Geometric Computing with Chain Complexes
Design and Features of a Julia Package

Francesco Furiani[1], Giulio Martella[2] and Alberto Paoluzzi[1]
[1]*Department of Mathematics and Physics, Roma Tre University, Rome, Italy*
[2]*Department of Engineering, Roma Tre University, Rome, Italy*
{*f.furio@gmail.com, giuliomartella1995@gmail.com, alberto.paoluzzi@uniroma3.it*

Keywords: Geometric Computing, Computational Topology, Solid Modeling, Cellular Complex, Chain Complex, LAR.

Abstract: Geometric computing with chain complexes allows for the computation of the whole chain of linear spaces and (co)boundary operators generated by a space decomposition into a cell complex. The space decomposition is stored and handled with LAR (Linear Algebraic Representation), i.e. with sparse integer matrices, and allows for using cells of a very general type, even non convex and with internal holes. In this paper we discuss the features and the merits of this approach, and describe the goals and the implementation of a software package aiming at providing for simple and efficient computational support of geometric computing with any kind of meshes, using linear algebra tools with sparse matrices. The library is being written in Julia, the novel efficient and parallel language for scientific computing. This software, that is being ported on hybrid architectures (CPU+GPU) of last generation, is yet under development.

1 INTRODUCTION

1.1 What, When, Why

In the four decades-long history of solid modeling, providing computer representations for mathematical models of manufactured objects, the standard characterization of a solid normally required some specialized data structure, called a B-rep, to represent a cell decomposition of its boundary. When non-manifold solids are considered, in order to make their operations closed with respect to object configurations with touching boundaries, the computer representation becomes much more convoluted, as having to store and maintain both the incidence relations between boundary cells, and the cyclic orderings between incidence pairs.

In this paper we discuss a largely different approach. Our computer model of a solid (a) may contain a decomposition of either the boundary or the interior of the solid; (b) the type of cells of the decomposition (i.e. their shape or topology) is not fixed a priori; so allowing for (c) a very general covering of most application domains that require a computer model of the geometric shape and physical behaviour of the object. This approach encompasses (i) computer graphics and rendering, (ii) solid modeling and its common operations, (iii) mesh decompositions and physical simulations, as well as (iv) 3D imaging for medical and other applications. Even more, the computer representation itself does not need some weird data structures, typical of non-manifold descriptions, but only requires few very sparse binary matrices (either one or two, depending on the topology of cells), used to codify the cells as subsets of vertices, i.e., to store the subset of vertices on the boundary of each cell.

The software package discussed in this paper finds conceptual roots in the reduction of physical world behaviour to geometric shape and its properties—consider, e.g., the gravitation force due to the curvature of the space-time manifold—and the reduction of differential geometry and physical laws to topological facts. It is well-known that a discrete topological framework may support all/most physical simulations (Tonti, 1976; Tonti, 2013; Tonti, 2014), simply by leveraging the coboundary operator for all conservation / balance / compatibility / equilibrium type laws, and linear (metric) operators for constitutive / measured laws (Shapiro, 2017).

The development of a prototype literate software library supporting LAR (the Linear Algebraic Representation) (DiCarlo et al., 2014a) started in Python in 2012, and was interrupted in 2016 by A.P.'s lack of time. Few months later we started a new development project in Julia, the novel efficient computer language

for scientific computing, with F.F. and G.M. as main developers. The package core, concerning the merging of cellular complexes, will be discussed here and is close to conclusion. The porting of other parts of the package from Python to Julia is planned for the next months.

1.2 Chain-based computing

Chain-based modeling and computing is based on representation of d-cell subsets as *chains*, elements of linear spaces C_d. Chains can be simply represented as arrays of signed integers, one of simplest and more efficient data structure of most languages, particularly when oriented to scientific computing. Therefore, basic algebraic operations on chains as vectors (sum and product times a scalar) are implemented over arrays.

Often we must deal with oriented cells, again naturally represented as arrays of signed indices. This representation applies to very general kinds of connected cells: p-simplicies, p-cubes, polytopal, non-convex, even punctured, i.e. homeomorphic to cells with internal holes. Given an ordering over every set of p-cells, (linear) topological operators, like boundary and coboundary (see Section 2), and hence discrete gradients, curls and divergence, may be represented as either integer or real matrices, depending on the type of represented fields.

Incidence operators between chain spaces of different dimension are easy to compute by matrix products of characteristic matrices (see Section 2.1), possibly transposed. Data validity is easy to test by checking for satisfaction of basic equations $\partial\partial = \emptyset$ of a chain complex. Since both characteristic and operator matrices are very sparse, their products are computed with specialized algorithms for very sparse matrices, whose complexity is roughly linear in the size of the output sparse matrix, i.e., in the number of its stored non-zero elements.

Of course, several software packages are available for efficient linear algebra with sparse matrices defined over big geometric data sets, and most of such softwares are already ported to last-generation mixed HPC architectures (CPU + GPU).

Because of the above points (sparse-array- and GPU-based), the chain space approach seems to fit well with Neural Networks architectures, where the typical data type for input/output from neurons is either a dense or a sparse array. Novel operators (both linear and non-linear) are being investigated: for example, the topological traversal of a set of boundary cycles, returning the chain of interior cells. This last operator is a sort of pseudoinverse of the boundary operator, that goes from chains to boundary cycles.

Incidence queries and other types of geometric or topological computations are not performed element-wise, that necessarily require iterative or recursive programming patterns, but by matrix product on whole chains (sets of cells), so adapting naturally to parallel and/or dataflow computational patterns found in HPC and CNN architectures. Last but not least, our chain-based representation crosses the boundaries of all traditional geometric disciplines. It would be easy to show that chains of cells may represent *any subset* of 2D/3D images, as well as both boundary and decompositive schemes for solid modeling, as well as data structures used for domain meshing and physical simulation, as wells as the data structures used for geographical systems and outdoor mapping, and so on.

