Regularized arrangements of cellular complexes

ALBERTO PAOLUZZI, Roma Tre University, Rome, Italy
VADIM SHAPIRO, University of Wisconsin-Madison & ICSI, United States
ANTONIO DICARLO, CECAM-IT-SIMUL Node, Rome, Italy

In this paper we propose novel algorithms to combine two or more cellular complexes, providing a minimal fragmentation of the cells of the resulting complex. We introduce here the idea of arrangement generated by a collection of cellular complexes, producing a cellular decomposition of the embedding space. The algorithmic pipeline that executes this computation contains the Merge of complexes, as well as the Topological Gift Wrapping and the computation of Relative Containment between unconnected components. The arrangements of line segments in 2D and polygons in 3D are special cases, as well as the combination of closed triangulated surfaces or meshed models. Of course, our algorithms must be applied to valid input, i.e., such that may generate a partition of space. This approach has several important applications, including Boolean and other set operations over large geometric models, the extraction of solid models of biomedical structures at the cellular scale, the detailed geometric modeling of buildings, the combination of 3D surface or volumetric meshes, and the repair of graphical models. The algorithm is efficiently implemented using the Linear Algebraic Representation (LAR) of argument complexes, i.e., on sparse representation of binary characteristic matrices of d-cell bases, well-suited for implementation in last generation accelerators and GPGPU applications.

CCS Concepts: • **Computing methodologies → Volumetric models**; *Modeling methodologies*;

Additional Key Words and Phrases: Computational topology, Solid modeling, Linear Algebraic Representation, LAR, Arrangements, Cellular Complexes.

ACM Reference format:
Alberto Paoluzzi, Vadim Shapiro, and Antonio DiCarlo. 2018. Regularized arrangements of cellular complexes. 1, 1, Article 1 (April 2018), 36 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Given a finite collection $\mathcal{S}$ of cellular complexes[1] in $\mathbb{E}^d$, $d \in \{2, 3\}$, the arrangement $\mathcal{A}(\mathcal{S})$ is the decomposition of $\mathbb{E}^d$ into connected cells of dimensions $0, 1, \ldots, d$ induced by $\mathcal{S}$.

[1] A precise mathematical definition of a cellular complex is not trivial, and will be given in Section 2.1.1. As for now, the reader may rely on the intuitive idea of constructing a space by gluing together a number of building blocks of different dimensions, called cells.

This work is partially supported from SOGEI S.p.A. — the ICT company of the Italian Ministry of Economy and Finance, by grant 2016-17, and by the ERASMUS+ EU project medtrain3dmodsim. V.S. is supported in part by National Science Foundation grant CMMI-1344205 and National Institute of Standards and Technology.
Authors addresses: A. Paoluzzi, Department of Mathematics and Physics, Roma Tre University; V. Shapiro, University of Wisconsin at Madison, and International Computer Science Institute, Berkeley; A. DiCarlo, CECAM-IT-SIMUL Node.

XXXX-XXXX/2018/4-ART1
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In this paper, we discuss the computation of the arrangement produced by a given set of cellular complexes in either 2D or 3D. Our goal is to provide a complete description of the plane or space decomposition induced by the input, into cells of dimensions 0, 1, 2 or 3.

A planar collection $\mathcal{S}$ may include line segments, open or closed polygonal lines, polygons, two-dimensional meshes, and discrete images in 2D. A space collection may include 3D polygons, polygonal meshes, B-reps of solid models—either manifold or non-manifold, three-dimensional CAE meshes, and volumetric images in 3D.

In discrete geometry, an arrangement is the decomposition of the d-dimensional affine, or projective space into connected and relatively open cells of lower dimensions, induced by an intersection of a finite collection of geometric objects. The result of the computation discussed in this paper is the arrangement $\mathcal{A}(\mathcal{S}) = X$, with $X := \bigcup_{k=0}^{d} X_k$, where X_k is called the k-skeleton of the cellular complex X, usually with $d \in \{2, 3\}$, providing a cellular decomposition of the space $\mathbb{E}^d$ where the underlying space (point-set) of $\mathcal{S}$ is embedded.

For example, you may consider a set $\mathcal{L} = \{l_h\}$ of line segments, and the plane arrangement generated by it (see [13]), i.e., the 2-dimensional complex made by (relatively) open cells of dimension 0, 1 and 2. Analogously, you may consider a set $\mathcal{P} = \{p_k\}$ of planar polygons in $\mathbb{E}^3$. In this case the space arrangement $\mathcal{A}(\mathcal{P})$ is a 3-dimensional complex X made by (relatively) open 0-, 1-, 2- and 3-cells. A similar result $\mathcal{A}(\mathcal{S})$ is produced by any set $\mathcal{S}$ of 3D meshes and/or 2D meshes decomposing either open or closed surfaces, and with any kind of connected cells. Finally, we recall that, given a set A, the regularized set A^* is the closure of the interior of A. Our generated arrangements are regularized by construction, since all cells not contained in an d-dimensional cell are removed.

In order to help the reader to understand the following: the p-cells in a cellular complex are open and connected subsets of a topological space; are homeorphic to a p-ball; must be manifold; when embedded in Euclidean d-space they may be either convex or non-convex; linear (i.e. the set of solutions of a set of linear equalities and inequalities) or non-linear; bounded or unbounded (e.g. the exterior unbounded cell). When a cellular complex is represented using the LAR data structure (see Section 2.2) the condition of homeomorphism of cells to p-ball may be relaxed, allowing the cells be multiply connected, i.e. with holes. A very good characterization of an oriented p-cell is to define its boundary as an oriented $(p-1)$-cycle[2], possibly including other $(p-1)$-cycles with opposite orientation as internal holes, or other features internal to holes.

The algorithms discussed in this paper, and the underlying representation, work with chains (sets of cells) and with global linear operators. Before going ahead, the reader should read and try to roughly understand the (very simple) examples A.1 and A.2 in the Appendix. The actual data structures used are just sparse arrays. The only basic operations are the multiplication of two sparse matrices and the multiplication of a sparse matrix times a sparse vector. After having read the Section 2, the reader should return to A.1 and A.2 examples for complete understanding.

Even if our algorithms use global algebraic operations, in particular multiplications of matrices and vectors, local queries are always possible, and are computationally efficient. For example, the query about the incident oriented faces on a given oriented edge, is readily computed by product of the edge coordinate vector (a sparse vector storing only one 1 or -1 term), i.e. a 1-chain, times the sparse coboundary matrix δ_1 going from 1-chains to 2-chains. The result is the coordinate vector of the 2-chain incident on the edge, including two 2-cells if manifold, or more if the edge is non manifold. The product of a sparse matrix times a sparse vector with exactly one non-zero is $O(\ell)$, where ℓ is the number of non-zeros in the output sparse vector, hence is very efficient. In order to disclose the list of oriented edges on the 1-cycles corresponding to each incident face, again

[2]Concept of cycle = closed chain: see Section 2.1.

a multiplication of the ∂_2 boundary sparse matrix[3], times the coordinate representation of each above face (a single non-zero in its sparse coordinate vector) will again provide the answer in time linear with the size of the result.

1.1 Problem statement and results

The problem we discuss in this paper is how to compute the arrangement $\mathcal{A}(\mathcal{S})$ and hence the cellular complex $X := \mathcal{A}(\mathcal{S})$ generated by a set $\mathcal{S}$ of d- or $(d-1)$-complexes of linear, bounded and connected cells in $\mathbb{E}^d$. This problem can be identified with the construction, by induction, of a d-dimensional cellular complex, through the stratification of its skeletons $X_0, X_1, \ldots, X_{d-1}, X_d$. In fact, the unknown d-cells of the output arrangement are generated starting from 0-cells, used as 0-chains[4] to compute the coboundary operator δ_0, used in turn to compute the 1-cells and the δ_1 operator 0-chains to 1-chains, and so on, by iteratively growing in dimension.

The above introduces an important feature of the algorithmic workflow presented in this paper, since in computing the arrangement $\mathcal{A}(\mathcal{S})$ induced by $\mathcal{S}$, we actually compute the whole chain complex $C_\bullet$ generated by the cell complex $X := \mathcal{A}(\mathcal{S})$. For example, in 3D we compute all objects and arrows (morphisms) in the diagram below, and hence we obtain a computational knowledge of space subdivision homology [36, 37], including the Euler number. Given the input collection $\mathcal{S}$, in this paper we discuss how to compute and represent the chain complex

$$C_\bullet := C_3 \underset{\partial_3}{\overset{\delta_2}{\leftrightarrows}} C_2 \underset{\partial_2}{\overset{\delta_1}{\leftrightarrows}} C_1 \underset{\partial_1}{\overset{\delta_0}{\leftrightarrows}} C_0, \tag{1}$$

where C_p $(0 \leq p \leq d)$, with $d \in \{2, 3\}$, is a linear space of p-chains (sets with algebraic structure of p-cells), and where $\delta_{p-1} = \partial_p^\top$. Per se, boundary operators $\partial_1, \ldots, \partial_d$ belong in the chain complex, while coboundary operators $\delta_0, \ldots, \delta_{d-1}$ belong in a dual cochain complex. As a consequence, taking the coboundary of a chain only makes sense after chains and cochains have been identified, as explained in Section 2.1. Note that, in computing the d-cells of the arrangement, we actually compute the sparse matrix of the operator ∂_d, and for this construction we need the $(d-1)$-cells and ∂_{d-1}, and so on backward, until ∂_1 is constructed from the most elementary data (0- and 1-cells).

One component of our algorithmic workflow is the topological gift-wrapping, reminiscent of the "gift-wrapping" algorithm for computing convex hulls of 2D and 3D discrete sets of points [17, 35]. Actually, our topology-based algorithm broadly generalizes the former, being applicable also to non-convex contractible $(p-1)$-cells to construct the boundary of p-cells.

The robustness of geometrical and topological computations is approached here by lowering the dimension of numerical computations whenever it is possible, and by solving independently the resulting set of subproblems. E.g., the intersection of 3-cells is reduced to a set of intersections of bounding 2-cells, and each of those to a set of pairwise intersections of bounding line segments in 2D. The topology is finally reconstructed bottom-up, by successive identification of geometrical elements (reduction to quotient sets) by nearest neighborhood queries on vertices, and syntactical identification of coincident cells in canonical form, i.e., as sorted lists of vertex indices.

The problem studied in this paper has a number of useful geometric applications, including the motion planning of robots, the variadic[5] computation of Boolean operations (union, intersection, difference, symmetric difference), and the topology repair of graphical meshes, all starting from a set of cellular complexes embedded in the same Euclidean space. In particular, we are currently using chain complexes and (co)boundary operators to extract the models of neurons and vessels from extreme-resolution 3D images of brain tissue, and to dramatically reduce the complexity

[3] Note that $\delta_1 = \partial_2^\top$.
[4] Informally, a d-chain can be seen as a set of cells of dimension d.
[5] Which accepts a variable number of arguments.

of their representation, while preserving the homotopy type, in order to contribute to piecewise computation of the connectome of brain structures [15, 42].

1.2 Previous work

The construction of arrangements of lines, segments, planes and other geometrical objects can be found in [27] together with a description of the CGAL software [26], implementing 2D/3D arrangements with Nef polyhedra [30]. A wide analysis of papers and algorithms concerning the construction and counting of cells may be found in the dedicated survey chapter on Arrangements in the Handbook of Discrete and Computational Geometry [29]. The arrangements of polytopes, hyperplanes and d-circles are discussed in [9]. The above references deal with space arrangements generated by analytical subspaces.

The "Incidence Graph" data structure (IG) described in Edelsbrunner's book Algorithms in Combinatorial Geometry [24], which is an implementation of the Hasse diagram of the cells of a d-complex [20], is the standard way to look to topological data structures. Actually, our chain of boundaries is another implementation of it, associating each two adjacent levels of the Hasse diagram to a different boundary operator, implemented as a sparse matrix. The memory occupancy is also similar.

Some early papers were concerned with efficient representation of 3D cellular decompositions. In particular, [23] defined the polygon-edge data structure, to represent orientable and non-punctured 3D decompositions and their duals, manipulated by intricate operations with specialized Euler operators. Several other systems have been developed two/three decades ago, to handle the merging of complexes in the context of solid modeling and manufacturing automation. In particular, Alan Middleditch and colleagues in UK have proposed the Djin API [2, 41], and discussed the theoretical foundations for merging operations of cellular complexes. In 1988 Rossignac and O'Connor proposed their SGC (Selective Geometric Complex) formulation [45], providing a precise definition of geometric complexes, similar to the one used in this paper, but using more general topologies (lower dimensional cells inside the interior of cells, and not only on the boundary). Such authors also proposed their "merge" algorithm, called compatible subdivision, that computes the common subdivision of two complexes by subdividing the cells of the first complex against the cells of the second complex, and vice-versa, while maintaining the consistency of a common data structure.

Recently, a systematic recipe has been proposed in [51] for constructing a family of exact constructive solid geometry operations, but starting from a collection of boundary triangle meshes, and not from complexes of a general kind.

All merge algorithms, by definition, must follow the same steps. The advantages of formulating in terms of (co)chain complexes and operations are that this (1) reveals the common and general algebraic topological nature of this operation; (2) hides implementation specific low level details and algorithms; (3) provides explicit connection to sparse numerical linear algebra that can be efficiently implemented on GPU or other HPC platforms; (4) supports systematic and rigorous development of the algorithms that are correct by construction.

Other approaches aiming to integration of domain modeling, differential topology, mathematical modeling, and physical simulations are also based on chains and cochains. In particular, the Discrete Exterior Calculus (DEC) with simplicial complexes was introduced by [33] and made popular by [19] and [25]. Our approach to modeling with chains and cochains was introduced in [20–22].

Most of the above algorithms and procedures work with specific data structures optimized for representation of selected classes of geometric objects under consideration. In contrast, our formulation is cast in terms of (co)chain complexes and (co)boundary operators that may be

applied to very different geometric object, ranging from solid models, to engineering meshes, to geographical systems, to biomedical images. Our reference implementation relies on Linear Algebraic Representation (LAR) [22], that is well suited for efficient implementation of topological operations in terms of matrix algebra over cellular complexes with cells of a quite general kind.

1.3 Paper preview

In Section 2 the main definitions concerning cell complexes and chain complexes are recalled, together with the characteristic features of the Linear Algebraic Representation (LAR) scheme. In Section 3 a gentle introduction to our computational approach is given, supported also by a cartoon chronicle of a simple 3D example. The main contribution of the paper, i.e., the novel algebraic algorithm for computing the merging of d-complexes, is explained in Section 4. Section 5 provides some pseudocode and discusses the complexity of the main computational steps. For clarity, Section 6 exemplifies the developed algorithms on simple examples. The closing Section 7 summarizes the work and highlights its salient features. Some very simple examples of LAR input/output and topology computations are given in the Appendix.

2 BACKGROUND

For the sake of readability, let us introduce the meaning of most symbols: $\Lambda = \Lambda(X)$ is a cellular decomposition of the topological space X, i.e., a quasi-disjoint union of cells; Λ_p is the set of p-cells; C_p is a linear space of p-chains of cells over a field of coefficients; X_p is a p-complex, i.e. the p-skeleton of the d-complex $X_d := \mathcal{A}(\mathcal{S})$. Finally, $\mathcal{S}$ is a collection of cellular d-complexes and/or $(d-1)$-complexes embedded in $\mathbb{E}^d$ space. We use greek letter for cells and latin letters for chains, i.e., for signed combinations of cells. With some abuse of language, cells in Λ_p and singleton[6] chains in C_p are often identified. Also, let us remind the reader that the characteristic function $\chi_A : S \to \{0, 1\}$ is a function defined on a set $S = \{s_j\}$, that indicates membership of an element s_j in a subset $A \subseteq S$, having the value 1 for all elements of A and the value 0 for all elements of S not in A. We call characteristic matrix M of a collection of subsets $A_i \subseteq S$ $(i = 1, \ldots, n)$ the binary matrix $M = (m_{ij})$, with $m_{ij} = \chi_{A_i}(s_j)$. A matrix M_p, whose rows are indexed by singleton p-chains and columns are indexed by singleton 0-chains, provides the coordinate representation of a basis for the linear space C_p. Permutating either the rows or the columns would provide a different basis.

2.1 Cellular complex and chain spaces

2.1.1 Definitions. Let X be a topological space, and $\Lambda(X) = \bigcup_p \Lambda_p$ $(p \in 0, 1, \ldots, d)$ be a cellular decomposition of X, with Λ_p a set of (relatively) open and connected p-cells. A CW-structure on the space X is a filtration $\emptyset = X_{-1} \subset X_0 \subset X_1 \subset \ldots \subset X_{d-1} \subset X = \bigcup_p X_p$, such that, for each p, the skeleton X_p is homeomorphic to a space obtained from X_{p-1} by attachment of p-cells in $\Lambda_p = \Lambda_p(X)$ [31].

A CW-complex is a space X endowed with a CW-structure, and is also called a cellular complex. A cellular complex is finite when it contains a finite number of cells. A regular d-complex is a complex where every p-cell $(p < d)$ is contained in the boundary of a d-cell. Two d-cells are coherently oriented when their common $(d-1)$-cells have opposite orientations. A d-complex is orientable when its d-cells can be coherently oriented.

