Topological computing of arrangements with (co)chains

ALBERTO PAOLUZZI, Roma Tre University, Rome, Italy
VADIM SHAPIRO, University of Wisconsin-Madison & ICSI, United States
ANTONIO DICARLO, CECAM-IT-SIMUL Node, Rome, Italy
FRANCESCO FURIANI, Roma Tre University, Rome, Italy
GIULIO MARTELLA, Roma Tre University, Rome, Italy
GIORGIO SCORZELLI, Scientific Computing and Imaging Institute (SCI), Salt Lake City, Utah, USA

In many areas of applied geometric/numeric computational mathematics, including geo-mapping, computer vision, computer graphics, finite element analysis, medical imaging, geometric design, and solid modeling, one has to compute incidences, adjacencies and ordering of cells, generally using disparate and often incompatible data structures and algorithms. This paper introduces computational topology algorithms to discover the 2D/3D space partition induced by a collection of geometric objects of dimension 1D/2D, respectively. Methods and language are those of basic geometric and algebraic topology. Only sparse vectors and matrices are used to compute both spaces and maps, *i.e.*, the chain complex, from dimension zero to three. The prototype software is written in Julia, the novel language for scientific computing. The applications may vary from 3D graphics to 3D printing, from images to scene understanding, and from games to building information modeling (BIM).

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

Additional Key Words and Phrases: Computational Topology, Chain Complex, Cellular Complex, Arrangement, Solid Modeling, Linear Algebraic Representation, LAR, Image Understanding.

ACM Reference Format:
Alberto Paoluzzi, Vadim Shapiro, Antonio DiCarlo, Francesco Furiani, Giulio Martella, and Giorgio Scorzelli . 2020. Topological computing of arrangements with (co)chains. *ACM Trans. Spatial Algorithms Syst.* 1, 1, Article 1 (January 2020), 29 pages. https://doi.org/10.1145/3401988

1 INTRODUCTION

Given a collection $\mathcal{S}$ of geometric objects[1], the subject of this paper is computing the topology of their space arrangement $\mathcal{A}(\mathcal{S})$ as a *chain complex*, *i.e.*, as a short exact sequence of linear spaces C_i

[1]Examples include, but are not limited to: line segments, quads, triangles, polygons, meshes, pixels, voxels, volume images, B-reps, *etc.* In mathematical terms, a geometric object is a topological space embedded in some $\mathbb{E}^d$ [38].

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. V.S. is supported in part by National Science Foundation grant CMMI-1344205 and National Institute of Standards and Technology.
Authors' addresses: Alberto Paoluzzi Roma Tre University, Rome, Italy; Vadim Shapiro University of Wisconsin-Madison & ICSI, United States; Antonio DiCarlo CECAM-IT-SIMUL Node, Rome, Italy; Francesco Furiani Roma Tre University, Rome, Italy; Giulio Martella Roma Tre University, Rome, Italy; Giorgio Scorzelli Scientific Computing and Imaging Institute (SCI), Salt Lake City, Utah, USA.

2374-0353/2020/1-ART1 $15.00
https://doi.org/10.1145/3401988

of (co)chains and linear boundary/coboundary maps ∂_p and $\delta_p = \partial_{p+1}^{\top}$ between them:

$$C_\bullet = (C_p, \partial_p) := C_3 \underset{\partial_3}{\overset{\delta_2}{\leftrightarrows}} C_2 \underset{\partial_2}{\overset{\delta_1}{\leftrightarrows}} C_1 \underset{\partial_1}{\overset{\delta_0}{\leftrightarrows}} C_0.$$

The $C_\bullet$ chain complex fully characterizes the topology of the space partition (arrangement) induced within the ambient Euclidean space by a collection of geometric objects embedded in it. To construct $C_\bullet$, the collection of given geometric objects (cell complexes) must be merged into a common structure. The data structures needed for such computational program are sparse multi-arrays, and their standard algebraic operations. In this way, we introduce a novel approach based on piecewise-linear algebraic topology [51, 89], that allows us to treat rather general cellular complexes, with cells homeomorphic to polyhedra, *i.e.*, to triangulable spaces [89], and hence possibly non convex and multiply connected.

We believe that basic geometric algebraic topology already provides the right set of mathematical concepts and tools to compute and explore the cells of the space partition induced by a set of geometric objects, as well as the related incidence/neighborhood relations. The current escalation of quantity/quality of data, and their drift through pipelines of micro/macro-services that need simple interfaces, along with the fast diffusion of hybrid architectures for advanced applications, also motivate this paper. The notions we deal with include geometric complexes, linear spaces of chains and cochains, the *chain complex* of linear operators between pairs of spaces, and their compositions. The discussion is restricted to piecewise-linear topology and to space dimensions less or equal to three. The paper formalizes the algorithms to generate the matrices of (co)boundary operators on a cellular arrangement in Euclidean space.

To our knowledge, algorithms for computing chain complexes only include the recent paper [2], where combinatorial generalized maps [30], a quite intricate data structure, are used for cycles/boundary calculation of the homology of a cellular complex. Is is well known [38, 61, 70, 74] that for simplicial complexes, *i.e.*, triangulations, boundary operators are defined as linear extensions of basic boundary operators which act on simplices.

Construction of arrangements of lines, segments, planes and other geometrical objects is discussed in [43], with a description of CGAL software [41], implementing 2D/3D arrangements with Nef polyhedra [17, 50]. A review of papers and algorithms concerning construction and counting of cells may be found in the chapter on Arrangements in the 'Handbook of Discrete and Computational Geometry' [47]. Arrangements of polytopes, hyperplanes and d-circles are discussed in [19].

The standard way to look at combinatorial data structures is the IG (Incidence Graph) data structure described in the book 'Algorithms in Combinatorial Geometry' [39]. IG is an implementation of the Hasse diagram [18] of the cells of a d-complex [34]. Our (co)chain complex provides an algebraic representation of the Hasse diagram with sparse matrices, associating each two adjacent levels with (co)boundary maps to traverse up and down the hierarchy.

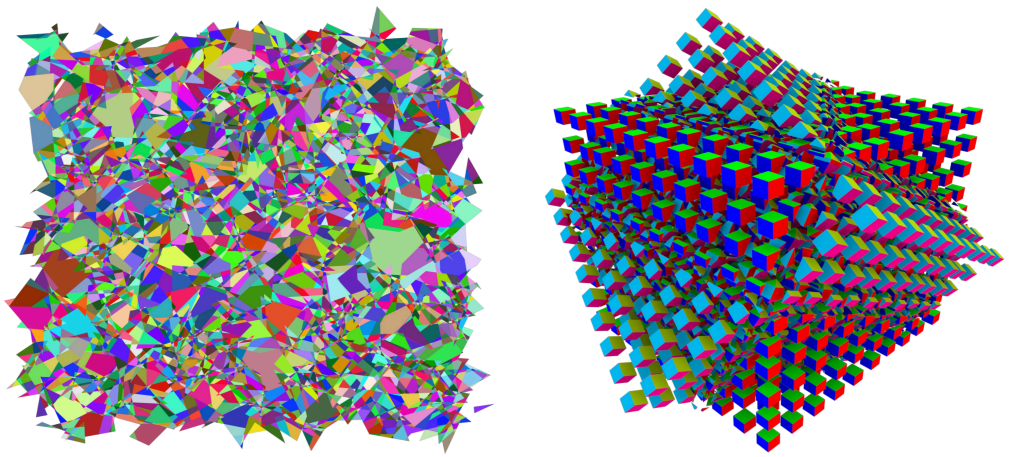
Some early papers are concerned with the efficient representation of 3D cellular decompositions [12, 69]. In particular, [37] defines the polygon-edge data structure to represent orientable and contractible 3D decompositions and their duals. Several other systems have been developed about three decades ago, to handle the merging of complexes in the context of solid modeling and manufacturing automation. The merging of intersecting shells of polyhedral solids is computable by finding all intersecting bounding boxes of faces in $O(n\,log^2(n) + k)$, where n is the number of boxes and k is the number of box-pair intersections [55]. The Selective Geometric Complex (SGC) was introduced [87] in 1989, allowing for lower dimensional cells contained inside the interior of cells. Later, scientists in UK proposed the Djinn API [3, 67], with the theoretical foundations for merging operation of cellular complexes, but (non-manifold) boundary representations remained

the standard in the field. Recently, a systematic procedure has been proposed in [109] for constructing a family of exact constructive solid geometry operations, starting from a collection of boundary triangular meshes.

Most of earlier algorithms and procedures [1, 8, 12, 17, 20, 21, 23, 24, 34, 37, 45, 48, 54–57, 60, 62–64, 74, 78–88, 91, 92, 96, 102–109] work with data structures optimized for selected classes of geometric objects. By contrast, our formulation, representation, and algorithms, cast in terms of (co)chain complexes of (co)boundary maps, may be applied to very different geometric objects, ranging from solid models to engineering meshes, geographical systems, biomedical images.

Numerical methods aiming to integrate domain modeling, differential topology and mathematical modeling with physical simulations are also based on chains and cochains [71, 72]. In particular, Discrete Exterior Calculus (DEC) with simplicial complexes was introduced by [53] and made popular by [33, 40]. FEEC is a recent advance [4–6] in the mathematics of finite element methods that employs differential complexes to construct stable numerical schemes. The Cell Method (CM) is a purely algebraic computational method for modeling and simulation [42, 99, 100] based on boundary/coboundary maps and a direct discrete formulation of field laws. Our own research in geometrical and physical modeling with chain and cochain complexes was introduced in [34–36].

Our arrangement computation may be characterized as a merging of cellular complexes. All merge algorithms, by nature, follow the same steps. The advantages of formulating them in terms of (co)chain complexes and operations on sparse matrices are that (1) the common and general algebraic topological nature of this operation is revealed; (2) implementation specific low-level details and algorithms are hidden; (3) explicit connection to SpMV kernels (sparse matrix-vector multiplication) and to sparse numerical linear algebra systems [13, 16, 25, 31] on GPU and HPC platforms is provided; (4) systematic and rigorous development of the algorithms, that are correct by construction, is supported.



(a) (b)

Fig. 1. Examples of space arrangements by merging cellular complexes: (a) 2D arrangement X_2 of $\mathbb{E}^2$ generated by a set of random line segments. The Euler characteristic is $\chi = \chi_0 - \chi_1 + \chi_2 = 11361 - 20813 + 9454 = 2$; (b) arrangement of $\mathbb{E}^3$ by merging of two 3-complexes with 2×10^3 3-cells. The Euler characteristic (of the non-exploded resulting 3-complex) is $\chi = \chi_0 - \chi_1 + \chi_2 - \chi_3 = 8787 - 26732 + 26600 - 8655 = 0$. This count includes the outer (unbounded) 2-cell or 3-cell, respectively, that are also computed by the Topological Gift Wrapping (TGW) algorithm (see Section 2.3). The Euler characteristic of the d-sphere is $\chi = 1 + (-1)^d = 2$ or 0 for either even or odd space dimension d.

The paper is organized as follows. A brief introduction to the proposed computational pipeline is given in Section 2, going from the arrangement of $\mathbb{E}^2$ induced by a set of line segments, to the arrangement of $\mathbb{E}^3$ induced by a collection of open/closed piecewise-linear surfaces and/or meshes. In the short subsection 2.5, we show that representing chains as sparse arrays is compact and flexible. In Section 3, the algorithms for computing (co)boundary operators are presented in pseudocode format. In Section 4 we compare our approach with other published relevant results, explaining its key differences and advantages. Past development and current prospects of this project are outlined in Section 5. The closing section presents a summary of contents, and outlines possible applications of ideas. The Appendix gives terse summary of standard notions from algebraic topology related to chains calculus and detailed examples of topological computations.

2 COMPUTATIONAL PIPELINE

Let us start with an input collection $\mathcal{S}$ of piecewise-linear *cellular complexes* of $(d-1)$ dimension, embedded in $\mathbb{E}^d$ space, with $d \in \{2, 3\}$. Examples include soups of lines or polygons, triangled surfaces, quads from cubical meshes, 1-, 2-, or 3-cells from 2D or 3D image elements (pixels or voxels, respectively), 2-skeletons/boundaries of triangulated polyhedra, non manifold B-reps or decompositive reps of solid models. These objects are *geometric complexes*, *i.e.*, pairs (X, μ), where X is a cellular complex[2] specifying the topology and $\mu : X_0 \longrightarrow \mathbb{E}^d$ is the embedding function of 0-cells, sufficient for a piecewise-linear geometry. The data may contains both $(d-1)$- and d-complexes: the combinatorial union of their $(d-1)$-skeletons is selected as the actual input to the pipeline. An admissible input collection $\mathcal{S}$ of geometric complexes will mutually intersect and partition $\mathbb{E}^d$ into a cellular complex $X = \bigcup X_p$ $(0 \leq p \leq d)$, called the *arrangement* $\mathcal{A}(\mathcal{S})$ induced by $\mathcal{S}$.