1.3 Previous Work

An up-to-date review of the mapping schemes used to provide computer representations of manufactured objects can be found in the dedicated survey chapter on Solid Modeling, by Rossignac and Shapiro, in the Handbook of Discrete and Computational Geometry (Goodman et al., 2017). The challenges introduced by the novel frontiers of computational modeling of material structures are discussed in (Regli et al., 2016). A description of the computational geometry CGAL software (Fabri et al., 2000), implementing 2D/3D arrangements with Nef polyhedra (Hachenberger et al., 2007) and the construction of arrangements of lines, segments, planes and other geometrical objects can be found in (Fogel et al., 2007). Discrete Exterior Calculus (DEC) with simplicial complexes was introduced by (Hirani, 2003) and made popular by (Desbrun et al., 2006) and (Elcott and Schroder, 2006). More recently, a systematic recipe has been proposed in (Zhou et al., 2016) for constructing a family of exact constructive solid geometry operations starting from a collection of triangle meshes. Conversely, the dimension-independent algorithms implemented in `Larlib` and discussed here may apply to any collection of both boundary and decompositive representations, including 2/D/3D computer images.

1.4 Paper Contents

In Section 2 we shortly recall the main concepts regarding LAR (Linear Algebraic Representation) as a general representation scheme for geometric and topological modeling, and the main features of Julia language, including its native support for parallel processing. In Section 3 after describing a partial list of the main subpackages of the Python prototype li-

brary, we discuss the design goals of the new Julia implementation, and present the *Merge* algorithm. In Section 4 we show by examples the meaning of both the characteristic matrix of a basis of d-cells, and the (co)boundary matrix. In the Conclusion section 5 the next development steps are outlined, and our view of the evolution of this software is provided.

2 BACKGROUND

A large amount of research was recently invested into the rewriting of standard algorithms over very-large graphs (i.e. cellular 1-complexes, made by 0- and 1-cells) using linear algebra methods based on sparse matrices (Buono et al., 2016; Buluç and Gilbert, 2011; Kepner and Gilbert, 2011; Davis, 2006). Our approach allows to work with linear algebra and sparse matrices over cellular d-complexes.

2.1 Characteristic matrices and (co)boundary operators

Let X be a topological space, and $\Lambda(X) = \bigcup_p \Lambda_p$ ($p \in 0,1,\ldots,d$) be a cellular decomposition of X, with Λ_p a set of closed and connected p-cells. A *CW-structure* on the space X is a filtration $\emptyset = X_{-1} \subset X_0 \subset X_1 \subset \ldots \subset X = \bigcup_p X_p$, 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 $\Lambda_p(X)$. A *cellular complex* is a space X provided with a CW-structure.

With abuse of language, we consider a finite cellular complex X as generated by a discrete partition of an Euclidean space. In computing a cellular complex as the space *arrangement* of a collection of geometric objects $\mathcal{S}$, i.e. when $X := \mathcal{A}(\mathcal{S})$, we actually compute the whole *chain complex* $C_\bullet$ generated by X, i.e.:

$$C_3 \underset{\partial_3}{\overset{\delta_2}{\longleftrightarrow}} C_2 \underset{\partial_2}{\overset{\delta_1}{\longleftrightarrow}} C_1 \underset{\partial_1}{\overset{\delta_0}{\longleftrightarrow}} C_0,$$

where C_p ($0 \geq p \geq 3$) is a linear space of *p-chains* (subsets of p-cells with algebraic structure), the linear operators ∂_p and δ_p are called *boundary* and *coboundary*, respectively, with

$$\partial_{p-1} \circ \partial_p = \emptyset = \delta_p \circ \delta_{p-1},$$

and where $\delta_{p-1} = \partial_p^\top$, $(1 \leq p \leq 3)$. For a discussion about the identification, performed here, between chains and cochains, see (Paoluzzi et al., 2017).

A *characteristic function* $\chi_A : S \to \{0,1\}$ is a function defined on a finite set $S = \{s_j\}$, that indicates membership of an element s_j in a subset $A \subseteq S$, having the value 1 for all elements of A and the value 0 for all elements of S not in A. We call *characteristic matrix* M of a collection of subsets $A_i \subseteq S$ ($i = 1,\ldots,n$) the binary matrix $M = (m_{ij})$, with $m_{ij} = \chi_{A_i}(s_j)$, that provides a basis for a linear space C_p of *p-chains*.

2.2 Sparse matrices as representations

The LAR *representation scheme* (Requicha, 1980), i.e. our mapping between mathematical models of a solid and their computer representation, uses linear *chain spaces* C_p as models, and sparse *characteristic matrices* M_p of p-cells as symbolic representations, where the p-cell $\sigma^k \in \Lambda_p$ is represented as the k-th binary row of the sparse matrix $M_p : C_0 \to C_p$. Since M_p matrices are *very* sparse, we can compactly represent the basis of a C_p space, when the ordering of 0-cells (vertices) and p-cells have been fixed, as an *array*, indexed by cell indices, *of arrays* of indices of vertices. Vertex positions are represented by a 2-array of d real coordinates. The reader may find in the Appendix A a LAR representation of both the input and the output cellular complex of Figure 1.

2.3 Features of LAR (Linear Algebraic Representation)

In development since several years, in a joint collaboration between Roma Tre University and the University of Wisconsin at Madison, the chain-based modeling approach (DiCarlo et al., 2009b; DiCarlo et al., 2009a) provides the first complete representation of solid models making advantage of linear topological algebra, using only sparse matrices.

The LAR scheme (DiCarlo et al., 2014b) provides the capability of using few pivotal algorithms—basically sparse matrix-vector multiplication (Buluç and Gilbert, 2012)—to compute and analyse the space arrangements and chain complexes generated by cellular complexes. In particular, this approach produces linear operators to compute incidences between cells of any p dimension ($0 \leq p \leq d$), allowing for fast algebraic computation of boundary, coboundary and mutual incidence of any *subsets of cells*, called *chains*, to consider as elements of linear spaces of chains (Hatcher, 2002).