In this paper, the algorithm called "topological gift wrapping" produces CW-complexes, despite the fact that our merge operation may generate punctured d-cells, i.e., cells with holes (but without isolated points). As a matter of fact, such spaces may be handled by combining standard CW-complexes (i.e., with cells homeomorphic to balls) by the adjunction of d-cells to the interior of

[6] A set having exactly one element.

d-cells. In few words, each void in a cell or disconnected sub-complex is considered as a single oriented cell, counterclockwise or clockwise oriented, respectively. The orientation is handled depending on the parity of their relative containment relation. This handling mostly concerns in our approach the adjoining of sets of rows (columns) in the final (co)boundary matrix. See, for this purpose, the Algorithm 2 of Section 5.2.

2.1.2 Chains. Let $(G, +)$ be a nontrivial commutative group, whose identity element will be denoted 0. A p-chain of X with coefficients in G is a mapping $c_p : X \longrightarrow G$ such that, for each $\sigma \in X_p$, reversing a cell orientation changes the sign of the chain value:

$$c_p(-\sigma) = -c_p(\sigma).$$

Chain addition is defined by addition of chain values: if c_{p_1}, c_{p_2} are p-chains, then $(c_{p_1} + c_{p_2})(\sigma) = c_{p_1}(\sigma) + c_{p_2}(\sigma)$, for each $\sigma \in X_p$. The resulting group is denoted $C_p(X; G)$. When clear from the context, the group G is often left implied, writing $C_p(X)$.

Let σ be an oriented cell in X and $g \in G$. The elementary chain whose value is g on σ, $-g$ on $-\sigma$ and 0 on any other cell in X is denoted $g\sigma$. Each chain can then be written in a unique way as a sum of elementary chains. With abuse of notation, we do not distinguish between cells and singleton chains (i.e., the elementary chains whose value is 1 σ for some cell σ), used as elements of the standard bases[7] of chain groups.

Chains are often thought of as attaching orientation and multiplicity to cells: if coefficients are taken from the group $G = (\{-1, 0, 1\}, +) \simeq (\mathbb{Z}_3, +)$, then cells can only be discarded or selected, possibly inverting their orientation (see [20]). A p-cycle is a closed p-chain, i.e. a p-chain without boundary. It is useful to select a conventional choice to orient the singleton chains (single cells) automatically. 0-cells are considered all positive. The p-cells, for $1 \leq p \leq d-1$, can be given a coherent (internal) orientation according to the orientation of the first $(p-1)$-cell in their canonical representation sorted on indices of their $(p-1)$-cycle. Finally, a d-cell may be oriented as the sign of its oriented volume.

2.1.3 Cochains. Cochains are dual to chains: a p-cochain attaches additively an element of the group G to each p-chain. Singleton cochains, attaching the identity element 1 to singleton chains, form the standard bases of cochains groups. The groups of p-chains and those of p-cochains may be identified with each other in infinitely different ways. Different legitimate identifications, while affecting the metric properties of the chain-cochain complex [21], do not change the topology of finite complexes. Since we shall only use the topological properties of finite chain-cochain complexes, we feel free to chose the simplest possible identification, obtained by identifying the elements of the standard chain bases with the corresponding elements of the standard cochain bases. In this paper, we take for granted that chains and cochains are identified in this trivial way.

2.1.4 Examples. For reader's convenience, we include here some very simple examples of cellular complexes, and some basic computations with boundary and coboundary operators. In Figure 1 we show a space partition into cells of dimension 0 (v_i, $1 \leq i \leq 6$), dimension 1 (η_j, $1 \leq j \leq 8$), and dimension 2 (γ_k, $1 \leq k \leq 3$), associated with different additive groups of coefficients.

In Figure 1c the 0-cells are given arbitrary numbers, e.g., the values of an arbitrary scalar field; whereas the values for 1-cells and 2-cells were computed from these using the co-boundary relations δ_0 and δ_1. Note that they are discrete gradients and curl values associated to 1-cells and 2-cells, respectively. In discrete geometric calculus, we are interested in cochains as functions from chains

[7] A standard basis, also called a natural basis, is a special orthonormal vector basis in which each basis vector has a single nonzero entry with value 1.

to reals. The colored numbers on 1- and 2-cells are exactly the evaluation of the dual elementary cochain on each elementary chain.

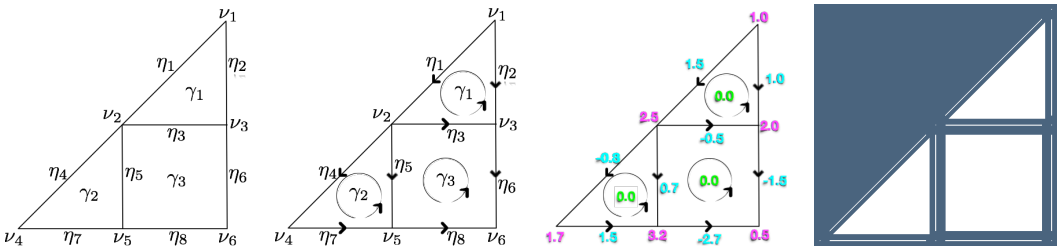


Fig. 1. Cellular complexes with 0-cells in $\Lambda_0 = \{\nu_1, \ldots, \nu_6\}$, 1-cells in $\Lambda_1 = \{\eta_1, \ldots, \eta_8\}$, and 2-cells in $\Lambda_2 = \{\gamma_1, \gamma_2, \gamma_3\}$: (a) non-oriented complex, with cell coefficients in $\mathbb{Z}_2 = \{0, 1\}$; (b) oriented complex, with cell coefficients in $G = \{-1, 0, +1\}$; (c) oriented complex, with cell coefficients in $\mathbb{R}$, using different colors for the maps from Λ_0, Λ_1, and Λ_2 to $\mathbb{R}$. To interpret the real numbers here, see Example 2.5; (d) the exploded 2-complex with $|\Lambda_0| + |\Lambda_1| + |\Lambda_2| = 6 + 8 + 3$ bounded cells, plus the exterior unbounded cell.

Example 2.1 (Chains). Unoriented chains take coefficients from $\mathbb{Z}_2 = \{0, 1\}$. E.g., a 0-chain $c \in C_0$ shown in Figure 1a is given by $c = 1\nu_1 + 1\nu_2 + 1\nu_3 + 1\nu_5$. Hence, the coefficients associated to all other cells are zero. The coordinate vector of c with respect to the (ordered) basis $(\nu_1, \nu_2, \ldots, \nu_6)$ is hence $[1, 1, 1, 0, 1, 0]^t$. Analogously for the 1-chain $d \in C_1$ and the 2-chain $e \in C_2$, written by dropping the 1 coefficients, as $d = \eta_2 + \eta_3 + \eta_5$ and $e = \gamma_1 + \gamma_3$, with coordinate vectors $[0, 1, 1, 0, 1, 0, 0, 0]^t$ and $[1, 0, 1]^t$, respectively.

Example 2.2 (Orientation). In Figure 1b it is shown an oriented version of the cellular complex $\Lambda = \Lambda_0 \cup \Lambda_1 \cup \Lambda_2$, where the 1-cells are oriented from the vertex with lesser index to the vertex with greater index, and where all 2-cells are counterclockwise oriented. Let us note that the orientation of every cell may be fixed arbitrarily, since can always be reversed by the associated coefficient, that is now taken from the set $\{-1, 0, +1\}$. So, the oriented 1-chain having first vertex ν_1 and last vertex ν_5 is now given as $d' = \eta_2 - \eta_3 + \eta_5$, with coordinate vector $[0, 1, -1, 0, 1, 0, 0, 0]^t$

Example 2.3 (Boundary). The boundary operators are maps $C_p \to C_{p-1}$, with $1 \leq p \leq d$, hence for a 2-complex we have two operators, denoted as $\partial_2 : C_2 \to C_1$ and $\partial_1 : C_1 \to C_0$, respectively. Since they are linear maps between linear spaces, may be represented by matrices of coefficients $[\partial_2]$ and $[\partial_1]$ from the corresponding groups. For the unsigned and the signed case (Figures 1a and 1b) we have, respectively:

$$[\partial_2] = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \text{ and } \quad [\partial_2'] = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2}$$

Analogously, for the unsigned and the signed matrices of the ∂_1 operator, we have:

$$[\partial_1] = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}, \text{ and } \quad [\partial_1'] = \begin{pmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}. \quad (3)$$

As a check, let compute the 0-boundary of the coordinate representations of the unsigned 1-chains $[d] = [0, 1, 1, 0, 1, 0, 0, 0]^t$ and of the signed 1-chain $[d'] = [0, 1, -1, 0, 1, 0, 0, 0]^t$:

$$[\partial_1][d]^t \ mod\ 2 = [1, 0, 0, 0, 1, 0]^t = v_1 + v_5 \in C_0,$$

where the matrix product is computed *mod* 2, and where

$$[\partial_1'][d']^t = [-1, 0, 0, 0, 1, 0]^t = v_5 - v_1 \in C_0'.$$

Example 2.4 (Dual cochains). The concept of cochain ϕ^p in a group C^p of linear maps from chains C_p to $\mathbb{R}$ allows for the association of numbers not only to single cells, as done by chains, but also to assemblies of cells. A cochain is hence the association of discretized subdomains of a cell complex with a global numeric quantity, usually resulting from a discrete integration over a chain. Each cochain $\phi^p \in C^p$ can be seen as a linear combination of the unit p-cochains $\phi_1^p, \ldots, \phi_k^p$[8] whose value is 1 on a unit p-chain and 0 on all others. The evaluation of a real-valued cochain is denoted as a duality pairing, in order to stress its bilinear property:

$$\phi^p(c_p) = \langle \phi^p, c_p \rangle.$$

This mapping is orientation-dependent, and linear with respect to the assemblies of cells, modeled by chains [32]. Also, remember here that we identify chain and cochain spaces (see Section 2.1.3).

Example 2.5 (Coboundary). The coboundary operator $\delta^p : C^p \longrightarrow C^{p+1}$ acts on p-cochains as the dual of the boundary operator ∂_{p+1} on $(p+1)$-chains. For all $\phi^p \in C^p$ and $c_{p+1} \in C_{p+1}$, it is:

$$\langle \delta^p \phi^p, c_{p+1} \rangle = \langle \phi^p, \partial_{p+1} c_{p+1} \rangle.$$

Recalling that chain-cochain duality means integration, the reader will recognize this defining property as the combinatorial archetype of Stokes' theorem. See also that, in Figure 1 we have, $(\delta_2 \circ \delta_1)(\gamma_1) = 0.0$. This property, i.e. $\delta \circ \delta = 0$, is the discrete archetype by which the curl of gradient is zero. Note that a scalar field, in the discrete version, becomes a real valued 0-cochain to be valued on 0-cells, i.e., on 0-chains.

It is possible to see [21] that since we use dual bases, matrices representing dual operators are the transpose of each other: for all $p = 0, \ldots, d-1$,

$$[\delta^p]^t = [\partial_{p+1}].$$

In Figure 1c, coefficients from $\mathbb{R}$ are associated to elementary (co)chains, as resulting from the evaluation of cochain functions on elementary chains. When cochain coefficients are taken from $G = \{-1, 0, +1\}$, we get, from Eqs. (2) and (3):

$$[\delta^1] = [\partial_2']^t \quad \text{and} \quad [\delta^0] = [\partial_1']^t$$

so that, with $f = [0, 0, 0, 0, 1, 0, 0, 0] \in C^1 \equiv C_1$, we get

$$[\delta^1][f]^t = [0, -1, 1]^t = \gamma_3 - \gamma_2 \in C^2 \equiv C_2,$$

as you can check by looking to Figure 1b.

[8] Coincident with $\eta_p^1, \ldots, \eta_p^k$ by identification of primal and dual bases.

Example 2.6 (Application: geographical map). Let us consider a geographical map as the plane arrangement $\mathcal{A}(\mathcal{S})$ generated by a quasi-disjoint set of regions (2-complex R_2) superimposed with road networks (1-complex R_1).

For this purpose, we take as input the collection of data $\mathcal{S} = \{R_2, R_1\}$, and select the combinatorial union of their 1-skeletons $R_2^1 \cup R_1^1$. From this set of 1-cells—which is not a 1-complex since cells may intersect out of their boundary vertices—we compute, using the algorithms presented in this paper, the cellular complex $X_2 = \mathcal{A}(\mathcal{S})$ and the associated chain complex $C_\bullet$.

Here, 1D roads are simply a particular chain in the linear space C_1, whereas each region is a chain in C_2, i.e.,a sum of singleton 2-chains, with abuse of language: a sum of 2-cells[9]. The length of any portion of road is the real number produced by a 1-cochain on that particular 1-chain. Analogously, the area of a region is the real number produced by a 2-cochain evaluated on that 2-chain. By linearity, that number is the sum of areas of basis 2-chains, i.e., with abuse of language, a sum of areas of 2-cells in X_2.

2.2 Linear Algebraic Representation

LAR (Linear Algebraic Representation) [22] is a representation scheme [44] for geometric and solid modeling. The domain of the scheme is provided by cellular complexes, while its codomain is the set of sparse matrices. The topology in LAR is given just by the binary characteristic matrix M_d of d-cells for polytopal complexes, or by the pair M_d, M_{d-1} for more general (non convex and/or non contractible) types of cells [22].

The LAR polyhedral domain coincides with complexes of connected d-cells, including non-convex and multiply connected cells. Sparse matrices are stored in memory using either the CSR (Compressed Sparse Row) or the CSC (Compressed Sparse Column) memory format [10]. Note that the sparsity of matrices representing a cellular complex grows quadratically with the number n of cells, so that the sparse matrix representation of a cellular complex is $O(n)$.

The quite general shape allowed for cells makes the LAR scheme notably appropriate for biomedical applications like the modeling of neuronal tissues [15, 42] and the solid modeling of buildings and their components [39]. E.g., the whole facade of a building, including the simplified openings of windows and doors, can be described by a single 3-cell of its solid model. Also, the algebraic foundation of LAR allows not only for fast queries about incidence and adjacency of cells, but also for extracting—via fast SpMV computational kernels—the boundary or coboundary of any 3D subset of the building model.

In summary, LAR provides a direct management of all subsets of cells and their physical properties through the linear spaces of chains induced by the model partitioning, and their dual spaces of cochains. The linear operators of boundary and coboundary between such linear spaces, suitably implemented by sparse matrices, directly supply the discrete differential operators of gradient, curl and divergence, while their combination gives the Laplacian [20]. A word of warning is in order here: contrary to gradient, curl and divergence, the Laplacian operator substantially depends on metric properties, hence on the specific chain-cochain identification.

2.3 LAR of a cellular complex

A common representation of a d-complex in both commercial and academic systems is some—often very intricate—data structure storing its d-boundary. For 3D meshes discretising the domain of a simulation, or when the representation scheme is decompositive [44], the object of the storage structure is the set of either d- or $(d-1)$-cells, or both. The representation of cells is normally

[9]For this specific application, we may add to $\mathcal{A}(\mathcal{S})$ the dangling terminal segments of roads, if any, removed by the merge procedure to get a regular partition of the map.

supplemented by the storage of subsets of the incidence relations between topological elements, often paired with some ordering though linked lists of pointers.

The details of such data structures vary greatly depending on the type of cells, dimension, or specific intended applications, leading to many specialized and ad hoc computational procedures that must be redesigned for each new data structure. For example, boundary evaluation or boundary traversal algorithms strongly depend on many specific assumptions in the data structure and do not generalize across dimensions.

LAR is a concise and simple representation that supports the same queries and operators (incidence, boundary, coboundary, etc.) without additional computational overhead. In Appendix A we show some very simple examples of LAR definition of cellular complexes and computation of their properties. With the LAR scheme, only the characteristic functions of d-cells as vertex subsets are necessary for representing polytopal complexes. Examples include simplicial, cubical, and Voronoi complexes. In all cases, boundary and coboundary operators, and all derived topological queries, can be supported by computational kernels for sparse matrix multiplication implemented on GPUs [5, 18].

It is worth stressing that this dimension-independent representation supports a description of cellular and chain complexes, including topological operators, without special data structures, but simply using signed integers or arrays of signed integers, either dense or sparse depending on the size of the complex. The LAR scheme enjoys several useful properties, including simplicity, compactness, readability, and direct usability for calculus. It may by useful to distinguish between the input/output representation, and the internal representation of a cellular complex. The human-readable i/o representation is made, for each dimension p of cells ($1 \leq p \leq d$), by an array of arrays of indices of vertices of p-cells ; the internal representation is the corresponding sparse binary matrix, implementing the characteristic matrix M_p. The reader is referred to [22] for further discussion and details.

Example 2.7 (2D cellular complex). The LAR of the simple example in Figure 1 is given below. Here, arrays `V`, `EV`, `FV` contain, respectively, the coordinates of vertices, and the indices of cell vertices for edges, and faces.

```
V = [[1,1],[0.5,0.5],[1,0.5],[0,0],[0.5,0],[1,0]]
EV = [[0,1],[0,2],[1,2], [1,3],[1,4],[2,5], [3,4],[4,5]]
FV = [[0,1,2],[1,3,4],[1,2,4,5]]
```

It is worth noting that, by using the chain complex generation introduced in this paper, the complete representation of geometry and topology of the example 2.1.4 can be reduced to

```
V = [[1,1],[0.5,0.5],[1,0.5],[0,0],[0.5,0],[1,0]]
EV = [[0,1],[0,2],[1,2], [1,3],[1,4],[2,5], [3,4],[4,5]]
```

since the 2-cells `FV` may be computed as the 2-cells of the arrangement $\mathcal{A}(\mathcal{E})$ of 2-space induced by the 1-skeleton codified here by `V` and `EV` arrays. See, e.g., the Example 6.3.

