Leaf blocks for a Stellar tree decompositions over NEPTUNE triangle mesh. Each leaf block indexes up to $k_V = 100$ mesh vertices.

8. Evaluation of storage costs and generation times

In this section, we evaluate the storage costs and generation times of Stellar trees. First, we compare the cost of different Stellar tree encodings (Section 8.1), then we compare the Stellar tree against several state-of-the-art topological mesh data structures (Section 8.2), and, finally, we evaluate the generation times of the Stellar tree (Section 8.3).

Table 2
Overview of our generated Stellar trees for each dataset. For each Stellar tree, we list the thresholds k_V, the number of blocks in the index (total $|\mathbb{H}|$ and leaf $|\mathbb{H}_L|$) and the average spanning number χ.

Data		k_V	$\lvert\mathbb{H}\rvert$	$\lvert\mathbb{H}_L\rvert$	χ
TRIANGULAR					
NEPTUNE	k_S	100	73.7K	58.8K	1.37
	k_L	500	15.0K	12.2K	1.17
STATUETTE	k_S	100	182K	147K	1.36
	k_L	500	39.8K	32.7K	1.17
LUCY	k_S	100	464K	374K	1.35
	k_L	500	88.8K	70.3K	1.16
QUADRILATERAL					
NEPTUNE	k_S	100	407K	322K	1.47
	k_L	800	55.0K	44.3K	1.17
STATUETTE	k_S	100	1.10M	883K	1.47
	k_L	800	146K	120K	1.17
LUCY	k_S	100	3.53M	2.85M	1.54
	k_L	800	329K	265K	1.17
TETRAHEDRAL					
BONSAI	k_S	400	45.2K	39.5K	1.58
	k_L	800	17.9K	15.7K	1.44
VISMALE	k_S	400	32.8K	28.7K	1.52
	k_L	800	17.7K	15.5K	1.45
FOOT	k_S	400	88.8K	77.7K	1.75
	k_L	800	17.1K	15.0K	1.43
HEXAHEDRAL					
F16	k_S	100	1.11M	972K	3.08
	k_L	1000	113K	99.0K	1.90
SAN FERN	k_S	100	2.02M	1.77M	3.15
	k_L	1000	247K	216K	1.88
VISMALE	k_S	100	7.39M	6.46M	2.80
	k_L	1000	800K	700K	1.72
PROBABILISTIC					
5D	k_S	100	37.4K	36.1K	4.39
	k_L	500	2.79K	2.68K	2.55
7D	k_S	100	10.8K	4.87K	4.98
	k_L	500	2.02K	1.00K	3.78
40D	k_S	100	15.2K	4.32K	36.2
	k_L	1000	1.56K	550	34.0
V-RIPS					
VISMALE 7D	k_S	400	32.8K	28.7K	1.44
	k_L	800	17.7K	15.5K	1.37
FOOT 10D	k_S	400	88.8K	77.7K	2.02
	k_L	800	17.1K	15.0K	1.56
LUCY 34D	k_S	100	464K	374K	2.47
	k_L	500	88.8K	70.3K	1.73

8.1. Storage comparison among stellar tree encodings

We begin by comparing the EXPLICIT and COMPRESSED Stellar tree encodings as well as a VERTEX-COMPRESSED encoding, similar to the PR-star encoding for tetrahedral meshes [41], that compresses the vertex array but not the top cells arrays. Table 3 lists the storage costs for the indexed representation of the complex ('Base Complex') as well as the additional costs required for the three Stellar tree encodings, in terms of megabytes (*MBs*). In the following, we assume that pointers require 64 bits and indices 32 bits, the de-facto standard in modern computing hardware. Stellar trees based on the COMPRESSED encoding are always the most compact.

We first consider the storage requirements of the hierarchical structures with respect to our tuning parameter k_V and observe that higher values of k_V always yield reductions in memory requirements. As expected, this effect is more pronounced for the COMPRESSED encoding than for the other two encodings. Specifically, the EXPLICIT and VERTEX-COMPRESSED k_L trees achieve a 20–50% reduction in storage requirements compared to their k_S counterparts, while the COMPRESSED k_L trees are a factor of 3–10 smaller than their k_S counterparts. For example, on the triangular NEPTUNE dataset, storage requirements for the EXPLICIT Stellar tree reduces from 32.0 MB (k_S) to 26.2 MB (k_L), while the COMPRESSED Stellar trees reduces by more than a factor of 4 from 5.76 MB (k_S) to 1.24 MB (k_L).

When comparing the three encodings, we see that compressing the vertices alone, as in the VERTEX-COMPRESSED representation, achieves only 10–20% reduction in storage requirements compared to the EXPLICIT representation, in most cases. In contrast, compressing the vertices and top cells, as in our COMPRESSED representation, yields an order of magnitude improvement, requiring a factor of 10–20 less storage than their EXPLICIT counterparts. This trend is nicely tracked for each dataset by the differences between its average references number μ and its average spanning number χ. This is particularly evident on our *probabilistic* datasets, for which it is difficult to calibrate k_V in order to reduce χ values. However, after SRE compression, μ values are always very small, leading to significant storage reductions in the COMPRESSED representation.

Considering the hierarchical storage requirements against those of the original indexed base complex, we observe that EXPLICIT Stellar trees require about 50% to 80% the storage of the base complex, while COMPRESSED Stellar trees require only around 10% (k_S) and 1% (k_L) the storage of the base complex. Thus, for reasonable k_V values, COMPRESSED Stellar trees impose only a negligible storage overhead with respect to the underlying indexed complex, which the Stellar tree representation does not modify. In the remainder of this paper, we restrict our attention to the COMPRESSED Stellar Tree, which we refer to as the Stellar tree, for simplicity.

8.2. Storage comparison with respect to other data structures

We compare the Stellar tree with several dimension-independent topological data structures as well as dimension-dependent topological data structures for 2D and 3D simplicial complexes. Figs. 14–16 compare the storage requirements for the different data structures normalized against the storage costs of the indexed base complex. The analysis compares the topological overhead of the data structures, and thus, we omit the cost of the geometry of the underlying complex, which is common to all the data structures.

Based on our analysis of the literature (see Section 3.1), the most relevant dimension-independent topological data structures that scale to our experimental datasets are: the Incidence Graph (IG) [8], the Incidence Simplicial (IS) [10], the Simplex tree [28], and the Generalized Indexed data structure with Adjacencies (IA*) [26]. Since Canino et al. [11] demonstrated that the IA* data structure is more compact than the IG and IS data structures for low and high-dimensional datasets, we restrict our comparisons to the IA* and Simplex tree data structures.

The *IA* data structure* has been defined for dimension-independent simplicial complexes, and for our experiments, we have extended it to dimension-independent CP complexes. It explicitly encodes all vertices and top CP k-cells in Σ, with $0 < k \leq d$, as well as the following topological relations:

(i) boundary relation $R_{k,0}(\sigma)$, for each top CP k-cell σ;
(ii) adjacency relation $R_{k,k}(\sigma)$, for each top CP k-cell σ;

Table 3
Storage costs (in *MB*s) and average spanning (χ) and reference (μ) numbers for different Stellar tree encodings.

Data		Base Complex	**Stellar tree**					
			EXPLICIT		V_COMPR.		COMPR.	
			cost	χ	cost	χ	cost	μ
TRIANGULAR								
NEPTUNE	k_S	45.9	32.0	1.37	24.3	1.37	5.76	0.16
	k_L		26.2	1.17	18.6	1.17	1.24	0.04
STATUETTE	k_S	114	79.2	1.36	60.2	1.36	14.6	0.17
	k_L		65.6	1.17	46.6	1.17	3.41	0.04
LUCY	k_S	321	220	1.35	166	1.35	34.5	0.12
	k_L		181	1.16	128	1.16	6.18	0.02
QUADRILATERAL								
NEPTUNE	k_S	183	132	1.47	86.0	1.47	28.0	0.20
	k_L		102	1.17	56.3	1.17	3.86	0.03
STATUETTE	k_S	458	333	1.47	219	1.47	76.0	0.22
	k_L		255	1.17	141	1.17	10.4	0.03
LUCY	k_S	1.3K	976	1.54	656	1.54	245	0.26
	k_L		710	1.17	389	1.17	23.1	0.03
TETRAHEDRAL								
BONSAI	k_S	373	166	1.58	150	1.58	6.55	0.05
	k_L		151	1.44	135	1.44	2.65	0.02
VISMALE	k_S	405	173	1.52	156	1.52	4.87	0.03
	k_L		165	1.45	147	1.45	2.69	0.02
FOOT	k_S	450	220	1.75	201	1.75	13.0	0.08
	k_L		181	1.43	161	1.43	2.60	0.02
HEXAHEDRAL								
F16	k_S	775	456	3.08	349	3.08	151	1.03
	k_L		296	1.90	189	1.90	18.0	0.13
SAN FERN	k_S	1.7K	999	3.15	765	3.15	275	0.86
	k_L		646	1.88	412	1.88	33.1	0.10
VISMALE	k_S	3.8K	2.2K	2.89	1.7K	2.89	887	1.15
	k_L		1.4K	1.72	858	1.72	106	0.15
PROBABILISTIC								
5D	k_S	607	448	4.39	446	4.39	63.7	0.61
	k_L		259	2.55	258	2.55	3.57	0.03
7D	k_S	7.9K	4.9K	4.98	4.9K	4.98	101	0.10
	k_L		3.7K	3.78	3.7K	3.78	12.2	0.01
40D	k_S	2.6K	2.3K	36.2	2.3K	36.2	55.7	0.87
	k_L		2.1K	34.0	2.1K	34.0	0.45	0.01
V-RIPS								
VISMALE 7D	k_S	134	56.2	1.44	37.0	1.44	7.38	0.26
	k_L		53.7	1.37	34.6	1.37	4.54	0.18
FOOT 10D	k_S	2.1K	604	2.02	586	2.02	65.1	0.33
	k_L		431	1.56	413	1.56	11.5	0.12
LUCY 34D	k_S	2.0K	416	2.47	363	2.47	86.2	0.92
	k_L		292	1.73	238	1.73	19.0	0.53

(iii) co-boundary relation $R_{k-1,k}(\tau)$, for each non-manifold $(k-1)$-cell τ bounding a top CP k-cell;
(iv) partial co-boundary relation $R^*_{0,k}(v)$, for each vertex v, consisting of an arbitrary top CP k-cell σ from each *k-cluster* in the star of v. A k-cluster is a $(k-1)$-connected component of the star of v restricted to its top CP k-cells.