Also, LAR can be used for both boundary representation and cellular decomposition of objects, so unifying the management of engineering meshes, 2D/3D images, solid and graphical models in both 2D and 3D, and even in higher dimensions. In particular, this scheme can be used for simplicial, cubical and polytopal (e.g. Voronoi) meshes, and for cellular decompositions with any shape of cells, also including holes of any dimension. In a fair comparison with

B-rep solid models (Woo, 1985; Baumgart, 1972), LAR needs (before the possible bitwise compression) a space proportional to the double of the number of boundary edges. Similar improvements hold for decompositive modeling schemes.

2.4 Scientific Computing with Julia Language

Julia (Bezanson et al., 2017) is a novel language for scientific computing, aiming at producing high performance code based on readable and compact sources, in a programming style similar to Python or Matlab. Julia also provides an excellent support for parallel and distributed computing environments using both low-level and high-level accelerated libraries. The `Larlib.jl` package aims to become the Julia's library for dealing with big geometric environments and models, and with meshes of any type.

3 THE LARLIB PACKAGE

This current project aims are (a) to provide a prototype parallel implementation of a new technique to compute the d-cells of the unknown space arrangement generated by $(d$-1)-dimensional geometric structures (Paoluzzi et al., 2017), and (b) to apply this new technology and software to the multilevel extraction of structures and models emerging from high-resolution biomedical imaging (Paoluzzi et al., 2016), in the framework of next-generation neural networks and image analysis research.

The breadth of the LAR scheme (DiCarlo et al., 2014b), planned by design for replacing with sparse matrices and linear algebra the intricate data structures and algorithms used for non-manifold solid modeling, has a broad covering range. The application opportunities opened by this approach seem pretty large, spanning from product lifecycle management to biochemical and biomedical applications, and from new engines for 3D printing and modeling from images, to scene understanding, and beyond.

3.1 Python Prototype

A previous Python prototype was written using literate programming methods (Knuth, 1984), to make the code more easily accessible to other developers. In particular, we used Python 2.7, Latex and Nuweb. The software package and its documentation are accessible at `https://github.com/cvdlab/lar-cc`.

The `Larlib` package, requires `Scipy`, `Pyplasm`, `PyOpenGL`, and includes several subpackages, for support of dimension-independent simplicial modeling, integration of polynomials over triangulated domains, basic operations on cellular complexes, the generation of simplicial or cuboidal grids, the interfacing of PLaSM language primitives (Paoluzzi et al., 2001), the computation of topological operators, the management of assemblies of solid objects, affine operations overs solids and assemblies, the experimental development of 2D/3D Boolean operations, the decoration of cells of a complexes with their indices (for testing), the graphics rendering of exploded complexes, the mapping of polynomial and rational splines over decomposed grid domains, and experimental applications for prototype architectural modeling, IOT modeling, and other applications and interfaces.

3.2 LarLib.js Design Goals

We posit that basic data and algorithms used by `Larlib.jl` may find a proper fitting among the common architectures and representations of convolutional neural networks, based on sparse matrices and linear algebra, in order to properly combine image understanding and geometric modeling.

In particular, our approach grounded on LAR to discover the d-cells[1] of an unknown space arrangement (Paoluzzi et al., 2017), seems to match well with deep NNs (Goodfellow et al., 2016; Boxel, 2016). In fact, the computation may proceed by discovering in parallel the 0-cells, then by locally detecting the incident edges, then by computing the equivalence relations and quotient sets of 1-cells, then by proceeding to local (planar) reconstruction of 2-cells, and hence again to identification of quotient sets of 2-cells in 3D, and finally to local (parallel) reconstruction of 3-cells, and so on, even for higher dimensions.

When applied to very-high resolution imagery of biological structures (e.g. neuronal images) or to next-generation medical 3D imaging, this approach may produce 3D models of micro-level structures, say capillary veins and small nerve structures, locally reconstructing their dendritic meso-structures, including possible inclusions of other components, and then proceed bottom-up to hierarchically assemble more and more complex macro-structures.

3.3 Implementation of the Merge Algorithm

The computation of the arrangement of $\mathbb{E}^d$ generated by a collection of $(d-1)$-complexes, is called

[1] Of very general type, even non convex and with any number of internal holes.