Example 2.8 (Cell with hole). Let us consider a square of side three with an internal hole of side one shown in Figure 2, and the cellular complex defined as:

```
V = [[0.,0.],[3.,3.],[1.,2.],[2.,1.],[3.,0.],[1.,1.],[0.,3.],[2.,2.]]
FV = [[0,1,2,3,4,5,6,7],[2,3,5,7]]
EV = [[0,4],[0,6],[1,4],[1,6],[2,5],[2,7],[3,5],[3,7]]
```

Its LAR is written above in canonical form (as sorted arrays of sorted array). The unsigned matrix of the boundary operator $\partial_2 : C_2 \to C_1$, computed by filtering elements of value 2 in the matrix

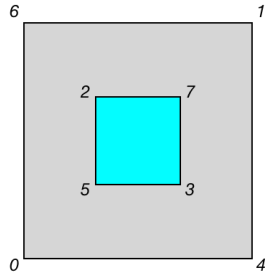


Fig. 2. A cellular 2-complex with two 2-cells, eight 1-cells, and eight 0-cells.

$M_1\, M_1^t$, and transposed here for typographical reasons, is:

$$[\partial_2]^t = \text{filter}(M_1\, M_1^t, 2) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix},$$

where the first row represents the non-convex cell with the hole, and the second row represents the convex cell within the hole. The reader may easily check that the four ones in positions from fifth to eighth in the second row of $[\partial_2]^t$ correspond to the last four edges in the LAR array EV.

By (mod 2) matrix multiplication of $[\partial_2]$ times the coordinate representation $[c]$ of the 2-complex in Figure 2, i.e., times the "total" 2-chain $\begin{pmatrix} 1 & 1 \end{pmatrix}^t$, we get the coordinate representation

$$[\partial_2][c] = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}^t$$

of the 1-boundary of the 2-chain c. Of course, this one is the chain of the first four edges in EV.

2.4 LAR of a chain complex

A common use of LAR is to represent the characteristic matrices M_p $(1 \leq p \leq d)$ of a cellular d-complex, as either arrays of arrays of integer indices to 0-cells, or as sparse binary matrices storing the same information. But more important is its use to fully represent a chain of boundary operators between linear spaces, i.e. a chain complex. The first use stores the bases of chain spaces; the second use provides a full representation of the topology of the complex. In particular, when one such sparse matrix is used, say $[\partial_3]$, we have by columns the representation of the basis of oriented 3-cells as cycles of oriented 2-cells. The approach put forth in this paper may be seen — and is actually implemented — as the generation of the whole hierarchy of boundary operators and bases of p-cells (arrows and objects, in categorical language) of the arrangement of the d-space induced by the input. See Eq. 1.

3 A GENTLE INTRODUCTION

To perform a Boolean set operation on two or more solids represented by their boundaries, you need to compute how two (or more) cellular complexes (boundaries of solids) break each other into pieces, and select the 3D pieces that enter the Boolean result. We start with the first step of this process, which is commonly called merge operation. This problem becomes harder with three-dimensional meshes decomposing solids, i.e., when using 3D decompositions (see, e.g., [23, 41, 45]). Whereas B-reps and object decompositions were treated independently in the past [44], a unified approach is preferable in order to deal with increasingly complex and diverse models that may include variety of cellular models. For example, you might combine two or more 3D meshes together with open or closed surfaces which separate the interesting parts of a model.

In other words, you may need to merge two or more cellular complexes, even of different dimensions, and to compute the resulting space arrangement. This computation, using simple methods well founded on basic mathematical concepts of algebraic topology, is the topic of this paper. We adopt here a dimension-independent notation, since concepts and methods are the same in 2D and in 3D, so that their implementation can be unified and made simpler, and even used for applications in higher dimensions.

The main idea of this paper is to reduce the intersections of higher dimensional discrete varieties to a number of much simpler intersections, between pairs of lines in the 2D plane. For this purpose, we (i) reduce the intersection of a $(d-1)$-cell σ with all the possibly intersecting others to an independent set of intersections with the $z = 0$ plane, (ii) calculate the resulting (easily computed) arrangements in 2D, and finally (iii) come back to higher dimension, while at the same time identifying the possibly coincident instances of lower-dimensional cells, that were independently generated from each other.

A cartoon chronicle of the computation of the arrangement of $\mathbb{E}^3$ (space decomposition including the unbounded exterior cell) generated by two very simple cellular 2-complexes, i.e., by the boundaries of two translated and rotated unit cubes, is shown in Figures 3, 4, 5, and 6.

3.1 Input and setup

First we need to assemble the input collection of data into a single representation, by properly concatenating the arrays of vertices and cells. Figure 3a shows the LAR input, made by the vertex array `V` and by the arrays `CV`, `FV` of indices of vertices of each 3-cell and 2-cell of the cubes. `CV`, `FV` stand for "cells by vertices" and "faces by vertices", respectively. Note that the input is not a cellular complex, since cells intersect outside of their boundary. The relative positions of cubes, and both their (explosed) boundary 2-complex X_2^h ($h \in \{1, 2\}$) are shown in Figures 3b and 3c. The bounding boxes of each $\sigma \in B$, with $B = \Lambda_2^1 \cup \Lambda_2^2$, and Λ_2^h the sets of 2-cells, are displayed in yellow in Figure 3d. Note that some 3D bounding boxes of 2-cells degenerate to rectangles, since their 2-faces are aligned with the coordinate planes, while others are not.

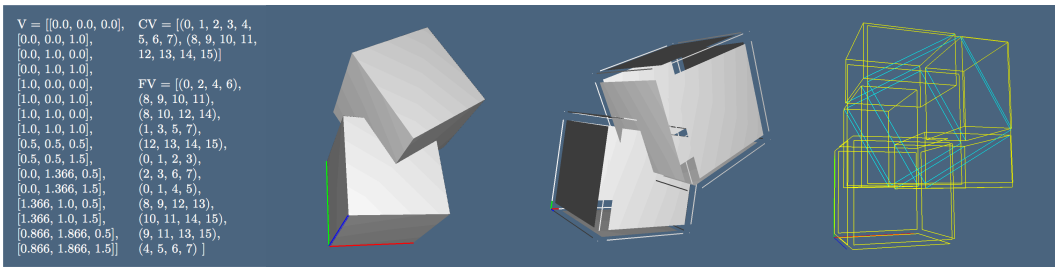
3.2 Decomposition of input 2-cells (facets: $d - 1 = 2$)

Figure 4 gives the sequence of computations on a generic 2-cell σ (i.e. a cell of dimension $d - 1$) in the input set B. First the subset $\mathcal{I}(\sigma)$ of 2-cells of possible intersection with σ is computed, by intersecting the results of queries upon three (i.e., d) interval trees, based on sides of containment boxes of 2-cells. Then the set $\Sigma = \{\sigma\} \cup \mathcal{I}(\sigma)$ is transformed in such a way that the 2-cell σ is mapped into the $z = 0$ subspace, i.e. to $x_3 = 0$. In this space $\mathbb{E}^2 \times \mathbb{E}$ the 1-cells in Σ are used to compute a set of line segments in $\mathbb{E}^2$, generated by intersection of edges of each 2-cell in $\mathcal{I}(\sigma)$ with $z = 0$, and by join (convex combination) of alternate pairs of intersection points of boundary edges along the intersection line of a 2-face with $z = 0$.

3.3 Construction of ∂_2 boundary matrix

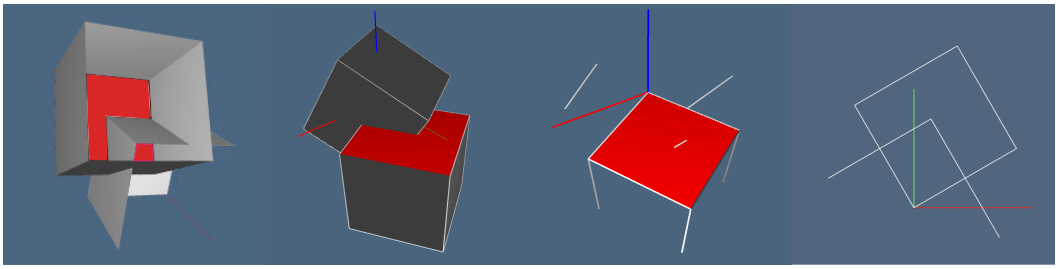
The planar processing of the 2-cell σ continues in Figure 5, where the arrangement $X_2(\Sigma) := \mathcal{A}(\Sigma)$ induced by Σ on $\mathbb{E}^2$ is computed (see Figure 5b). This computation by intersection of lines produces a linear graph, shown exploded in Figure 5a. The dangling edges (and dangling tree subgraphs) are removed, by computing the maximal 2-point-connected subgraphs [49] via the [34] algorithm.

Only these biconnected components enter the ensuing computations, since the remaining graph parts are certainly dangling trees (1-connected subgraphs), and cannot contribute to the space arrangement. If required by the envisaged application, such trees of 1-cells may be added a posteriori



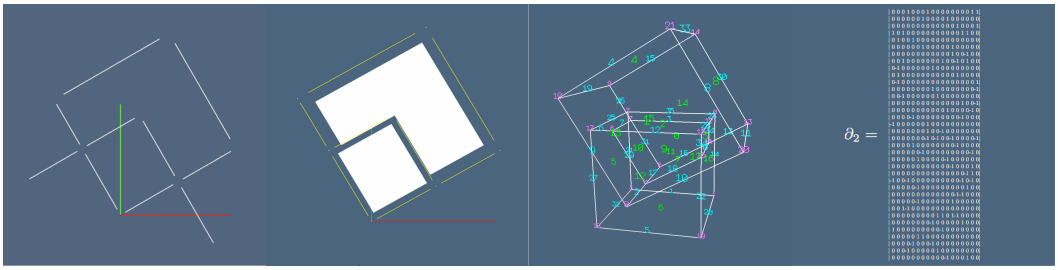
(a) (b) (c) (d)

Fig. 3. Input of two 3D cellular complexes $\{X_3^1, X_3^2\}$ in $\mathbb{E}^3$: (a) LAR input data: V := vertices; CV := 3-cells; FV := 2-cells; (b) rotated and translated unit cubes $\mathcal{S}_3 = \{ X_3^h,\ h \in \{1,2\} \}$ in $\mathbb{E}^3$; (c) the (exploded) set of 2-complexes $B = \bigcup_{h\in\{1,2\}} X_2^h$. Note that the 2-cells in 3-space have no orientation; (d) image of the (exploded) spatial index over 2-cells, using (yellow) 3D containment boxes & three 1D interval trees (not included in the image).



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

Fig. 4. Single facet 2D decomposition: (a) the (red) reference facet σ and the set $\mathcal{I}(\sigma)$ of possibly intersecting facets; (b) $\Sigma = \{\sigma\} \cup \mathcal{I}(\sigma)$ after the transformation mapping σ into $x_3 = 0$; (c) σ facet and 1-cells in Σ intersecting the subspace $x_3 = 0$; (d) the six 1D generated intersections in $\mathbb{E}^2 \times \mathbb{E}$ between $\mathcal{I}(\sigma)$ and $x_3 = 0$. The output is a $\mathcal{L}(\Sigma)$ collection of six line segments in the $x_3 = 0$ subspace (the 2D plane)

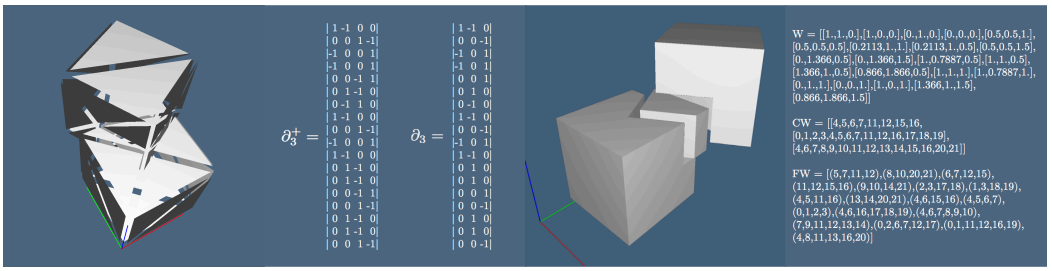


(a) (b) (c) (d)

Fig. 5. 3D reconstruction: (a) view (exploded) of 2D line segments produced by pair-wise intersection of elements in $\mathcal{L}(\Sigma)$; (b) 2-complex $X_2(\Sigma)$ generated as $\mathcal{A}(X_1(\Sigma))$, where the X_1 skeleton is generated by computing the maximal biconnected subgraph induced by $\mathcal{L}(\Sigma)$, with nodes at the edge intersections; (c) numbered 0-, 1- and 2-cells (with different colors) of the reconstructed $X_2(B)$ after identification of coincident cells generated by the embedding in $\mathbb{E}^3$ of all $X_2(\Sigma)$, for each $\sigma \in B$; (d) matrix $[\partial_2]$, assembling by column the signed 1-chain representation of 2-cells, before computing the quotient 0- and 1-cells. The reader should remember that a quotient set is a set derived from another by an equivalence relation. In this case, two cells are equivalent if (and only if) they have the same support. The representatives of equivalence classes are computed by identification of coincident vertices (for 0-cells), and by equality of canonical representations (for 1-cells). Note that 2-cells are uniquely generated, whereas 3-cells are still undetected.

to the generated arrangement. The resulting graph is actually a representation of the $\partial_1(\sigma)$ and $\delta_0(\sigma)$ operators of the chain 2-complex associated to a plane arrangement.

Then, the oriented 2-cells of the partition $\mathcal{A}(\Sigma)$ are computed as shown in Figure 5b, so generating the $\partial_2(\sigma)$ and $\delta_1(\sigma)$ operators of the plane arragement. The process is repeated for each $\sigma \in B$, each $X_2(\sigma) = \mathcal{A}(\Sigma)$ is mapped back in $\mathbb{E}^3$, and coincident 0- and 1-cells are identified numerically or syntactically, making use of their unique canonical LAR representation. The canonical representation of a cell is the tuple (array) of sorted indices of cell vertices, after identification of numerically nearby-coincident vertices using a *kd*-tree. The resulting 2-complex $X_2(B)$, embedded in $\mathbb{E}^3$, is shown in Figure 5c. Its 2-cells, written as 1-chains, i.e. as linear combinations of signed 1-cells, are stored by column in the matrix of the operator $\partial_2 : C_2 \longrightarrow C_1$, shown in Figure 5d. Let us remind that a 1-cell τ, is written[10] by convention as $v_k - v_h$ when $k > h$, and is oriented from v_h to v_k. The conventional rules used in this paper about sign and orientation of cells are summarized at the end of Section 2.1.2.



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

Fig. 6. Output: (a) Triangulation of 2-cells, needed to compute the 1-cell coboundaries, in particular when non-convex cells are generated, and where the 2-cells are sorted on angles around their common 1-cell; (b) $\partial_3^+ : C_3 \longrightarrow C_2$, and boundary operator ∂_3 removed of exterior cell, providing a basis for chains of bounded cells; (c) 3-cells (exploded) in $X_3 := \mathcal{A}(\mathcal{S})$; (d) output LAR model of the cellular complex X_3.

3.4 ∂_3 boundary matrix and $\mathcal{A}(X_3)$ output

Finally, the 3-cells of the 3-space partition induced by B are computed, using the topological gift wrapping algorithm of Sections 4.2.2 and 4.2.3, pseudo-coded in Section 5.1, whose results are shown in Figure 6b. As discussed there, the correct boundary operator is ∂_3, whereas the "augmented" ∂_3^+ (one more column, linear combination of the others) is directly generated by that algorithm.

For each 1-cell in the X_1 skeleton we compute the cyclic order of 2-cells incident on it, in order to extract the 3-cells via our topological gift wrapping algorithm (see Figure 9) [11]. Permutations $Next : C_2 \longrightarrow C_2$ and $Prev := Next^{-1}$ are used to compute the signed 2-cycle representation of basis 3-cells. Such 2-cycles are detected and stored by column in the ∂_3^+ matrix (see Figure 6b). This one is rewritten as the ∂_3 operator by finding and removing the column of the exterior unbounded cell. The bounded 3-cells of the complex $X_3 := \mathcal{A}(B)$ are displayed in Figures 6c and 6d as the array `CW`.

The LAR representation of the output complex X_3 is finally given in Figure 6d, using a user-readable version of CSR (Compressed Sparse Row) format for binary matrices, where the non-zeros

[10] As a 0-chain of signed 0-cells in the matrix representation of $\partial_1 : C_1 \longrightarrow C_0$.

[11] In our current implementation, a CDT triangulation [47] of all 2-cells is used to compute correctly, also for non-convex cells, the cyclic orders of 2-cells about 1-cells. This triangulation, while not mandatory, is the quickest robust method we found to implement exact cyclic permutations of non-convex 2-cells around 1-cells. Alternatively, neighborhood classification via offsetting and point-in-set methods could be used.

of binary data do not need storage. This representation is useful for input/output of LAR models, as the reader may see in Appendix A, and for their long-term storage in document format.