Note that for pure CP complexes, the non-manifold co-boundary relation $R_{k-1,k}$ is empty. Further, for pseudo-manifold complexes, the partial vertex co-boundary relation $R^*_{0,k}$ has cardinality 1, and the IA* is identical to the IA data structure [20].

The *Simplex tree* encodes all j-simplices in Σ, with $0 \leq j \leq d$, like the IG, while storing a subset of the incidence relations encoded by the IG. The Simplex tree is defined over a total order on the vertices of Σ, and thus, each simplex σ is uniquely represented as an ordered path in a trie whose nodes correspond to the boundary vertices of σ. Thus, the nodes are in bijection with the simplices of the complex, and a Simplex tree over a simplicial complex with $|\Sigma|$ simplices (of any dimension) contains exactly $|\Sigma|$ nodes. This provides an efficient representation for extracting all boundary relations of simplices in Σ. We compare the Stellar tree to the implementation of the Simplex tree provided in [30], where each node of a Simplex tree requires a reference to the label of the vertex and three references to the tree structure (pointers to the parent node, to the first child and to the next sibling node) for a total of $4|\Sigma|$ references.

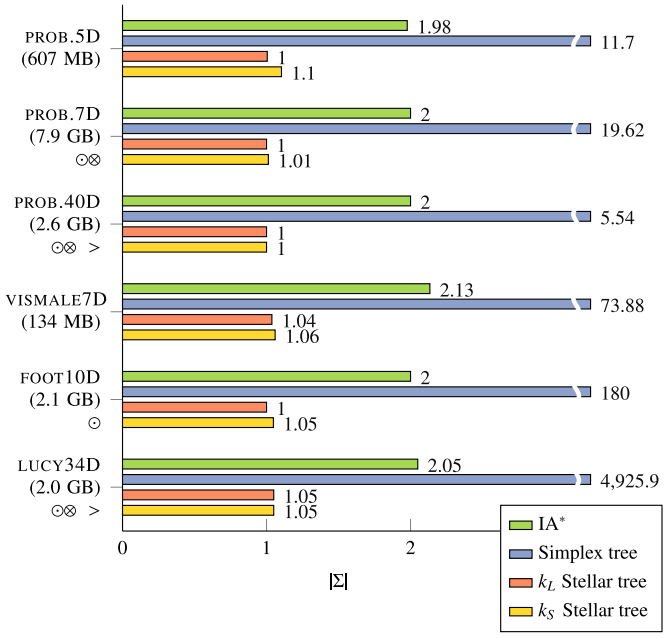

Fig. 14. Storage costs for high dimensional *probabilistic-refinement* simplicial complexes (PROB.5D, PROB.7D and PROB.40D) and *V-Rips* simplicial complexes (VISMALE7D, FOOT10D and LUCY34D). Costs (labels to right of each bar) are normalized to the indexed mesh representation (listed along y-axis). Note that: (1) the x-axis is truncated to a factor of 3; (2) datasets marked with ⊙ or ⊗ could not be directly generated on our test machine for the Simplex tree or IA* (respectively); and (3) the Simplex tree results for the PROB.40D and LUCY34D dataset are partial (>).

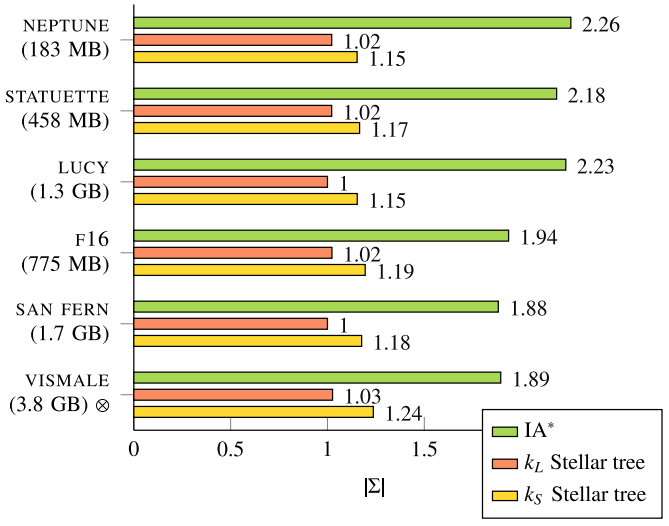

Fig. 15. Storage costs for manifold quadrilateral (NEPTUNE, STATUETTE and LUCY) and hexahedral (BONSAI, VISMALE and FOOT) complexes. Costs (labels to right of each bar) are normalized to the indexed mesh representation (listed along y-axis). Datasets marked with ⊗ could not be directly generated on our test machine using the standalone IA*.

Note that the Stellar tree and our extended IA* data structure can both represent CP complexes in arbitrary dimension and, thus, have the same expressive power, while the Simplex tree can represent only simplicial complexes. Another difference is that Stellar trees require the complex to be embedded in an ambient space $\mathbb{A}$, while the other data structures are purely topological and do not require a spatial embedding. We note, however, that while this is a requirement for Stellar trees, it is not a requirement for the more general Stellar decomposition.

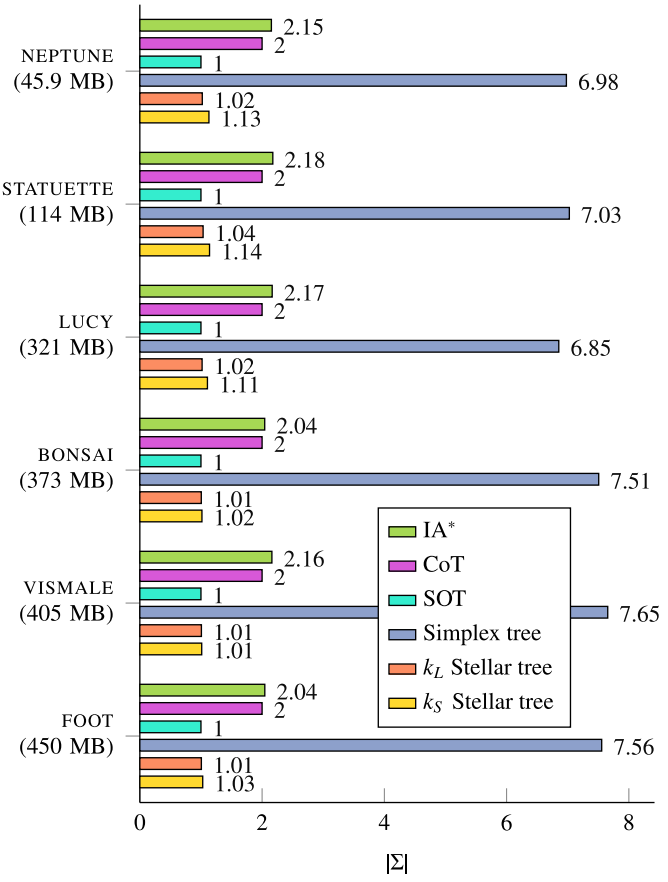

Fig. 16. Storage costs for manifold triangle (NEPTUNE, STATUETTE and LUCY) and tetrahedral (BONSAI, VISMALE and FOOT) complexes. Costs (labels to right of each bar) are normalized to the indexed mesh representation (listed along y-axis).

In terms of storage requirements, the Stellar tree is always more compact than the IA* data structure, requiring approximately half of the storage, nearly all of which is used for encoding boundary relation $R_{k,0}$ for the top cells (i.e., the indexed representation that they share in common). It is worth noting that we were unable to directly generate the IA* data structure for several of our larger datasets on our 64 GB test machine. We generated the IA* data structure on these datasets indirectly using our Stellar tree representation (see Section A.3 in the Supplementary materials) and we have marked these datasets with an ⊗ in Figs. 14 and 15.

When comparing the Stellar tree to the Simplex tree, we observe that the Stellar tree is significantly more compact: by an order of magnitude on manifold and pure models, and by two orders of magnitude or more on non-manifold models. Here too, we were unable to generate Simplex trees for several of the higher dimensional models on our test machine. For these datasets (marked with ⊙ in Fig. 14), we estimated the storage requirements based on the number of simplices of each dimension in the model. On two of these datasets, PROB 40D and LUCY 34D, we were unable to extract all simplices in all dimensions (even indirectly, see Section 9.1), and thus, the storage shown in Fig. 14 is a lower bound of the real storage requirements. In contrast, we had no difficulty generating Stellar Trees for any of our test datasets.