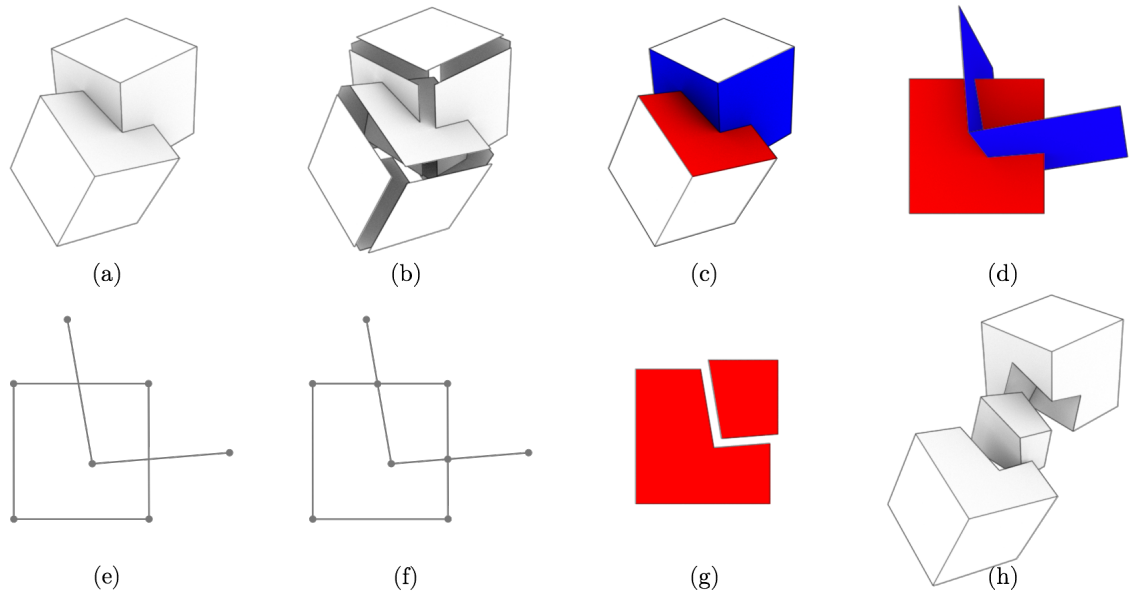


Figure 1: Cartoon display of the Merge algorithm: (a) the two input solids; (b) the exploded input collection B of 2-cells embedded in $\mathbb{E}^3$; (c) 2-cell σ (red) and the set $\Sigma(\sigma)$ (blue) of possible intersection; (d) affine map of $\sigma \cup \Sigma$ on $z = 0$ plane; (e) reduction to a set of 1D segments in $\mathbb{E}^2$; (f) pairwise intersections; (g) regularized plane arrangement $\mathcal{A}(\sigma \cup \Sigma)$; (h) exploded 3-complex extracted from the 2-complex $X_2(\cup_{\sigma \in B} \mathcal{A}(\sigma \cup \Sigma))$ in $\mathbb{E}^3$. The LAR representation of both the input data (not a complex) and the output data (3-complex) is given in Appendix A

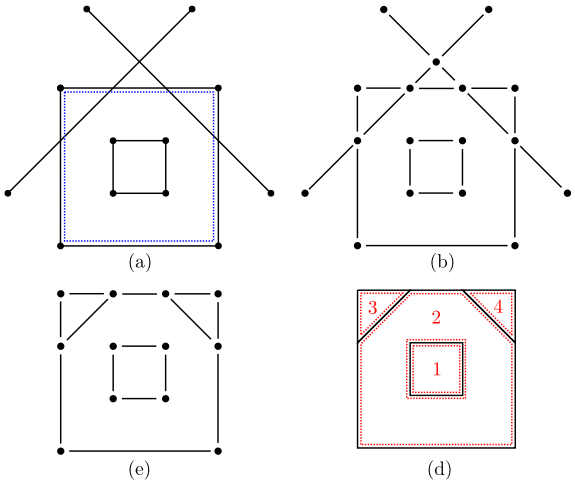


Figure 2: Basic case: computation of the regularized arrangement of a set of lines in $\mathbb{E}^2$: (a) the input, i.e. the 2-cell σ (blue) and the line segment intersections of $\Sigma(\sigma)$ with $z = 0$; (b) all pairwise intersections; (c) removal of the 1-subcomplex external to σ; (d) interior cells of the regularized 2-complex $X_2 = \mathcal{A}(\sigma \cup \Sigma)$ generated as arrangement of $\mathbb{E}^2$ produced by $\sigma \cup \Sigma$.

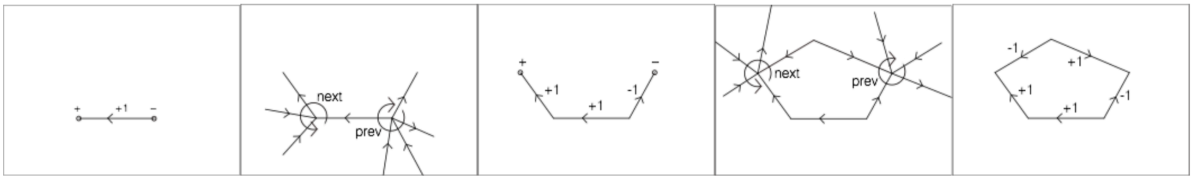
Merge of complexes. We give here a simplified outline of it, since this algorithm is the actual core of the `Larlib.jl` implementation. For all the details, the reader is referred to (Paoluzzi et al., 2017) paper. Using dimension-independent language, we say that the input is a collection of cellular complexes of dimension $(d-1)$ embedded in the Euclidean space $\mathbb{E}^d$, while the output is the arrangement of $\mathbb{E}^d$ generated by them, represented as a single cellular complex of dimension d.

Assemble the input complexes. First we assemble the n input $(d-1)$-complexes in a single LAR representation B. This one is not a cellular complex, since cells may intersect outside of their boundaries. The purpose of the first part of the algorithm is to fragment the cells properly, so that they become a single $(d-1)$-complex embedded in $\mathbb{E}^d$.

Compute a spatial index. Then we efficiently compute a spatial index that associate to each $(d-1)$-cell σ in the input, the set $\Sigma(\sigma)$ of $(d-1)$-cells with possible intersection with it. For this purpose we compute d interval trees, allowing for fast computation of intersecting bounding boxes.

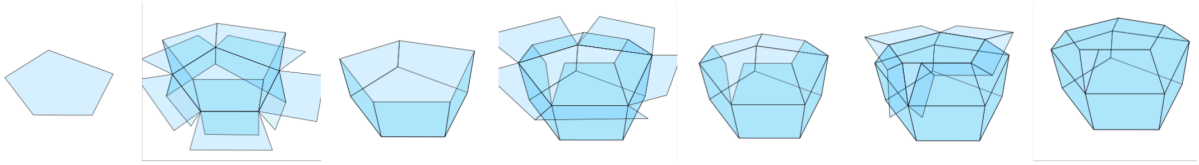
Compute the facet arrangments in $\mathbb{E}^{d-1}$. For each set $\Sigma(\sigma)$ of $(d-1)$-cells, embedded in $\mathbb{E}^d$, we compute the planar arrangement generated by them in the affine hull of σ. This arrangment is a $(d-1)$-complex having σ as support space, i.e. properly decomposing it in a cellular complex.