It is easy to see that, for B-reps of 2D manifolds without boundary, the space required by LAR is $|FV| = 2|E|$ [50], i.e., 1/4 of the well-known winged-edge representation [4]. Note that $|FV|$ is also the cardinality of the incidence relation between faces and vertices. In LAR, it is given by the sum of lengths of elements of the array `FV`. As an example, consider the B-rep of a 3-cube: each face contains 4 vertices ($6 \times 4 = 24 = 2 \times 12$), where 12 is the number of edges. Note that the actually computed ∂_3^+ matrix contains one more column (the exterior 3-cell) which is a linear combination of the other columns [40]. Hence, in order to get a basis for the linear space C_3, and the matrix representation of ∂_3 with the correct number of independent elements, this column has to be located (see Section 4.2.6) and removed, as shown in Figure 6b.

The time complexity of sequential $[\partial_3]$ construction from $[\partial_2]$ is $O(nm \log m)$, where n, m are the numbers of 3- and 2-cells, in the worst case of unbounded complexity of 3-cells (see Section 5.1.1), and $O(nk \log k)$ where k is an upper bound on the number of 2-cells forming on the boundary of a single 3-cell.

4 ALGORITHMIC WORKFLOW

The computation of the arrangement of $\mathbb{E}^d$ generated by a collection of d-complexes and/or $(d-1)$-complexes, is discussed in this section using some mathematical tool (chain complexes) from algebraic topology, i.e., the linear spaces of chains (of cells) and the linear (co)boundary maps between such spaces. At the best of our knowledge, the present approach is the first of this kind.

It is worth noting that the support space (point-set union) of the $(d-1)$-skeleton of the output complex equals the union of the corresponding input skeletons. In particular, our approach produces the minimal subdivision of the output boundaries, since no part of the boundary of any output d-cell was not present on input $(d-1)$-cells. By contrast, approaches based on spatial indices, like [3, 48], have much more object cuts, and lead to much larger fragmentation of the underlying space and correspondingly larger size of the computed cellular arrangement. A quantitative comparison cannot be done in general, since the algorithm performance depends on the shape of the objects combined. However, we may say that with the algorithm used for Booleans in our library `PyPlasm`[12], based on a combination of octrees and BSP-trees, the number of (d-1)-cells increases in a range between 1 and 2 orders of magnitude.

In this section we adopt a dimension-independent writing style. Actually, in a full-fledged dimension-independent implementation, the algorithm would iterate on dimensions, starting from dimension 2. Consequently, no recursion is needed, and this allows for parallel implementations. We leave this matter to a further paper.

4.1 Merge of 2-skeletons

Let d be the dimension of the embedding space $\mathbb{E}^d$, and

$$\mathcal{S}_d = \{ X^h,\ h \in H \}$$

the input collection of d-complexes, and denote with $X^h_{d-1} \subset X_d$ the $(d-1)$-th skeleton of the h-th d-complex. We proceed to compute the arrangement of $\mathbb{E}^d$ generated by $\mathcal{S}_d$, or, more precisely, the regularized d-complex $X = \mathcal{A}(\mathcal{S}_{d-1})$.

For the sake of concreteness, the reader may assume here $d = 3$. It is worth noting that our merge algorithm actually computes the sparse matrix $[\partial_{d-1}]$ of the boundary operator of the generated

[12] `https://github.com/plasm-language/pyplasm`

space arrangement. This differentiate our approach from others in the literature. Our topological gift wrapping algorithm (Section 4.2) computes the operator $[\partial_d]$ starting from $[\partial_{d-1}]$ as input.

4.1.1 Assemble the $(d-1)$-skeletons. We start by considering the set $B = \bigcup_{h\in H} X^h_{d-1}$, combinatorial union of $(d-1)$-cells in $\mathcal{S}_d$, where H is a set indexing the input complexes. Note that B may not be a complex, since $(d-1)$-cells may geometrically intersect outside their boundaries. So, the actual input is a collection B of $(d-1)$-complexes.

4.1.2 Compute the spatial index $\mathcal{I} : B \to 2^B$. An interval tree is a data structure to hold intervals [43], that allows to efficiently find all intervals that overlap with any given interval or point. It is mainly used for windowing queries. We compute a spatial index $\mathcal{I} : B \to 2^B$, mapping each $(d-1)$-cell σ to the subset of $(d-1)$-cells τ such that $box(\sigma) \cap box(\gamma) \neq \emptyset$, where $box(\gamma)$ is the containment box of $\gamma \in B$, i.e. the minimal d-parallelepiped parallel to the Cartesian frame, and such that $\gamma \subseteq box(\gamma)$. The $\mathcal{I}$ map is computed by set intersection of the outputs of d elementary queries on 1D interval trees.

4.1.3 Compute the facet arrangments in $\mathbb{E}^{d-1}$. We compute the set $\mathcal{X}_{d-1}$ of facet arrangements $\mathcal{A}(X_{d-1}(\sigma))$ induced in $\mathbb{E}^{d-1}$ by σ elements of B, fragmented by all the incident cells $\mathcal{I}(\sigma)$:

$$\mathcal{X}_{d-1} = \{\, \mathcal{A}(\{\sigma\} \cup \mathcal{I}(\sigma)),\ \sigma \in B \,\}.$$

To compute such arrangements, for each $\sigma \in B$, a submanifold map[13] $M : \mathbb{E}^d \to \mathbb{E}^d$ is easily determined, mapping σ into the subspace with implicit equation $x_d = 0$.

Accordingly, for each $\sigma \in B$:

(1) compute an affine transformation M that maps σ into the subspace $x_d = 0$;
(2) consider the set $\mathcal{S}_{d-1} = M(\Sigma)$, with $\Sigma = \{\sigma\} \cup \mathcal{I}(\sigma)$;
(3) compute the arrangement $\mathcal{A}(\Sigma) = X^\sigma_{d-1}$ of the σ cell, including its interior and boundary;
(4) transform back the resulting $(d-1)$-complex to $\mathbb{E}^d$, i.e. compute $M^{-1}(X^\sigma_{d-1})$.

All such $(d-1)$-complexes embedded in $\mathbb{E}^d$ are accumulated in $\mathcal{X}_{d-1} = \{M^{-1}(X^\sigma_{d-1}), \sigma \in B\}$, represented as the proper LAR structure with different indexing of cells, for each fragmented σ. In our current implementation this iteration is parallelized.

4.1.4 Assemble the output $(d-1)$-skeleton. A kd-tree, or k-dimensional tree, is a binary search tree used to organize a a discrete set of k points in a d-dimensional space. Kd-trees are very useful for range and nearest neighbour searches [6]. Here, a kd-tree is computed over the vertices of the collection of complexes $\mathcal{X}_{d-1}$, in order to collapse into a single point all the vertices sharing the same ϵ-neighborhood, with small $\epsilon \in \mathbb{R}$. As a consequence of this reduction of the vertex set, we

- rewrite the representation of each $(d-1)$-cell using the indices of new vertices;
- sort the vertex indices of each $(d-1)$-cell to identify and remove the duplicated $(d-1)$-cells.

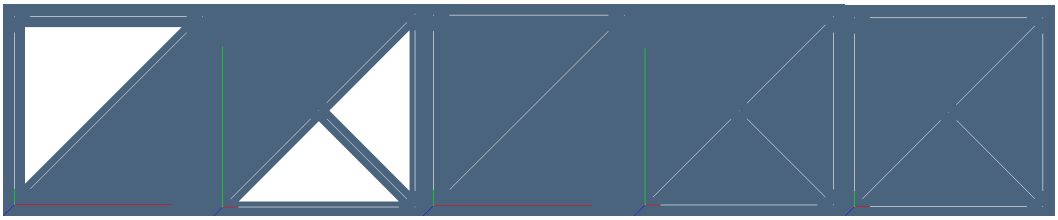
The resulting $(d-1)$-complex in $\mathbb{E}^d$ is the $(d-1)$-skeleton X_{d-1} of the output complex $X = \mathcal{A}(\mathcal{S}_d)$ generated by the input collection $\mathcal{S}_d$. The LAR representation is also reduced in canonical form, where the vertex indices are sorted in each cell.

4.1.5 Validity of output $(d-1)$-skeleton. The merge algorithm in 3D gets any one 2-cell in the input; calculate its 2D arrangement, induced by all possibly incident 2-cells; accumulate all such new 2-cells in a single data array; create the quotient set of 0-cells with respect to closeness; rewrite in canonical form the 1-cells and the 2-cells; eliminates the duplicates so creating the quotient set of 1-cells and 2-cells. When working in 3D it halts here: the output 2-skeleton is so generated.

[13] Submanifold map is a function that maps some submanifold to a coordinate subspace of a chart.

It is unique, since each subset of a partition of output 2-cells is generated uniquely by each single input 2-cell. Of course, a plane polygon, i.e. a 2-cell, intersected twice with the same data, cannot give different partitions (arrangements) of its interior and boundary. It is valid because, by construction, there are no lower dimensional cells within the relative interior of each output 2-cell, or outside from the 2-skeleton as a whole.

Example 4.1 (Merge of two complexes with incompatible boundaries). In this example we show the merge result when the input collection is defined by a pair of abutting complexes with incompatible sub-complexes along their interface. In particular, a 2D example is displayed in Figure 7. Another simple example could be made by two tetrahedralized unit cubes that are incident on a planar face that is triangulated into two triangles but along different diagonals. Similarly to the 2D case (edges), each boundary triangle would be fragmented against all the incident ones, producing fragmented output faces, so that the result on the common affine support (the common vertical plane) would be exactly four triangles, because each input triangle is fragmented by the other diagonal.



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

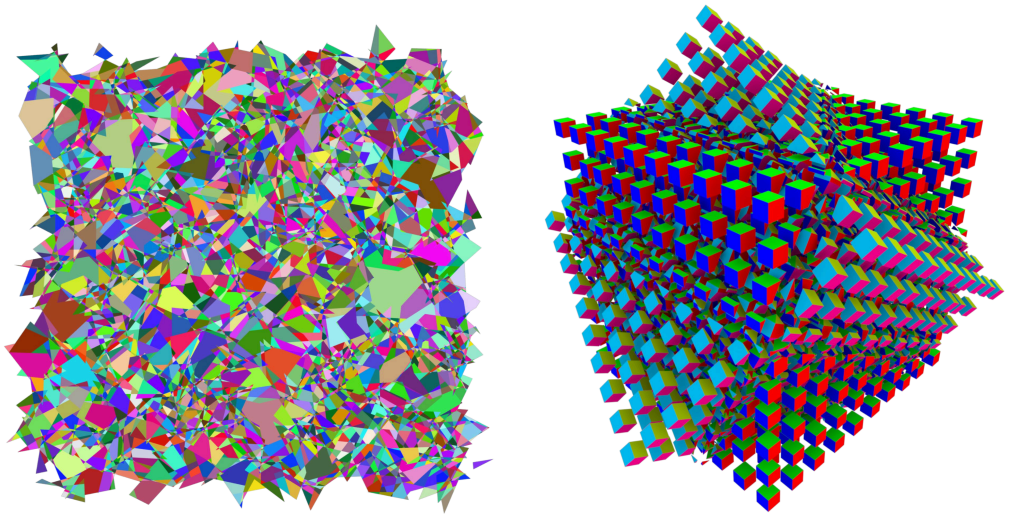
Fig. 7. Merge of two complexes with incompatible boundaries: (a) First input complex (2D); (b) second input complex (2D); (c) 1-skeleton ; (d) incompatible 1-skeleton; (e) 1-skeleton of the 2D arrangement generated by the merge algorithm.

4.1.6 Robustness of the merge algorithm. The regularized arrangement of cellular complexes was (quite) fully implemented in all its parts (see https://github.com/cvdlab/LARLIB.jl), including the merge in 2D and in 3D, and the reader may see some very complex examples in Reference [28]. For discussions of numerical and topological robustness problem in geometric computations, see [16] and [11]. A very simple handling, partially numerical and partially symbolical, of the robustness problem was adopted in our merge algoritm.

The numerical part. The merge of complexes starts with (a) independent fragmentation of $(d-1)$ cells of $(d-1)$-skeleton, generating a regularized arrangement of each of them (see Figure 5b). Then, (b) they are glued to each other in $\mathbb{E}^d$ on the boundary, starting from 0-cells. For this purpose, a k-d-tree of the whole set of vertices is built, and vertices within a small ϵ-disk are identified with each other and given the same integer index.

The symbolic part. Finally, the possibly repeated cells of dimension $p > 0$ are reduced to a single instance. As a matter of fact, it is possible that various input cells (belonging to different complexes) have the same affine support, and a partial or total mutual covering. In such case, after the fragmentation, two or more output cells may totally cover each other. The same happens certainly for the final fragmented cells of lower dimension (on the boundary of input $(d-1)$-cells). So, we reduce each set of p-cells Λ_p $(1 \leq p \leq d-1)$ to its quotient set w.r.t. the total covering relation. For this purpose, (a) each cell is reduced to its canonical form (see Section 4.1.4), and (b) duplicates are removed in each Λ_p set of fragmented cells.

Exact topological results. Topological inconsistencies do not seem to appear in the implementation of the two algorithms discussed in this Section. The generated plane and space arrangements produced by the whole workflow are homeomorphic to the punctured 2-sphere and 3-sphere, respectively[14], in all the examples we have run, since we generate by construction an exact partition of the embedding space. In particular, the implementation always produces the correct Euler characteristics χ (2 for $d = 2$, and 0 for $d = 3$). See Figure 8. For a deeper discussion of the implementation, see [28].



(a) (b)

Fig. 8. Let us recall that the Euler characteristic of the d-sphere is $\chi = 1 + (-1)^d = 2$ or 0 for either even or odd space dimension d: (a) regularized 2D arrangement X_2 of the plane generated by a set of random line segments. The Euler characteristic is $\chi = \chi_0 - \chi_1 + \chi_2 = 11361 - 20813 + 9454 = 2$; (b) merge of two 3-complexes with 2×10^3 3-cells. The Euler characteristic (of the non-exploded resulting 3-complex) is $\chi = \chi_0 - \chi_1 + \chi_2 - \chi_3 = 8787 - 26732 + 26600 - 8655 = 0$. Of course, the count includes the exterior unbounded 2-cell or 3-cell, respectively, that are actually computed by the topological gift wrapping.

4.2 Space arrangement via topological gift wrapping

4.2.1 Extraction of d-cells from $(d-1)$-skeleton. At the beginning of this stage of the algorithmic reconstruction of the space arrangement induced by a collection $\mathcal{S}$ of cellular complexes, we have a complete [44] representation in $\mathbb{E}^d$ of the skeleton X_{d-1} of $\mathcal{A}(\mathcal{S})$.

Let us consider the boundary map $\partial_d : C_d \to C_{d-1}$, with $\partial_d = \delta_{d-1}^\top$. Each column in ∂_d is the representation of one element of the C_d basis (i.e., a d-cell in X_d), in the basis of C_{d-1}, i.e., as a linear combination of the cells in X_{d-1}. The construction of d-cells is carried out by using two tools:

- the property that every $(d-1)$-cell in a d-complex is shared at most by two d-cells [31]. Note that if the complex includes also the exterior (unbounded) cell, then such incidence relation between cells in X_{d-1} and in X_d holds exactly;
- the computation of the cyclic subgroups of permutations of $(d-1)$-cells around their common $(d-2)$-cell, which implies the existence of a *next* and *prev* $= next^{-1}$ bijections on such subsets.

[14]The 2-space $\mathbb{R}^2$ is homeomorphic the the punctured 2-sphere S^2 via the stereographic projection. This one works also in superior dimensions, hence the claim holds in any dimension.

Such functions are computed by calculating the cyclic ordering (on angles) of the coboundary 1-cells of a 0-cell around the 0-cell, as well as the ordering of the coboundary 2-cells of a 1-cell around the 1-cell itself, respectively in the 2D case and in 3D case (see Figures 11 and 9). Specifically, the orientation of single coboundary elements is that provided by the matrix multiplication operator.

Cyclic subgroups δc, with elementary (singleton) $c \in C_{d-2}$, have the following properties:

(1) are in one-to-one correspondence with subsets of cells in X_{d-2},
(2) $next(-c) = next^{-1}(c) = prev(c)$.

Since we mostly deal with oriented chain complexes, we normally want to build a coherently oriented cellular complex X from the arrangement $\mathcal{A}(\mathcal{S}_d)$, enforcing the condition that every (d-1)-cell shared by two d-cells in X_d should have opposite orientations on them. The construction of d-cells is done by computing the columns of the $[\partial_d^+]$ matrix (see Section 5.1 and Algorithm 1). Hence the reorientation of a d-cell just requires to invert the sign of non-zero coefficients of its sparse column, or (symbolically) to multiply the column index times a -1 coefficient.

4.2.2 d-cells are bounded by minimal $(d-1)$-cycles. A chain $c \in C(X)$ is said to be a cycle if $\partial c = \emptyset$. A formal algorithm for extraction of d-cycles starting from $(d-1)$-chains is given in Section 5.1. In particular, we generate the linear representation of a basis d-cell (more exactly: of a singleton d-chain, element of a C_d basis) as minimal combination of coherently oriented $(d-1)$-chains.

We say that a basis of d-chains is canonical (or minimal) when the sum of cardinalities of its cycles, as combinations of elementary (d-1)-chains with non-zero coefficients, equals the double of the number of elementary (d-1)-chains. In other terms, if we denote the cardinality of a p-basis as $\#X_p$, we have, for the matrix $[\partial_d^+] = (a_{ij})$ with $a_{ij} \in \{-1, 0, 1\}$, where $[\partial_d^+]$ denotes the matrix including the exterior unbounded cell, i.e., its column representation:

$$\sum_{i=1}^{\#X_{d-1}} \sum_{j=1}^{\#X_d} |a_{ij}| = 2\,(\#X_{d-1}). \tag{4}$$

Note that every facet ($(d$-1)-cell) is used exactly twice in constructing the rows of $[\partial_d^+]$. This property, well known in solid modeling, is normally represented [50] as an identity between the cardinalities of incidence relations between vertices, edges and faces in 3D boundary representations and/or on graphs drawn on a 2-manifold.