For our dimension-dependent comparisons on manifold simplicial complexes, we also considered the *Corner Table* (*CoT*) [22] and the *Sorted Opposite Table* (*SOT*) [25] data structures, both defined only for manifold triangle and tetrahedral complexes. The *CoT* data structure is similar to the IA data structure and explicitly encodes boundary relation $R_{d,0}(\sigma)$ and adjacency relation $R_{d,d}(\sigma)$ of each

top d-simplex σ. The *SOT* extends the *CoT* by implicitly encoding boundary relation $R_{d,0}(\sigma)$. It only explicitly encodes adjacency relation $R_{d,d}(\sigma)$.

When comparing the Stellar tree to corner-based data structures, we observe that the *CoT* data structure has similar storage requirements as the IA and is roughly twice as large as the Stellar tree, while the *SOT* has similar storage requirements as the Stellar tree, requiring about 1% to 10% less space.

Finally, we consider the effect of different bucketing threshold on the size and efficiency of the Stellar tree representation. For our experimental datasets, there was only about a 10% difference in storage requirements between the large (k_L) and small (k_S) bucketing factors. Clearly, this is not always true, especially in the limit cases, i.e., with $k_V = 1$ and $k_V = \infty$. Very low bucketing thresholds (with k_V near 1) yield deeper trees whose leaf blocks index only a few entities, leading to a high topological overhead but more efficient execution for individual mesh processing operations. Conversely, really large bucketing threshold values lead to lower storage overhead at the expense of increased query and execution times for individual operations. At the limit, when $k_V = \infty$, the Stellar tree is effectively identical to the indexed representation.

These results confirm that the Stellar tree can efficiently represent both low- and high-dimensional complexes with only a slight storage overhead relative to that of the indexed base complex. This is largely due to the Stellar tree's exploitation of the complex's spatial locality via SRE compression.

8.3. Evaluation of stellar tree generation times

In this section, we evaluate the generation times for the Stellar tree. Table 4 shows the timings of the four generation phases and the overall *total* timings. The two *insert* columns show the time for creating the base indexing structure $\mathbb{H}$ over the vertices Σ_V of the complex Σ, or the time for inserting the top cells Σ_T into $\mathbb{H}$, while *reindex* columns show the timings for reordering and SRE compressing the indexed lists and arrays in $\mathbb{H}$ and Σ.

We first consider the relative cost of each of the generation phases. In general, the vertex reindexing phase consumes less than 10% of the overall timings. For the *triangle*, *quadrilateral*, *hexahedral* complexes, and the lower dimensional *Vietoris-Rips* complex, generating $\mathbb{H}$ is the most expensive phase, while for the *tetrahedral*, *probabilistic-refinement* and the two higher dimensional *Vietoris-Rips* models, reindexing the top cells is the most expensive phase. These results can be understood by considering the relative sizes of Σ_V and Σ_T. When the number of vertices is greater than or equal to the number of top cells, it is more expensive to generate the spatial hierarchy $\mathbb{H}$. Otherwise, reindexing and compressing the top cells arrays dominates.

Finally, considering the effect of the bucketing thresholds (k_V) on generation times, we find that Stellar trees with higher bucketing thresholds (k_L) can be generated in less time than those with lower bucketing thresholds (k_S). This is expected since high values of k_V tend to produce coarser spatial subdivisions with lower average spanning numbers χ.

Table 4
Generation timings (in seconds) for the Stellar tree.

Data	k_V	Timings				
		vertices		top CP cells		total
		insert	reindex	insert	reindex	
TRIANGULAR						
NEPTUNE	k_S	4.52	0.68	1.64	3.23	10.1
	k_L	3.83	0.67	1.24	2.77	8.51
STATUETTE	k_S	11.6	1.77	3.42	7.99	24.8
	k_L	10.1	1.74	2.74	6.70	21.3
LUCY	k_S	34.6	1.32	8.85	21.9	66.7
	k_L	30.3	0.48	7.45	18.1	56.3
QUADRILATERAL						
NEPTUNE	k_S	32.2	4.39	6.64	11.3	54.5
	k_L	27.5	4.36	4.63	8.58	45.1
STATUETTE	k_S	82.7	12.3	14.0	29.1	138
	k_L	73.8	12.2	10.7	22.7	119
LUCY	k_S	263	2.17	37.0	61.8	364
	k_L	223	2.02	29.5	35.5	290
TETRAHEDRAL						
BONSAI	k_S	6.69	1.66	7.99	20.8	37.2
	k_L	6.25	1.65	7.12	19.3	34.3
VISMALE	k_S	7.25	1.82	8.35	22.1	39.6
	k_L	6.96	1.81	7.88	21.2	37.8
FOOT	k_S	8.55	2.00	10.8	27.9	49.2
	k_L	7.34	1.97	8.52	23.4	41.2
HEXAHEDRAL						
F16	k_S	103	14.2	77.7	53.9	249
	k_L	94.1	13.9	46.7	35.1	190
SAN FERN	k_S	154	27.6	52.1	102	336
	k_L	140	27.5	37.1	67.8	273
VISMALE	k_S	337	72.8	118	222	751
	k_L	324	71.8	85.3	147	628
PROBABILISTIC						
5D	k_S	0.50	0.58	40.9	53.0	95.0
	k_L	0.37	0.58	20.9	32.7	54.5
7D	k_S	0.55	5.98	332	612	950
	k_L	0.45	5.97	203	471	681
40D	k_S	1.32	1.73	972	769	1.7K
	k_L	1.02	1.73	529	448	980
V-RIPS						
VISMALE 7D	k_S	7.20	1.80	2.65	3.04	13.0
	k_L	6.94	1.81	2.50	2.75	12.3
FOOT 10D	k_S	9.01	1.99	41.0	57.1	108
	k_L	7.91	1.98	30.6	35.9	75.2
LUCY 34D	k_S	35.9	1.63	36.8	42.2	117
	k_L	30.7	0.81	28.7	24.3	84.5

9. Topological queries on a stellar tree

In this section, we describe how to perform topological queries on a CP complex Σ in the Stellar tree representation. These queries are the fundamental building blocks for locally traversing and processing the underlying complex.

Since these queries often depend on *all* cells in the complex, rather than just the explicitly represented top cells, we first describe how we obtain and represent the implicitly encoded boundary cells of the complex from the Stellar tree representation (Section 9.1). We then present algorithms for extracting the coboundary (Section 9.2). For brevity, we omit a description of how to extract adjacency relations, but in the Supplementary materials, we describe how to extract the $R_{d,d}$ adjacency relations to generate the IA* data structure from a Stellar tree.

9.1. Extracting boundary relations

The Stellar tree's underlying indexed representation of a CP complex Σ explicitly encodes only the vertices and top CP k-cells of Σ for $k \leq d$ (see Section 4.2.1). However, many applications require access to non-top cells within the complex. Since they are implicitly encoded within the Stellar tree representation, we must

Algorithm 2 EXTRACT_P_CELLS(p,r,Σ) .

Input: p is the cell dimension to extract
Input: r is a leaf block in $\mathbb{H}$
Input: Σ is the CP complex indexed by $\mathbb{H}$
Variable: m_p maps a p-cell vertex tuple to its local index
Require: Extract boundary p-cells of top k-cells, $0 < p \leq k \leq d$
1: **for all** top CP k-cells σ in $\Phi_{TOP}(r)$ (with index i_σ in Σ_T) **do**
2: **for all** p-faces τ in $R_{k,p}(\sigma)$ (with face index i_τ in σ) **do**
// Rearrange τ's vertices into a canonical order
3: $v\_tuple \leftarrow$ CANONICAL_TUPLE($R_{p,0}(\tau)$)
// If τ is indexed by r*, add it to the local p-faces map*
4: **if** there exists $v \in R_{p,0}(\tau)$ such that $v \in \Phi_{VERT}(r)$ **then**
// Insert τ as a new p-cell, if not already present
5: **if** v_tuple is not in m_p **then**
6: $id_\tau \leftarrow$ SIZE(m_p) *// id_τ is τ's local index in* r
7: $m\_p[v\_tuple] \leftarrow id_\tau$

create a local (explicit) representation to support algorithms for processing and attaching data to such cells.

Our strategy for extracting all p-cells is to iterate through the top k-cells of a leaf block for each dimension k, $0 < p \leq k \leq d$ and to extract an ordered set of p-cells (see Algorithm 2). We use an associative array m_p to track the unique set of encountered p-cells with at least one vertex indexed by r (row 4). Array m_p maps the tuple of vertices for a p-cell τ to an integer index id_τ in the set, accounting for changes in ordering and orientation through the CANONICAL_TUPLE routine (row 3). In some applications, it is useful to also explicitly maintain the boundary relation $R_{p,0}$ for the p-cells and/or the incidence relations $R_{k,p}$ or $R_{p,k}$ for the top k-cells. These are encoded using the local indices within the ordered set of extracted p-cells.

We note that, for truly high-dimensional datasets, it is not feasible to extract p-cells in all cases. For example, there are $\binom{41}{21}$ 20-simplices within a single 40-simplex. Encoding these 269 billion simplices would require more than 40TB of storage. However, even on these datasets, we can still extract the lowest and highest dimensional p-cells. This highlights an advantage of only encoding the top cells of the complex (as in the Stellar tree and IA* data structure) compared to representations that encode all cells of the complex (as in the IG or Simplex tree). Stellar trees have no difficulty encoding and processing such high-dimensional complexes, despite the combinatorial explosion in the number of overall cells.

Experimental results. We now analyze the effectiveness of the Stellar tree representation for (batched) p-cell extractions against our implementation of the IA* data structure and the Simplex tree (as implemented in the *GUDHI* framework [30]). Table 5 lists the aggregate times and storage requirements for extracting all non-top p-cells from our experimental datasets. Notice that we do not consider the higher dimensional *probabilistic* dataset and the LUCY 34D V-Rips complex, as extracting all p-cells on these datasets is unfeasible due to its computational and storage requirements.