The object of this paper is the computation of the chain complex $C_\bullet(X) = (C_p, \partial_p)$, starting from some representation[3] of $\mathcal{S}$. In particular, we compute the matrices of the linear maps ∂_p (and their duals δ_{p-1}) between chain spaces C_p. Definitions and examples are given in Appendix A. Since the matrix of a linear map $C_p \longrightarrow C_{p-1}$ between linear spaces contains by columns the target space representation of the domain space basis elements, the paper also provides constructive algorithms to generate a sparse matrix representation of basis elements $u_p \in C_p$, which are one-to-one with p-cells σ_p in X_p skeletons. Note that cells in X, specifically in X_d, are not known in advance.

The computation is correct because the boundaries of adjacent decomposed 2-cells are compatible as cellular complexes by construction. A requirement of the standard definition of a cellular complex [51, 70] demands *boundary compatibility* to hold. This fact is guaranteed here, since when abutting subsets of 2-cells have non-empty intersection, they generate congruent 0- and 1-cells on their common boundary. A summary of the computational steps for $d = 3$ follows.

2.1 2D arrangements generated by 2-cells (Merge)

Let $\mathcal{S}_2 \subseteq \mathcal{S}$ be the set of 2-cells of input geometric complexes[4], embedded in $\mathbb{E}^3$. Note that $\mathcal{S}_2$ is not required to be a cellular complex, since cells may intersect outside of their boundaries. It is only required that each cell is connected and manifold. Each $\sigma \in S_2$ is mapped to subspace $z = 0$ by an affine transformation $\mathbf{Q}_\sigma$, together with the set $\mathcal{I}(\sigma) \subset S_2$ of cells potentially intersecting it. The set $\Sigma = X_2(\sigma \cup \mathcal{I}(\sigma))$ is intersected with $z = 0$ subspace, producing a set $\mathcal{S}_1(\sigma)$ of line segments in $\mathbb{E}^2$. First, these are mutually intersected, producing the chain complex $C_\bullet(\sigma) = (C_{2,1,0}, \partial_{2,1})$ generated by $\mathcal{A}(\mathcal{S}_1)$ (see Figure 3d). Care must be taken to identify the 1-cycles around holes within partitioned 2-cells, in order to remove their outer boundary cycles, by removing their columns,

[2] See Appendix A.1 for this and related definition(s).

[3] Our prototype implementation in https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/tree/julia-1.0, makes use of LAR representation [36], on which this approach strongly relies.

[4] Cell complex embedded in Euclidean space via association of position vectors to 0-cells.

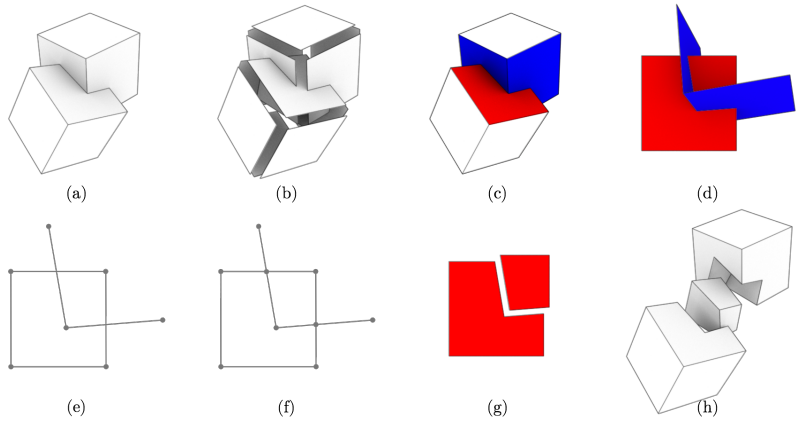


Fig. 2. Cartoon display of the computational pipeline: (a) two solids in $\mathcal{S}$; (b) the exploded input collection $\mathcal{S}_2$ in $\mathbb{E}^3$; (c) 2-cell σ (red) and the set $\Sigma(\sigma)$ (blue) of possible intersection; (d) $\sigma \cup \Sigma$ affinely mapped on $z = 0$; (e) reduction to a set of 1D segments in $\mathbb{E}^2$ via intersection with $z = 0$; (f) pairwise intersections; (g) exploded U_2 basis of C_2 generated as columns of $\partial_2 : C_2 \longrightarrow C_1$, and (h) exploded U_3 basis of C_3 generated as columns of operator's $\partial_3 : C_3 \longrightarrow C_2$ sparse matrix, both via the TGW algorithm in 2D / 3D, respectively (see Section 2.3).

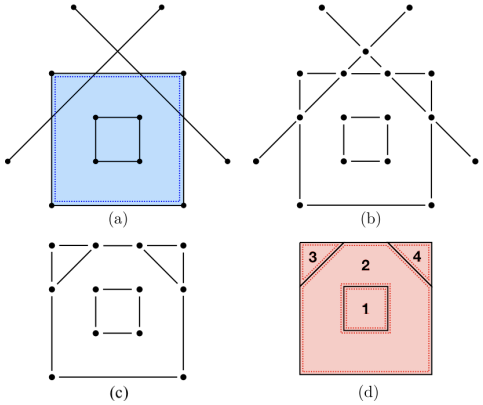


Fig. 3. Basic case: computation of the regularized arrangement of a set of lines in $\mathbb{E}^2$: (a) the input, *i.e.*, the 2-cell σ (signed blue) and the line segment intersections of $\Sigma(\sigma)$ with $z = 0$; (b) all pairwise intersections of 1-cells; (c) removal of the 1-subcomplex external to $\partial\sigma$; (d) 2-chain generated by $\sigma \cup \Sigma$ via TGW in 2D (see 2.3).

as well the outer cycle, from the operator matrix. Identification is easy: each hole produces two opposite columns summing to 0. Finally, the geometric 2-complex X_σ is transformed back in $\mathbb{E}^3$ by $\mathbf{Q}_\sigma^{-1}$. In summary, Algorithm 1 executes the one-to-one map $\sigma \mapsto X_\sigma$, by computing the maps $\sigma \mapsto C_\bullet(\sigma)$ independently from each other. The output is a set $\mathbb{C} := \{C_\bullet(\sigma), \sigma \in \mathcal{S}_2\}$.

2.2 Quotient set computations (Congruence)

The idea allowing us to compute independent fragmentations of 2-cells comes from a similitude between homology and congruence. Two $(d-1)$-spaces (curves, surfaces, *etc.*) embedded in $\mathbb{E}^d$ are topologically *homologous* when their boundaries can be glued, enclosing a portion of the ambient space, and subdivide $\mathbb{E}^d$ in two parts, inner and outer. Two geometric figures are geometrically

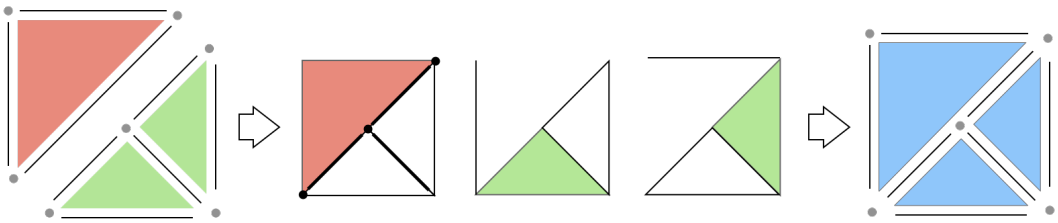
congruent iff one can be transformed into the other by an isometry [29]. The congruences R_p between p-cells of geometric complexes in $\mathbb{C}$ are equivalence relations, so we may compute the chain complex of quotient chain spaces,

$$C_2(U_2/R_2) \xrightarrow{\partial_2} C_1(U_1/R_1) \xrightarrow{\partial_1} C_0(U_0/R_0),$$

over which subsequently build the yet unknown basis of C_3. Note that $U_p = \bigcup_\sigma U_p^\sigma$, with $\sigma \in \mathcal{S}_2$, is the *union of bases* of "fragmented p-chains[5]" in $\mathbb{C}$, module the *congruence relations* R_p. $C_p(U_p/R_p)$ stands for the chain space generated by $X_p = U_p/R_p$. In this stage of the computational pipeline, we compute, for each $\sigma \in \mathcal{S}_2$, the quotient sets and the maps ∂_p in-between, for $p = 0, 1, 2$.

As usual, we proceed by an inductive procedure: each stage consists in glueing cells of given dimension to the result of the previous stage [9]. The construction of the sparse matrix of the signed operator $\partial_1 : C_1 \to C_0$ is straightforward: for each $u_1^h = u_0^{k_2} - u_0^{k_1}$, just write $\partial_1[k_2, h] = 1$ and $\partial_1[k_1, h] = -1$, by convention for $k_2 > k_1$. For details of quotient operations, see Section 3.2.

Example 2.1 (Merge of two complexes with incompatible boundaries). Here we discuss the merge result when the input collection is defined by a pair of adjacent complexes with incompatible sub-complexes along their interface. Such a 2D example is displayed in Figure 4. Another simple example may include two tetrahedralized unit cubes that are incident on a planar face triangulated into two triangles but along different diagonals. Similarly to the 2D case, each boundary triangle would be fragmented against all the incident ones, producing fragmented output faces, so that the result on the common affine support (say, the vertical plane) would be exactly four triangles, because each input triangle is fragmented by the other diagonal. A larger example is shown in Figure 1a, where the 2D arrangement generated by a number of random line segments is displayed.



(a) (b) (c)

Fig. 4. Merge of two 2D complexes with incompatible boundaries: (a) Input data $\mathcal{S}_2 = \Sigma^1 \cup \Sigma^2$; (b) independent fragmentation of 2-cells $\sigma \in \mathcal{S}_2$ induced by $\mathcal{I}(\sigma)$; (c) *local* arrangement $X_2 = \mathcal{A}(\mathcal{S}_1(\mathcal{S}_2))$ generated by *Merge*, 2D *TGW*, and *Congruence* algorithm pipeline.

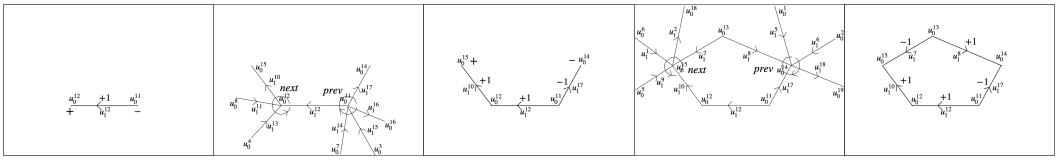
Note in Figure 4 that: (i) each 2-cell σ is processed *independently* as $\Sigma = \{\sigma\} \cup \mathcal{I}(\sigma)$; (ii) the bigger diagonal is fragmented by the normal 1-cell; (iii) the unit 2-chains are reconstructed in 2D (before reduction by congruence) as 1-cycles by the TGW algorithm; (iv) 0-cells and 1-cells from fragmented diagonal are finally identified *mod congruence*.

2.3 Topological gift wrapping (TGW)

The algorithm discussed here is used to compute topologically in 2D/3D, respectively, the sparse matrices of signed operators ∂_2 and ∂_3, starting from ∂_1 and ∂_2 input, respectively. The matrix $[\partial_d]$ of the linear map $C_d \to C_{d-1}$ between linear spaces contains by columns the target space representation of domain space basis elements, as closed $(d-1)$-chains, *i.e.*, $(d-1)$-cycles. Within the computational pipeline discussed in this paper, TGW is used *locally* for each 2-cell to be decomposed,

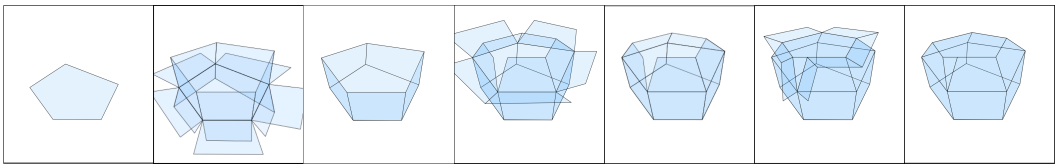
[5]We use this term to denote the chain spaces in each $C_\bullet(\sigma)$, $\sigma \in \mathcal{S}_2$, after fragmentation.

and *globally* to generate the 3-cells of the arrangement of the ambient space $\mathbb{E}^3$. The algorithm pseudocode is given and discussed in Section 3.3. The needed extensions for handling holes and non-connected components are detailed in Section 3.4.2.



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

Fig. 5. 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 $\partial c = \emptyset$. The step-by-step computation is discussed in Example A.1.



(a) (b) (c) (d) (e) (f) (g)

Fig. 6. Extraction of a minimal 2-cycle from $\mathcal{A}(X_2)$: (a) initial (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.