Assemble the output $(d-1)$-skeleton. All planar arrangements $X_{d-1} = \mathcal{A}(\sigma)$, for $\sigma \in B$,



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

Figure 3: Dimension-independent topological gift-wrapping: 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 $\partial c = \emptyset$. (Image from (Paoluzzi et al., 2017))



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

Figure 4: Dimension-independent topological gift-wrapping: extraction of a minimal 2-cycle from $\mathcal{A}(X_2)$: (a) 0-th value for $c \in C_2$; (b) cyclic subgroups on $\delta\partial c$; (c) 1-st value of c; (d) cyclic subgroups on $\delta\partial c$; (e) 2-nd value of c; (f) cyclic subgroups on $\delta\partial c$; (g) 3-rd value of c, such that $\partial c = 0$, hence stop. (Image from (Paoluzzi et al., 2017))

are syntactically merged into a single proper $(d-1)$-complex, providing the $(d-1)$-skeleton of the output complex. This merging of p-cells $(0 \leq p \leq d-2)$ is performed via identification of quotient subcomplexes with a single instance of the common cells. The equivalence relation is computed looking for cells sharing the same portion of space, starting from identification of vertices with the same coordinates (or very close, due to numerical round-offs). N-d-trees of vertices are used for this purpose, and cells are written in canonical form (sorted arrays of unsigned vertex indices) to allow for syntactical identification.

Extract d-cells from $(d-1)$-skeleton. Finally, the unknown d-cells of the output d-complex $X_d = \mathcal{A}(X_{d-1})$ are extracted. First note that a cycle is a chain with empty boundary. Each σ_d is a minimal $(d-1)$-cycle, extracted from X_{d-1} using the *Topological Gift Wrapping* algorithm displayed in Figure 4. Each basis element of the chain space C_d, i.e., every d-cell, represented as a $d-1$-cycle (see Section 4.1), is stored by columns in the matrix $[\partial_d]$ of the linear operator $\partial_d : C_d \to C_{d-1}$.

Basis d-cells bounded by minimal $(d$-1$)$-cycles. The topological property used to compute the basis of d-cells is the fact that each $(d-1)$-cell is shared by at most two d-cells. The property holds exactly if considering also the exterior unbounded cell, that must later discovered, with its column removed from the boundary matrix, since it is a linear combination of the other columns.

Topological gift wrapping. Reminds the gift wrapping algorithm (Paoluzzi et al., 2017), but is more general, applying also to non-convex cells. It is implemented by iteration of application to a singleton chain of the operator $\delta_{d-2} \circ \partial_{d-1}$, as shown in Figure 4.

Cyclic subgroups of the $\delta\partial c$ chain. The permutation of the elements of the chain returned by the application of $\delta_{d-2} \circ \partial_{d-1}$ operator, can be expressed as the composition of m cyclic permutations, each one sharing one of $(d-2)$-elements of the boundary of the current chain value, where m is the length of its boundary. Iin Figure 4 the corolle can be subdivided into the sets of petals around each boundary edge.

Linear independence of the extracted cycles. The basis of elementary d-cycles corresponding to the interior d-cells of the generated d-complex is linearly independent. The minimality of basis cycles is guaranteed by the fact that the cardinality of incidence relation between d-cells and $(d-1)$-cells is exactly twice the number of $(d-2)$-cells. For any other basis of cycles this cardinality is higher.

Poset of isolated boundary cycles. When the boundary of the sum of all basis d-cells (called *total d-chain* in what follows) is an unconnected set of $(d-1)$-cycles, these n *isolated* boundary components have to be compared with each other, to determine the possible relative containment and consequently their orientation. To this

purpose an efficient point classification algorithm is used, in order to compute the poset (partially ordered set) induced by the containment relation of isolated components.

Base case: arrangement of lines in 2D. Most of the actual numerical computing is performed in $\mathbb{E}^2$ by generating the 2D arrangement generated by a set of lines X_1^σ that, with $d = 2$, provide exactly a collection of $(d-1)$-complexes. See Figure 2 for an illustration of the steps below. The computation described here is repeated for each X_1^σ subcomplex generated from input data, i.e. for each 2-cell σ.

Segment subdivision: linear graph. The set of line segments in X_1^k is pairwise intersected. The endpoints and the intersection points become the vertices of a graph, and the segment fragments the edges of it.

Maximal 2-point-connected subgraphs. Using the Hopcroft-Tarjan algorithm (Hopcroft and Tarjan, 1973) the maximal biconnected subgraphs are computed. The dangling edges and dangling trees are discarted.

Topological extraction of X_2^σ. The topological gift-wrapping algorithm in 2D, introduced in (Paoluzzi et al., 2017), is used to compute the 1-chain representation of cells in X_2^σ, i.e. its (local) arrangement, codified as $[\partial_2^\sigma]$. See Figures 3 and 2 for two graphical illustration in 2D of the dimension-independent topological gift wrapping algorithm.

4 EXAMPLES

4.1 Meaning of a Boundary Matrix

When constructing the matrix of a linear operator between linear spaces, say $\partial_d : C_d \to C_{d-1}$, by building it column-wise, we actually construct an element of the basis of the domain space, represented as a linear combination of basis elements of the codomain space. In this sense we "build" or "extract" one d-cell as a $(d-1)$-cycle, i.e. by constructing its boundary or frontier. We would like to note that a *cycle* ia a closed chain, i.e. a chain without boundary, and that the boundary of a boundary is empty.

The `LAR` representation of d-cells, i.e. the characteristic matrix M_d, is generated after the construction of the matrix $[\partial_d]$, by executing, for each column of $[\partial_d]$, and for each non-zero element of it (indices of $(d-1)$-cells of the cycle), the elementwise union of the corresponding rows of the characteristic matrix M_{d-1}, so finally getting the list of vertices of the "built" or "extracted" d-cell. See the Appendix A for the LAR representation of both the input (Figure 1a) and the output (Figure 1h) of the cartoon example in Figure 1, and for its boundary operator matrix $[\partial_3]$.

4.2 Merge of 3D cuboidal complexes

A first example of the *Merge* algorithm from the prototype implementation of `larlib.jl` is provided here and displayed in Figures 5 and 6.