First, we analyze the influence of the bucketing threshold k_V for Stellar trees. Smaller k_V values lead to faster extractions on all our experimental datasets. This speedup increases with the dimension of the complex since the auxiliary data structure encoding a p-face type becomes smaller, and thus, checking for the presence of duplicates has a lower computational cost.

The IA* data structure follows a similar strategy as the Stellar trees for extracting its implicit p-cells since both data structures use an indexed representation for encoding the boundary relations of a CP complex. Table 5 demonstrates the computational and storage advantages of the Stellar tree over the IA* for this task. Namely, Stellar trees require from 20% to 55% less time for the two-dimensional datasets and approximately 10% less time on the higher dimensional ones. Notice, however, that the IA* data structure is a global data structure over the entire complex and runs out of memory (OOM) on our *hexahedral* datasets and on the 7D *probabilistic* and FOOT 10D V-Rips datasets. In addition, the Stellar tree's auxiliary storage requirements are negligible compared to those of the IA* data structure.

The Simplex tree explicitly encodes all simplices of a simplicial complex, thus, its p-cells can be enumerated by traversing all simplices at the p-th level of the tree. Explicitly encoding boundary relation $R_{p,0}$ would require the same auxiliary storage as the IA* data structure, since both data structures require global structures. Table 5 demonstrates that Stellar trees are slower than Simplex trees at boundary cell extraction, but, still, competitive with

Table 5
Summed timings (seconds) and additional storage requirements (number of references) to extract boundary p-cells from Stellar tree, IA* and Simplex tree data structures. Datasets marked with an ⊗ could not be directly generated on our test machine by the IA*.

Data	k_V	Time			Storage	
		IA*	Simplex tree	Stellar tree	IA* / Simplex	Stellar tree
TRIANGULAR						
NEPTUNE	k_S	4.93	1.82	1.90	12.0M	0.70K
	k_L			2.20		3.24K
STATUETTE	k_S	9.21	3.73	4.90	30.0M	0.72K
	k_L			5.55		3.22K
LUCY	k_S	25.3	9.94	13.8	84.1M	0.82K
	k_L			16.2		3.28K
QUADRILATERAL						
NEPTUNE	k_S	40.8	n/a	6.61	96.2M	0.52K
	k_L			7.43		3.37K
STATUETTE	k_S	91.3	n/a	15.9	240M	0.50K
	k_L			19.0		3.38K
LUCY	k_S	251	n/a	43.2	673M	0.53K
	k_L			53.4		3.41K
TETRAHEDRAL						
BONSAI	k_S	49.6	22.7	45.6	204M	20.9K
	k_L			47.8		42.5K
VISMALE	k_S	54.5	25.1	52.2	222M	21.4K
	k_L			53.7		36.5K
FOOT	k_S	59.5	29.7	50.9	246M	21.2K
	k_L			57.5		43.3K
HEXAHEDRAL						
F16	k_S	OOM	n/a	49.6	OOM	2.64K
	k_L			71.1		18.9K
SAN FERN	k_S	OOM	n/a	109	OOM	2.89K
	k_L			143		21.1K
VISMALE⊗	k_S	OOM	n/a	263	OOM	1.77K
	k_L			340		17.4K
PROB.						
5D	k_S	456	123	316	970M	152K
	k_L			425		1.94M
7D⊗	k_S	OOM	OOM	21.2K	OOM	51.3M
	k_L			24.6K		167M
V-RIPS						
VISMALE 7D	k_S	179	149	156	1.43B	267K
	k_L			162		318K
FOOT 10D	k_S	OOM	OOM	16.6K	OOM	12.0M
	k_L			21.4K		15.9M

Algorithm 3 EXTRACT_RESTRICTED_VERTEX_CBDRY(r,Σ) .

Input: r is a leaf block in $\mathbb{H}$
Input: Σ is the mesh indexed by $\mathbb{H}$
Variable: r_0_k encodes $R_{0,k}$ relation for the vertices in r
Ensure: Relation $R_{0,k}$ is *locally* reconstructed $\forall\sigma \in \Phi_{VERT}(r)$
1: **for all** top k-simplex σ in $\Phi_{TOP}(r)$ (with index i_σ in Σ_T) **do**
2: **for all** vertices v in σ (with index i_v in Σ_V) **do**
3: **if** $v \in \Phi_{VERT}(r)$ **then**
4: add i_σ to $r\_0\_k[i_v]$

respect to a representation that explicitly encodes all cells. This is possible thanks to the smaller local auxiliary data structures used by Stellar trees. Note that the Simplex tree runs out of memory (OOM) on our workstation for the 7D *probabilistic* dataset and the FOOT 10D V-Rips complex. Since a Simplex tree can only represent simplicial complexes, it does not support p-cell extraction on *quad* and *hexahedral* datasets.

9.2. Extracting co-boundary relations

Co-boundary queries arise in a variety of mesh processing applications, including those requiring mesh simplification and refinement [63–65], or the *dual* of a complex [66–68].

Co-boundary queries are naturally supported by the Stellar decomposition model. By definition, all regions of the decomposition that contain at least one vertex of a CP cell τ must index all CP cells in the star of τ (see Eq. 1). Since the top cells are explicitly represented in Σ, we first describe how to extract the vertex co-boundary relation $R_{0,k}$ restricted to the top k-cells of Σ, which we will refer to as the *restricted co-boundary relation* $R_{0,k}$. We will then discuss how to extend this to extract vertex co-boundary relation $R_{0,p}$ over *all* p-cells in Σ, and the general co-boundary relation $R_{p,q}$ with $0 \le p < q \le d$.

The *restricted vertex co-boundary relation* $R_{0,k}$ in a leaf block r is generated by inverting boundary relation $R_{k,0}$ on the top CP k-cells in $\Phi_{TOP}(r)$. Since the indexed vertices in the leaf blocks of a COMPRESSED Stellar tree are contiguous, with indices in the range $[v_s, v_e)$, we encode our local data structure using an array of size $|\Phi_{VERT}(r)| = v_e - v_s$. Each position in the array corresponds to a vertex indexed by r and points to an (initially empty) list of indexes from Σ_T. As shown in Algorithm 3, we populate these arrays by iterating through relation $R_{k,0}$ of the top CP k-cells in $\Phi_{TOP}(r)$. For each cell σ such that relation $R_{k,0}(\sigma)$ contains a vertex v with index $i_v \in [v_s, v_e)$, the index of σ is added to vertex v's list.

Extending the vertex co-boundary relation to all p-cells in r is complicated by the fact that we only have an explicit representation for the top cells in Σ. A simple strategy we have developed for extracting $R_{0,p}$ on all p-cells in r is to first extract the explicit set of all p-cells in r, as in Algorithm 2 (see Section 9.1). We then invert $R_{p,0}$ to obtain the complete relation $R_{0,p}$ for the vertices in r.

In some applications, we prefer to express $R_{0,p}$ entirely in terms of top cells from Σ. Thus, another strategy we have developed is to extract the restricted co-boundary relation $R_{0,k}$ for all top k-cells in r, with $p \le k \le d$. This redundant representation is thus used as an intermediate representation for $R_{0,p}(v)$ since each k-cell in $R_{0,k}(v)$ contains one (or more) p-face in the co-boundary of v. For example, this provides a convenient representation for the star of a vertex v as a union of restricted co-boundary relations $R_{0,k}(v)$, where $1 \le k \le d$.

Similarly, we have defined and implemented a strategy for generating the general co-boundary relation $R_{p,q}$, where $p < q$. First, the sets of all q-cells, which is expressed as $R_{q,0}$, is extracted. This implicitly provides also boundary relation $R_{q,p}$. Then, co-boundary relation $R_{p,q}$ is extracted by inverting $R_{q,p}$.

Table 6
Times (seconds) and additional storage requirements (number of references) for restricted co-boundary relations $R_{0,k}$ extractions from Stellar tree and IA* representations. Datasets marked with an ⊗ could not be directly generated on our test machine by the IA*.

Data	k_V	Time		Storage
		IA*	Stellar	Stellar
TRIANGULAR				
NEPTUNE	k_S	5.02	0.66	0.61K
	k_L		0.64	3.00K
STATUETTE	k_S	10.2	1.66	0.61K
	k_L		1.58	3.01K
LUCY	k_S	24.8	4.20	0.61K
	k_L		4.17	3.01K
QUADRILATERAL				
NEPTUNE	k_S	27.5	2.86	0.41K
	k_L		2.65	3.21K
STATUETTE	k_S	63.6	7.04	0.41K
	k_L		7.22	3.22K
LUCY	k_S	156	20.4	0.42K
	k_L		19.3	3.22K
TETRAHEDRAL				
BONSAI	k_S	14.5	3.10	9.58K
	k_L		2.81	18.5K
VISMALE	k_S	16.1	3.38	9.57K
	k_L		3.07	18.2K
FOOT	k_S	17.3	3.83	9.62K
	k_L		3.32	18.6K
HEXAHEDRAL				
F16	k_S	145	11.8	0.83K
	k_L		10.8	7.51K
SAN FERN	k_S	157	26.9	0.93K
	k_L		22.0	8.51K
VISMALE⊗	k_S	254	44.5	0.75K
	k_L		47.7	7.54K
HEXAHEDRAL				
PROBABILISTIC	k_S	17.9	4.88	33.0K
	k_L		2.73	243K
7D⊗	k_S	415	46.1	1.62M
	k_L		35.7	9.01M
40D⊗	k_S	206	56.1	2.64M
	k_L		51.4	14.3M
V-RIPS				
VISMALE 7D	k_S	25.8	2.22	3.20K
	k_L		2.16	5.04K
FOOT 10D	k_S	376	19.0	55.7K
	k_L		16.0	72.6K
LUCY 34D⊗	k_S	334	22.9	13.0K
	k_L		23.2	43.8K

Experimental results. We now analyze the effectiveness of the Stellar tree representation for co-boundary extractions. Specifically, since the main co-boundary extraction in our applications (see Section 10) is the restricted vertex co-boundary relation and most of the other co-boundary extractions can be posed in terms of this primitive extraction, we compare the performance of the Stellar tree against our implementation of the IA* data structure for this query and against the Simplex tree. Table 6 lists the extraction

times and storage requirements for the vertex co-boundary relation $R_{0,d}$ on our manifold (*triangular*, *quad*, *tetrahedral* and *hex*) and pure (*probabilistic*) complexes and the sum of extraction times for the restricted vertex co-boundary relations $R_{0,k}$ for each dimension k with top cells on our non-manifold (V-RIPS) complexes.