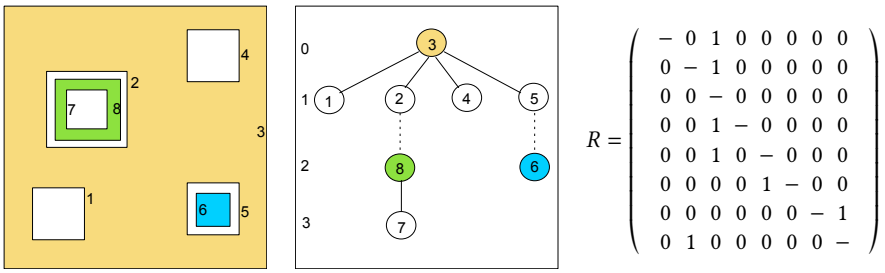
The topological method introduced here, reminiscent of the "gift-wrapping" algorithm [28, 59] for computing convex hulls of 2D and 3D discrete sets of points, is detailed and formalized in Section 3.3. The TGW algorithm takes a sparse matrix $[\partial_{d-1}]$ as input and produces in output a sparse matrix $[\partial_d^+]$, augmented with the outer cell. A geometric embedding function $\mu : X_0 \longrightarrow \mathbb{E}^d$ is used to compute the angular ordering, around some $(d-2)$-cells, of $(d-1)$-basis elements in the boundary's coboundary, while wrapping up a $(d-1)$-cycle, as illustrated in Figures 5 and 6. The built (minimal) cycles are set as columns of $[\partial_d]$, in the construction of the C_d basis. Note also that, once the ordered sets of basis elements is fixed, columns contain the *coordinate representation* of $(d-1)$-cycles, built from group coefficients $(\{-1, 0, 1\}, +) \simeq \mathbb{Z}/3\mathbb{Z} = \mathbb{Z}_3$. Analogously, boundaries and coboundaries of chains are calculated by multiplication of operator matrices times proper coordinate vectors of such coefficients.

2.4 Non-connected components (Holes)

The outer cell of the space arrangement $X = \mathcal{A}(\mathcal{S})$ might have a non-connected boundary, compraising more than one $(d-1)$ cycle[6], like a 3-ball minus a smaller concentric 3-ball. Analogously, X might contain both non-connected and possibly nested components. The TGW algorithm actually computes all of the boundary cycles, that must be properly handled in order to produce a single $[\partial_d]$ matrix: first decompose the input $[\partial_{d-1}]$ into connected components; then assemble/remove the empty cycles.

[6]Called a *shell* in the literature of solid modeling.

2.4.1 Decomposition of 2-skeleton. Consider a bipartite graph $G = (N, A)$, with $N \simeq \Lambda_2 \cup \Lambda_0$, and $A \subseteq \Lambda_2 \times \Lambda_0$, associated with the sparse characteristic matrix encoding the *incidence* relation. G has one node for each 2-cell, one node for each 0-cell, and one arc for each incident pair. Therefore, the arcs in G are one-to-one with the nonzero elements of the A matrix. By computing the maximal point-connected components of G, we subdivide the X_2 skeleton into h connected components: $\mathcal{X}_2 = \{X_2^p\}$, such that $1 \leq p \leq h$. 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 2. Then, subdivide $[\partial_3^+]^p$ 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 outer cell $\sigma^p \in X_3$. The set $S = \{c^p\}$ of h disjoint 2-cycles, is the initialization of the set of X_d *shells*. Other (empty) shells of X_d can be discovered later from mutual containment of S elements.



(a) (b) (c)

Fig. 7. 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. Removing the dashed arcs produces a *forest* of small *trees*; (c) matrix of transitively reduced R. Note that the ones equate the number of edges in the graph.

2.4.2 Shell discovering and assembling/removing. Then add to the set S the empty cycles (shells) already included within some non-contractible cell. We call them *input holes*. We discover the (unmodified) input holes by direct inspection of matrices $[\partial_3^p]$, since holes are represented there as *pairs of columns with zero sum*. In fact, each input hole, if non intersected by other data, returns unmodified, and produces *two opposite columns* in the component matrix $[\partial_3^p]$ it belongs to. Each corresponding pair of columns has all non-zero elements on the same rows, but with opposite signs (orientations). The inspection is done by a sort of scan-line algorithm working on the rows of each $[\partial_3^p]$ matrix, that recognizes the emerging pairs of opposite columns, and stores each pair (candidate hole) until its contents eventually differ. The algorithm proceeds by moving from a row to the next one, until the bottom row is reached and the set, possibly empty, of holes is returned. One column from each pair (with proper sign—see 2.4.3) is removed from the component matrix $[\partial_3^p]$, and its opposite 2-cycle is added to the set S of shells.

2.4.3 Transitive reduction of shell poset. The antisymmetric *containment* relation between shells (2-cycles) in S is computed for all $(c_3^i, c_3^j) \in S^2$, by the containment test $PointInShell(u_0^i, c^j)$ between a single vertex $u_0^i \in c_3^i$ and the cycle c_3^j.

The general 3D case is described here. To answer whether u_0^i, and hence c_3^i, is *internal* or not to the shell c_3^j, let consider the point $\mathbf{p}^i = \mu(u_0^i)$ and the 2-cells in $[c_2^j] = [\partial_3][c_3^j]$, with μ the embedding function $U_0 \to \mathbb{E}^3$. A naive containment computation, where each point $\mathbf{p}^i$ is tested for containment against each c_2^j cycle, $1 \leq i, j \leq s = \#S$, would be computed in quadratic time $O(s^2)$.

An efficient $O(s \log s)$ procedure is instead embraced here, by using two (i.e., $d-1$) one-dimensional *interval-trees* for the containment boxes of $\cup c_2^j$ elements, $1 \leq j \leq s$, in order to select only the terms in each $[c_2^j]$, whose boxes intersect a ray (degenerate box) from the tested point. The containment test in 3D is then executed by intersecting the ray from $\mathbf{p}^i$ with the planes containing the 2-cells in the query output, and testing for 2D point-polygon containment in their planes (via maps to the $z = 0$ subspace). Each planar point-polygon test is executed in time linear with the number of edges on the boundary of the 2-cell. All elements of column i of $[S^2]$ are so produced after a single query on interval trees, by considering the parity of positive answers to containment tests.

Then, the directed graph of transitive reduction R of the containment relation S^2 is extracted. Since S^2 is antisymmetric, the graph of reduced R is a tree. If the edge-set of this tree is empty, no disjoint component of X_3 is contained inside another one, and both X_3 and ∂_3 may by assembled by disjoint union of 3-cells of X_3^p and columns of $[\partial_3]^p$, $(1 \leq p \leq h)$, respectively. If, conversely, the above is not true, then reduce the R graph to a forest of small trees by cancelling the arcs at even distance from root (see Figure 7) and, for each arc (i, j), discover which *cell* of the container component X^i actually contains the contained component X^j, *i.e.*, its shell c_2^j and its possibly non-empty interior. More complex intersecting situations are impossible by construction, since the components are a priori disjoint. Therefore, in case of containment, one component is necessarily contained in some *empty cell* of the other.

2.4.4 About robustness of computations. Robustness issues appear everywhere in geometric computing, due to both numerical and/or topological problems. In our case we may claim that the TGW algorithm terminates correctly every time that its input is topologically correct. This is always true in 2D by construction, via elimination of subgraphs which are not 2-connected. In 3D, the input matrix $[\partial_2]$ may, or may not, be topologically correct. It is flawed when it contains some 1-cell incident to just one 2-cell, i.e. some columns with a single non-zero. In this case the TGW loops and does not terminate. When the algorithm terminates, the output $[\partial_3^+]$, i.e., the basis of independent 3-cells and the exterior cycles, are correctly computed by construction. With infinite precision computations the TGW algorithm would always terminate correctly. The possible flaws of 2-skeleton (hence of $[\partial_2]$) depend on congruence errors, i.e. on incompatible common boundaries between decomposed 2-cells. Such topological errors are generated by numerical errors. Sources of numerical errors in our pipeline may only reside, by design, in the pairwise intersection of 2D line segments, and in the ϵ bound allowed to identify very closed vertices. For example, there are some configurations where the diagonal of the red triangle in Figure 4 and the orthogonal line segment do not intersect numerically, so creating a topological error. Boundary compatibility may be coerced, once discovered through the above column check, by: (a) accepting intersection parameters slightly outside their $[0, 1]$ domain; (b) enforcing the subdivision process via `Float128` variables later casted to `Float64`; (c) immediate identification of the two point instance generated; (d) locally using a greater diameter for numerical identification of (quasi-)congruent vertices.

2.5 Efficiency of array representation of topology

For reader's convenience, we give in Appendix A definitions and facts about computing with chains and cochains, and provide some examples of elementary topological computation with chains, cochains and their operators. Be it noted about our *topology representations* that:

- p-cells (as either 0-chains or p-cycles) are given by *sparse 1-arrays* of signed numbers;
- p-complexes (as cell-sets, or bases of linear spaces, or graded linear transformations) are represented by *sparse 2-arrays* of signed numbers.

They have smaller space complexity than common data structures of well-known efficient representations [36, 105] of Solid Modeling. For example, with regard to the B-rep of a closed 2-manifold A, we have $Space(A) = Space([\partial_2]) + Space([\partial_1]) = 2\#\mathsf{E} + 2\#\mathsf{E}$, where $[\partial_2], [\partial_1]$ are sparse matrices of boundary operators, and $\#\mathsf{E}$ is the number of unit 1-chains (edges). Hence the storage required by $Space(A)$ is equal to 2/3 of *half-edge* [69], largely used in Computational Geometry, and 1/2 of *winged-edge* [12], often used as a reference representation for manifold Solid Modeling [1].

The cardinality of *all* the incidence/adjacency relations between p-cells and q-cells ($1 \leq p, q \leq 3$) is also minimal, according to [105]. For example, $\#\mathsf{FV} = O(Space([\partial_2][\delta_1]) = 2\#\mathsf{E})$, where V are the vertices of a complex and EV, FV are binary incidences of edges and faces with vertices. Therefore, every set of local queries about the 3×3 incidences/adjacencies between p-cells can be answered by multiplication, via software kernels for *sparse matrix* product and transposition, just by collecting the coordinate vectors of unit chains, "subject" of elementary queries, as *columns* of a sparse Q matrix, and by left-multiplying Q times one/two operator matrices $[\partial_1]$ and/or $[\partial_2]$, suitably ordered and/or transposed [36], to get the algebraic equivalent of multiple database queries at once.

3 CHAIN-BASED ARRANGEMENT ALGORITHMS

In this section, we provide a slightly simplified pseudocode of the main algorithms introduced in the previous section, and discuss their worst-case complexity. An implementation, very similar to this pseudocode, is available as open-sourced Julia package in Github[7].

The pseudocode style is a blend of Python and Julia styles. Some words about notations: greek letters are used for the *cells* of a space partition, and roman letters for *chains* of cells, all coded as either signed integers or sparse arrays of signed integers. In order to provide a formalized writing of the *pseudocoded algorithms* given in this section, we need to introduce the following conventions.

$[\partial_d]$ or $[c]$ stand for general matrices or column matrices, respectively, whereas $\partial_d[h, k]$ or $c[\sigma]$ stand for their indexed elements. Also, $|c|$ stands for *unsigned* (nonzero) indices of the (sparse) array $[c]$. 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 $A = A - B$.

3.1 Arrangements of 2-cells (Merge algorithm)

The sequence of computations performed on each 2-cell $\sigma \in \mathcal{S}_2 \subseteq \mathcal{S}$ is discussed in the following. A visualization of the decomposition process, discussed in Example 2.1, is shown in Figure 4.

3.1.1 Fragmentation of a 2-cell. In a first stage, the subset $\mathcal{I}(\sigma)$ of 2-cells *potentially intersecting* σ is computed (see Figure 2c). This is done by intersection of results of three queries about the σ bounding box, against the three 1D *interval trees* generated at the beginning of the pipeline. Each 1D interval-tree was built using one of side intervals of the 3D containment boxes of input 2-cells. In a second stage, the set $\Sigma = \{\sigma\} \cup \mathcal{I}(\sigma)$ is transformed so that σ is mapped into the $z = 0$ subspace (see Figure 2d). The mapped 2-cells are used to compute a *set of line segments* in $\mathbb{E}^2$, generated by intersection of edges of 2-cells in Σ with the 2D plane. Alternate pairs[8] of such intersection points are finally joined, along the intersection line with $z = 0$ of each 2-cell in Σ (Figures 2e and 3a).

The planar processing of the 2-cell σ continues by pairwise intersecting all computed line segments and producing a linear graph, as shown in Figure 3b. The dangling [9] edges are removed,

[7] https://github.com/cvdlab/LinearAlgebraicRepresentation.jl

[8] The 2-cell being intersected with $z = 0$ may be non-convex, and its 1-cells may produce and even number $k > 2$ of intersection points along the same line.

[9] In a d-complex, *dangling edges* are p-cells, p<d, that are not contained in any boundary cycle of a d-cell. They are the interior structures of SGC cells. In Solid Modeling terminology, they are called non-regular subsets, whence the term *regularized* Boolean operation.