The input data are two $3 \times 3 \times 3$ solid meshes of 3-cubes, both centred on the centroid, with the second rotated by $\pi/6$ about the x axis and by $\pi/6$ about the z axis. Both the topologies are described in LAR as the pair `(FV,CV)`, i.e., as *face-by-vertices* and *cell-by-vertices*. As it is possible to see, there are 108 quadrilateral faces and 27 cubical cells. Of course by column we have the indices to vertices of a single cell.

```
julia> FV
4 x 108 Array{Int64,2}:
 1  2  3   5   6   7   9  10  11  17 ... 36  37  38  39  40  41  42  43  44
 2  3  4   6   7   8  10  11  12  18     40  41  42  43  44  45  46  47  48
 5  6  7   9  10  11  13  14  15  21     52  53  54  55  56  57  58  59  60
 6  7  8  10  11  12  14  15  16  22     56  57  58  59  60  61  62  63  64

julia> CV
8 x 27 Array{Int64,2}:
  1   2   3   5   6   7   9  10  11 ... 33  34  35  37  38  39  41  42  43
  2   3   4   6   7   8  10  11  12     34  35  36  38  39  40  42  43  44
  5   6   7   9  10  11  13  14  15     37  38  39  41  42  43  45  46  47
  6   7   8  10  11  12  14  15  16     38  39  40  42  43  44  46  47  48
 17  18  19  21  22  23  25  26  27     49  50  51  53  54  55  57  58  59
 18  19  20  22  23  24  26  27  28 ... 50  51  52  54  55  56  58  59  60
 21  22  23  25  26  27  29  30  31     53  54  55  57  58  59  61  62  63
 22  23  24  26  27  28  30  31  32     54  55  56  58  59  60  62  63  64
```

The 3-cells of the output 3-complex generated by the *Merge* algorithm are actually codified inside the coboundary sparse matrices $[\delta_0], [\delta_1], [\delta_2]$. The algorithm actually codifies the (signed or unsigned, depending on the need) topological incidences called `(WE,EF,FC)`, where `W` is the array of vertices after the merge, i.e., the vertices of the fragmented cells, and `(WE,EF,FC)` respectively denote the *Edges by vertices*, the *Faces by edges*, and the *Cells by faces*.

The sparse matrix matrix $[\delta_2]$ of the linear operator $\delta_2 : C_2 \to C_3$ is displayed as color image in Figure 6. Such (236×816) matrix has Julia's type `SparseMatrixCSC{Int8,Int64}` and is stored in CSC (Compressed Sparse Column) format, with 1.632 stored (non-zero) entries in $\{-1, 1\}$.

It may be worthwhile to note that the stored entries are exactly the double of the 2-face number, since each face provides in his column for exactly two (opposite) non-zero elements column of δ_2, external 2-cycle included.

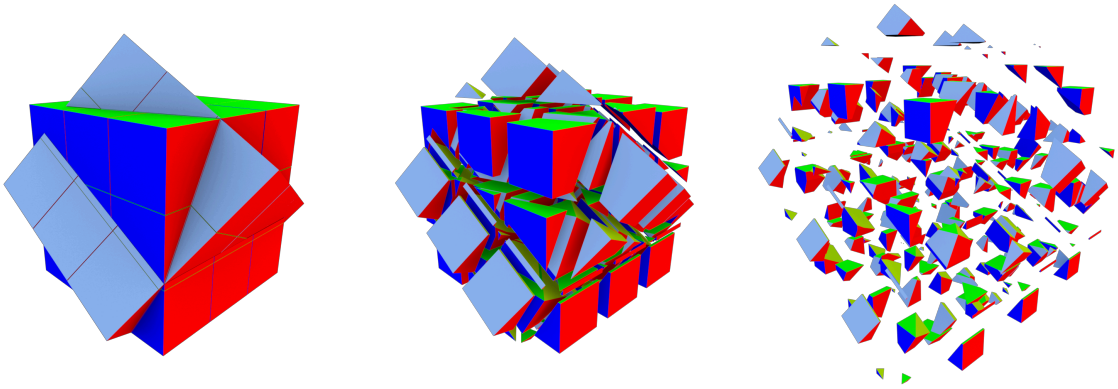


Figure 5: Merge of two rotated $3 \times 3 \times 3$ cuboidal 3-complexes: (a) the initial positions of the two input 3-complexes, each with 3^3 unit cubic cells; (b) 3-cells of merged complex, with space explosion of small scaling parameter; (c) larger space explosion. Each exploded cell is translated by a scaling-induced movement of its centroid. Output cells are not necessarily convex. In the merged complex we get 236 3-cells and 816 2-cells, both possibly non-convex.

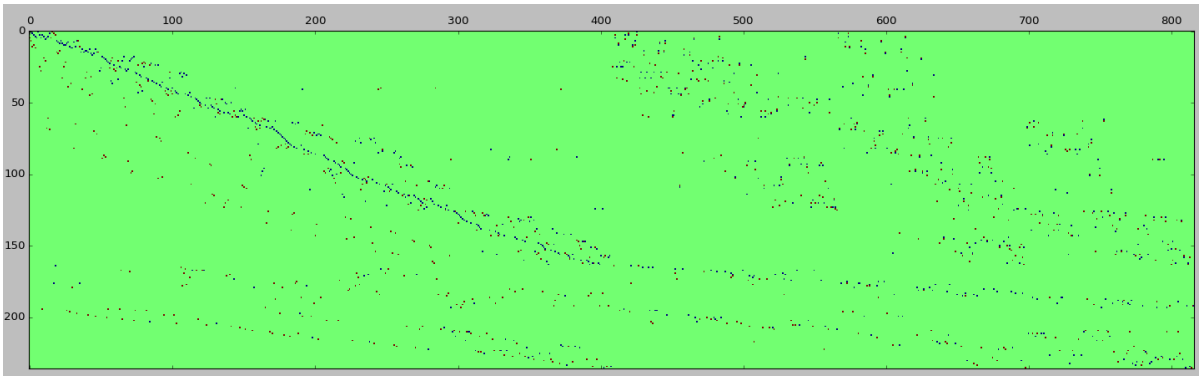


Figure 6: Sparse matrix $[\delta_2]$ of the operator $\delta_2 : C_2 \to C_3$ for the cell-decomposition in Figure 5. We have a (236×816) matrix of Julia's type `SparseMatrixCSC{Int8,Int64}` (CSC $\equiv$ Compressed Sparse Column) with 1.632 stored (non-zero) entries $\in \{-1, 1\}$. Let us note that there are exactly *two* opposite *non-zero* elements in each column of δ_2, external 2-cycle included, with each face providing a pair of non zeros.