4.2.3 Topological gift wrapping. The algorithm discussed in this section reminds of the "gift-wrapping" algorithm for computing convex hulls [17, 35] from sets of points. The topology-based algorithm given here is a broad generalization, since it may generate non-convex d-cells.

In particular, we start a $(d-1)$-cycle c from a single $(d-1)$-cell and update c iteratively by summing to it a suitable $(d-1)$-chain next(∂c), until c becomes closed. See Examples 6.1, 6.2 and Section 5.1. The minimality of each basis d-cell is guaranteed in Algorithm 1 by the repeated application of the map

$$corolla : C_{d-1} \to C_{d-1},$$

that takes as input a $(d-1)$-chain c, starting from an initial "seed" $(d-1)$-cell, and returns a subset of "petals" from a "corolla" (see Figures 9c, 9e, 9f) of boundary cells at each repeated application, until $\partial c = \emptyset$. In formulas, we may write

$$corolla(c) = c + \text{next}(\,\partial c\,) = c + \sum_{\tau \in \partial c} \text{next}\big((\delta\tau) \cap c\big).$$

In words, a basis element of C_d, represented as a minimal $(d-1)$-cycle in C_{d-1}, is generated from X_{d-1} by: (i) choosing a cell $\sigma \in X_{d-1}$ and setting $c := 1\,\sigma$, then (ii) by repeating $c := c + corolla(c)$, until (iii) $\partial c = \emptyset$. See Figure 9 and Algorithm 1.

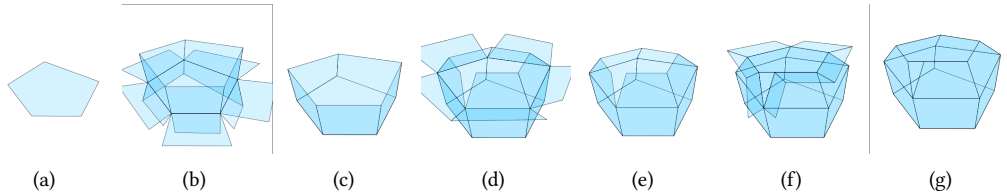


Fig. 9. Extraction of a minimal 2-cycle from $\mathcal{A}(X_2)$: (a) 0-th value for $c \in C_2$; (b) cyclic subgroups on $\delta\partial c$; (c) 1-st value of c; (d) cyclic subgroups on $\delta\partial c$; (e) 2-nd value of c; (f) cyclic subgroups on $\delta\partial c$; (g) 3-rd value of c, such that $\partial c = 0$, hence stop.

4.2.4 Domain of valid input. The algorithm cannot work with any input. In particular, the input $(d-1)$-skeletons must be regular, i.e. without dangling parts, so that every $(d-1)$-cell belongs at most to two $(d-1)$-cycles. In 2D, this fact is guaranteed by applying the algorithm separately to each 2-maximally connected component of the 1-skeleton, considered as a graph, and then by merging the results (clearly disconnected). Analogously, in 3D, the adjacency graph of 2-cells should not contain dangling parts.

The validity set of the input contains the 2-skeleton of a 3-complex, and the boundary of a solid model, and the set of manifold components of the boundary of a non-manifold solid model. Of course, to apply the algorithm to data which do not determine a partition of the embedding space does not make sense[15] and produces an empty result.

When applied to correct input, as described above, the algorithm is always valid, because always produces an (augmented) basis of C_d that satisfies Eq. 4. The results are also unique, since otherwise two different bases for the linear space C_d would produce two boundary operators that, applied to the total 3-chain (vector of all ones) would return the same exterior cell, which is not possible.

There are no ambiguities in the algorithm, since in a d-complex every two d-cells either share at most two $(d-1)$-cells, or exactly two if the exterior unbounded d-cell is considered. Also, it halts when this last condition is exactly reached. Note that the suitable choice of the next petals from the corolla implies that a 2-cell cannot be used more that twice.

4.2.5 Cyclic subgroups of the $\delta\partial c$ chain. In computing the representation $c \in C_{d-1}$ of a C_d basis element, we iterate over "corollas" — $(d-1)$-chains — of coherently oriented cells in X_{d-1}, generated by the coboundary $\delta\partial c \in C_{d-1}$ of a cycle $\partial c \in C_{d-2}$, with $n = \#\delta\partial c$ and $m = \#\partial c$. In the following we iterate over subsets of "petals" associated to each element of ∂c.

Let us consider the group S_n of permutations of elements of $\delta\partial c$. By linearity of δ, the group elements can be partitioned into m cyclic subgroups $R^k \subset X_{d-1}$ of "petals", one-to-one mapped to each element $\tau \in \partial c$, with $\sum_{k=1}^{m} \#R^k = \#\delta\partial c = n$. In the concrete case with $d = 2$ of Example 6.1, such subgroups are represented in Figures 11b and 11d by arrowed circles.

It is easy to prove that each R_k contains one and only one c component, computable as $\sigma = R_k \cap c$, so that each $corolla(c)$ application to c adds to such (d-1)-chain exactly m cells, each one generated either by $\underline{next}(\sigma)(R_k)$ or by $\underline{prev}(\sigma)(R_k)$, depending on the orientation of each "hinge" cell $\tau \in C_{d-2}$ with respect to the orientation of ∂c, codified by either a $+1$ or a -1 coefficient in ∂c. It is worth stressing that the chains producing the $c \in C_{d-1}$ cycle (see Example 6.1) are computed by sparse

[15] Like, e.g., an open 2-manifold surface.

matrix[16] multiplications times sparse vectors, and not by traversal of the data structure representing the cellular complex. This feature renders the present algorithm not directly comparable with the others documented in the literature.

The computation of the minimal boundary $(d-1)$-cycle of an elementary d-chain is shown for $d = 2$ in Example 6.1 and Figures 11 and 12. The above description also holds for higher values of d, and in particular in 3-space, where the cyclic subgroups are around $(d-2)$-cells—see Figures 9b,d,f .

4.2.6 Linear independence of the extracted cycles. First remind that, with abuse of language, we often use cell as synonym of singleton chain, element of a basis of a chain space.

Therefore, when constructing the matrix of a linear operator between linear chain spaces, say $\partial_d : C_d \longrightarrow C_{d-1}$, by building it column-wise, we actually construct an element of the basis of the domain space, represented as a linear combination of basis elements of the target space. In this sense we "build" or "extract" one d-cell as a $(d-1)$-cycle. The extraction of the LAR representation (as a set of vertices) of a d-cell is then completed, by union of the corresponding rows of the characteristic matrix M_{d-1}, finally getting the list of vertices of the "built" or "extracted" d-cell.

4.2.7 Removal of the exterior d-cycle. Now, the $(d-1)$-cycles generated in Section 4.2.2 to construct the basis d-cells are not linearly independent, since one of them, and in particular the external unbounded cycle, is computable as the sum of all the other ones [40]. To remove the external cycle from the basis requires a dedicated test. We find in linear time the vertices having an extremal (max or min) value of one coordinate, and select for each extremum vertex the incident subset of $(d-1)$-cycles, i.e., the incident subset of $[\partial_d^+]$ columns. Their intersection will necessary contain only the exterior cell. In the worst case—the wildest non convex cells, having one vertex in all extrema positions—it might be necessary to compute the absolute value of the signed volume of cells [7, 12]. The cell with the largest volume is then removed.

4.2.8 Poset of isolated boundary cycles. When the d-boundary operator, applied to the chain sum of all basis d-cells — called total d-chain in this paper — produces a disconnected chain of $(d-1)$-cycles, these n isolated boundary components have to be compared with each other, to determine the possible relative containment and consequently their orientation. To this purpose an efficient point classification algorithm is used, in order to compute the poset (partially ordered set) induced by the containment relation of isolated components.

The $n \times n$ binary and antisymmetric matrix $M = (m_{ij})$ of the containment relation between the external cycles of the n connected components is constructed, computing each element of the matrix with a single point-cycle containment test, because the two corresponding cycles (the one associated to row i of M, from which the vertex point is extracted, and the one associated to column j) certainly do not intersect. Then the attribute of each cycle c_j as either external or internal (and hence its relative orientation) is given by the parity of $\sum_{p=1}^{n} m_{pj}$. Further details on this point and pseudocode are given in Section 5.2 and Algorithm 2, respectively.

4.3 Base case: arrangement of lines in $\mathbb{E}^2$

The actual base case in $\mathbb{E}^2$, starting from a collection $\mathcal{L}$ of 1D complexes, i.e., from a collection of line segments, is discussed in depth in this section. Example 6.3 and Figure 13 illustrate this point. A similar procedure, using the bipartite graph of incidence between vertices and cells, is used in higher dimensions to regularize the result, i.e., to remove the higher dimensional dangling parts.

[16] The boundary and coboundary operators ∂_2 and $\delta_1 = \partial_2^\top$ are computed once and for all before the algorithm application. In the actual implementation, only one matrix is mantained, and it is alternatively post- and pre-multiplicated.

4.3.1 Segment subdivision: linear graph. Each input line segment in $\mathcal{L}$ is subdivided against all the intersecting segments, detected using a spatial indexing based on interval trees (Figure 13b). Each generated sub-segment produces a pair of unique vertices. Quasi-coincident vertices—i.e. very close numerically— are finally identified, and a single 1-complex, stored as vertices and edges in a complex X_1, is created. Remember that a cellular 1-complex is just a graph, and the generation of line sub-segments (graph edges) is embarrassingly parallel.

4.3.2 Maximal 2-point-connected subgraphs. The X_1 complex is reduced to the union of its maximal 2-point-connected subgraphs, discovered by using the [34] algorithm, in order to remove all dangling edges and tree subgraphs. As a consequence, $\mathcal{A}(X_1)$ generates a partition of $\mathbb{E}^2$ with open, connected, and regularized, but still undetected, 2-cells (Figure 13c) to be computed in the next step.

4.3.3 Topological extraction of X_2. The topological gift-wrapping algorithm in 2D is used for computation of the 1-chain representation of cells in X_2, providing also the signed ∂_2 and δ_1 operators, is discussed in Section 4.2.2, and shown in Figures 11 and 12. In Figures 13d and 13e we show the extraction of 2-cell elements in X_2, starting from a random set of line segments.

4.4 Comments

Our approach is embarrassingly parallel, since no effort is spent to separate the problem into a number of independent tasks: if B_2 is the collection of 2-cells in $\mathcal{S}$, with $n_2 = \#B_2$, we have n_2 independent tasks for the computation of planar arrangements X_2^σ, $\sigma \in B_2$.

For example, given two complexes in $\mathbb{E}^3$, intersect all their polygonal facets with each possibly intersecting facet, to produce the 0-, 1-, and 2-cells of their planar arrangements, independently from each other. Then, sort the symbolic representations of 2-cells (as lists of indices of 1-cells) both internally and externally, to discover quotient sets and to compute X_2 in $\mathbb{E}^3$. Finally, extract a basis of 3-cells for the linear space C_3 and compute the (co)boundary operators $\partial_3 \equiv \delta_2^\top$.

All geometric computations on each target facet are essentially two-dimensional, since the focus, after a suitable affine map, is on partitioning the $z = 0$ plane against the affine hulls of incident polygons, or, more precisely, against the line segments generated by intersecting $z = 0$, so always reducing the problem to the arrangement of line segments in 2D (see Figure 13).

If computing would entail only k-planes, e.g. lines in 2D and planes in 3D, then all cells in their arrangement would be connected and convex [9], leading to straightforward geometric and topological computations. Conversely, in our case, because the arrangements are generated by a finite number of geometric objects, the resulting cellular complex may contain general non convex cells, possibly multiply-connected, and needs more complicated algorithms for inclusion of hole vertices in LAR cells, that we remind are just sorted sets of vertex indices. A Constrained Delaunay Triangulation [14, 46] of every 2-cell in 2D is performed in our implementation, using the Shewchuk's Triangle library both for correct computation of the permutation subgroups 4.2.5, and for visualization of non convex cells.

5 PSEUDOCODES AND COMPLEXITY

Here we provide a slightly simplified pseudocode[17] of some algorithms presented in the previous sections, and discuss their worst case complexity. The used pseudocode style is a blend of Python and Julia. Of course, the accumulated assignment statement A += B stands for $A = A + B$, where the meaning of "+" symbol depends on the context, e.g. may stand either for sum (of chains), or for union (of sets), or for concatenation (of matrix columns). Analogously, A -= B stands for $A = A - B$.

[17]The actual implementation, extremely similar to this pseudocode, can by found in https://github.com/cvdlab/LARLIB.jl.

5.1 Signed boundary $\partial_d : C_d \rightarrow C_{d-1}$ computation

This section deals with the pseudo-coded Algorithm 1, which describes the generation of the signed matrix $[\partial_d]$ of the boundary operator that maps C_d into C_{d-1}, whose construction was discussed in Sections 4.2.1–4.2.5. Algorithm 1 is written in a dimension-independent way, and works for both $d = 2$ and $d = 3$. The reader may find beneficial to mentally map $(d, d-1, d-2)$ onto $(2, 1, 0)$ or onto $(3, 2, 1)$, according to Figures 11 or 9, respectively.

Some preliminary words about pseudocode notations: we use greek letters for the cells of a space partition, and latin letters for chains of cells, all actually coded in LAR as either signed integers or arrays of signed integers. $[\partial_d]$ or $[c_d]$ stand for general matrices or column matrices, respectively, whereas $\partial_d[h, k]$ or $c_d[\sigma]$ stand for their indexed elements. Also, $\{|c_d|\}$ stands for the set of unsigned (nonzero) indices of the (sparse) array $[c_d]$. It may be also useful to recall that column $\partial_d[\cdot, \sigma]$ of the operator matrix is the chain representation of the d-cell σ of C_d basis, written by using the C_{d-1} basis, i.e. as linear combination of $(d-1)$-cells of the space partition.

Note the precondition, warning that the algorithm can only be used to compute the ∂_d matrix for a cell decomposition of a d-space. In fact, only in this case the $(d-1)$-cells are shared by exactly two d-cells, including the exterior cell. The termination predicate is a consequence of this property: the algorithm terminates when all incidence numbers in the *marks* array equal 2, so that their sum is exactly $2n$, where n is the number of $(d-1)$-cells. Of course, the actual implementation in scientific languages like Python or Julia uses sparse arrays and coordinates in $\{-1, 0, 1\}$ to achieve an actual efficient execution in storage space and computation time.

5.1.1 Complexity of 3-cells extraction. In three dimensions, Algorithm 1 constructs one basis 3-cell at a time (as a 2-cycle, i.e., as a closed 2-chain), building the corresponding column of the matrix $[\partial_3^+]$, so including one more boundary column for each connected component of the output complex, as detailed in Section 5.2. The following Algorithm 2 operates on $[\partial_3^+]$, used to actually generate the operator matrix $[\partial_3]$.

The complexity of a 3-cell is measured by a set of triples (row, column, value) for non-zero values[18], associated with its cycle of 2-cells, where each 2-cell is always shared by two 3-cells. Hence the total number of triples, i.e. the space complexity of the COO representation of $[\partial_3^+]$, is exactly $2n$, where n is the number of 2-cells in the X_2 skeleton.

The construction of a single 3-cell requires the search of the adjacent *adj* 2-cell for each *pivot* 2-cell in the boundary shell. The search for *next* or *prev* 2-cell as *adj* requires the circular sorting of each permutation subgroup of 2-cells incident to each 1-cell on each boundary of an incomplete 2-cycle. Consequently, we need a total number of sorts of small sets (normally bounded by a small integer—4 for cubical 3-complexes—hence $O(1)$ timewise) that is bounded by the number of 1-cells.

The subsets to be sorted are encoded in the rows of the incidence relation between 1-cells and 2-cells, i.e., by the i, j indices of non-zero elements of $[\partial_2]$. The computation of (unsigned) $[\partial_2]$ is in turn performed through *SpMSpM* multiplication of two sparse matrices (see [22]), and hence in time linear with the complexity of the output, i.e., in the number of non-zero elements of the $[\partial_2]$ matrix. Summing up, if n is the number of d-cells and m is the number of $(d-1)$-cells, the time complexity of this algorithm is $O(nm \log m)$ in the worst case of unbounded complexity of d-cells, and roughly $O(nk \log k)$ if their face complexity is bounded by k.

5.2 Managements of contents of non-intersecting shells

The cellular 3-complexes considered in this paper may contain cells with a number of holes, and/or inclusions of smaller cells and sub-complexes. In the general case, deep hierarchies of inclusion are

[18] The coordinate (COO) representation of sparse matrices [10] is an array of triples (i, j, v).