ALGORITHM 1: *Subdivision of 2-cells*

Input: $\mathcal{S}_2 \subseteq \mathcal{S}_{d-1}$ # *collection* of all 2-cells from $\mathcal{S}_{d-1}$ input in $\mathbb{E}^d$
Output: $[\partial_2]$ # CSC (Compressed Sparse Column) 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
  $\Sigma = M(\mathcal{I}(\sigma) \cup \{\sigma\})$ # 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 $\lambda \not\subset \{\mathbf{q} \,|\, x_3(\mathbf{q}) = 0\}$ **then** $\mathcal{P}(\tau)$ += $\{\mathbf{p}\}$ # append 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 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
$[\partial_2] = TGW([\partial_1])$ # output computation via TGW algorithm in 2D
return $[\partial_2]$

by computing the maximal 2-vertex-connected subgraphs[10] [101], with the Hopcroft's and Tarjan's algorithm [58]. Only the non-external biconnected components enter the following computations, since the other graph parts are either external to σ or certainly dangling (1-connected subgraphs), and will contribute separately to the space arrangement (Figure 3b). Finally, the oriented 2-chain of the partition $\mathcal{A}(\Sigma)$ is computed as shown in Figures 2g and 3d, using the TGW in 2D, so generating the $\partial_2(\sigma)$ matrix from $X(\sigma)$. The fragmentation process is repeated for each $\sigma \in \mathcal{S}_2$, with each geometric map $\mu(\sigma) : X_0(\sigma) \longrightarrow \mathbb{E}^2$ composed with its inverse transformation back to $\mathbb{E}^3$.

3.1.2 Complexity of 2-cells subdivision. The time complexity of Algorithm 1 is bounded by the number n of 2-cells in the input collection $\mathcal{S}_{d-1}$ times the worst-case cost required by the subdivision of one of them. In turn, this cost depends on the size and the distribution of the actual input, *i.e.*, on the number of potentially intersecting 2-cells in $\mathcal{I}(\sigma)$. The computation of each $\mathcal{I}(\sigma)$ set is done in the query time of interval trees, i.e., in time $O(\log n + k)$, where $k << n$ is the average length of the result. Since $\log n$ is usually dominated by k, we may append this factor to our bound for the search of all potentially incident sets. Therefore, the computation of $\{\mathcal{I}(\sigma) \mid \sigma \in \mathcal{S}_2\}$ for all input 2-cells is $O(kn)$, with k depending on the density of data, and $O(n^2)$ in the worst case $k = n$.

In all the regular cases we usually meet in computer graphics, CAD meshes and engineering applications, the number of 2-cells incident in (even on the boundaries of) a given cell σ is bounded by a constant number k_1. If the maximum number of 1-cells on the boundary of a 2-cell is k_2, then the whole computation of Algorithm 1 requires time $O(k_1 k_2 n + A)$, where A is the time needed to compute the quotient sets, *i.e.*, to glue all $X_2(\sigma)$ in $\mathbb{E}^d$ space. When $d = 3$, the affine transformations $\mathbf{Q}_\sigma$ of each set Σ (see 2.1) are computable in $O(1)$ time; building a static *kd*-tree

[10] A connected graph G is *2-vertex-connected* if it has at least three vertices and no articulation points. A vertex is an *articulation point* if its removal increases the number of connected components of G.

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 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 to canonical form (sorted 1-array of integers) is done in $O(1)$ for each edge, so giving $A = O(m \log^2 m) + O(m \log m) + O(1) = O(m \log m)$. In conclusion, the worst-case running time of Algorithm 1 is $O(kn + k_1k_2n + m \log m) = O(n(k + k_1k_2) + m \log m)$, degenerating to $O(n^2)$, which is known to be the worst-case bound [66] for hidden line removal.

3.2 Quotient sets computation (Congruence algorithm)

Small sparse matrices of signed operators $\partial_2(\sigma) : C_2(\sigma) \longrightarrow C_1(\sigma)$ have already been assembled *independently* in 2D for each fragmented σ, *i.e.*, for each X_2^σ, as detailed in the previous Section 3.1.1. The output of that pipeline stage is a collection $\mathbb{C} := \{C_\bullet(\sigma), \sigma \in \mathcal{S}_2\}$ of small chain complexes, one for each input 2-cell, embedded in $\mathbb{E}^3$. They were built by repeatedly applying in 2D the TGW algorithm (see Section 3.3) and mapping back the results in 3D. The quotients of chain spaces modulo the p-congruence relations are calculated at this point, starting from $p = 0$. Therefore, the unit 0-chains are identified numerically via their geometric maps and snap rounded by numerical identification of nearby-coincident points using a kd-tree. The congruent unit 1-chains and 2-chain are identified symbolically, making use of their unique canonical indexed representation. The canonical representation of a unit d-chain is the array of sorted indices of the unit elements of its $(d-1)$-cycle. The 2-cells of the output 2-complex $X_2(\mathcal{S}_2)$ embedded in $\mathbb{E}^3$, written as 1-cycles, *i.e.*, as linear combinations of signed 1-cells, are stored by column in the matrix of the operator $\partial_2 : C_2 \longrightarrow C_1$. A 1-cell τ, is written[11] by convention as $1u^i_{0k} - 1u^i_{0h}$ when $k > h$, and is oriented from u^i_{0h} to u^i_{0k}. The conventional rules used in this paper about sign and orientation of cells are summarized at the end of Section A.1.2.

3.3 Computation of ∂_2 and ∂_3 (TGW Algorithm)

3.3.1 Topological Gift Wrapping. The algorithm was introduced in Section 2.3. Here we provide a readable pseudocode, with the only *caveat* that it actually computes a redundant set of generators for C_3 (resp. C_2), as minimal connected 2-cycles (resp. 1-cycles) from a ∂_2 (resp. ∂_1) matrix. A step-by-step formalized example of computation of a unit 2-chain as 1-cycle, using the TGW algoritm, is discussed in Example A.1. The given pseudocode makes use of math symbols and high-level math operations; the actual implementation in Julia uses sparse arrays and discrete coordinates in $\{-1, 0, 1\}$, to achieve an efficient execution in terms of storage space and computation time.

Note the precondition of Algorithm 2, warning that the method used will compute the ∂_d matrix only for a cell decomposition of d-space. In fact, only in this case the $(d-1)$-cells are shared by exactly *two* d-cells, including the outer cell. This condition implies that the input cellular complex it applies to should be a (possibly non-connected) CW-complex, with all cells homeomorphic to spheres. Note also that the matrix of the boundary operator for the d-chain space of a cellular complex with holes as well as inner and outer components will be built starting from output of Algorithm 2 in a later stage. The termination predicate of Algorithm 2 is a consequence of the above property: the algorithm terminates when all incidence numbers in the *marks* array are 2, so that their sum is exactly $2n$, where n is the number of $(d-1)$-cells, equal to the number of columns in the input matrix $[\partial_{d-1}]$.

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

3.3.2 Valid input and unique output. The algorithm works properly with legitimate input. In particular, 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 subgraphs.

The validity set of the input may contain 2-skeletons of 3-complexes, boundaries of solid models, sets of manifold boundary components of non-manifold solid models. Of course, to apply the algorithm to data which do not determine a partition of the embedding space does not make sense and produces an empty result. When applied to valid input, as described above, the TGW algorithm is always *correct*, because always produces the set of generators for C_d that satisfies the Eq. 1 below:

$$[\partial_d] = (a_{ij}) \quad \text{where} \quad \sum_{i=1}^{\#X_{d-1}} \sum_{j=1}^{\#X_d} |a_{ij}| = 2\,(\#X_{d-1}), \tag{1}$$

The results are also *unique*, modulo reordering, 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 boundary cycle, which is impossible. There are no ambiguities in the algorithm, since in a d-complex every two d-cells share at most two $(d-1)$-cells, or exactly two if the outer d-cell is considered. Also, it halts when this last condition is exactly reached. Note that the suitable choice of the next "petals" from "corolla" (see the pseudocode in Algorithm 2) implies that a 2-cell cannot be used more that twice.

3.3.3 Complexity of 3-cell extraction. In three dimensions, Algorithm 2 constructs iteratively (outer **while**) one unit 3-chain (represented as a 2-cycle, *i.e.*, as a closed 2-chain), building the corresponding column of the matrix $[\partial_3^+]$, and so adding one outer boundary column for each connected component of the input complex, as detailed in 3.4.

The space complexity of a 3-cell is measured by a set of triples (`row`, `column`, `value`) implemented as a triple of arrays (`I`, `J`, `Values`) for non-zero values[12] in 3-cell column, *i.e.*, with its representation as cycle of unit 2-chains. 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* unit 2-chain in the boundary shell. The search for *next* or *prev* 2-cell as *adj* for each pivot requires the circular sorting of this permutation subgroup of 2-cells incident to each 1-cell on each boundary of an incomplete 2-cycle. A naive circular sorting of 2-cells, using the angles between normals to their planes, would always work only with convex 2-cells; in our case, since 2-cells may be non-convex, this ordering may go wrong when computing the face normal with badly chosen third vertex. For this reason a CDT (Constrained Delaunay Triangulation) of each incident 2-face is needed. It is implemented using the `Triangle` library, ported[13] by our group to Julia language. Consequently, we have several sorts of triangle sets around an edge, where each set is bounded by a very small integer, hence each sort is $O(1)$ timewise. The total number of such sorts is upper bounded by the number of $(d-1)$-cells on the d-cell boundary (equal to 6 for cubical 3-complexes, and to 4 for simplicial 3-complexes, and to a small integer in general).

The subsets to be sorted are encoded in the columns of the incidence matrix from 2-cells to 1-cells, *i.e.*, by the i, j indices of non-zero elements of $[\partial_2]$. The computation of the (unsigned) $[\partial_2]$

[12] The coordinate (`COO`) representation of sparse matrices [25] is an array of triples $(i, j, value)$.
[13] See Notes 15 and 16

ALGORITHM 2: *Computation of signed* $[\partial_d^+]$ *matrix*

```
/* Pre-condition: d equals the space E^d dimension, such that (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 of (d − 1)-cycles
[∂_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^+]
```

may be performed through SpGEMM[14] multiplication of two sparse matrices (see [36]), hence in time linear with the size of the output, *i.e.*, with 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 $(d-1)$-cycle complexity is bounded by k.

3.4 Isolated shells (Holes algorithm)

In the general case, both the outer cell and the inner cells may contain isolated holes and/or isolated components, i.e., subcomplexes whose outer boundaries do not intersect each other. Talking of isolated holes is quite improper, since holes are *never* empty within an arrangement, *i.e.*, a partition of the ambient space, but contain an isolated component within their boundary represented as a $(d-1)$-cycle. The aim of this section is to discuss the handling of isolated components and their boundaries, to be considered holes within their container space.

The TGW Algorithm 2 produces CW-complexes, despite the fact that the subdivision of Merge Algorithm 1 may generate non contractible d-cells, *i.e.*, cells with holes. These spaces are handled by

[14] SpGEMM is a subroutine for matrix multiplication between two general sparse matrices [25], *i.e.*, no banded, nor Hermitian, *etc.* Name derived from BLAS rules [10].

ALGORITHM 3: *Non-intersecting shells*

Input: $\mathsf{LAR}_{d-1}, [\partial_{d-1}]$ # for $d = 3$: FV, ∂_2
Output: $\mathsf{LAR}_d, [\partial_d]$ # 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
$S = []$ # 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
  S += $[c]^p$ # append the boundary shell to the shell array
end
for $i, j \in \{1, \ldots, h\}, i < j$ **do** # for each shell pair $(c^i, c^j) \in R$,
  $(R[i,j], R[j,i]) :: Bool \times Bool \leftarrow PointSet(u^i_0 \in c^i, c^j)$ # containment test of u^i_0 in c^j
end
$R = \{(i,j)\} \leftarrow Tree(TransitiveReduction(R))$ # set of arcs of reduced containment tree of shells
if $R \neq \emptyset$ **then** # if the containment tree of shells is not empty
  for $(i,j) \in R$ **do** # for each shell pair (c^i, c^j) such that $dist(c^j)\%2$!= 0
    $\rho = FindContainerCell(u^i_0, c^j, \mathsf{LAR}_{d-1})$ # look for a d-cell ρ such that $u^i_0 \in |c^i| \subseteq |\rho| \subseteq |c^j|$
    $[\partial_d]^j$ -= $\partial^j_d[\rho]$ # 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 $\mathsf{LAR}_d, [\partial_d]$

combining standard CW-complexes, *i.e.*, with cells homeomorphic to spheres, and by adding d-cells to the interior of d-cells. In other words, the boundary of holes in a cell coming from disconnected sub-complexes is merged in the container cell. The orientation is handled depending on the parity of relative containment relation. The management of isolated boundaries concerns essentially the adjoining/removal of columns in the final boundary matrix.