4.3 Posets of Unconnected Subcomplexes

The `Larlib.jl` package is providing a full implementation of the *Merge* algorithm (Paoluzzi et al., 2017), including unconnected components in the output, either contained or not within some boundary shells of other components. To manage the very general case, the coboundary matrices of every isolated d-components are computed, each including one redundant row (i.e. linearly dependent on the other rows) corresponding to the boundary $(d-1)$-cycle of the component. Such redundant d-cycles constitute a poset (partial ordered set) with respect to the relative containment relation. The poset's lattice is reconstructed via a point-shell containment algorithm, and the parity of each boundary d-cycle in the lattice graph is used to merge each container (even) and contained (odd) cycle pairs in the aggregated (co)boundary matrix .

5 CONCLUSION

In this paper we have discussed the design goals, the features and the implementation state of the novel `Larlib.jl` Julia package for topological computing with cellular and chain complexes. The distintive character of this approach is to deal with global operations over the space decomposition into connected cells, according to the old orientation of graphics hardware to give support to global / local affine or projective transformations. A d-chain is in fact a subset of cells of the same dimension d, and its coordinate representation is a sparse mapping ranging on the whole set of d-cells, and taking values either in $\{0,1\}$ or in $\{-1,0,+1\}$, depending on the possible consideration of cell orientation. Most topological operations and queries, and in particular the application of boundary or coboundary operators, can be therefore performed on a subset (even the whole set) of cells at the same time, as a single SpMV (sparse matrix-vector) multiplication. This approach will favour the rewriting of most topological/geometrical algorithms as dataflow processes of operator applications, according to the powerful hardware/software computational architectures of the last generation.

5.1 How to Contribute

The project discussed here is open-sourced and provides a liberal (MIT) licence, according to the common style of Julia's packages. New developers are needed in the short time to get the library to take momentum. The best approach is to fork the project on `https://github/julia/geometry/larlib.jl` and use the current development release into your own project or application, then abstracting some generally useful functionality, testing it deeply, and finally asking for its pull into the master branch. In case of your use of the software, please do not forget to mention this paper as a reference on your articles.

5.2 Next Steps

Presently, the `Larlib.jl` implementation reached a quite complete enactement of the 2D / 3D *Merge* algorithm of cellular complexes (Paoluzzi et al., 2017), and generates a whole chain complex, i.e. the whole set of (co)boundary operators acting on the space decomposition (i.e. the space arrangement) induced by the merged set of input complexes.

The next step concerns the use of this result for reducing the resolution of any variadic Boolean formula between solids (including simple Boolean expressions) to the fast evaluation of a Boolean predicate over a truth table having by columns the coordinate representation of the input arguments within the space decomposition generated by the merge. The development plan of `Larlib.jl` also includes an efficient evaluation of the pseudoinverse of the boundary operator, in order to compute the field of distances from actual boundaries, to be applied over any fixed solid chain in the chain space. This operation is needed in order to perform mesh-free simulations of physical equations on cellular models.

Also, in the current and in the next semester, the students and collaborators of CVDLab at Roma Tre University will be interested in a porting project of the already implemented parts of `Larlib.py` to Julia, including the applications related to extraction of geometric models from 3D medical images (Paoluzzi et al., 2017), and the design and/or material decomposition of building architectures (Paoluzzi et al., 2017).

ACKNOWLEDGEMENTS

We like to thank the friendly and warm assistance by Antonio DiCarlo and Vadim Shapiro, that contributed to Roma Tre approach to discrete modeling, which provided the foundations of LAR and the main algorithms over it. Giorgio Scorzelli, Enrico Marino and Federico Spini provided several types of support, mainly by writing and maintaining the `pyplasm` package (G.S.) and testing some prototype `larlib` ideas in Javascript (E.M and F.S.). We also gratefully acknowledge the partial finantial support from SOGEI, the ICT company of the Italian Ministry of Economy and Finance.

REFERENCES

Baumgart, B. G. (1972). Winged edge polyhedron representation. Technical report, Stanford, CA, USA.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98.

Boxel, D. V. (2016). *Deep Learning with TensorFlow*. Packt Publishing.

Buluç, A. and Gilbert, J. R. (2011). The combinatorial BLAS: design, implementation, and applications. *The International Journal of High Performance Computing Applications*, 25(4):496–509.

Buluç, A. and Gilbert, J. R. (2012). Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. *SIAM Journal of Scientific Computing (SISC)*, 34(4):170 – 191.

Buono, D., Petrini, F., Checconi, F., Liu, X., Que, X., Long, C., and Tuan, T.-C. (2016). Optimizing sparse matrix-vector multiplication for large-scale data analytics. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, pages 37:1–37:12, New York, NY, USA. ACM.

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

Desbrun, M., Kanso, E., and Tong, Y. (2006). Discrete differential forms for computational modeling. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, pages 39–54, New York, NY, USA. ACM.

DiCarlo, A., Milicchio, F., Paoluzzi, A., and Shapiro, V. (2009a). Chain-based representations for solid and physical modeling. *Automation Science and Engineering, IEEE Transactions on*, 6(3):454 –467.

DiCarlo, A., Milicchio, F., Paoluzzi, A., and Shapiro, V. (2009b). Discrete physics using metrized chains. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, SPM '09, pages 135–145, New York, NY, USA. Acm.

DiCarlo, A., Paoluzzi, A., and Shapiro, V. (2014a). Linear algebraic representation for topological structures. *Computer-Aided Design*, 46:269–274.

DiCarlo, A., Paoluzzi, A., and Shapiro, V. (2014b). Linear algebraic representation for topological structures. *Comput. Aided Des.*, 46:269–274.