ALGORITHM 1: Computation of signed $[\partial_d^+]$ matrix

```
/* Pre-condition: d equal to space dimension, s.t. (d − 1)-cells are shared by two d-cells */
/*                                                                                        */
Input: [∂_{d−1}]  # Compressed Sparse Column (CSC) signed matrix (a_ij), where a_ij ∈ {−1, 0, 1}
Output: [∂_d^+]  # CSC signed matrix
[∂_d^+] = [] ; m, n = [∂_{d−1}].shape ; marks = Zeros(n)    # initializations
while Sum(marks) < 2n do
    σ = Choose(marks)          # select the (d − 1)-cell seed of the column extraction
    if marks[σ] == 0 then [c_{d−1}] = [σ]
    else if marks[σ] == 1 then [c_{d−1}] = [−σ]
    [c_{d−2}] = [∂_{d−1}] [c_{d−1}]          # compute boundary c_{d−2} of seed cell
    while [c_{d−2}] ≠ [] do          # loop until boundary becomes empty
        corolla = []
        for τ ∈ c_{d−2} do          # for each "hinge" τ cell
            [b_{d−1}] = [τ]^t [∂_{d−1}]          # compute the τ coboundary
            pivot = {|b_{d−1}|} ∩ {|c_{d−1}|}          # compute the τ support
            if τ > 0 then adj = Next(pivot, Ord(b_{d−1}))          # compute the new adj cell
            else if τ < 0 then adj = Prev(pivot, Ord(b_{d−1}))
            if ∂_{d−1}[τ, adj] ≠ ∂_{d−1}[τ, pivot] then corolla[adj] = c_{d−1}[pivot]          # orient adj
            else corolla[adj] = −(c_{d−1}[pivot])
        end
        [c_{d−1}] += corolla          # insert corolla cells in current c_{d−1}
        [c_{d−2}] = [∂_{d−1}] [c_{d−1}]          # compute again the boundary of c_{d−1}
    end
    for σ ∈ c_{d−1} do marks[σ] += 1          # update the counters of used cells
    [∂_d^+] += [c_{d−1}]          # append a new column to [∂_d^+]
end
return [∂_d^+]
```

possible. A containment relation R between isolated boundary cycles, called shells in the following, must be handled. For this purpose we detect the shells, then discover the whole inclusion graph between shells, and compute its transitive reduction, i.e. the smallest relation having the transitive closure of R as its transitive closure [1].

An example of the transitive reduction tree of R is given in Figure 10b for a 2D case. When $d = 3$, first we compute the set of disjoint parts of the X_2 complex in $\mathbb{E}^3$, given by the connected subgraphs, called components, of the FV relation. For each connected component of X_2, we extract the $[\partial_3^+]$ matrix and remove from it the column that corresponds to the boundary of the component, i.e., to its exterior unbounded cell. Note that each such "isolated" component defines a connected boundary shell. We have to detect their relative containment, and possibly remove some cycles from the boundary operator matrix, so implicitly moving the cycle to the (disconnected) boundary of the exterior cell.

5.2.1 Preview of algorithm. We need to consider two main concepts here: (i) the maximal connected components of X_{d-1}, producing disconnected d-components of the output complex X_d; (ii) the possible inclusions of components within single cells of the output d-complex. We list in the following the main stages of the algorithm. Our goal is the computation of both the X_d skeleton, and

the ∂_d operator. In other words, we "extract" the d-cells of a cellular complex from the knowledge of its X_{d-1} skeleton:

(1) First, we have to split the (isolated) connected components of the input X_{d-1} skeleton, using its LAR representation;
(2) For each (connected) component p of $(d-1)$-skeleton $(1 \leq p \leq h)$ compute the matrix $[\partial_d^+]^p$ and its boundary cycle, and remove it from the matrix, moving into a shell array D;
(3) compute the containment relation graph between shells in D, and the tree T of its transitive reduction R.
(4) for each arc $(c^i, c^j) \in T$ with even distance from the root, look for the d-cell ρ of the component with bounding shell c^j at minimal distance from the shell c^i, and remove the ρ's bounding cycle from the component matrix $[\partial_d]^j$, i.e. declare ρ is empty.
(5) produce the final $[\partial_d]$ matrix, by concatenation of component matrices $[\partial_d]^p$, $(1 \leq p \leq h)$, and the LAR representation of X_d.

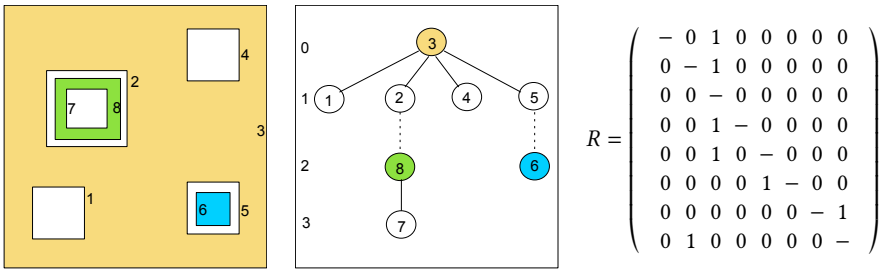


Fig. 10. Non intersecting cycles within a 2D cellular complex with three connected components and only three cells, denoted by the image colors: (a) cellular complex; (b) graph of the reduced containment relation R between shells, with dashed arcs of even depth index; (c) matrix of transitively reduced R. Note that the ones equate the number of edges in the graph.

5.2.2 Pseudocode discussion. For the sake of clarity, this subsection discusses the $d = 3$ case.

Consider the bipartite graph $G = (N, A)$, with $N = \Lambda_2 \cup \Lambda_0$, and $A \subseteq \Lambda_2 \times \Lambda_0$, associated with the sparse matrix encoding the FV relation. G has one node for each facet (2-cell), one node for each vertex (0-cell), and one arc for each incident pair. Therefore, the arcs in A are in one-to-one correspondence with the nonzero elements of the FV matrix. By computing the maximal 2-connected components of G, we subdivide the X_2 skeleton into h connected components: $\mathcal{X}_2 = \{X_2^p\}, 1 \leq p \leq h$.

For each component X_2^p, repeat the following actions. First assemble the $[\partial_2]^p$ sparse matrix, and compute the corresponding $[\partial_3^+]^p$ generated by Algorithm 1. Then split it into the boundary operator $\partial_3^p : C_3^p \to C_2$ and the column matrix $c^p = \partial_3^+[\sigma^p] \in C_2$ of the exterior cell $\sigma^p \in \Lambda_3$. The set $D = \{c^p\}$ of h disjoint 2-cycles, is the initialization of the set of X_d shells. Some further (empty) shells of X_d can be discovered later, resulting from mutual containment of D elements.

Then we compute the transitive reduction R of the point-containment relation between shells in D, by computing for $i, j \in range(h), i < j$, the containment test $pointSet(v, c^j)$ between any vertex $v \in c^i$ and the cycle c^j, with $(c^i, c^j) \in R$. If the edge set of the graph of R is empty, then no disjoint component of X_3 is contained inside another one, and both X_3 and ∂_3 may by assembled by disjoint union of cells of X_3^p and columns of $[\partial_3]^p$, respectively, for $1 \leq p \leq h$.

If conversely the above is not true, then for each arc (i, j) in the tree of R, with odd distance of i node from the root, it is necessary to discover which cell of the container component X^j actually

ALGORITHM 2: Non-intersecting shells

Input: LAR_{d-1}, $[\partial_{d-1}]$ # for $d = 3$: FV, ∂_2
Output: LAR_d, $[\partial_d]$ # for $d = 3$: CV, ∂_3
$N = \Lambda_2 \cup \Lambda_0$; $A \subseteq \Lambda_2 \times \Lambda_0$; $G = (N, A)$ # initializations
$\mathcal{G} = \{G^p \mid 1 \le p \le h\} \leftarrow ConnectedComponents(G)$ # partition of G into h connected components
$\mathcal{X}_{d-1} = \{(X^p_{d-1}, \partial^p_{d-1}) \mid 1 \le p \le h\} \leftarrow Rearrange(\mathcal{G})$ # partition of X_{d-1} into h connected components
$D = []$ # initialize the sparse array of shells
for $p \in \{1, \ldots, h\}$ **do** # for each connected component of $(d-1)$-skeleton
  $[\partial^+_d]^p = Algorithm\_1([\partial_{d-1}]^p)$ # compute the minimal d-cycles of a component of complex
  $(c^p, \partial^p_d) = Split([\partial^+_d]^p)$ # split the component into the exterior $(d-1)$-cycle and the boundary ∂^p_d
  D += $[c]^p$ # append the boundary shell to the shell array
end
for $i, j \in \{1, \ldots, h\}, i < j$ **do** # for each shell pair $(c^i, c^j) \in R$,
  $(R[i,j], R[j,i]) :: Bool \times Bool \leftarrow PointSet(v \in c^i, c^j)$ # containment test of v in c^j
end
$T = \{(i,j)\} \leftarrow Tree(TransitiveReduction(R))$ # set of arcs of reduced containment tree of shells
if $T \neq \emptyset$ **then** # if the containment tree of shells is not empty
  for $(i,j) \in T$ **do** # for each shell pair (c^i, c^j) such that $dist(c^j)\%2$!= 0
    $\rho = FindContainerCell(v, c^j, \mathsf{LAR}_{d-1})$ # look for a d-cell ρ such that $v \in |c^i| \subseteq |\rho| \subseteq |c^j|$
    $[\partial_d]^j$ -= $\partial^j_d[\rho]$ # remove ρ from ∂^j_d
  end
end
$\partial_d = [\partial^1_d \cdots \partial^p_d \cdots \partial^h_d]$ # return the aggregate ∂_d operator
$\mathsf{LAR}_d = [\cup_k \mathsf{LAR}_{d-1}(c^k = \partial_d[\cdot, k]),\ for\ k \in Range(Cols(\partial_d))]$ # for $d = 3$: LAR_d = CV
return LAR_d, $[\partial_d]$

contains the contained component X^i, i.e., its shell c^i. More complex intersecting situations are not possible by construction, since we know that the components are disjoint. Therefore, in case of containment, one component is necessarily contained in some empty cell of the other.

To identify the 3-cell ρ that is contained within a 2-cycle $c^j \in D$ and in turn contains a 0-cell $v \in c^i \in D$, and hence the whole c^i, is not difficult. It is achieved by shooting a ray (halfline) in any direction from v (e.g., in positive x_1 direction) against the 2-cells in X^j_2, i.e., the appropriate subset of rows of the sparse matrix FV, and ordering parametrically the resulting intersection points, and hence the 2-cells of X^j_2 intersecting the ray. The closest 2-cell will be shared by two 3-cells: the 3-cell $\rho \in X^j_3$ that does contain v, and the one that does not.

Of course, ρ is the empty cell of X^j_3 that contains the whole X^i_3. As a consequence, the column $\partial^j_3[\rho]$ has to be removed from $[\partial^j_3]$. We can check that (implicitly) the 2-cycle $\partial_3\rho$ is automatically added, with reversed coefficients, to the boundary of X^j_3, i.e., to the boundary cycle $[\partial_3]^j[c]^j \in C_2$ of the solid component X^j_3. Note that the resulting boundary is non connected.

When the above cancellation of empty cells has been performed for all arcs of the relation tree, the new simplified matrices $[\partial_3]^p$ $(1 \le p \le h)$ can be assembled into the final ∂_3 operator matrix, whose column 2-cycles, suitably transformed into the union of corresponding LAR representations (subsets of vertices) of 2-cells, will provide the LAR representation of 3-cells in X_3.

Example 5.1. Consider the example in Figure 10, where $d = 2$. In this case, by analysing the connection of $X_{d-1} = X_1$, we obtain 8 connected subgraphs. If we had directly applied Algorithm 1

to the whole data set X_1, i.e. to the whole matrix $[\partial_1]$, then we would have generated a matrix $[\partial_2^+]$ with dimension $e \times 16$, where e is the number of edges and 8 is the number of shells. In fact, in this case every edge belongs to two minimal adjacent 1-cycles (shells from connected components of X_1 graph and their exterior cycles).

According to the discussion above, about preliminary extraction of connected d-components, and to Algorithm 2, we actually extract from X_1 8 shells, then we consider the 8×8 matrix of the transitively reduced containment relation R. This one is antisymmetric, and we do not care about the reflexive part (see Figure 10c). The final boundary matrix $[\partial_2]$ gets size $e \times 3$, after the restructuring induced by the containment relation R of shells, and considering also the parity of nodes of the tree of R, that determines the alternation between full and empty spaces.

The resulting boundary matrix corresponds to a 2-complex with three 2-cells, denoted as σ_2^1 (yellow), σ_2^2 (green), and σ_2^3 (blue) in Figures 10a and 10b. Let us note that we have reduced the set of 8 isolated shells to three non-reducible 2-cells (in this example coincident one-to-one with isolated 2-components of the input complex).

5.2.3 Complexity of shell management. The computation of the connected components of a graph G can be performed in linear time [34]. The recognition of the h shells requires the computation of $[\partial_d^+]^p$ $(1 \leq p \leq h)$ and the extraction of the boundary of each connected component X_d^p . To compute the reduced relation R we execute $h^2/2$ point-cycle containment tests, linear in the size of a cycle, so spending a time $O(h^2\, k)$, where h is the number of shells, and k is the average size of cycles. Actually, the point-cycle containment test can be easily computed in parallel, with a minimal transmission overhead of the arguments. The restructuring of boundary submatrices has the same cost of the read/rewrite of columns of a sparse matrix, depending on the number of non-zeros of $[\partial_3]$, and hence is $O(k\, \#\Lambda_d)$, i.e., linear with the product of the number of d-cells $\#\Lambda_d$ and their average size k as chains of $(d-1)$-cells. The parameter k is a small constant for simplicial and cubical complexes.

5.3 Subdivision of 2-cells

Algorithm 3 was already introduced in Section 4.3 for the basic case concerning the arrangement of lines in $\mathbb{E}^2$. We give in this section the pseudocode and a more detailed exposition, relative to the handling of data elements producing a subdivision of a $(d-1)$-cell σ. Of course, when $d = 3$, we have $\mathcal{S}_{d-1} = \mathcal{S}_2$, and only this case is discussed here. In the multidimensional case, after that the 2-cells have been topologically extracted from the $(d-1)$-cells, Algorithm 3 still works, reducing to the arrangement of the 2-cell σ induced by a soup $\mathcal{S}_1$ of 1-complexes, i.e., of line segments, as shown in Figures 4d, 5a, and 5b, as well as in the more complex Example 6.3 and Figure 13.

5.3.1 Complexity of 2-cells subdivision. The time complexity of Algorithm 3 is given by the number of 2-cells times the worst case cost required by the subdivision of one of them. In turn this depends on the size of the actual input, i.e., on the number of 2-cells possibly intersecting each other. It is fair to say that, in all the regular cases we usually meet in computer graphics, CAD meshes and engineering applications, the number of 2-cells incident on (even on the boundaries of) a fixed one is bounded by a constant number k_1. If k_2 is the maximum number of 1-cells on the boundary of a 2-cell, then the whole computation of Algorithm 3 requires time $O(k_1 k_2 n + A)$, where n is the number of the 2-cells in the input, and A is the time needed to glue all the $X_2(\sigma)$ in $\mathbb{E}^d$ space. When $d = 3$, the affine transformations of each set Σ (see Section 4.1.3) are computable in $O(1)$ time; building a static kd-tree generated by m points requires $O(m \log^2 m)$; and each query for finding the nearest neighbor in a balanced kd-tree requires $O(\log m)$ time on average. The number of occurrences of the same vertex on incident 2-cells is certainly bounded by a small constant

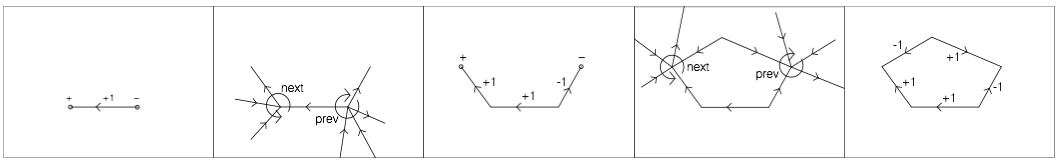
ALGORITHM 3: Subdivision of 2-cells

Input: $\mathcal{S}_2 \subset \mathcal{S}_{d-1}$ # collection of all 2-cells from $\mathcal{S}_{d-1}$ input in $\mathbb{E}^d$
Output: $[\partial_2]$ # CSC signed matrix
$\widetilde{\mathcal{S}_2} = \emptyset$ # initialisation of collection of local fragments
for $\sigma \in \mathcal{S}_2$ **do** # for each 2-cell σ in the input set
  $M = SubManifoldMap(\sigma)$ # affine transform s.t. $\sigma \mapsto x_3 = 0$ subspace
  $\Sigma = M\,\mathcal{I}(\sigma)$ # apply the transformation to (possible) incidencies to σ
  $\mathcal{S}_1(\sigma) = \emptyset$ # collection of line segments in $x_3 = 0$
  for $\tau \in \Sigma$ **do** # for each 2-cell τ in Σ
    $\mathcal{P}(\tau), \mathcal{L}(\tau) = \emptyset, \emptyset$ # intersection points and int. segment(s) with $x_3 = 0$
    for $\lambda \in X_1(\tau)$ **do** # for each 1-cell λ in $X_1(\tau)$
      if $\lambda \not\subset \{\mathbf{q} \mid x_3(\mathbf{q}) = 0\}$ **then** $\mathcal{P}(\tau)$ += $\{\mathbf{p}\}$ # append the intersection point of λ with $x_3 = 0$
    end
    $\mathcal{L}(\tau) = Points2Segments(\mathcal{P}(\tau))$ # Compute a set of collinear intersection segments
    $\mathcal{S}_1(\sigma)$ += $\mathcal{L}(\tau)$ # accumulate intersection segments with σ generated by τ
  end
  $X_2(\sigma) = \mathcal{A}(\mathcal{S}_1(\sigma))$ # arrangement of σ space induced by a soup of 1-complexes
  $\widetilde{\mathcal{S}_2}$ += $M^{-1}\,X_2$ # accumulate local fragments, back transformed in $\mathbb{E}^d$
end
$[\partial_1] = QuotientBases(\widetilde{\mathcal{S}_2})$ # identification of 0- and 1-cells using kd-trees and canonical LAR
$[\partial_2] = Algorithm\_1([\partial_1])$ # output computation
return $[\partial_2]$

k_3, approximately equal to m/v, where $v = \#X_0$ is the number of 0-cells after the identification processing. The transformation of output LAR in canonical form (sorted 1-array of integers) is done in $O(1)$ for each edge, so giving $A = O(m \log^2 m) + O(m \log m) + O(1) = O(m \log^2 m)$. In conclusion, the total running time of Algorithm 3 is $O(k_1 k_2 n + m \log^2 m)$.