We first consider the influence of the bucketing threshold k_V for Stellar trees. While there is not much difference in extraction times for the two-dimensional complexes, larger k_V values lead to faster extractions for three-dimensional and non-manifold datasets in most cases. While this comes with a slight increase in storage requirements for encoding the relation (see right column in Table 6), the overall storage cost per block is pretty low, requiring at most a few megabytes for the probabilistic models, and a few kilobytes in all other cases.

The IA* data structure extracts co-boundary relations through a traversal along the face adjacencies of its top cells (encoded in the $R_{k,k}$ adjacency relation). The traversal for a given vertex v is seeded by one top k-cell per k-cluster (encoded by partial relation $R^*_{0,k}(v)$, see Section 8.2; we refer to [26] for more details). Since each such traversal is run on demand, there is a negligible memory impact for this query. Table 6 demonstrates that Stellar trees are significantly faster at extracting $R_{0,k}$ relations, which can be performed in about one tenth of the time in most cases. However, it is important to note that the Stellar tree extraction is batch-based (by leaf blocks of $\mathbb{H}$), and individual co-boundary extractions would likely be faster on the IA* data structure.

The Simplex tree extracts co-boundary relations through a traversal of the underlying trie. Given a vertex v, the procedure for extracting its restricted co-boundary first identifies the simplices incident in v (i.e., its star), and then extracts just the top simplices from the star. The former requires a trie traversal, with a worst-case complexity linear in the number of nodes in the trie, since, as stated in the GUDHI documentation [30], this corresponds to a depth-first search of the trie starting from the node with value v. Identifying the top simplices in the star of a vertex has a negligible cost on low dimensional meshes, while it becomes a costly operation on higher-dimensional ones, where it accounts for nearly 50% of the overall extraction time. As with the IA*, since this traversal is done on demand, this query imposes negligible memory impact. On our experimental datasets, the Simplex tree was able to complete the extraction of restricted vertex co-boundary relations only on the smaller triangle mesh NEPTUNE, for which it requires nearly 72 hours. To provide a comprehensive performance comparison against the Stellar tree, we consider two additional smaller datasets for this query: a tetrahedral mesh (FIGHTER2) with 256 thousand vertices and 1.4 million tetrahedra, and a probabilistic-refinement CP complex with six thousand vertices and two million top 7-simplices. The results, shown in Fig. 17, highlight the Stellar tree's significant advantage over the Simplex tree for restricted vertex co-boundary extraction (i.e., less than a second vs hours).

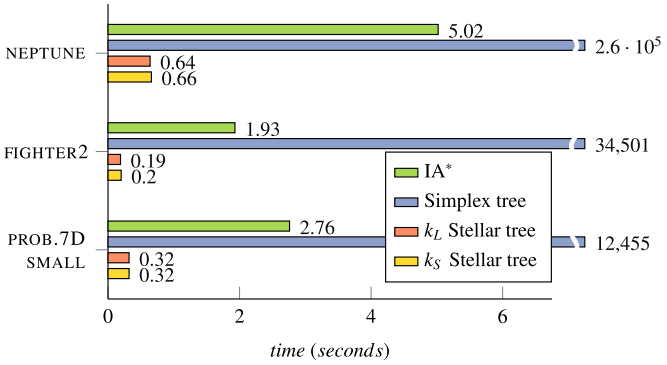

Fig. 17. Extraction times (in seconds) for the restricted vertex co-boundary relations. The top dataset is the triangle mesh used in our main comparison, the second is a tetrahedral mesh with 256 thousand vertices and 1.4 million tetrahedra, and the last dataset is a probabilistic-refinement CP complex with 7-dimensional top simplices.

10. A brief tour of applications in the stellar universe

Stellar decompositions and Stellar trees have been successfully applied in several mesh processing applications. In this section, we provide a high-level overview of several such applications over large CP complexes with a focus on how Stellar trees uniquely benefit the application. As we will describe, each such application utilizes local topological data structures designed for the underlying application. Due to the streamed processing approach discussed in Section 6, the storage requirements for these data structures are proportional to the geometry indexed within a leaf block of the tree and the generation costs are amortized over all processed cells in the block.

10.1. Validation of geometric and topological properties

Many popular topological mesh data structures are valid only for a restricted class of complexes due to assumptions they exploit in their encodings, such as the cardinality of the adjacency relation among top cells. For example, popular edge-based and adjacency-based data structures, such as the *half-edge* [12–14], *Corner-Table* [22], SOT [25] and IA [20,21], require the underlying complex to be pseudo-manifold.

While one can verify such topological conditions using local checks on the *star* or *link* of the vertices of the complex, it can be infeasible to reconstruct such relations on large meshes without the aid of an efficient topological data structure. On the other hand, global approaches that directly build the required relations do not scale to larger complexes.

In contrast, Stellar trees are ideally suited to verify topological properties of large CP complexes, even in memory-limited environments, since each leaf block of the Stellar tree only requires a list of vertices and their incident top cells (i.e., those in the star of the vertices). A simple local topological verification operation was utilized in [41] to mark boundary vertices of a tetrahedral mesh by checking properties of its link, such as its Euler number. This was extended in [88] to a full suite of topological checks on a CP complex, implemented using global Stellar tree traversals. In particular, a graph traversal of the 1-skeleton was used, in conjunction with a *Union-Find* data structure [69], to count the connected components of a pure simplicial complex. A similar traversal of the 1-skeleton of its dual complex (i.e., the graph of the d-adjacency relation) was used to verify the d-connectedness of the complex and whether it is pseudo-manifold. Simplified checks for combinatorial manifolds (when applicable) were then performed on the links of each vertex to check that they were locally homeomorphic to $(d-1)$-spheres (for internal vertices) or to $(d-1)$-balls (for boundary vertices).

10.2. Topology-preserving simplification

One of the earliest applications of the Stellar tree (actually, its predecessor, the PR-star octree [41]) was to accelerate a mesh simplification algorithm for tetrahedral meshes based on *edge collapses*. An edge collapse is a local topological operation defined in terms of the stars of an edge's two vertices. This operation identifies the pair of vertices along an edge, removes all tetrahedra in the star of that edge and updates the mesh connectivity within this local region [70]. Edge collapses are valid when they satisfy the so-called *link condition* [71], which consists of local checks on the links of the edge and its vertices.

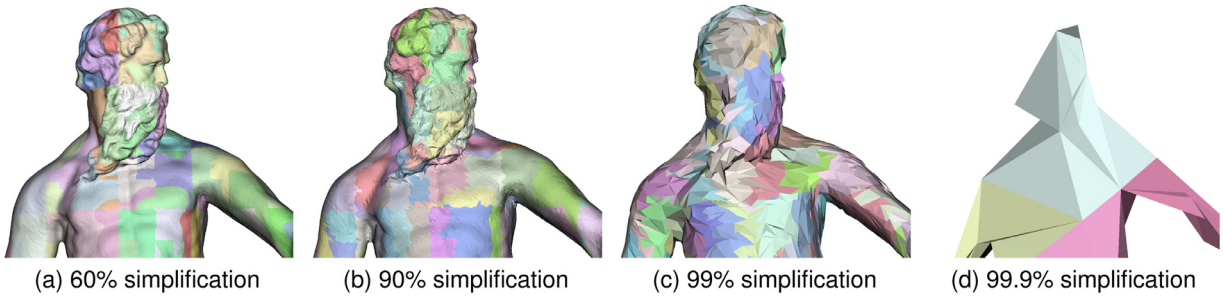

Fig. 18. Homology-preserving edge-contractions retain topological invariants of a complex, such as its number of loops, even after extreme simplification. (a)-(d) Genus-3 NEPTUNE complex at various simplification percentages. Triangles within the same octree block of the Stellar tree have the same color.

The simplification procedure in [41] was implemented as an iterative process that alternated between: (i) streaming through the Stellar tree blocks, where it collapsed eligible candidates, and (ii) rebuilding the Stellar tree's index over the simplified mesh. It used discrete distortion [72] and a quadric error metric [63] to organize eligible edges into a priority queue. Applying this simplification algorithm to the leaf blocks of a Stellar tree rather than to the entire mesh provides a significant space savings due to the reduced sizes of the edge queues. In many cases, the simplification was 10–50% faster and required only 0.1% of the additional memory for auxiliary data structures. Moreover, this speedup was more pronounced as the mesh size increased.