3.4.1 Synthesis of the Holes algorithm. We need to consider two main issues: (a) the computation of maximal connected components of X_{d-1} may produce $h > 1$ disconnected *d-components* of the output complex X_d; (b) the inclusion of components within single cells of the output d-complex: see Figure 7. In the following we list the main stages of the *Holes* algorithm to take care of these issues. Our goal is the computation of both the X_d skeleton, and the ∂_d operator for spaces with multiple components nested into holes. Note that exactly the same points apply (scaled in dimension) before and after TGW execution, for both *local* arrangements in 2D generated by every input 2-cell σ, and the *global* arrangement in 3D generated by the whole set of input complexes, respectively. This fact gives a cue for a possible multidimensional extension.

3.4.2 Non-intersecting shells. If the shell-set $S \neq \emptyset$, then the h *isolated* boundary components $(0 \le p \le h)$ in S must be compared with each other, to determine their relative containment, if any, and consequently their orientation. The $h \times h$ binary and antisymmetric matrix $M = (m_{ij})$ of the relation is built, by computing each element m_{ij} $(i < j)$, with a single point-cycle ray firing, because the two corresponding cycles (columns i and j) are guaranteed not to intersect. The attribute of

c_j as outer/inner, and hence its relative orientation is given by the parity of c^j in R. When the cancellation 3.4.1.4 of empty cells has been performed for all "solid" arcs of R (see Figure 7), the updated matrices $[\partial_d]^p$ can be assembled into the final ∂_d operator matrix.

3.4.3 Complexity of shell management. The computation of the connected components of a graph G can be performed in linear time [58]. 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 $O(h^2)$ point-cycle containment tests, linear in the size of a cycle, so spending a time $O(h^2 n)$, with h number of shells, and n average size a cycle. 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 $O(n\#X_d)$, *i.e.*, linear with the product of the number of d-cells and their average size n as chains of $(d-1)$-cells, with n size of the average isolated cycle.

3.5 The whole picture

A short synthesis of sequential steps of the whole computational pipeline, from input collection to chain complex output, follows in the more general setting, with both isolated components (within the outer cell), and possibly nested isolated components (within holes in inner cells).

Input Facet selection, *i.e.*, construction of the collection $\mathcal{S}_{d-1}$ from $\mathcal{S}_d$, using LAR.
Indexing Spatial index made by intersection of d interval-trees on bounding boxes of $\sigma \in \mathcal{S}_{d-1}$.
Decomposition Pairwise $z = 0$ intersection of line segments in $\sigma \cup \mathcal{I}(\sigma)$, for each $\sigma \in \mathcal{S}_{d-1}$.
Congruence Graded bases of equivalence classes $C_k(U_k)$, with $U_k = X_k/R_k$ for $0 \leq k \leq 2$.
Connection Extraction of $(X_{d-1}^p, \partial_{d-1}^p)$, maximal connected components of X_{d-1} $(0 \leq p \leq h)$.
Bases Computation of redundant cycle basis $[\partial_d^+]^p$ for each p-component, via TGW.
Boundaries Accumulation into H += $[o]^p$ (hole-set) of outer boundary cycle from each $[\partial_d^+]^p$.
Containment Computation of antisymmetric containment relation S between $[o]^p$ holes in H.
Reduction Transitive R reduction of S and generation of forest of flat trees $\langle [o_d]^p, [\partial_d]^p \rangle$.
Adjoining of roots $[o_d]^r$ to (unique) outer cell, and non-roots $[\partial_d^+]^q$ to container cells.
Assembling Quasi-block-diagonal assembly of matrices relatives to isolated components $[\partial_d]^p$.
Output Global boundary map $[\partial_d]$ of $\mathcal{A}(\mathcal{S}_{d-1})$, and reconstruction of 0-chains of d-cells in X_d.

4 COMPARISON WITH OTHER APPROACHES

In this section we mention some relevant connections of the present approach with recent papers concerning similar topics, and discuss some remarks in relation to our own work.

In [2] a topological approach to homology is introduced for subclasses of subdivided spaces constructed by combinatorial and generalized maps. Generalized map (*Gmap*) is a combinatorial model which allows for representing and handling subdivided objects [30] via "connecting darts" between cell pairs. Gmaps are used to describe the topology of manifold-like cellular objects where p-cells are homeomorphic to p-spheres. Alayrangues, Damiand, Lienhardt, and Peltier give an algorithm to build signed boundary maps for $0 \leq i \leq 3$, focusing on the equivalence between computing homology via Gmaps and via simplicial complexes.

The time complexity of boundary maps in [2] is linear in the number of incidence numbers. In [36], we already obtained the same result, linear in the sparse output size, for computation via SpGEMM multiplication (see, *e.g.*, the Example A.6) when the input complex is known. The complexity of TGW for computing the *unknown* $X_d = \mathcal{A}(\mathcal{S}_{d-1})$ is obviously higher (see Section 3.3.3), and equates the standard in Solid Modeling. The main difference with our approach is that Alayrangues and colleagues start from a *given* Gmap cellular model, whose construction is quite complex and

requires interactive operations with a graphical user interface or a symbolic logic systems (such as INRIA's Coq [32]) with a formal specification language. If simplicity metric matters, our linear algebraic representation of chains with sparse arrays compares well with chains of Gmaps.

With Selective Geometric Complex [88], Rossignac and O'Connor proposed a significant extension of topological CW-complexes, allowing for p-cells with structures of dimension $0 \leq k < p$ internal to cells, *i.e.* not necessarily embedded in a cell boundary. The selection bit associated to each cell allows selective choice of substructures. The association of incidence and local ordering among incident cells is maintained via hierarchical links, analogous to the Hasse diagram between vertices, edges, and faces, as well with geometric extents of non-linear surface patches. Two attributes for *c.boundary* and *c.star* of a cell return the cells on its boundary and those it is on the boundary of. The search for boundary of more complex substructures is algorithmic.

In the present paper, we conversely use graded and combinable linear operators for boundary and coboundary to traverse, both locally and globally, the incidence hierarchy. Hence we obtain, via multiplication of sparse matrices and vectors, a complete linear characterization of the space topology. Even local updates to topology, via Euler operators, can be done algebraically [34]. Another significant difference concerns the large amount of information and pointers associated by SGC to each cell, including extent, dimension, boundary, activity bit, and extendable attributes. Conversely, in the present paper an oriented unit p-chain is characterized only by a signed integer index to U_p basis, and by its signed $(p-1)$ boundary cycle, stored as a sparse column in $[\partial_p]$. Of course, all topological queries, both local and global, are allowed by suitable SpGEMM multiplication. We allow for nesting inner cycles (holes) and substructures to the cells, but not for explicit containment of internal edges and points. The most part of topological algorithms is algebraic in nature.

[109], by Zhou, Grinspun, Zorin, and Jacobson, computes mesh arrangements for solid geometry, takes as input any number of triangle meshes, resolves triangle intersections in 3D, and assigns a winding number vector to subdivided cells, to evaluate variadic Boolean expressions. Their data are represented by (small) BSPs enriched with explicit convex surface patches on nodes, and adjacency structure between nodes, together with a large amount of additional information. The [109] approach can be split into two stages: first, *adding iteratively meshes* to an arrangement; second, executing all classic Boolean operations. Contrariwise, we do not add each input to the previous result but, in decomposition stage, operate *independently* on each input 2-cell (see 2.2), according to an embarrassingly parallel data-driven approach. It is also remarkable that the present approach works with more general meshes: sets of 2-manifolds with- and/or without-boundary, sets of non-manifolds, sets of 3-manifolds, *etc.*, versus just sets of triangle meshes. We do not discuss it here, but extending our approach to Boolean operations and to Boolean functions is straightforward.

An extremely fast mesh repairing algorithm with guaranteed topology is described in [7], mostly based on floating point arithmetics, and requiring exact arithmetics only in relatively few situations. At variance with Attene [7], we not discriminate between manifold and non-manifold case, and do not use special data structure in any steps of the pipeline, except 1D interval-trees and kd-trees for acceleration. By identifying the conditions that make floating-point arithmetics not reliable, J.R. Shewchuk, the author of the Triangle library[15] [94, 95] used in our method, identifies the key for fast robust geometric predicates in adaptive precision floating-point arithmetic [93]. In fact, the numerical results we obtained on triangulations with large arrangements of 2D line segments are very fast. We ported the CDT (Constrained Delaunay Triangulation) functions from his C library to

[15] https://www.cs.cmu.edu/ quake/triangle.html

Julia language[16], and used it to triangulate on-the-fly each non-convex 2-cell, in order to correctly compute the ordering of "corollas" 2-cells around "pivot" 1-cells in the 3D TGW.

In [26], Campen and Kobbelt present a technique to implement operators that modify the topology of polygonal meshes at intersections and self-intersections, by combining an adaptive octree with nested binary space partitions. An analogous decompositive technique was introduced in the geometric language PLaSM [73, 78][17] since 2004 by Scorzelli, Paoluzzi and Pascucci, in the contest of progressive geometry detailing allowing parallel modeling with BSP trees [77, 90]. The technique is now being substituted by methods given in the present paper, since it does not guarantee sufficient robustness and speed. In [49], Guibas and Marimont describe a dynamic algorithm to compute the arrangement of a set of line segments in the *digital plane*, and to snap the intersection points at the pixel centers. We snap small clusters of very close (numerically "quasi-congruent") points, rounding at the center of their ϵ-neighboroughs in 2D, with average diameter of 10^{-16}, close to the resolution of IEEE-754 binary floating point.

Barki, Guennebaud, and Foufou claim that [11] presents an exact, robust, and efficient method to execute regularized Booleans on general 3D meshes. They use a triangulation of all faces, and reduce the intersection of two surfaces to the 3D intersection of two triangles. Their simple decomposition process for intersecting faces is very similar to the old paper [79] by Paoluzzi and his students. Note that both [11] and [79] contain a procedure for computation of regularized Boolean operations including isolated shells. The novel point about this matter here is that in the present paper outer and inner oriented shells, as well as isolated components, are handled through signed sum of closed chains and implemented as sum or difference of their vector coordinates in $\mathbb{Z}$ or $\mathbb{Z}/3\mathbb{Z}$.

Finally, we recall that *Half-edge*, the smallest known efficient representation [69] of topology of planar graphs and closed 2-manifolds by Muller and Preparata, largely used in Computational Geometry for triangulations and Voronoi diagrams, as well as in meshes for games, requires 6#E space. With the $[\partial_1]$ *and* $[\delta_1]$ operators given in the present paper, we obtain for this class the optimal size $\Omega(n) = 4\#\mathsf{E}$, equal to the input size: two vertices and two faces per edge. It is well known [105] that both EV and EF relations weight for 2#E, i.e., equal to the space occupied by $[\partial_1]$ or $[\partial_2]$ maps, which also allow for the algebraic equivalent of multiple database queries at once.

5 PAST DEVELOPMENT AND PROSPECTS OF THIS PROJECT

The first three authors started this project about computing with chains, cycles, cochains, and (co)boundary in a seminar series on novel algebraic methods for physical simulation and optimization of geometric design [year 2000]. This project was awarded the IBM SUR award in 2003. LAR sparse arrays, big geometric data and geometric services were discussed in many meetings in Rome, Paris, Madison, Berkeley, and Berlin. Our conversations produced some papers [34–36] that started a lasting sequence of web and face-to-face discussions, algebraic experiments and software tests, producing in recent years three open-sourced partial implementations in Python and Julia. Partial implementations were used for software-based experiments of user-tracking and interior geo-mapping in LAR-based Building Information Modeling (BIM), meta-design of a general hospital, and delivery of web services aiming at deconstruction and reuse of buildings [65, 76, 97, 98].

Currently, some of the authors have materialized a Julia package[18] [44] for topological and geometric design, including a first implementation of the algorithms in this paper. A second version is including vectorization on the GPU and task-based concurrency using native Julia [14, 15]. Next, our plan is to port the chain-maps pipeline on Nvidia's GPX-1, to merge with deep learning

[16] https://github.com/cvdlab/Triangle.jl
[17] https://github.com/plasm-language/pyplasm
[18] https://github.com/cvdlab/LinearAlgebraicRepresentation.jl

from imaging. We are already using the modeling approach introduced here relying on the Julian open-source library, for rapid development of building models from analysis of Italian Cadastre documents and building models from 3D images scanned by flying of drones.

We hope that the basic structures and algorithms discussed in this paper may also find some appropriate use when combined with representations for convolutional neural networks, based as well on tensors and linear algebra, in order to properly combine image understanding and geometric modeling. In particular, our approach to compute the chain complex[19] of an unknown space arrangement should match well with deep NNs [22, 46]. Also our first experiments with topological methods in medical imaging [27, 75] look promising.