6 APPLICATIONS AND EXAMPLES

An important application of the Merge algorithm is for implementing Boolean operations between solids represented as cellular complexes, by using decompositions of either the boundary or the interior. For example, the free space for motion planning of robots in 2D or 3D can be obtained by merging the (grown) models of obstacles within a cubical mesh of the workspace, and by computing the generated arrangement. Other important applications may be found in biomedical applications, e.g., the extraction of solid models of small-scale biological structures from high-resolution 3D medical images [15, 42].



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

Fig. 11. Extraction of a minimal 1-cycle from $\mathcal{A}(X_1)$: (a) the initial value for $c \in C_1$ and the signs of its oriented boundary; (b) cyclic subgroups on $\delta\partial c$; (c) new (coherently oriented) value of c and ∂c; (d) cyclic subgroups on $\delta\partial c$; (e) final value of c, with $\partial c = \emptyset$.

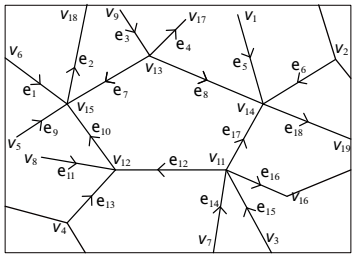


Fig. 12. A portion of the 1-complex used by Example 6.1.

Example 6.1. In Figure 12 we show a fragment of a 1-complex $X = X_1$ in $\mathbb{E}_2$, with cells $v_k \in X_0$ and $e_h \in X_1$. Here we compute stepwise the 1-chain representation $c \in C_1$ of a 2-cell of the complex $X_2 = \mathcal{A}(X_1)$. Look also at Figures 11a-e to follow stepwise the extraction of the 2-cell.

(a) Set $c = e_{12}$. Then $\partial c = v_{12} - v_{11}$;
(b) then $\delta\partial c = \delta v_{12} - \delta v_{11}$ by linearity. Hence, $\delta\partial c = (e_{10}+e_{11}+e_{12}+e_{13})-(+e_{12}+e_{14}+e_{15}+e_{16}+e_{17})$.
(c) Actually, by computing $\underline{\text{corolla}}(c)$ we get

$$\begin{aligned}\underline{\text{corolla}}(c) &= c + \underline{\text{next}}(c \cap \delta\partial c)\\ &= c + \underline{\text{next}}(e_{12})(\delta v_{12}) - \underline{\text{next}}(e_{12})(\delta v_{11})\\ &= e_{12} + \underline{\text{next}}(e_{12})(\delta v_{12}) + \underline{\text{prev}}(e_{12})(\delta v_{11})\\ &= e_{12} + e_{10} + e_{17}\end{aligned}$$

If we orient c coherently, we get $c = e_{10} + e_{12} - e_{17}$, and $\partial c = v_{15} - v_{12} + v_{12} - v_{11} + v_{11} - v_{14} = v_{15} - v_{14}$.
(d) As before, we repeat and orient coherently the computed 1-chain:

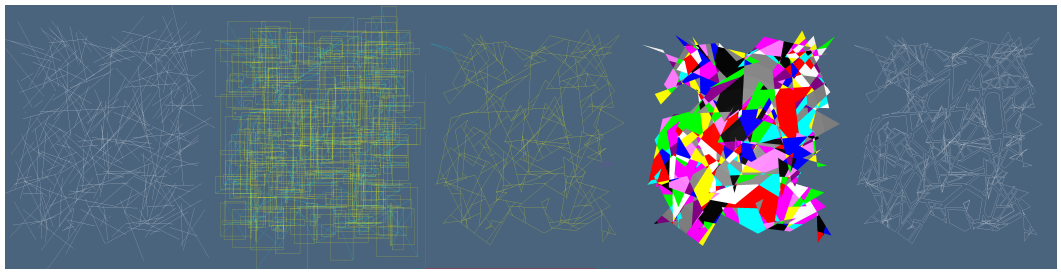
$$\begin{aligned}\underline{\text{corolla}}(c) &= c + \underline{\text{next}}(c \cap \delta\partial c)\\ &= c + \underline{\text{next}}(e_{10})(\delta v_{15}) - \underline{\text{next}}(e_{17})(\delta v_{14})\\ &= e_{10} + e_{12} - e_{17} + \underline{\text{next}}(e_{10})(\delta v_{15}) + \underline{\text{prev}}(e_{17})(\delta v_{14})\\ &= e_{10} + e_{12} - e_{17} - e_7 + e_8\end{aligned}$$

(e) Finally, $\partial\, \underline{\text{corolla}}(c) = \emptyset$, and the extraction algorithm terminates, giving $e_{10} + e_{12} - e_{17} - e_7 + e_8$ as the $C_1(X)$ representation for a basis element of $C_2(X)$, with $X = \mathcal{A}(X_1)$, and hence as a column for the oriented matrix of the unknown $\partial_2 : C_2 \to C_1$.

Example 6.2. An example of extraction of a minimal 2-cycle c, representation in C_2 of a basis element of C_3, is shown in Figure 9. The coefficients of the linear combination provide a matrix column of the coordinate representation of the operator $\partial_3 : C_3 \to C_2$.

Let us preliminary recall some notions about p-chains as linear combinations of oriented $(p-1)$-chains, for $0 \leq p \leq d$ (see Section 2.1). It is possible to see that the ordering (numbering) of vertices, edges, and faces (i.e. of 0-, 1-, and 2-cells) completely determines the pattern of signs in the matrices of boundary/coboundary operators.

Example 6.3 (Arrangement of line segments). In Figure 13 we show the whole algorithm pipeline for the computation of the plane arrangement $X_2 = \mathcal{A}(\mathcal{L})$ generated by a set $\mathcal{L}$ of random line segments in $\mathbb{E}^2$. The merge algorithm introduced in this paper directly produces the regularized arrangement. However, the non-regularized solution could readily be recovered by a posteriori addition of the dangling edges and trees, previously removed.



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

Fig. 13. Computation of the plane arrangement $\mathcal{A}(\mathcal{L})$ generated by a set of line segments: (a) set of random lines in $\mathbb{E}^2$; (b) spatial index based on containment boxes; (c) plane partition $\mathcal{A}(X_1)$, where the 1-complex X_1 is the union of maximal 2-node-connected subgraphs in $\mathcal{A}(\mathcal{L})$, drawn here in yellow and cyan; (d) cells of X_2 in $X = \mathcal{A}(X_1)$, randomly colored; (e) exploded boundaries of the 2-cells in X_2.

7 CONCLUSION

In this paper we have introduced a computational topology method for constructing the space arrangement induced by a collection of cellular complexes. The most attractive feature of this approach consists in its minimal subdivision of the output d-cells. Any (d-1)-cell of the output is a portion of a (d-1)-cell of some input complex. By construction, at variance with other merge algorithms, the set of "cuts" is the absolute minimum. In other words, the total area of the $(d-1)$-skeleton of resulting subdivision equates the total area of the $(d-1)$-skeletons of input data. Also, there are no redundant $(d-p)$-cells $(1 \leq p \leq d)$ in the output arrangement, and there is no need for simplification of output cells.

Our algorithmic pipeline applies to any kind of decompositions of either the interior or the boundary of the merged spaces, including polytopal (e.g. Voronoi) complexes or complexes with non convex cells, even punctured, i.e., non contractible to a point (with holes).

The output of our algorithms produces not only a complete generation of the cellular d-complex induced by a collection of d-complexes, but also the associated chain complex, including all the signed boundary/coboundary operators, a consistent orientation of all cells, and hence the complete knowledge of the homology of the space subdivision.

The introduced methods differs significantly from other approaches, not only because of the language, based on chain complexes and their operators, but also for the actual computer implementation, that uses only two classical data structures (kd-tree of vertices and 1D interval trees) as computation accelerators, without introducing representations typical of non-manifold solid modeling, that might seem mandatory in this kind of algorithms and applications.

The actual implementation, based on LAR [22] and implemented in Julia language [8] as open-source software[19], efficiently describes cells and chains with either dense or sparse arrays of signed integers, and their linear operators with sparse matrices. It is currently being used for deconstruction modeling of buildings [39], as well as for the extraction of solid models of biomedical structures at the cellular scale [15, 42].

ACKNOWLEDGEMENTS

This work is supported in part by a grant 2016/17 from SOGEI, the ICT company of Italian Ministry of Economy and Finance, and by the EU project medtrain3dmodsim. Vadim Shapiro is supported in part by National Science Foundation grant CMMI-1344205 and National Institute of Standards

[19] https://github.com/cvdlab/LARLIB.jl

and Technology. The authors would like to thank the anonymous reviewers who careful read our manuscript and provided us with many useful comments. Finally, they wish to acknowledge their gratitude to Giorgio Scorzelli, implementor and maintainer of the dimension-independent library `pyplasm`[20], without which this work could have not been possible.

A APPENDIX: EXAMPLES OF LAR

In this Appendix we give some examples of LAR use, to show the reader how a cellular complex is defined, how to compute a signed boundary operator, and how some topological adjacency relations may be computed in 2D and 3D, respectively, by multiplication of sparse matrices.

Example A.1 (2D cellular complex). In Figure 14 we show an example of 2D cellular complex, with `#V` = 22 vertices (0-cells), `#EV` = 34 edges (1-cells), `#FV` = 13 faces (2-cells). The full user-readable representation is given below. Note that `EV` and `FV` respectively provide the canonical (sorted) representations of edges and faces as lists of lists of vertex indices, that can be interpreted as the user-readable CSR sparse characteristic matrices M_1 and M_2 of the bases of 1-chains and 2-chains, respectively.

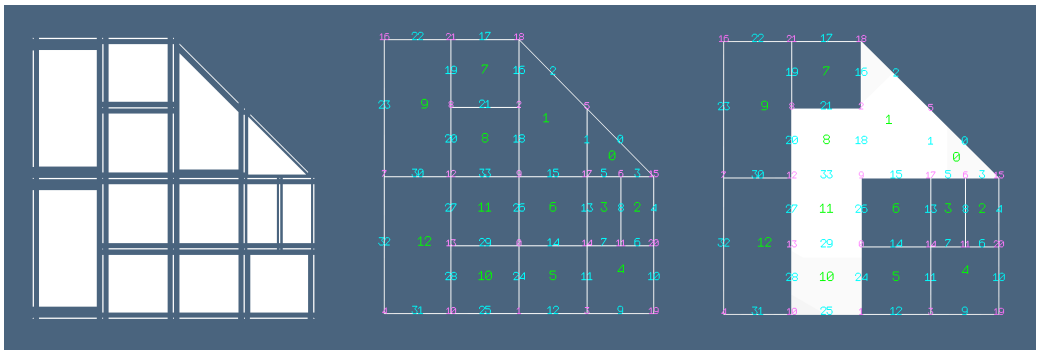
```
V = [[0.5,0.2475],[0.5,0.0],[0.5,0.7525],[0.7525,0.0],[0.0,0.0],[0.7525,0.7475],
[0.8787,0.5],[0.0,0.5],[0.2475,0.7525],[0.5,0.5],[0.2475,0.0],[0.8787,0.2475],
[0.2475,0.5],[0.2475,0.2475],[0.7525,0.2475],[1.0,0.5],[0.0,1.0],[0.7525,0.5],
[0.5,1.0],[1.0,0.0],[1.0,0.2475],[0.2475,1.0]]

EV = [[5,15],[5,17],[5,18],[6,15],[15,20],[6,17],[11,20],[11,14],[6,11],[3,19],
[19,20],[3,14],[1,3],[14,17],[0,14],[9,17],[2,18],[18,21],[2,9],[8,21],[8,12],
[2,8],[16,21],[7,16],[0,1],[1,10],[0,9],[12,13],[10,13],[0,13],[7,12],[4,10],
[4,7],[9,12]]

FV = [[5,6,15,17],[2,5,9,17,18],[6,11,15,20],[6,11,14,17],[3,11,14,19,20],
[0,1,3,14],[0,9,14,17],[2,8,18,21],[2,8,9,12],[7,8,12,16,21],[0,1,10,13],
[0,9,12,13],[4,7,10,12,13]]
```

We would like to note that, by means of the canonical representation, we can execute efficiently, via string syntax, both equality tests and/or set operations on the textual representation of chains and cells. This approach is used, e.g., in order to perform the identification of coincident cells and their reduction to quotient sets.

Example A.2 (Boundary computation). In order to compute the boundary of a 2-chain $g = c_0 + c_1 + c_8 + c_{11} + c_{12}$, white in Figure 14c, simply represented in LAR as `[0,1,8,11,12]`, i.e. the 1-cycle bounding its cells, we must multiply the coordinate matrix representation $[\partial_2]$ of the boundary operator $\partial_2 : C_2 \to C_1$, times the coordinate representation of g. Once fixed an ordering of the C_2 basis, the coordinate representation of any cell is unique. In this case we have

$$[g] = [1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1]^t,$$

[20] `https://github.com/plasm-language/pyplasm`



(a) (b) (c)

Fig. 14. LAR representation of a computation on a cellular complex: (a) 2-complex exploded; (b) numbering of basis elements of C_0 (magenta), C_1 (cyan), and C_2 (green); (c) the 2-chain $\mathsf{LAR}([g]) = [0, 1, 8, 10, 11]$ (white) in C_2, and its boundary $\mathsf{LAR}([\partial_2][g]) = [-0, 2, 3, -5, 15, -16, 20, 21, -24, -25, 26, 27, -28]$, 1-cycle in C_1.

and

$$[\partial_2]^t = \begin{pmatrix}
-1 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & -1 & 1 & -1 & 0
\end{pmatrix}$$

By multiplication of $[\partial_2] : C_2 \to C_1$ times $[g] \in C_2$ we get the boundary 1-cycle in signed coordinate notation:

$$[\partial_2][g] = [-1, 0, 1, 1, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0, 0, 0, 1, 1, 0, 0, -1, -1, 1, 1, -1, 0, 0, 0, 0, 0]^t$$

that, in the LAR formalism of signed integers for cells and chains, becomes the 1-array:

```
[-0, 2, 3, -5, 15, -16, 20, 21, -24, -25, 26, 27, -28]
```

i.e.[21], in mathematical notation for chains:

$$\partial_2 g = (-c_0 + c_2 + c_3 - c_5 + c_{15} - c_{16} + c_{20} + c_{21} - c_{24} - c_{25} + c_{26} + c_{27} - c_{28}) \in C_1$$

that the reader may readily check as counterclockwise coherently oriented.

Example A.3 (Adjacency of vertices). Here we compute the adjacency relation of vertices of Example A.2, simply represented by a sparse matrix $\mathsf{VV} = M_1^t M_1$, defined by this product of the characteristic matrix M_1 of edges (given in readable form by the array EV of Example A.2). The reader is sent to [22] for details and other topological relationships.

[21] Of course, -0 is just a symbolic notation here. In Python, where array indexing is 0-based, the indices of cells and their sign are maintained separated within sparse arrays. Conversely, the notation of d-chains as arrays of signed (nonzero) integers can be used directly in Julia, where arrays are 1-based.

$$VV = M_1^t M_1 = \begin{pmatrix}
4&1&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0&0&0&0\\
1&3&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&3&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&0&0&0\\
0&1&0&3&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0\\
0&0&0&0&2&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&3&0&0&0&0&0&0&0&0&0&1&0&1&1&0&0&0\\
0&0&0&0&0&0&3&0&0&0&0&1&0&0&0&1&0&1&0&0&0&0\\
0&0&0&0&1&0&0&3&0&0&0&0&1&0&0&0&1&0&0&0&0&0\\
0&0&1&0&0&0&0&0&3&0&0&0&1&0&0&0&0&0&0&0&0&1\\
1&0&1&0&0&0&0&0&0&4&0&0&1&0&0&0&0&1&0&0&0&0\\
0&1&0&0&1&0&0&0&0&0&3&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&3&0&0&1&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&1&1&1&0&0&4&1&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&1&0&1&3&0&0&0&0&0&0&0&0\\
1&0&0&1&0&0&0&0&0&0&0&1&0&0&4&0&0&1&0&0&0&0\\
0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&3&0&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&2&0&0&0&0&1\\
0&0&0&0&0&1&1&0&0&1&0&0&0&0&1&0&0&4&0&0&0&0\\
0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&3&0&0&1\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&1&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&3&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&0&0&3
\end{pmatrix}$$

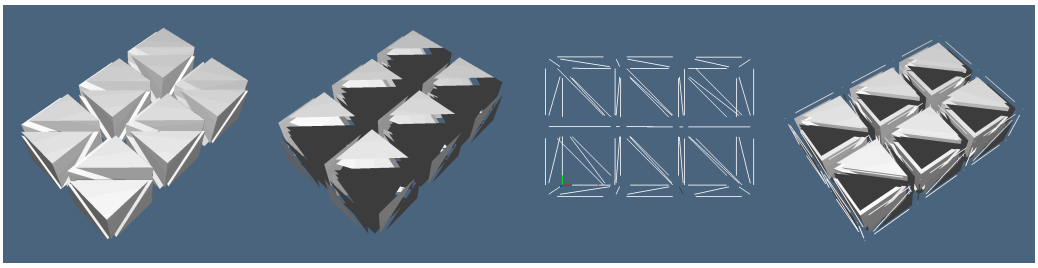
Of course, the incidence relation VV in form of array of arrays of integers is immediately derivable,

```
VV = [[1,9,13,14],[0,3,10],[8,9,18],[1,14,19],[7,10],[15,17,18],[11,15,17],
[4,12,16],[2,12,21],[0,2,12,17],[1,4,13],[6,14,20],[7,8,9,13],[0,10,12],
[0,3,11,17],[5,6,20],[7,21],[5,6,9,14],[2,5,21],[3,20],[11,15,19],[8,16,18]]
```

but it is more useful and efficient to solve several queries at the same time, on modern computational architectures, by multiplication of sparse matrices [18, 38].