More recently, Stellar trees have been used to perform *homology-preserving* edge-contractions on general simplicial complexes of arbitrary dimension [90]. The core edge contraction algorithm was implemented using a custom local topological data structure over the *top star* of the vertices and edges within each leaf block of the tree, similar to the *restricted co-boundary relations* $R_{0,k}$ and $R_{1,k}$ (c.f. Section 9.2). To avoid regenerating these topological relations, it utilized an LRU cache for the expanded leaf blocks as it traversed the tree. In this mesh simplification application, Stellar trees were applied to datasets with dimensions up to 70 and were able to remove more than 90% of the simplices of the mesh, significantly reducing the dimensionality of the complex while preserving important topological invariants of the dataset. Compared to existing state-of-the-art data structures for edge contractions [32], Stellar trees were able to simplify complexes using comparable or less runtime and memory in all cases, and requiring significantly less memory and/or processing time in several cases. Notably, in one case, the Stellar tree was able to successfully complete the simplification process in less than 30 minutes, while [32] did not complete after more than 24 hours. Fig. 18 shows simplified versions of the genus-3 NEPTUNE triangle mesh. Each simplified mesh preserves the homology of the complex, such as its number of connected components, loops and cavities. We note that the above application incorporates only topological considerations into its simplification error metric. Incorporating mesh quality considerations into the error metric could significantly improve the mesh quality [63].

10.3. Shape analysis and morphological feature extraction

While topological validation and simplification operations can be implemented in terms of local operations on the star of a vertex, shape analysis applications, such as watershed analysis [73] and visibility queries on terrain datasets [74,75] often require algorithms that are seeded locally and span vast interwoven regions of the complex. This section discusses how Stellar trees have aided in the generation and simplification of the *discrete Morse gradient field* and of the associated *Morse* and *Morse-Smale* complexes of triangulated terrains [91] and of tetrahedralized volumetric data [68].

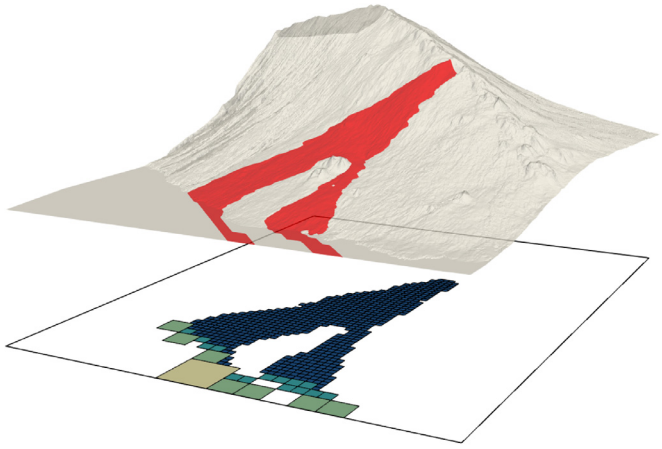

Fig. 19. Manifolds of a Morse complex can span vast regions of the domain. This subset of the 4M triangle MAUI terrain dataset highlights the 73K triangles of a single 2-manifold of the Morse complex (shown in red on top) along with the Stellar tree blocks indexing these triangles (rectangles on bottom, colored by quadtree depth). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The discrete Morse gradient field [76] is composed of *arrows* (ordered pairs) between incident cells of the complex and can be computed locally using scalar values associated with cells incident in the star of a vertex [77]. Since the encoding of [68] compactly encodes the discrete Morse gradient field as a scalar field on the top simplices of the complex, the latter can be computed using a local traversal of a Stellar tree's blocks. Compared to an IA implementation, Stellar trees were able to extract the discrete Morse gradient of scalar fields defined over tetrahedral meshes in about half the time (see [68] for details). While the experiments in [68] were performed on VERTEX-COMPRESSED Stellar trees, yielding a 30% storage savings over the IA data structure, a COMPRESSED Stellar tree encoding would likely provide a 50% total memory savings while maintaining similar performance improvements.

Extracting the Morse complex from a Stellar tree-based encoding is more complicated since it involves traversing the directed acyclic graphs (DAGs) induced by the discrete Morse gradient field's arrows. Specifically, in the encoding of [68], each k-dimensional critical point of the discrete Morse gradient field corresponds to a k-cell of the d-dimensional Morse complex. The disjoint regions of influence of each such critical point, referred to as the *k-manifolds* of the Morse complex, are extracted by

traversing the DAG rooted at a given critical point of the discrete Morse gradient field. Since each such graph traversal can visit the blocks of a Stellar tree multiple times, an LRU cache was used in [68] to support global extraction algorithms for each k-manifold of the Morse complex.

Further, since the extraction algorithm for each dimension's manifolds depends on different topological connectivity relations, Morse complex extraction benefits from the Stellar tree's ability to generate customized local topological data structures. For example, since extracting the 2-manifolds from a tetrahedral complex requires only the $R_{2,2}$ adjacency relation, its extraction was optimized by directly starting from $R_{2,2}$, rather than the $R_{3,3}$ adjacency relation, as in the IA data structure.

This approach was extended to terrain datasets in [91], which also introduced a persistence-based simplification algorithm for noise removal. Fig. 19 highlights the 73K triangles in the largest 2-manifold of the Morse complex for the Maui terrain dataset (in red) and the blocks of the Stellar tree that were visited when extracting this region (blue-green squares). While manifold extraction and persistence-based simplification operations were slightly more expensive than their IA counterparts for volumetric [68] and terrain [91] datasets, the Stellar tree's vast memory savings and hierarchical encoding open the door to efficient parallel implementations on huge datasets, which we hope to explore in future work.

11. Concluding remarks

We have introduced the Stellar decomposition as a model for topological data structures over *Canonical-Polytope (CP)* complexes, a class of complexes that includes simplicial complexes and certain classes of cell complexes, like quadrilateral and hexahedral meshes. Stellar decompositions cluster the vertices of a complex into *regions* that contain sufficient information to locally reconstruct the *star* of their vertices. The model is agnostic about the domain of the complex (e.g., manifold, pure, non-manifold) and we have demonstrated the scalability of this model to large mixed-dimensional datasets in high dimension.

We introduced the Stellar tree as a concrete realization of the Stellar decomposition model over spatially embedded CP complexes. Stellar trees couple a spatial index $\mathbb{H}$ decomposing the complex's embedding space with a simple tuning parameter that limits the number of vertices indexed by a leaf block.

Stellar trees effectively exploit the spatial coherence of a CP complex Σ by using the clustering structure of $\mathbb{H}$ to reorder the arrays of top cells of Σ and to compress the resulting ranges of sequential indexes within the lists of vertices and top cells in the leaf blocks of $\mathbb{H}$. We have demonstrated over a wide range of datasets that this process typically produces compressed Stellar trees that are only 1–10% larger than the original indexed base mesh for Σ while still retaining sufficient information to efficiently reconstruct all topological connectivity relations. The source code for our Stellar tree implementation will be released in the public domain.

In terms of storage size, Stellar trees compare quite favorably with state-of-the-art topological data structures. They are consistently half the size of their IA* data structure counterparts [26] and one to two orders of magnitude smaller than their Simplex tree counterparts [28]. This is especially notable for high dimensional Vietoris-Rips complexes, a target application for the Simplex tree, for which Stellar trees have very low overhead. While Stellar trees support a much broader class of complexes, they have similar storage requirements as the dimension-specific SOT data structure [25,78], which supports only static pseudo-manifold triangle and tetrahedral complexes. In future work, it would be interesting to compare the Stellar tree against top-based extensions of the Simplex tree, such as the *MST* and the *SAL* [31], if public-domain implementations become available.

Despite the simplicity of their leaf block representation, Stellar trees provide a great deal of flexibility to customize the structure and layout of their *expanded* topological data structures to meet the needs of a given application. Such data structures are typically constructed by composing several local topological incidence and adjacency relations. We described efficient algorithms for reconstructing these relations within the subcomplex indexed by the leaf blocks of a Stellar tree and demonstrated the advantages of this approach compared to similar algorithms on the IA* and Simplex tree data structures. Stellar trees can also be used as an intermediary representation to generate topological data structures in a memory-constrained environment. For example, we used Stellar trees to generate IA* and Simplex tree representations for several of our larger complexes in Section 8 (as we discuss in the Supplementary materials). We also provided an overview of several mesh processing applications, ranging from mesh validation, to topology and shape preserving simplification and morphological analysis that have benefited from the Stellar trees representation.

One direction of future work would involve extending the Stellar tree representation to support a broader class of cell complexes. For example, it would not be difficult to extend support to indexed polyhedral cell complexes which define their cells in terms of their boundary polyhedral faces which are, in turn, defined by oriented lists of vertex indices [79].

Another avenue for investigation is to extend our processing algorithms for parallel, distributed and/or out-of-core environments, which could be used for applications like multicore homology computation [80] on point cloud data. The Stellar tree's compact leaf block representation is already geared towards a parallel execution pattern since each block already has sufficient resources to query the connectivity of its local subcomplex. Preliminary results along this line look promising. A simple unoptimized OpenMP [81] adaptation of boundary and restricted vertex coboundary queries yielded a 3-4x speedup compared to our serial approach on our 6 core machine.

Finally, while Stellar trees require their underlying complex to be spatially embedded, there is no such restriction on the Stellar decomposition model. Thus, we plan to investigate Stellar decompositions for *abstract CP* complexes, such as simplicial complexes representing social networks. Social network representation and processing poses new challenges in the social big data domain, such as the identification of key-players and *communities* in the dataset, as well as extracting topological properties of the network, like its homology or k-connectivity. Due to the irregularities of non-spatial datasets, one key challenge would be to define efficient decompositions (i.e., with a low average spanning number χ) using only the complex's connectivity information. A preliminary attempt for geolocalized social networks can be found in [92], where the social network was represented in terms of its *maximal cliques*, i.e., sets of mutually related entities, corresponding to top simplices over the network's flag complex. The Stellar tree was built over the 2D embedding provided by the geospatial locations of the entities and simplified using homology-preserving edge-contractions [90], enabling a study of the network's topological structure on a significantly reduced dataset.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Riccardo Fellegara: Conceptualization, Software, Validation, Investigation, Writing – original draft, Visualization. **Kenneth Weiss:** Conceptualization, Software, Validation, Investigation, Writing – original draft, Visualization. **Leila De Floriani:** Conceptualization, Supervision, Funding acquisition, Methodology, Writing – original draft.