6 SUMMARY OF RESULTS AND CONCLUSION

We have introduced a novel view on topology computation of space arrangements, that may find good use in disparate subdomains of geometric and visual computing, discussed an original computational architecture based on linear topological algebra, and claim that our approach is in tune with current trends towards hybrid hardware and its more advanced software applications. In particular, in this paper we provide a pseudocode implementation of the full computational pipeline: from a collection of virtual geometric objects to the chain complex (C_p, ∂_p) of their partition of space, giving a full characterization of the topology induced by the input. This result is obtained going beyond simplicial complexes, and working with general piecewise-linear topology with non-contractible cells. Among the strong points we cite: the compact representation; the combinable nature of maps, allowing for *multiple* queries about the 3×3 local topology relations, via fast sparse kernels for multiplication and transposition; the independent fragmentation of input cells through cell congruence; and the topological gift wrapping algorithm. Last but not least, the whole approach may be extendable to higher dimension. We believe that a real-time implementation of our algorithms on GPUs may generate new techniques for image understanding, in particular when inputs come from next-generation 3D cameras, going to be installed on self-driving mobile vehicles. Part of this work was developed within the framework of the IEEE standardization of model extraction from medical images [68].

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers who carefully read our manuscript and provided us with many useful comments. Alberto Paoluzzi and Antonio DiCarlo gratefully acknowledge the support of Antonio Bottaro CTO of R&D at Sogei, now CEO at Geoweb, who believed in our work, and the EU project medtrain3dmodsim. Vadim Shapiro is supported in part by National Science Foundation grant CMMI-1344205 and National Institute of Standards and Technology.

A APPENDIX

For readers' convenience, we recall here a few definitions and facts about computing with chains and cochains, mainly from [34] and [36]. Some simple examples of computations conclude this appendix. We use greek letter for *cells* and roman letters for *chains*, *i.e.*, for signed combinations of cells. With some abuse of language, cells in Λ_p and unit (singleton[20]) chains in C_p are often identified.

A.1 (Co)chain Complexes

[19] Of very general type, with basis cells possibly non convex and multiply connected.
[20] A set having exactly one element.

A.1.1 Cellular complex. Let X be a topological space, and $\Lambda(X) = \bigcup \Lambda_p$ ($p \in \{0, 1, \ldots, d\}$) a partition of X, with Λ_p a set of (relatively) open, connected, and manifold 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 $\Lambda_p = \Lambda_p(X)$ [51]. 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 cellular d-complex X is *orientable* when its d-cells can be coherently oriented. The *support space* $|\sigma|$ of a cell σ is its compact point-set.

A.1.2 Chain groups. Chains are defined by attaching coefficients to cells. Since one wishes to add chains, one has to pick coefficients from a set endowed with the structure of a commutative group, or stronger. Let $(G, +, 0)$ be a nontrivial commutative group, whose identity element is denoted 0. A p-chain of X with coefficients in G is a mapping $c_p : X \to G$ 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 be written in a unique way as a sum of elementary chains. Chains are often thought of as attaching orientation and/or multiplicity to cells: if coefficients are taken from the group $G = (\{-1, 0, 1\}, +, 0) \simeq (\mathbb{Z}/3\mathbb{Z}, +, 0)$, then cells can only be discarded or selected, possibly inverting their orientation (see [34]). A *p-cycle* is a *closed* p-chain, *i.e.*, a p-chain without boundary. It is useful to select a conventional orientation to orient cells automatically. 0-cells are considered all positive. Closed p-cells can be given a coherent (internal) orientation in according with the orientation of the first $(p-1)$-cell in their *canonical representation* sorted on indices of their $(p-1)$-cycles. Finally, a d-cell may be oriented as the sign of its oriented volume.

A.1.3 Chain spaces. To allow not only for chain addition, but also for linear combination of chains, coefficients should be taken from a set endowed with the structure of a field, such as $(\mathbb{F}, +, \times, 0, 1)$, where 0 and $1 \neq 0$ denote, respectively, the additive and multiplicative identities. *Unit* chains are elementary chains whose value is $u = 1\sigma$ for some cell σ. Each chain can be written in a unique way as a linear combination of unit chains $u \in U$, if the outer cell is not taken into account. Hence, the space of p-chains C_p is endowed with a standard (or natural) basis, comprised of all the independent unit p-chains. In particular, $\#U_d = \#\Lambda_d - 1$. Often, with some abuse of notation, one does not distinguish between a p-cell and the corresponding unit p-chain.

A.1.4 Characteristic matrices. Given a set $S = \{s_j\}$, the *characteristic function* $\chi_A : S \to \{0, 1\}$ takes value 1 for all elements of $A \subseteq S$ and 0 at 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 unit p-chains and columns are indexed by unit 0-chains, provides a useful representation of a basis for the linear space C_p. Permuting (reindexing) either rows or columns provides a different basis. While chains are mostly presented as formal sums of cells, in the actual implementation their signed coordinate vectors are used as *sparse* arrays, and in particular as CSC (Compressed Sparse Column) maps : $\mathbb{N} \to \{-1, 0, 1\}$.

A.1.5 Cochain spaces. Cochains are dual to chains: p-cochains map linearly p-chains to the underlying field $\mathbb{F}$. Unit p-cochains, that yield 1 when evaluated on one unit p-chain and 0 when evaluated on all the others, form the standard basis of the space of p-cochains C^p. The linear spaces C_p and C^p, being isomorphic, can be identified with each other in infinitely many ways. Different legitimate identifications, while affecting the *metric* properties of the chain-cochain complex, do not change the *topology* of finite complexes.[21] Since we shall use only the topological properties of finite chain-cochain complexes defined by piecewise linear cell complexes in Euclidean space, we feel free to chose the simplest possible identification, consisting in identifying each element of the standard basis of C_p with the corresponding element of the standard basis of C^p. In this paper, we take for granted that chains and cochains are identified in this trivial way.

A.2 Topology computing with chains

Example A.1. Figure 8 shows a fragment of a 1-complex $X = X_1$ in $\mathbb{E}_2$, with unit chains $u_0^k \in C_0$ and $u_1^h \in C_1$. Here we compute stepwise the 1-chain representation $c \in C_1$ of the central 2-cell of the unknown complex $X_2 = \mathcal{A}(X_1)$, using the Topological Gift Wrapping Algorithm 2. Refer to Figure 5a-e, repeated below, to follow stepwise the extraction of the 2-cell as 1-cycle.

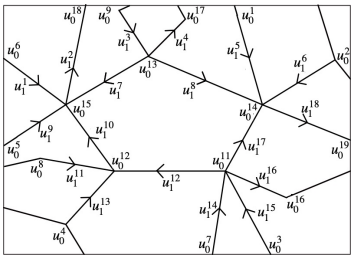
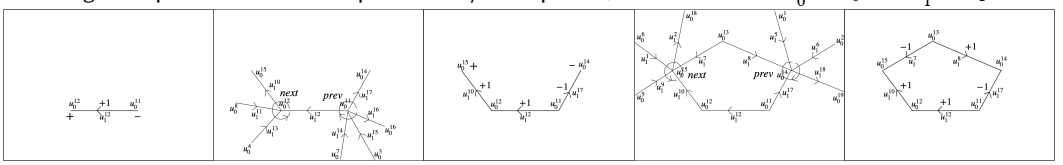


Fig. 8. A portion of the 1-complex used by Example A.1, with unit chains $u_0^h \in C_0$ and $u_1^k \in C_1$.



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

Step (a) Set $c = u_1^{12}$. Then $\partial c = u_0^{12} - u_0^{11}$.
Step (b) $\delta\partial c = \delta u_0^{12} - \delta u_0^{11}$ by linearity. Hence, $\delta\partial c = (u_1^{10} + u_1^{11} + u_1^{12} + u_1^{13}) - (+u_1^{12} + u_1^{14} + u_1^{15} + u_1^{16} + u_1^{17})$.
Step (c) By computing *corolla*(c), we get

$$\begin{aligned} corolla(c) &= c + next(c \cap \delta\partial c) \\ &= c + next(u_1^{12})(\delta u_0^{12}) - next(u_1^{12})(\delta u_0^{11}) \\ &= u_1^{12} + next(u_1^{12})(\delta u_0^{12}) + prev(u_1^{12})(\delta u_0^{11}) \\ &= u_1^{12} + u_1^{10} + u_1^{17}. \end{aligned}$$

If c is coherently orientd, then

$$c = u_1^{10} + u_1^{12} - u_1^{17}, \quad \text{and} \quad \partial c = u_0^{15} - u_0^{12} + u_0^{12} - u_0^{11} + u_0^{11} - u_0^{14} = u_0^{15} - u_0^{14}.$$

[21] The reader interested in the notions of measured and metrized chains is referred to [34, 35].

Step (d) Repeating and orienting coherently the computed 1-chain yields:

$$\begin{aligned} corolla(c) &= c + next(c \cap \delta\partial c) \\ &= c + next(u_1^{10})(\delta u_0^{15}) - next(u_1^{17})(\delta u_0^{14}) \\ &= u_1^{10} + u_1^{12} - u_1^{17} + next(u_1^{10})(\delta u_0^{15}) + prev(u_1^{17})(\delta u_0^{14}) \\ &= u_1^{10} + u_1^{12} - u_1^{17} - u_1^{7} + u_1^{8} \end{aligned}$$

Step (e) $\partial corolla(c) = \emptyset$, and the extraction algorithm terminates, giving

$$c = u_1^{10} + u_1^{12} - u_1^{17} - u_1^{7} + u_1^{8}$$

as the $C_1(X)$ representation of a basis element of $C_2(X)$, with $X = \mathcal{A}(X_1)$. The coordinate vector of this cycle is therefore accommodated as a new signed column of the yet partially unknown sparse matrix $[\partial_2]$ of the operator $\partial_2 : C_2 \to C_1$.

Example A.2 (Chains). Unoriented chains take coefficients from $\mathbb{Z}/2\mathbb{Z} = \mathbb{Z}_2 = \{0, 1\}$. *e.g.*, a 0-chain $c \in C_0$ shown in Figure 9a is given by $c = 1v_1 + 1v_2 + 1v_3 + 1v_5$. Hence, the coefficients associated to all other cells are zero. So, $[1, 1, 1, 0, 1, 0]^t$ is the coordinate vector of c with respect to the (ordered) basis $(u_1, u_2, \ldots, u_6) = (1v_1, 1v_2, \ldots, 1v_6)$. 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 A.3 (Orientation). Figure 9b shows an oriented version of the cellular complex $\Lambda = \Lambda_0 \cup \Lambda_1 \cup \Lambda_2$, where 1-cells are oriented from the vertex with lesser index to the vertex with greater index, and where all 2-cells are counterclockwise oriented. The orientation of each cell may be fixed arbitrarily, since it 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 v_1 and last vertex v_5 is given as $d' = \eta_2 - \eta_3 + \eta_5$, with coordinate vector $[0, 1, -1, 0, 1, 0, 0, 0]^t$.

Example A.4 (Dual cochains). The concept of cochain ϕ^p in a space C^p of linear maps from chains C_p to $\mathbb{R}$ allows for the association of a scalar 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 numerical quantity, resulting from a discrete integration over a chain. Each cochain $\phi^p \in C^p$ is a linear combination of the unit p-cochains $\phi_1^p, \ldots, \phi_k^p$[22] (Figure 9). 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 "assemblies of cells", modeled by chains [52].

A.2.1 Discrete differential operators. Let us consider a space partition into cells of dimension 0 (v_i, $1 \le i \le 6$), dimension 1 (η_j, $1 \le j \le 8$), and dimension 2 (γ_k, $1 \le k \le 3$), associated with different additive groups of coefficients. In particular, in Figure 9c 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 $\delta_0 = \partial_1^\top$ and $\delta_1 = \partial_2^\top$. 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 to reals. The colored numbers on 1- and 2-cells are exactly the evaluation $\phi^k(u_k) = \langle \phi^k, u_k \rangle$ of the dual elementary cochain on each elementary chain. In Example A.5 we show the numeric values of the matrices.

[22] Coincident with $\eta_p^1, \ldots, \eta_p^k$, due to the identification of primal and dual bases.

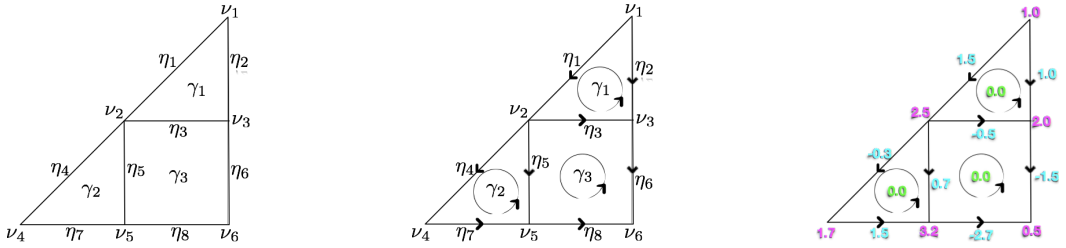


Fig. 9. Cellular complexes with 0-cells in $\Lambda_0 = \{\nu_1, \ldots, \nu_6\}$, 1-cells in $\Lambda_1 = \{\eta_1, \ldots, \eta_8\}$, and 2-cells in $\Lambda_2 = \{\gamma_1, \gamma_2, \gamma_3\}$: (a) non-oriented complex, with cell coefficients in $\mathbb{Z}_2 = \{0, 1\}$; (b) oriented complex, with 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 A.7.