Elcott, S. and Schroder, P. (2006). Building your own dec at home. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, pages 55–59, New York, NY, USA. ACM.

Fabri, A., Giezeman, G.-J., Kettner, L., Schirra, S., and Schönherr, S. (2000). On the design of cgal a computational geometry algorithms library. *Softw. Pract. Exper.*, 30(11):1167–1202.

Fogel, E., Halperin, D., Kettner, L., Teillaud, M., Wein, R., and Wolpert, N. (2007). Arrangements. In Boissonat, J.-D. and Teillaud, M., editors, *Effective Computational Geometry for Curves and Surfaces*, Mathematics and Visualization, chapter 1, pages 1–66. Springer.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. The MIT Press.

Goodman, J. E., O'Rourke, J., and Tòth, C. D., editors (2017). *Handbook of Discrete and Computational Geometry – Third Edition*. CRC Press, Inc., Boca Raton, FL, USA.

Hachenberger, P., Kettner, L., and Mehlhorn, K. (2007). Boolean operations on 3d selective nef complexes: Data structure, algorithms, optimized implementation and experiments. *Comput. Geom. Theory Appl.*, 38(1-2):64–99.

Hatcher, A. (2002). *Algebraic topology*. Cambridge University Press.

Hirani, A. N. (2003). *Discrete Exterior Calculus*. PhD thesis, Pasadena, CA, USA. AAI3086864.

Hopcroft, J. and Tarjan, R. (1973). Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378.

Kepner, J. and Gilbert, J. (2011). *Graph Algorithms in the Language of Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Knuth, D. E. (1984). Literate programming. *The Computer Journal*, 27(2):97–111.

Paoluzzi, A., DiCarlo, A., Furiani, F., and Jirík, M. (2016). CAD models from medical images using LAR. *Computer-Aided Design and Applications*, 13(6):747–759.

Paoluzzi, A., Shapiro, V., and DiCarlo, A. (2017). Regularized arrangements of cellular complexes. *ArXiv e-prints*. https://arxiv.org/pdf/1704.00142.pdf.

Paoluzzi, A., Vicentino, M., Baldazzi, C., and Pascucci, V. (2001). *Geometric Programming for Computer Aided Design*. John Wiley & Sons, Inc., New York, NY, USA.

Regli, W. C., Rossignac, J., Shapiro, V., and Srinivasan, V. (2016). The new frontiers in computational modeling of material structures. *Computer-Aided Design*, 77:73–85.

Requicha, A. G. (1980). Representations for rigid solids: Theory, methods, and systems. *ACM Comput. Surv.*, 12(4):437–464.

Shapiro, V. (2017). Personal communication.

Tonti, E. (1976). Sulla struttura formale delle teorie fisiche. *Rend. Sem. Mat. Fis. Milano*, 46:163–257 (1978).

Tonti, E. (2013). *The Mathematical Structure of Classical and Relativistic Physics*. Birkhäuser.

Tonti, E. (2014). Why starting from differential equations for computational physics? *J. Comput. Phys.*, 257:1260–1290.

Woo, T. (1985). A combinatorial analysis of boundary data structure schemata. *IEEE Comput. Graph. Appl.*, 5(3):19–27.

Zhou, Q., Grinspun, E., Zorin, D., and Jacobson, A. (2016). Mesh arrangements for solid geometry. *ACM Trans. Graph.*, 35(4):39:1–39:15.

A APPENDIX

The `Larlib.jl` code producing the example of Figures 1a and 1h, and the corresponding input and output LAR data are given below.

```
V = [[0.0,0.0,0.0],[0.0,0.0,1.0],[0.0,1.0,0.0],
[0.0,1.0,1.0],[1.0,0.0,0.0],[1.0,0.0,1.0],
[1.0,1.0,0.0],[1.0,1.0,1.0],[0.5,0.5,0.5],
[0.5,0.5,1.5],[0.0,1.366,0.5],[0.0,1.366,1.5],
[1.366,1.0,0.5],[1.366,1.0,1.5],[0.866,1.866,
0.5],[0.866,1.866,1.5]]

FV = [[0,2,4,6],[8,9,10,11],[8,10,12,14],[1,3,5,
7],[12,13,14,15],[0,1,2,3],[2,3,6,7],[0,1,4,5],
[8,9,12,13],[10,11,14,15],[9,11,13,15],
[4,5,6,7]]
```

The array `V` contains the input vertices; `FV` is the LAR representation of the characteristic matrix M_2, i.e. of the incidence relation between faces and vertices. The coboundary matrix $[\delta_2] : C_2 \to C_3 = \partial_3^\top : C_3 \to C_2$ between oriented chains, computed by the *Merge* algorithm starting from the above data, is given below, with $[\delta_2]^t = [\partial_3]$:

$$[\delta_2] = \begin{pmatrix} 1 & 0 & -1 & -1 & 0 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 1 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & -1 \end{pmatrix}$$

```
W = [[1.0,1.0,0.0],[1.0,0.0,0.0],[0.0,1.0,0.0],
[0.0,0.0,0.0],[0.5,0.5,1.0],[0.5,0.5,0.5],
[0.2113,1.0,1.0],[0.2113,1.0,0.5],[0.5,0.5,1.5],
[0.0,1.366,0.5],[0.0,1.366,1.5],[1.0,0.7887,
0.5],[1.0,1.0,0.5],[1.366,1.0,0.5],[0.866,1.866,
0.5],[1.0,1.0,1.0],[1.0,0.7887,1.0],[0.0,1.0,
1.0],[0.0,0.0,1.0],[1.0,0.0,1.0],[1.366,1.0,
1.5],[0.866,1.866,1.5]]

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

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

It is worth noting the variable numbers of vertices per face in the `FW` array, with some non-convex faces, and the non-convexity of two solid cells in `CW` (see Figure 1h). The reader is kindly asked to compare for simplicity and compactness this representation with any other solid representation scheme. It also reduces to the standard mathematical representation of simplicial complexes, when cells are simplices, but can be applied to more general cells, even containing holes.