Example A.4 (3D cellular complex). The LAR scheme is, of course, dimension-independent. Here we give the LAR description of a the simplest simplicial decomposition of a mesh $3 \times 2 \times 1$ of 3-cubes into tetrahedra, shown in Figure 15.

```
V = [[0,0,0],[1,0,0],[2,0,0],[3,0,0],[0,1,0],[1,1,0],[2,1,0],[3,1,0],[0,2,0],
[1,2,0],[2,2,0],[3,2,0],[0,0,1],[1,0,1],[2,0,1],[3,0,1],[0,1,1],[1,1,1],
[2,1,1],[3,1,1],[0,2,1],[1,2,1],[2,2,1],[3,2,1]]

TV = [0,1,4,12],[1,4,12,13],[4,12,13,16],[1,4,5,13],[4,5,13,16],[5,13,16,17],[1,2,
5,13],[2,5,13,14],[5,13,14,17],[2,5,6,14],[5,6,14,17],[6,14,17,18],[2,3,6,14],[3,6,
14,15],[6,14,15,18],[3,6,7,15],[6,7,15,18],[7,15,18,19],[4,5,8,16],[5,8,16,17],[8,
16,17,20],[5,8,9,17],[8,9,17,20],[9,17,20,21],[5,6,9,17],[6,9,17,18],[9,17,18,21],
[6,9,10,18],[9,10,18,21],[10,18,21,22],[6,7,10,18],[7,10,18,19],[10,18,19,22],[7,
10,11,19],[10,11,19,22],[11,19,22,23]]
```



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

Fig. 15. A small ($3 \times 2 \times 1$) mesh (exploded) of tetrahedra: (a) 3-cells; (b) 2-cells; (c) 1-cells; (d) the whole 3-complex. Note that to $(d-1)$-cells in $\mathbb{E}^d$ space cannot be given a preferred orientation.

The above TV array of "tetrahedra by vertices" is the user-readable format of the sparse characteristic matrix M_3 of 3-cells, and provides a complete LAR representation of the solid model, allowing for efficient queries on topology. For example the adjacency relation TT between tetrahedra is computed by filtering the elements with value 3 in the (sparse) matrix $M_3 M_3^t$. In this case we have:

$$\mathtt{TT} = \mathrm{filter}(M_3 M_3^t, 3) = \begin{pmatrix}
- & 1 & - \\
1 & - & 1 & 1 & - \\
- & 1 & - & - & 1 & - \\
- & 1 & - & - & 1 & - & 1 & - \\
- & - & 1 & 1 & - & 1 & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & 1 & - & - & - & 1 & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & 1 & - & - & - & 1 & - \\
- & - & - & - & - & - & 1 & - & 1 & 1 & - \\
- & - & - & - & - & 1 & - & 1 & - & - & 1 & - \\
- & - & - & - & - & - & - & 1 & - & - & 1 & - & 1 & - \\
- & - & - & - & - & - & - & 1 & 1 & - & 1 & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & 1 & - & - & - & 1 & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & 1 & - & - & - & 1 & - \\
- & - & - & - & - & - & - & - & - & - & 1 & - & 1 & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & 1 & - & 1 & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & 1 & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & 1 & 1 & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - \\
- & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & 1 & - & - & - & - & - & - & - & - & 1 & - & 1 & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & 1 & - & 1 & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & 1 & - & 1 & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & 1 & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & 1 & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & 1 & - & - & - & 1 & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & 1 & - & 1 & - & - & - & - & - & - & - & - & - & - \\
- & 1 & - & - & - & 1 & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & 1 & - & - & - & 1 & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & 1 & - & - & - & - & - & - & - & - & - & 1 & - & 1 & 1 & - & - & - & - & - & - \\
- & 1 & - & 1 & - & - & 1 & - & - & - & - & - \\
- & 1 & - & - & 1 & - & - & - & - & - \\
- & 1 & 1 & - & 1 & - & - & - & - \\
- & 1 & - & - & - & - & - \\
\end{pmatrix}$$

The symmetric TT matrix can be immediately transformed into the array of arrays of tetrahedra indices giving all pairs $(k \times \mathtt{TT}[k])$ $(0 \leq k \leq \#\mathtt{TV})$ of tetrahedra that share a 2-cell, i.e. a triangle. Clearly such product matrix returns in element (i, j) the number of vertices shared by tetrahedron $\mathtt{TV}[i]$ and tetrahedron $\mathtt{TV}[j]$. Of course, several queries on tetrahedra adjacency can be answered in parallel by a computational kernel SpMSpM (sparse matrix times sparse matrix) working on TT and a compatible matrix with unit columns.

```
TT = [[1],[0,2,3],[1,4],[1,4,6],[2,3,5,18],[4,8,19],[3,7],[6,8,9],[5,7,10],[7,10,
12],[8,9,11,24],[10,14,25],[9,13],[12,14,15],[11,13,16],[13,16],[14,15,17,30],
[16,31],[4,19],[5,18,20,21],[19,22],[19,22,24],[20,21,23],[22,26],[10,21,25],
[11,24,26,27],[23,25,28],[25,28,30],[26,27,29],[28,32],[16,27,31],[17,30,32,33],
[29,31,34],[31,34],[32,33,35],[34]]
```

REFERENCES

[1] Alfred V. Aho, Michael R. Garey, and Jeffrey D. Ullman. 1972. The Transitive Reduction of a Directed Graph. *SIAM J. Comput.* 1, 2 (1972), 131–137. https://doi.org/10.1137/0201008 arXiv:http://dx.doi.org/10.1137/0201008

[2] C. Armstrong, A. Bowyer, S. Cameron, J. Corney, G. Jared, R. Martin, A. Middleditch, M. Sabin, J. Salmon, and J. Woodwark. 1999. Djinn: a geometric interface for solid modelling. Technical Report. Information Geometers Ltd., Winchester, UK.

[3] D. Ayala, P. Brunet, R. Juan, and I. Navazo. 1985. Object Representation by Means of Nonminimal Division Quadtrees and Octrees. ACM Trans. Graph. 4, 1 (Jan. 1985), 41–59. https://doi.org/10.1145/3973.3975

[4] Bruce G. Baumgart. 1972. Winged Edge Polyhedron Representation. Technical Report. Stanford, CA, USA.

[5] Nathan Bell and Michael Garland. 2008. Efficient Sparse Matrix-Vector Multiplication on CUDA. NVIDIA Technical Report NVR-2008-004. NVIDIA Corporation.

[6] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. Commun. ACM 18, 9 (Sept. 1975), 509–517. https://doi.org/10.1145/361002.361007

[7] Fausto Bernardini. 1991. Integration of Polynomials over N-dimensional Polyhedra. Comput. Aided Des. 23, 1 (Feb. 1991), 51–58. https://doi.org/10.1016/0010-4485(91)90081-7

[8] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. 2017. Julia: A Fresh Approach to Numerical Computing. *SIAM Rev.* 59, 1 (2017), 65–98. https://doi.org/10.1137/141000671 arXiv:http://dx.doi.org/10.1137/141000671

[9] Anders Björner and Günter M. Ziegler. 1992. Combinatorial stratification of complex arrangements. J. Amer. Math. Soc. 5 (1992), 105–149.

[10] Aydın Buluç and John R. Gilbert. 2012. Parallel Sparse Matrix-Matrix Multiplication and Indexing: Implementation and Experiments. SIAM Journal of Scientific Computing (SISC) 34, 4 (2012), 170 – 191. https://doi.org/10.1137/110848244

[11] C.K. Yap C. Li, S. Pion. 2005. Recent progress in exact geometric computation. he Journal of Logic and Algebraic Programming 64 (2005), 85–111.

[12] Carlo Cattani and Alberto Paoluzzi. 1990. Boundary Integration over Linear Polyhedra. Comput. Aided Des. 22, 2 (Feb. 1990), 130–135. https://doi.org/10.1016/0010-4485(90)90007-Y

[13] Bernard Chazelle, Herbert Edelsbrunner, Leonidas Guibas, Micha Sharir, and Jack Snoeyink. 1993. Computing a Face in an Arrangement of Line Segments and Related Problems. SIAM J. Comput. 22, 6 (Dec. 1993), 1286–1302. https://doi.org/10.1137/0222077

[14] L. P. Chew. 1987. Constrained Delaunay Triangulations. In Proceedings of the Third Annual Symposium on Computational Geometry (SCG '87). ACM, New York, NY, USA, 215–222. https://doi.org/10.1145/41958.41981

[15] Giulia Clementi, Danilo Salvati, Giorgio Scorzelli, Alberto Paoluzzi, and Valerio Pascucci. 2016. Progressive extraction of neural models from high-resolution 3D images of brain. In 13th Int. Conf. on CAD & Applications. Vancouver, BC, Canada.

[16] Hoffmann C.M. 2001. Robustness in Geometric Computations. ASME. J. Comput. Inf. Sci. Eng. 1, 2 (2001), 143–155. https://doi.org/10.1115/1.1375815

[17] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd ed.). The MIT Press.

[18] Timothy A. Davis. 2006. Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[19] Mathieu Desbrun, Eva Kanso, and Yiying Tong. 2006. Discrete Differential Forms for Computational Modeling. In ACM SIGGRAPH 2006 Courses (SIGGRAPH '06). ACM, New York, NY, USA, 39–54. https://doi.org/10.1145/1185657.1185665

[20] Antonio DiCarlo, Franco Milicchio, Alberto Paoluzzi, and Vadim Shapiro. 2009. Chain-Based Representations for Solid and Physical Modeling. Automation Science and Engineering, IEEE Transactions on 6, 3 (july 2009), 454 –467. https://doi.org/10.1109/TASE.2009.2021342

[21] Antonio DiCarlo, Franco Milicchio, Alberto Paoluzzi, and Vadim Shapiro. 2009. Discrete Physics Using Metrized Chains. In 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling (SPM '09). ACM, New York, NY, USA, 135–145. https://doi.org/10.1145/1629255.1629273

[22] Antonio DiCarlo, Alberto Paoluzzi, and Vadim Shapiro. 2014. Linear Algebraic Representation for Topological Structures. Comput. Aided Des. 46 (Jan. 2014), 269–274. https://doi.org/10.1016/j.cad.2013.08.044

[23] David P. Dobkin and Michael Jay Laszlo. 1987. Primitives for the Manipulation of Three-dimensional Subdivisions. In Proceedings of the Third Annual Symposium on Computational Geometry (SCG '87). Acm, New York, NY, USA, 86–99. https://doi.org/10.1145/41958.41967

[24] Herbert Edelsbrunner. 1987. Algorithms in Combinatorial Geometry. Springer-Verlag New York, Inc., New York, NY, USA.

[25] Sharif Elcott and Peter Schroder. 2006. Building Your Own DEC at Home. In ACM SIGGRAPH 2006 Courses (SIGGRAPH '06). ACM, New York, NY, USA, 55–59. https://doi.org/10.1145/1185657.1185666

[26] Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra, and Sven Schönherr. 2000. On the Design of CGAL a Computational Geometry Algorithms Library. Softw. Pract. Exper. 30, 11 (Sept. 2000), 1167–1202. https://doi.org/10.1002/1097-024X(200009)30:11<1167::AID-SPE337>3.0.CO;2-B

[27] Efi Fogel, Dan Halperin, Lutz Kettner, Monique Teillaud, Ron Wein, and Nicola Wolpert. 2007. Arrangements. In Effective Computational Geometry for Curves and Surfaces, Jean-Daniel Boissonat and Monique Teillaud (Eds.). Springer, Chapter 1, 1–66.

[28] Francesco Furiani, Giulio Martella, and Alberto Paoluzzi. 2017. Geometric Computing with Chain Complexes: Design and Features of a Julia Package. CoRR abs/1710.07819 (2017). arXiv:1710.07819 http://arxiv.org/abs/1710.07819

[29] Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tòth (Eds.). 2017. Handbook of Discrete and Computational Geometry – Third Edition. CRC Press, Inc., Boca Raton, FL, USA.

[30] Peter Hachenberger, Lutz Kettner, and Kurt Mehlhorn. 2007. Boolean Operations on 3D Selective Nef Complexes: Data Structure, Algorithms, Optimized Implementation and Experiments. Comput. Geom. Theory Appl. 38, 1-2 (Sept. 2007), 64–99. https://doi.org/10.1016/j.comgeo.2006.11.009
[31] Allen Hatcher. 2002. Algebraic topology. Cambridge University Press. http://www.math.cornell.edu/~hatcher/AT/ATpage.html
[32] René Heinzl. 2007. Concepts for Scientific Computing. Ph.D. Dissertation. Wien, Österreich.
[33] Anil Nirmal Hirani. 2003. Discrete Exterior Calculus. Ph.D. Dissertation. Pasadena, CA, USA. Advisor(s) Marsden, Jerrold E. AAI3086864.
[34] John Hopcroft and Robert Tarjan. 1973. Algorithm 447: Efficient Algorithms for Graph Manipulation. Commun. ACM 16, 6 (June 1973), 372–378. https://doi.org/10.1145/362248.362272
[35] R. A. Jarvis. 1973. On the Identification of the Convex Hull of a Finite Set of Points in the Plane. 2, 1 (March 1973), 18–21.
[36] Tomasz Kaczynski, Konstantin Mischaikow, and Marion Mrozek. 2004. Computational homology. Number 157 in Appl. Math. Sci. Springer-Verlag.
[37] Tomasz Kaczynski, Konstantin Mischaikow, and Marion Mrozek. 2006. Computational homology. Bulletin (New Series) of the AMS 43, 2 (2006).
[38] Jeremy Kepner and John Gilbert. 2011. Graph Algorithms in the Language of Linear Algebra. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.
[39] Enrico Marino, Federico Spini, Danilo Salvati, Christian Vadalà, Michele Vicentino, Alberto Paoluzzi, and Antonio Bottaro. 2017. Modeling Semantics for Building Deconstruction. In Proc., 12wt International Conference on Computer Graphics Theory and Applications.
[40] Wataru Mayeda. 1972. Graph Theory. Wiley Interscience, New York, NY.
[41] A.E. Middleditch. 1992. Cellular models of mixed dimension. Technical Report BRU/CAE/92:3. Brunel University Centre for Geometric Modelling and Design.
[42] Alberto Paoluzzi, Antonio DiCarlo, Francesco Furiani, and Miroslav Jirík. 2016. CAD models from medical images using LAR. Computer-Aided Design and Applications 13, 6 (2016), 747–759. https://doi.org/10.1080/16864360.2016.1168216
[43] Franco P. Preparata and Michael I. Shamos. 1985. Computational Geometry: An Introduction. Springer-Verlag New York, Inc., New York, NY, USA.
[44] Aristides G. Requicha. 1980. Representations for Rigid Solids: Theory, Methods, and Systems. ACM Comput. Surv. 12, 4 (Dec. 1980), 437–464. https://doi.org/10.1145/356827.356833
[45] Jarek R. Rossignac and Michael A. O'Connor. 89. A dimension-Independent Model for Pointsets with Internal Structures and Incomplete Boundaries. Technical Report. IBM Research Division, Yorktown Heights, N.Y. 10598.
[46] Jonathan Richard Shewchuk. 2008. General-Dimensional Constrained Delaunay and Constrained Regular Triangulations, I: Combinatorial Properties. Discrete & Computational Geometry 39, 1 (01 Mar 2008), 580–637. https://doi.org/10.1007/s00454-008-9060-3
[47] Jonathan Richard Shewchuk. 2008. General-Dimensional Constrained Delaunay and Constrained Regular Triangulations,ÂăI: Combinatorial Properties. Discrete & Computational Geometry 39, 1 (2008), 580–637. https://doi.org/10.1007/s00454-008-9060-3
[48] William C. Thibault and Bruce F. Naylor. 1987. Set Operations on Polyhedra Using Binary Space Partitioning Trees. SIGGRAPH Comput. Graph. 21, 4 (Aug. 1987), 153–162. https://doi.org/10.1145/37402.37421
[49] T. Vialar. 2016. Handbook of Mathematics. HDBoM. https://books.google.com/books?id=-tcVMQAACAAJ (Def 805).
[50] Tony Woo. 1985. A Combinatorial Analysis of Boundary Data Structure Schemata. IEEE Comput. Graph. Appl. 5, 3 (March 1985), 19–27. https://doi.org/10.1109/MCG.1985.276337