Example A.5 (Boundary). The boundary operators are maps $C_p \longrightarrow C_{p-1}$, with $1 \leq p \leq d$. Hence for a 2-complex we have two operators, denoted as $\partial_2 : C_2 \longrightarrow C_1$ and $\partial_1 : C_1 \longrightarrow C_0$, respectively. Since they are linear maps between linear spaces, may be represented by matrices of coefficients $[\partial_2]$ and $[\partial_1]$ from the underlying field $\mathbb{F}$. For the unsigned and the signed case (Figures 9a and 9b) 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 ∂_1 and the signed ∂_1' operators we have:

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

As a check, let us compute (a) the 0-boundary of the coordinate representations of the unsigned 1-chains $[d] = [0, 1, 1, 0, 1, 0, 0, 0]^t$ (see Figure 9a) and (b) the signed 1-chain $[d'] = [0, 1, -1, 0, 1, 0, 0, 0]^t$ (see Figure 9b)

$$\partial_1 d = [\partial_1][d] \; mod \; 2 = [1, 0, 0, 0, 1, 0]^t = \nu_1 + \nu_5 \in C_0,$$

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

$$\partial_1' d' = [\partial_1'][d'] = [-1, 0, 0, 0, 1, 0]^t = \nu_5 - \nu_1 \in C_0'.$$

Example A.6 (Cell with a hole). Figure 10 shows an example of 2D cellular complex $X = X_2$, comprised of 8 unit 0-chains (0-cells) u_0^h, 8 unit 1-chains (1-cells) u_1^k, and 2 unit 2-chains (2-cells) u_2^j.

A user-readable representation of the geometric complex (X_2, ν) is given below. V is the array of vertices, that provides the embedding map $C_0 \longrightarrow \mathbb{E}^2$, implemented as array $\nu : \mathbb{N} \longrightarrow \mathbb{R}^2$. EV and FV respectively provide the *canonical* (sorted) LAR of 1-cells and 2-cells as lists of lists of 0-cells indices. These can be interpreted as user-readable CSR (Compressed Sparse Row) characteristic matrices M_1 and M_2 of the 0-generators of 1-cells and 2-cells, respectively, according to [36].

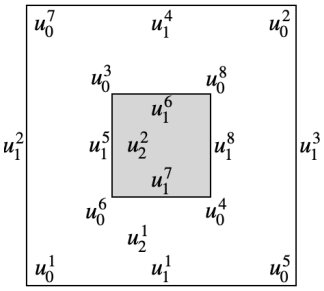


Fig. 10. Cellular 2-complex with two 2-cells, eight 1-cells, and eight 0-cells.

```
V = [[0.,0.],[3.,3.],[1.,2.],[2.,1.],[3.,0.],[1.,1.],[0.,3.],[2.,2.]]
FV = [[1,2,3,4,5,6,7,8],[3,4,6,8]]
EV = [[1,5],[1,7],[2,5],[2,7],[3,6],[3,8],[4,6],[4,8]]
```

The unsigned matrix of the boundary operator $\partial_2 : C_2 \longrightarrow C_1$, computed by filtering elements of value 2 in the matrix $M_1 M_2^t$, is:

$$[\partial_2] = \text{filter}(M_1 M_2^t, 2) = \text{filter}\left(\left(\begin{pmatrix} 1&0&0&0&1&0&0&0\\ 1&0&0&0&0&0&1&0\\ 0&1&0&0&1&0&0&0\\ 0&1&0&0&0&0&1&0\\ 0&0&1&0&0&1&0&0\\ 0&0&1&0&0&0&0&1\\ 0&0&0&1&0&1&0&0\\ 0&0&0&1&0&0&0&1 \end{pmatrix} \begin{pmatrix} 1&0\\ 1&0\\ 1&1\\ 1&1\\ 1&0\\ 1&1\\ 1&0\\ 1&1 \end{pmatrix}, 2\right) = \begin{pmatrix} 1&0\\ 1&0\\ 1&0\\ 1&0\\ 1&1\\ 1&1\\ 1&1\\ 1&1 \end{pmatrix}$$

where the first column represents the non-convex 2-cell with the hole, and the second column 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 column of $[\partial_2]$ correspond to the last four unit 1-chains in EV array. By multiplication (mod 2) of $[\partial_2]$ times the coordinate representation $[c]$ of the 2-complex in Figure 10, *i.e.*, times the *total* 2-chain $c = u_2^1 + u_2^2 = \begin{pmatrix} 1 & 1 \end{pmatrix}^t$, 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 c, *i.e.*, the cycle $u_1^1 + u_1^2 + u_1^3 + u_1^4$ made by the first four 1-cells in EV.

Example A.7 (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 $c_{p+1} \in C_{p+1}$:

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

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 9 we have $(\delta^1 \circ \delta^0)(\gamma_1) = 0$. This property, *i.e.*, $\delta \circ \delta = 0$, is the discrete archetype of the fact that 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-chains, *i.e.*, on 0-cells. It is possible to see [35] 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 9c, coefficients from $\mathbb{R}$ are associated to elementary cochains, as resulting from the evaluation of cochain functions on lower order basis chains. When cochain coefficients are taken from $G = \{-1, 0, +1\}$ we have coboundary matrices $[\delta^1] = [\partial_2']^t$ and $[\delta^0] = [\partial_1']^t$, so that, with $\phi = [0, 0, 0, 0, 1, 0, 0, 0] \in C^1 \equiv C_1$, we get $[\delta^1][\phi]^t = [0, -1, 1]^t = \gamma_3 - \gamma_2 \in C^2 \equiv C_2$, as you can check on Figure 9b.

Example A.8 (Application: geographical maps). Let us consider a geographical map as the plane arrangement $\mathcal{A}(\mathcal{S})$ generated by a quasi-disjoint set of regions (2-complex $\mathcal{R}^2$) superimposed with a road network (1-complex $\mathcal{R}^1$). To this purpose, we take as input the collection of data $\mathcal{S} = \{\mathcal{R}^2, \mathcal{R}^1\}$, and select the combinatorial union of their 1-skeletons $\mathcal{S}_1 = \mathcal{R}^2_1 \cup \mathcal{S}^1_1$. From this set of 1-cells — which is not a 1-complex, since cells may intersect away from their boundary vertices — we compute the cellular complex $X_2 = \mathcal{A}(\mathcal{S}_1)$ 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 basis 2-chains. The length of any portion of road is the real number attached by a 1-cochain to 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, the number is the sum of areas of basis 2-chains it is linear combination of.

REFERENCES

[1] Seshagiri Rao Ala. 1992. Performance Anomalies in Boundary Data Structures. *IEEE Comput. Graph. Appl.* 12, 2 (March 1992), 49–58. https://doi.org/10.1109/38.124288

[2] S. Alayrangues, G. Damiand, P. Lienhardt, and S. Peltier. 2015. Homology of Cellular Structures Allowing Multi-incidence. *Discrete & Computational Geometry* 54, 1 (01 Jul 2015), 42–77. https://doi.org/10.1007/s00454-015-9662-5

[3] 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.

[4] Douglas N. Arnold. 2018. *Finite Element Exterior Calculus.* CBMS-NSF Regional Conference Series in Applied Mathematics, Vol. 93. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

[5] Douglas N. Arnold, Richard S. Falk, and Ragnar Winther. 2006. Finite element exterior calculus, homological techniques, and applications. *Acta Numerica* 15 (2006), 1–155. https://doi.org/10.1017/S0962492906210018

[6] Douglas N. Arnold, Richard S. Falk, and Ragnar Winther. 2010. Finite element exterior calculus: from Hodge theory to numerical stability. *Bull. Amer. Math. Soc. (N.S.)* 47 (2010), 281–354.

[7] Marco Attene. 2014. Direct Repair of Self-intersecting Meshes. *Graph. Models* 76, 6 (Nov. 2014), 658–668. https://doi.org/10.1016/j.gmod.2014.09.002

[8] C. Bajaj, A. Paoluzzi, and G. Scorzelli. 2006. Progressive conversion from B-rep to BSP for streaming geometric modeling. *Computer-Aided Design and Applications* 3, 5-6 (2006).

[9] D.O. Baladze. 2012. CW-complex. In *Encyclopedia of Mathematics.* Springer & European Mathematical Society. http://www.encyclopediaofmath.org/

[10] Grey Ballard and Alex Druinsky. 2015. Sparse Matrix-Matrix Multiplication: Applications, Algorithms, and Implementations. In *SIAM Conference on Applied Linear Algebra.* Atlanta, GA.

[11] H. Barki, G. Guennebaud, and S. Foufou. 2015. Exact, robust, and efficient regularized Booleans on general 3D meshes. *Computers Mathematics With Applications* 70, 6 (2015), 1235 – 1254. https://doi.org/10.1016/j.camwa.2015.06.016

[12] Bruce G. Baumgart. 1972. *Winged edge polyhedron representation.* Technical Report Stan-CS-320. Stanford, CA, USA.

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

[14] Tim Besard. 2017. *High-Performance GPU Computing in the Julia Programming Language.* Technical Report. NVIDIA Developer Blog. https://devblogs.nvidia.com/gpu-computing-julia-programming-language/

[15] Tim Besard, Christophe Foket, and Bjorn De Sutter. 2018. Effective Extensible Programming: Unleashing Julia on GPUs. *IEEE Transactions on Parallel and Distributed Systems* (2018). https://doi.org/10.1109/TPDS.2018.2872064

[16] 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

[17] Hanspeter Bieri. 1995. Nef Polyhedra: A Brief Introduction. In *Geometric Modelling*, H. Hagen, G. Farin, and H. Noltemeier (Eds.). Springer Vienna, Vienna, 43–60.

[18] Garrett Birkhoff. 1948. *Lattice Theory (Revised ed.).* American Mathematical Society, New York, NY.

[19] A. Björner and G. Ziegler. 1992. Combinatorial stratification of complex arrangements. *J. Amer. Math. Soc.* 5 (1992), 105–149.

[20] A. Bowyer. 1995. *SvLis Set-theoretic Kernel Modeller: Introduction and User Manual.* Information Geometers. http://books.google.it/books?id=hYqwAAAACAAJ

[21] A. Bowyer and Geometric Modelling Society. 1995. *Introducing Djinn: A Geometric Interface for Solid Modelling.* Information Geometers [for] the Geometric Modelling Society. http://books.google.it/books?id=oCGGAAAACAAJ

[22] Dan Van Boxel. 2016. *Deep Learning with TensorFlow.* Packt Publishing.

[23] I. C. Braid. 1975. The synthesis of solids bounded by many faces. *Commun. ACM* 18, 4 (April 1975), 209–216. https://doi.org/10.1145/360715.360727

[24] E. Brisson. 1989. Representing geometric structures in *d* dimensions: topology and order. In *Proc. of the 5-th Annual Symposium on Computational Geometry (SCG '89).* Acm, New York, NY, USA, 218–227. https://doi.org/10.1145/73833.73858

[25] 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

[26] M. Campen and L. Kobbelt. 2010. Exact and robust (self-)intersections for polygonal meshes. *Computer Graphics Forum* 29 (2010), 397–406.

[27] 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.

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

[29] H. S. M. Coxeter and S. L. Greitzer. 1967. *Geometry Revisited.* Math. Assoc. Amer., Washington, D.C.

[30] Guillaume Damiand and Pascal Lienhardt. 2014. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing.* CRC Press.

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

[32] C. Dehlinger and J.F. Dufourd. 2014. Formal specification and proofs for the topology and classification of combinatorial surfaces. *Computational Geometry* 47, 9 (2014), 869 – 890. https://doi.org/10.1016/j.comgeo.2014.04.007

[33] 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

[34] A. DiCarlo, F. Milicchio, A. Paoluzzi, and V. 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

[35] 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 arXiv:http://doi.acm.org/10.1145/1629255.1629273

[36] 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

[37] D. P. Dobkin and M. J. 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

[38] C.J.A. Delfinado End H. Edelsbrunner. 1995. An incremental algorithm for Betti numbers of sinplicial complexes on the 3-sphere. *Computer Aided Geometric Design* 12 (1995), 771–784.

[39] H. Edelsbrunner. 1987. *Algorithms in Combinatorial Geometry.* Springer-Verlag New York, Inc., New York, NY, USA.

[40] 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

[41] 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

[42] E. Ferretti. 2014. *The Cell Method: A Purely Algebraic Computational Method in Physics and Engineering.* Momentum Press.

[43] 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.