Acknowledgments

We thank the reviewers for their thoughtful comments and thorough review which have improved the paper. This work has been developed while Riccardo Fellegara was with University of Maryland at College Park, USA. This work has been partially supported by the US National Science Foundation under grant number IIS-1910766 and by the University of Maryland under the 2017-2018 BSOS Dean Research Initiative Program. It has also been performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, and of the German Aerospace Center (DLR) under Grant DLR-SC-2467209. Datasets are courtesy of the *Volvis* repository (BONSAI, F16 and FOOT) [82], the *Volume Library* (VISMALE) [83], *CMU Unstructured Mesh Suite* (SAN FERNANDO) [84], *Aim@Shape* repository (LUCY, STATUETTE and NEPTUNE) [85], *Virtual Terrain Project* (*VTP*) (MAUI) [86], and Claudio Silva (FIGHTER2).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cag.2021.05.002.

References

[1] Schoof LA, Yarberry VR. EXODUS II: A finite element data model. Tech. Rep.. Albuquerque, NM: Sandia National Laboratories; 1994.
[2] Poirier D, Allmaras S, McCarthy D, Smith M, Enomoto F. The CGNS system. AIAA Fluid Dynamics Conference 1998. doi:10.2514/6.1998-3007.
[3] Remacle JF, Shephard MS. An algorithm oriented mesh database. Int J Numer Methods Eng 2003;58(2):349–74. doi:10.1002/nme.774.
[4] Celes W, Paulino GH, Espinha R. Efficient handling of implicit entities in reduced mesh representations. J Comput Inf Sci Eng 2005;5(4):348–59.
[5] Tautges TJ. Canonical numbering systems for finite-element codes. Int J Numer Method Biomed Eng 2010;26(12):1559–72. doi:10.1002/cnm.1237.
[6] De Floriani L, Hui A. Data structures for simplicial complexes: An analysis and a comparison. In: Proceedings of the third Eurographics symposium on Geometry processing. Eurographics Association; 2005. 119–es.
[7] Nguyen TT, Dahl VA, Bærentzen JA. Cache-mesh, a dynamics data structure for performance optimization. Procedia Eng 2017;203:193–205. doi:10.1016/j.proeng.2017.09.807.
[8] Edelsbrunner H. Algorithms in combinatorial geometry, vol 10. Springer Verlag; 1987.
[9] De Floriani L, Greenfieldboyce D, Hui A. A data structure for non-manifold simplicial d-complexes. In: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing. ACM; 2004. p. 83–92.
[10] De Floriani L, Hui A, Panozzo D, Canino D. A dimension-independent data structure for simplicial complexes. Proceedings of the 19th International Meshing Roundtable 2010a:403–20.
[11] Canino D, De Floriani L. Representing simplicial complexes with mangroves. In: Proceedings of the 22nd International Meshing Roundtable. Springer; 2014. p. 465–83.
[12] Mantyla M. An introduction to solid modeling. Computer Science Press; 1988.
[13] CGAL. Computational Geometry Algorithms Library (CGAL). 2020. https://www.cgal.org/ [Accessed on August 2020].
[14] OML. Openmesh library. 2015. http://www.openmesh.org [Accessed on March 2015].
[15] Sieger D, Botsch M. Design, implementation, and evaluation of the surface_mesh data structure. In: Quadros WR, editor. Proceedings of the 20th International Meshing Roundtable. Springer; 2011. p. 533–50. doi:10.1007/978-3-642-24734-7_29.
[16] Kremer M, Bommes D, Kobbelt L. OpenVolumeMesh–a versatile index-based data structure for 3d polytopal complexes. In: Jiao X, Weill JC, editors. Proceedings of the 21st International Meshing Roundtable. Springer; 2013. p. 531–48. doi:10.1007/978-3-642-33573-0_31.
[17] Lienhardt P. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. Int'l Journal of Computational Geometry and Applications 1994;4(3):275–324.
[18] Damiand G, Lienhardt P. Combinatorial maps: efficient data structures for computer graphics and image processing. CRC Press; 2014. ISBN 978-1482206524.
[19] Lawson CL. Software for C^1 surface interpolation. In: Rice JR, editor. Mathematical Software III. Academic Press; 1977. p. 161–94.
[20] Paoluzzi A, Bernardini F, Cattani C, Ferrucci V. Dimension-independent modeling with simplicial complexes. ACM Transactions on Graphics (TOG) 1993;12(1):56–102.
[21] Nielson GM. Tools for triangulations and tetrahedralizations and constructing functions defined over them. In: Nielson GM, Hagen H, Müller H, editors. Scientific Visualization: Overviews, Methodologies and Techniques. Silver Spring, MD: IEEE Computer Society; 1997. p. 429–525.
[22] Rossignac J, Safonova A, Szymczak A. 3D compression made simple: Edge-Breaker on a Corner Table. In: Proceedings Shape Modeling International 2001. Genova, Italy: IEEE Computer Society; 2001.
[23] Gurung T, Laney D, Lindstrom P, Rossignac J. SQuad: Compact representation for triangle meshes. In: Computer Graphics Forum, vol. 30. Wiley Online Library; 2011. p. 355–64. doi:10.1111/j.1467-8659.2011.01866.x.
[24] Luffel M, Gurung T, Lindstrom P, Rossignac J. Grouper: a compact, streamable triangle mesh data structure. IEEE Trans Vis Comput Graph 2014;20(1):84–98. doi:10.1109/TVCG.2013.81.
[25] Gurung T, Rossignac J. SOT: A compact representation for tetrahedral meshes. In: Proceedings SIAM/ACM Geometric and Physical Modeling. San Francisco, USA; 2009. p. 79–88. doi:10.1145/1629255.1629266.
[26] Canino D, De Floriani L, Weiss K. IA*: An adjacency-based representation for non-manifold simplicial shapes in arbitrary dimensions. Computers & Graphics 2011;35(3):747–53.
[27] Dyedov V, Ray N, Einstein D, Jiao X, Tautges TJ. AHF: Array-based half-facet data structure for mixed-dimensional and non-manifold meshes. Eng Comput 2015;31(3):389–404. doi:10.1007/s00366-014-0378-6.
[28] Boissonnat J-D, Maria C. The simplex tree: an efficient data structure for general simplicial complexes. Algorithmica 2014;70(3):406–27.
[29] Fredkin E. Trie memory. Commun ACM 1960;3(9):490–9.
[30] GUDHI. Geometric understanding in higher dimensions (GUDHI). 2018. http://gudhi.gforge.inria.fr/.
[31] Boissonnat J-D, Karthik CS, Tavenas S. Building efficient and compact data structures for simplicial complexes. Algorithmica 2017;79(2):530–67. doi:10.1007/s00453-016-0207-y.
[32] Attali D, Lieutier A, Salinas D. Efficient data structure for representing and simplifying simplicial complexes in high dimensions. International Journal of Computational Geometry & Applications 2012;22(04):279–303.
[33] Samet H. Foundations of multidimensional and metric data structures. Morgan Kaufmann; 2006. ISBN 978-0-12-369446-1.
[34] Samet H, Webber RE. Storing a collection of polygons using quadtrees. ACM Transactions on Graphics (TOG) 1985;4(3):182–222.
[35] De Floriani L, Facinoli M, Magillo P, Dimitri D. A hierarchical spatial index for triangulated surfaces. In: Proceedings of the Third International Conference on Computer Graphics Theory and Applications (GRAPP 2008); 2008. p. 86–91.
[36] Carlbom I, Chakravarty I, Vanderschel D. A hierarchical data structure for representing the spatial decomposition of 3d objects. IEEE Comput Graph Appl 1985;5(4):24–31.
[37] Navazo I. Extended octree representation of general solids with plane faces: model structure and algorithms. Computer & Graphics 1989;13(1):5–16.
[38] Fellegara R, Iuricich F, Floriani D. Efficient representation and analysis of triangulated terrains. In: Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM; 2017.
[39] De Floriani L, Fellegara R, Magillo P. Spatial indexing on tetrahedral meshes. In: Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM; 2010b. p. 506–9.
[40] Fellegara R, De Floriani L, Magillo P, Weiss K. Tetrahedral trees: a family of hierarchical spatial indexes for tetrahedral meshes. ACM Transactions on Spatial Algorithms and Systems 2020;6(4) 23:1–23:34. doi:10.1145/3385851.
[41] Weiss K, Fellegara R, De Floriani L, Velloso M. The PR-star octree: A spatio–topological data structure for tetrahedral meshes. In: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM; 2011. p. 92–101.
[42] Isenburg M, Lindstrom P. Streaming meshes. In: Proceedings IEEE Visualization. IEEE; 2005. p. 231–8. doi:10.1109/VISUAL.2005.1532800.
[43] Yoon SE, Lindstrom P, Pascucci V, Manocha D. Cache-oblivious mesh layouts. In: ACM Transactions on Graphics (TOG). In: Siggraph, vol. 24. ACM; 2005. p. 886–93. doi:10.1145/1073204.1073278.
[44] Yoon SE, Lindstrom P. Random-accessible compressed triangle meshes. IEEE Trans Vis Comput Graph 2007;13(6):1536–43. doi:10.1109/TVCG.2007.70585.
[45] Cignoni P, Montani C, Rocchini C, Scopigno R. External memory management and simplification of huge meshes. IEEE Trans Vis Comput Graph 2003a;9(4):525–37. doi:10.1109/TVCG.2003.1260746.
[46] Dey TK, Levine JA, Slatton A. Localized delaunay refinement for sampling and meshing. Comput Graphics Forum 2010;29(5):1723–32. doi:10.1111/j.1467-8659.2010.01781.x.
[47] Cignoni P, Ganovelli F, Gobbetti E, Marton F, Ponchio F, Scopigno R, et al. BDAM – Batched Dynamic Adaptive Meshes for high performance terrain visualization. Comput Graphics Forum 2003b;22(3):505–14.
[48] Cignoni P, Ganovelli F, Gobbetti E, Marton F, Ponchio F, Scopigno R. Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. ACM Trans Graph 2004a;23(3):796–803. doi:10.1145/1015706.1015802.