[44] 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

[45] Abel Gomes, Alan Middleditch, and Chris Reade. 1999. A mathematical model for boundary representations of n-dimensional geometric objects. In *Proceedings of the fifth ACM symposium on Solid modeling and applications (SMA '99)*. ACM, New York, NY, USA, 270–277. https://doi.org/10.1145/304012.304039
[46] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. The MIT Press.
[47] 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.
[48] Leonidas Guibas and Jorge Stolfi. 1985. Primitives for the manipulation of general subdivisions and the computation of Voronoi. *ACM Trans. Graph.* 4, 2 (April 1985), 74–123. https://doi.org/10.1145/282918.282923
[49] Leonidas J. Guibas and David H. Marimont. 1998. Rounding Arrangements Dynamically. *International Journal of Computational Geometry Applications* 8, 2 (1998), 157–178. https://doi.org/10.1142/S0218195998000096
[50] 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
[51] Allen Hatcher. 2002. *Algebraic topology*. Cambridge University Press.
[52] René Heinzl. 2007. *Concepts for Scientific Computing*. Ph.D. Dissertation. Wien, Österreich.
[53] Anil Nirmal Hirani. 2003. *Discrete Exterior Calculus*. Ph.D. Dissertation. Pasadena, CA, USA. Advisor(s) Marsden, Jerrold E. AAI3086864.
[54] Christoph M. Hoffmann. 1989. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
[55] Christoph M. Hoffmann, John E. Hopcroft, and Michael S. Karasick. 1987. *Robust Set Operations on Polyhedral Solids*. Technical Report. Ithaca, NY, USA.
[56] Christoph M. Hoffmann and Ku-Jin Kim. 2001. Towards valid parametric CAD models. *Computer-Aided Design* 33, 1 (2001), 81–90. https://doi.org/10.1016/S0010-4485(00)00073-7
[57] Christoph M. Hoffmann and George Vaněcek. 1991. *Fundamental Techniques for Geometric and Solid Modeling*. Technical Report Report Number: 91-044. Purdue University.
[58] 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
[59] R. A. Jarvis. 1973. On the Identification of the Convex Hull of a Finite Set of Points in the Plane. *Inform. Process. Lett.* 2, 1 (March 1973), 18–21.
[60] Y. E. Kalay. 1989. The hybrid edge: a topological data structure for vertically integrated geometric modelling. *Comput. Aided Des.* 21, 3 (April 1989), 130–140. https://doi.org/10.1016/0010-4485(89)90067-5
[61] Ravindran Kannan and Achim Bachem. 1979. Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix. *SIAM J. Comput.* 8 (1979), 1979. https://doi.org/10.1137/0208040
[62] Sang Hun Lee and Kunwoo Lee. 2001. Partial entity structure: a compact non-manifold boundary representation based on partial topological entities. In *Proceedings of the sixth ACM symposium on Solid modeling and applications (SMA '01)*. Acm, New York, NY, USA, 159–170. https://doi.org/10.1145/376957.376976
[63] Pascal Lienhardt. 1991. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Comput. Aided Des.* 23, 1 (Feb. 1991), 59–82. https://doi.org/10.1016/0010-4485(91)90082-8
[64] M. Mantyla. 1988. *Introduction to Solid Modeling*. W. H. Freeman & Co., New York, NY, USA.
[65] E. Marino, F. Spini, D. Salvati, C. Vadalà, M. Vicentino, A. Paoluzzi, and A. Bottaro. 2017. Modeling Semantics for Building Deconstruction. In *Proc., 12wt International Conference on Computer Graphics Theory and Applications*.
[66] Michael McKenna. 1987. Worst-case Optimal Hidden-surface Removal. *ACM Trans. Graph.* 6, 1 (Jan. 1987), 19–28. https://doi.org/10.1145/27625.27627
[67] A.E. Middleditch. 1992. *Cellular models of mixed dimension*. Technical Report BRU/CAE/92:3. Brunel University Centre for Geometric Modelling and Design.
[68] Young Lae Moon, Kazuomi Sugamoto, Alberto Paoluzzi, Antonio Di Carlo, Jaekeun Kwak, Dong Sun Shin, Dae Ok Kim, Dae Hyun Lee, and Jooyoung Kim. 2014. Standardizing 3D Medical Imaging. *Computer* 47, 4 (2014), 76–79. https://doi.org/10.1109/MC.2014.103
[69] D. E. Muller and F. P. Preparata. 1978. Finding the Intersection of Two Convex Polyhedra. *Theoretical Computer Science* 7 (1978), 217–236.
[70] J.R. Munkres. 1984. *Elements of Algebraic Topology*. Addison-Wesley, Reading, MA.
[71] Richard S. Palmer. 1995. Chain models and finite element analysis: An executable Chains formulation of plane stress. *Computer Aided Geometric Design* 12, 7 (1995), 733 – 770. https://doi.org/10.1016/0167-8396(95)00015-X Grid Generation, Finite Elements, and Geometric Design.
[72] Richard S. Palmer and Vadim Shapiro. 1993. Chain models of physical behavior for engineering analysis and design. *Research in Engineering Design* 5, 3 (01 Sep 1993), 161–184. https://doi.org/10.1007/BF01608361

[73] A. Paoluzzi. 2003. *Geometric Programming for Computer Aided Design.* John Wiley & Sons, Chichester, UK. https://doi.org/10.1002/0470013885

[74] A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. 1993. Dimension-independent modeling with simplicial complexes. *ACM Trans. Graph.* 12, 1 (Jan. 1993), 56–102. https://doi.org/10.1145/169728.169719

[75] Alberto Paoluzzi, Antonio DiCarlo, Francesco Furiani, and Miroslav JiuĬ^riĬk. 2016. CAD Models from Medical Images Using LAR. *Computer-Aided Design and Applications* (2016). https://doi.org/10.1080/16864360.2016.1168216

[76] Alberto Paoluzzi, Enrico Marino, and Federico Spini. 2015. LAR-ABC, a Representation of Architectural Geometry from Concept of Spaces, to Design of Building Fabric, to Construction Simulation. In *Advances in Architectural Geometry 2014*, Philippe Block, Jan Knippers, Niloy J. Mitra, and Wenping Wang (Eds.). Springer International Publishing, 353–372. https://doi.org/10.1007/978-3-319-11418-7_23

[77] A. Paoluzzi, V. Pascucci, and G. Scorzelli. 2004. Progressive Dimension-independent Boolean Operations. In *Proceedings of the Ninth ACM Symposium on Solid Modeling and Applications (SM '04)*. Eurographics Association, Aire-La-Ville, Switzerland, Switzerland, 203–211. http://dl.acm.org/citation.cfm?id=1217875.1217907

[78] Alberto Paoluzzi, Valerio Pascucci, and Michele Vicentino. 1995. Geometric programming: a programming approach to geometric design. *ACM Trans. Graph.* 14, 3 (July 1995), 266–306. https://doi.org/10.1145/212332.212349

[79] A. Paoluzzi, M. Ramella, and A Santarelli. 1989. Boolean algebra over linear polyhedra. *Comput. Aided Des.* 21, 10 (Oct. 1989), 474–484. http://dl.acm.org/citation.cfm?id=70248.70249

[80] Valerio Pascucci, Vincenzo Ferrucci, and Alberto Paoluzzi. 1995. Dimension-independent convex-cell based HPC: representation scheme and implementation issues. In *Proceedings of the third ACM Symposium on Solid Modeling and Applications (SMA '95)*. Acm, New York, NY, USA, 163–174. https://doi.org/10.1145/218013.218055

[81] Michael J. Pratt and Bill D. Anderson. 1994. A Shape Modelling API for the STEP Standard. In *in Fourteenth International Conference on Atomic Physics*. 1–7.

[82] Srinivas Raghothama and Vadim Shapiro. 1999. Consistent updates in dual representation systems. In *Proceedings of the fifth ACM symposium on Solid modeling and applications (SMA '99)*. Acm, New York, NY, USA, 65–75. https://doi.org/10.1145/304012.304019

[83] Ari Rappoport. 1997. The generic geometric complex (GGC): a modeling scheme for families of decomposed pointsets. In *Proceedings of the fourth ACM symposium on Solid modeling and applications (SMA '97)*. Acm, New York, NY, USA, 19–30. https://doi.org/10.1145/267734.267749

[84] A. A. G. Requicha and H. B. Voelcker. 1977. *Constructive solid geometry.* Technical Report TM-25. Production Automation Project, Univ. of Rochester.

[85] 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

[86] J. R. Rossignac and M. A. O'Connor. 1990. SGC: A Dimension-independent Model for Pointsets with Internal Structures and Incomplete Boundaries. In *Geometric modeling for product engineering*. North-Holland.

[87] 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.

[88] Jarek R. Rossignac and Aristides A. G. Requicha. 1991. Constructive non-regularized geometry. *Comput. Aided Des.* 23, 1 (Feb. 1991), 21–32. https://doi.org/10.1016/0010-4485(91)90078-B

[89] Colin Rourke and Brian Sanderson. 1982. *Introduction to Piecewise-Linear Topology.* Springer-Verlag, Berlin, Heidelberg. http://doi.org/10.1007/978-3-642-81735-9

[90] G. Scorzelli, A. Paoluzzi, and V. Pascucci. 2008. Parallel solid modeling using BSP dataflow. *International Journal of Computational Geometry and Applications* 18, 5 (October 2008), 441–467.

[91] Vadim Shapiro. 1991. *Representations of semi-algebraic sets in finite algebras generated by space decompositions.* Ph.D. Dissertation. Ithaca, NY, USA. UMI Order No. GAX91-31407.

[92] Vadim Shapiro and Donald L. Vossler. 1995. What is a parametric family of solids?. In *Proceedings of the third ACM symposium on Solid modeling and applications (SMA '95)*. ACM, 43–54. https://doi.org/10.1145/218013.218029

[93] Johnathan Richard Shewchuk. 1996. Robust Adaptive Floating-point Geometric Predicates. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry (SCG '96)*. Acm, New York, NY, USA, 141–150. https://doi.org/10.1145/237218.237337

[94] Jonathan Richard Shewchuk. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, Ming C. Lin and Dinesh Manocha (Eds.). Lecture Notes in Computer Science, Vol. 1148. Springer-Verlag, 203–222.

[95] Jonathan Richard Shewchuk. 2002. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry* 22, 1 (2002), 21 – 74. https://doi.org/10.1016/S0925-7721(01)00047-5

[96] C. E. Silva. 1981. *Alternative definitions of faces in boundary representations of solid objects.* Technical Report TM-36. Production Automation Project, Univ. of Rochester.

[97] Federico Spini, Enrico Marino, Michele D'Antimi, Edoardo Carra, and Alberto Paoluzzi. 2016. Web 3D Indoor Authoring and VR Exploration via Texture Baking Service. In *21st annual Web3D 2016 Conference*. Anaheim, CA.
[98] Federico Spini, Marco Sportillo, Marco Virgadamo, Enrico Marino, Antonio Bottaro, and Alberto Paoluzzi. 2015. HIJSON: Cartographic Document for Web Modeling of Interactive Indoor Mapping. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, Andrea Giachetti, Silvia Biasotti, and Marco Tarini (Eds.). The Eurographics Association. https://doi.org/10.2312/stag.20151290
[99] Enzo Tonti. 1975. *On the formal structure of physical theories*. Technical Report. Italian National Research Council.
[100] Enzo Tonti. 2013. *The Mathematical Structure of Classical and Relativistic Physics*. Birkhäuser.
[101] T. Vialar. 2016. *Handbook of Mathematics*. HDBoM. https://books.google.com/books?id=-tcVMQAACAAJ (Def 805).
[102] K. Weiler. 1985. Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments. *Computer Graphics and Applications, IEEE* 5, 1 (jan. 1985), 21 –40. https://doi.org/10.1109/MCG.1985.276271
[103] K. J. Weiler. 1986. *Topological structures for geometric modeling*. Ph.D. Dissertation. Rensselaer Polytechnic Institute.
[104] K. J. Weiler. 1988. The radial edge structure: A topological representation for non-manifold geometric modelling. In *Geometric modelling for CAD applications*, M. Wozny, H. McLaughlin, and J. Encarnacao (Eds.). Amsterdam, 3–12.
[105] T. Woo. 1985. A Combinatorial Analysis of Boundary Data Structure Schemata. *Computer Graphics & Applications, IEEE* 5, 3 (March 1985), 19–27.
[106] M.J. Wozny, J.U. Turner, and K. Preiss. 1990. *Geometric modeling for product engineering: IFIP WG 5.2/NSF Working Conf. on Geometric Modeling, Rensselaerville, U.S.A., 18-22 September, 1988*. North-Holland. http://books.google.it/books?id=-BkfAQAAIAAJ
[107] F. Yamaguchi and T. Tokieda. 1985. Bridge Edge and Triangulation Approach in Solid Modeling. In *Frontiers in Computer Graphics*, T.L. Kunii (Ed.). Springer Verlag, Berlin.
[108] Y. Yamaguchi and F. Kimura. 1995. Nonmanifold topology based on coupling entities. *Computer Graphics and Applications, IEEE* 15, 1 (1995), 42–50.
[109] Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh Arrangements for Solid Geometry. *ACM Trans. Graph.* 35, 4, Article 39 (July 2016), 15 pages. https://doi.org/10.1145/2897824.2925901