[49] Weiss K, De Floriani L. Simplex and diamond hierarchies: models and applications. Comput Graphics Forum 2011;30(8):2127–55. doi:10.1111/j.1467-8659.2011.01853.x.
[50] Devine K, Diachin L, Kraftcheck J, Jansen KE, Leung V, Luo X, Miller M, Ollivier-Gooch C, Ovcharenko A, Sahni O, Shephard MS, Tautges T, Xie T, Zhou M. Interoperable mesh components for large-scale, distributed-memory simulations. In: Journal of Physics: Conference Series, vol. 180. IOP Publishing; 2009.
[51] Ibanez DA, Seol ES, Smith CW, Shephard MS. PUMI: parallel unstructured mesh infrastructure. ACM Transactions on Mathematical Software (TOMS) 2016;42(3) 17:1–17:28. doi:10.1145/2814935.
[52] Anderson R, Andrej J, Barker A, Bramwell J, Camier J-S, Cerveny J, Dobrev V, Dudouit Y, Fisher A, Kolev T, et al. MFEM: A modular finite element methods library. Computers & Mathematics with Applications 2021;81(1):42–74. doi:10.1016/j.camwa.2020.06.009.
[53] Kirk BS, Peterson JW, Stogner RH, Carey GF. LibMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations. Eng Comput 2006;22(3–4):237–54. doi:10.1007/s00366-006-0049-3.
[54] Edwards HC, Williams AB, Sjaardema GD, Baur DG, Cochran WK. SIERRA toolkit computational mesh conceptual model. Tech. Rep.. Sandia National Laboratories; 2010.
[55] Poirier D, Bush R, Cosner R, Rumsey C, McCarthy D. Advances in the CGNS database standard for aerodynamics and CFD. In: 38th Aerospace Sciences Meeting and Exhibit; 2000. p. 681.
[56] Lawlor OS, Chakravorty S, Wilmarth TL, Choudhury N, Dooley I, Zheng G, Kalé LV. ParFUM: a parallel framework for unstructured meshes for scalable dynamic physics applications. Eng Comput 2006;22(3–4):215–35. doi:10.1007/s00366-006-0039-5.
[57] Ollivier-Gooch C, Diachin L, Shephard MS, Tautges T, Kraftcheck J, Leung V, Luo X, Miller M. An interoperable, data-structure-neutral component for mesh query and manipulation. ACM Transactions on Mathematical Software (TOMS) 2010;37(3) 29:1–29:28. doi:10.1145/1824801.1864430.
[58] Held G, Marshall T. Data compression; techniques and applications: hardware and software considerations. John Wiley & Sons; 1991.
[59] Bentley JL. Multidimensional binary search trees used for associative searching. Commun ACM 1975;18(9):509–17.
[60] Bellman R. Dynamic programming. Science 1966;153(3731):34–7.
[61] Hunter A, Willis P. Classification of quad-encoding techniques. In: Computer Graphics Forum, vol. 10; 1991. p. 97–112. doi:10.1111/1467-8659.1020097.
[62] Zomorodian A. Fast construction of the vietoris-rips complex. Computers & Graphics 2010;34(3):263–71. doi:10.1016/j.cag.2010.03.007.
[63] Garland M, Heckbert PS. Surface simplification using quadric error metrics. In: Proceedings of the 24th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co.; 1997. p. 209–16.
[64] Natarajan V, Edelsbrunner H. Simplification of three-dimensional density maps. Visualization and Computer Graphics, IEEE Transactions on 2004;10(5):587–97.
[65] Zorin D. Subdivision zoo. Subdivision for Modeling and Animation (ACM SIGGRAPH 2000 Conference Course Notes) 2000:65–102.
[66] Hirani AN. Discrete exterior calculus. California Institute of Technology; 2003.
[67] Mullen P, Memari P, de Goes F, Desbrun M. HOT: Hodge-optimized triangulations. In: ACM Transactions on Graphics (TOG), vol. 30. ACM; 2011. 103:1–103:12.
[68] Weiss K, Iuricich F, Fellegara R, De Floriani L. A primal/dual representation for discrete Morse complexes on tetrahedral meshes. In: Computer Graphics Forum, vol. 32; 2013. p. 361–70.
[69] Tarjan RE. Efficiency of a good but not linear set union algorithm. J ACM 1975;22(2):215–25. doi:10.1145/321879.321884.
[70] Cignoni P, De Floriani L, Magillo P, Puppo E, Scopigno R. Selective refinement queries for volume visualization of unstructured tetrahedral meshes. IEEE Trans Vis Comput Graph 2004b;10(1):29–45.
[71] Dey T, Edelsbrunner H, Guha S, Nekhayev D. Topology preserving edge contraction. Publications de l'Institut Mathematique (Beograd) 1999;60(80):23–45.
[72] Mesmoudi MM, De Floriani L, Port U. Discrete distortion in triangulated 3-manifolds. Comput Graphics Forum 2008;27(5):1333–40. doi:10.1111/j.1467-8659.2008.01272.x.
[73] Roerdink J, Meijster A. The watershed transform: definitions, algorithms, and parallelization strategies. Fundamental Informaticae 2000;41:187–228.
[74] Bittner J, Wonka P. Visibility in computer graphics. Environment and Planning B: Planning and Design 2003;30(5):729–55.
[75] De Floriani L, Magillo P. Algorithms for visibility computation on terrains: a survey. Environment and Planning B 2003;30(5):709–28.
[76] Forman R. Morse theory for cell complexes. Adv Math (N Y) 1998;134:90–145.
[77] Robins V, Wood PJ, Sheppard AP. Theory and algorithms for constructing discrete Morse complexes from grayscale digital images. IEEE Trans Pattern Anal Mach Intell 2011;33(8):1646–58. doi:10.1109/TPAMI.2011.95.
[78] Gurung T, Rossignac J. SOT: Compact representation for triangle and tetrahedral meshes. Tech. Rep.. College of Computing, Georgia Institute of Technology, Atlanta, GA, USA; 2010.
[79] Muigg P, Hadwiger M, Doleisch H, Groller E. Interactive volume visualization of general polyhedral grids. IEEE Trans Vis Comput Graph 2011;17(12):2115–24. doi:10.1109/TVCG.2011.216.
[80] Lewis RH, Zomorodian A. Multicore homology via Mayer Vietoris. arXiv preprint arXiv:14072275 2014.
[81] OpenMP. OpenMP API for parallel programming, version 4.5. 2015. http://openmp.org/wp/.
[82] Avila R., He T., Hong L., Kaufman A., Pfister H., Silva C., Sobierajski L., Wang S.. Volvis library. http://www.volvis.org [Accessed on June 2015]; 2015.
[83] Roettger S.. The Volume Library. http://www9.informatik.uni-erlangen.de/External/vollib[Accessed on June 2015]; 2015.
[84] CMUMeshSuite. CMU Unstructured Mesh Suite. http://www.cs.cmu.edu/~quake/meshsuite.html[Accessed on August 2020]; 1996.
[85] AIM@shape. Aim @ shape repository. http://visionair.ge.imati.cnr.it/ontologies/shapes[Accessed on August 2020]; 2015.
[86] VTerrain. Virtual terrain project. http://vterrain.org/BT [Accessed on August 2020]; 2015.
[87] Fellegara R. Stellar tree code repository. GitHub repository, GitHub. 2021. https://github.com/UMDGeoVis/Stellar_tree.
[88] Fellegara, R. and Weiss, K. and De Floriani, L. An efficient approach for verifying manifold properties of simplicial complexes. In: Canann S. editor. Proceedings of the 25th International Meshing Roundtable. Washington, D.C. 2016, http://imr.sandia.gov/papers/abstracts/Fe830.html.
[89] Fellegara R. Spatial Indexes for Simplicial and Cellular Meshes. New Trends in Databases and Information Systems. Springer International Publishing; 2014. p. 373–82.
[90] Fellegara R, Iuricich F, De Floriani L, Fugacci U. Efficient homology-preserving simplification of high-dimensional simplicial shapes. Computer Graphics Forum 2020;39:244–59, Wiley Online Library.
[91] Fellegara R, Iuricich F, De Floriani L, Weiss K. Efficient computation and simplification of discrete Morse decompositions on triangulated terrains. In: Proceedings of the 22th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM; 2014.
[92] Fellegara R, Fugacci U, Iuricich F, De Floriani L. Proceedings of the 9th ACM sigspatial workshop on location-based social networks. In: Analysis of Geolocalized Social Networks Based on Simplicial Complexes. New York, NY, USA: ACM; 2016. 5:1–5:8.