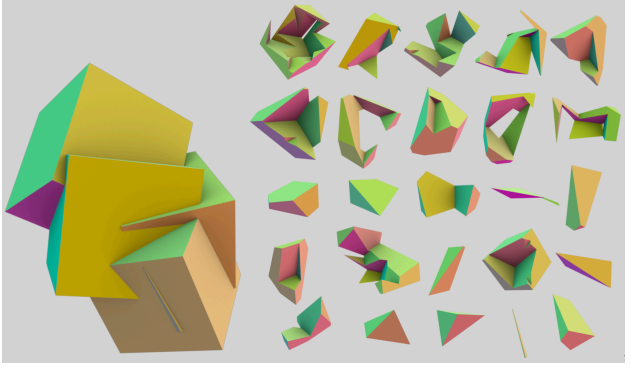
Graphical Abstract

Finite Boolean Algebras for Solid Geometry using Julia's Sparse Arrays

Alberto Paoluzzi,Vadim Shapiro,Antonio DiCarlo,Giorgio Scorzelli,Elia Onofri



The set of join-irreducible atoms of the Boolean Algebra generated by a partition of $\mathbb{E}^3$, composing a basis of 3-chains.

Highlights

Finite Boolean Algebras for Solid Geometry using Julia's Sparse Arrays

Alberto Paoluzzi,Vadim Shapiro,Antonio DiCarlo,Giorgio Scorzelli,Elia Onofri

- Basic tools of linear algebra and algebraic topology are used, namely, (sparse) matrices of linear operators and matrix multiplication and transposition. Interval-trees and kd-trees introduced only for acceleration.
- Validity of topological computations is guaranteed, since operator matrices satisfy by construction the (graded) constraints $\partial^2 = 0$ and $\delta^2 = 0$.
- The evaluation of CSG expressions of arbitrary complexity is done in a novel way. All input B-reps are accumulated in a single collection of 2-cells, each of which is operated *independently*, so generating a collection of local topologies, merged by boundary congruence.
- Once the global space partition is generated, and 3-cells are classified w.r.t. all component solids, via a single point-set containment test, *all CSG expressions*—of any complexity—can be evaluated simply by bitwise vectorized logical operations.
- Distinctions are removed between manifold and non-manifold representations, allowing for mixing B-reps, cellular decompositions of elementary solids, and/or regular grids. This allows for the generation of internal structures and mixed-dimensional objects needed in many applications, for example to make stronger and more resilient the ones to be produced via 3D printing.
- This approach can be extended to general dimensions and/or implemented on highly parallel computational engines, also using standard computing kernels.

Finite Boolean Algebras for Solid Geometry using Julia's Sparse Arrays[⋆,⋆⋆]

Alberto Paoluzzi[a,*], Vadim Shapiro[b], Antonio DiCarlo[d], Giorgio Scorzelli[c] and Elia Onofri[a]

[a]*Roma Tre University, Rome, RM, Italy*
[b]*University of Wisconsin, Madison & International Computer Science Institute (ICSI), Berkeley, California, USA*
[c]*Scientific Computing and Imaging Institute (SCI), Salt Lake City, Utah, USA*
[d]*CECAM-IT-SIMUL Node, Rome, Italy*

ARTICLE INFO

Keywords:
Computational Topology
Chain Complex
Cellular Complex
Solid Modeling
Constractive Solid Geometry (CSG)
Linear Algebraic Representation (LAR)
Arrangement
Boolean Algebra

ABSTRACT

The goal of this paper is to introduce a new method in computer-aided geometry of solid modeling. We put forth a novel algebraic technique to evaluate any variadic expression between polyhedral d-solids ($d = 2, 3$) with regularized operators of union, intersection, and difference, i.e., any CSG tree. The result is obtained in three steps: first, by computing an independent set of generators for the d-space partition induced by the input; then, by reducing the solid expression to an equivalent logical formula between Boolean terms made by zeros and ones; and, finally, by evaluating this expression using bitwise operators. This method is implemented in Julia using sparse arrays. The computational evaluation of every possible solid expression, usually denoted as CSG (Constructive Solid Geometry), is reduced to an equivalent logical expression of a finite set algebra over the cells of a space partition, and solved by native bitwise operators.

1. Introduction

The aim of this paper is to introduce a novel method for evaluation of Boolean algebraic expressions including solid objects, often called constructive solid geometries, by reduction to vectorized bitwise evaluation of coordinate representations within a linear space of chains of cells. This approach uses only basic algebraic topology and basic linear algebra, in accord with current trends in data science.

1.1. Motivation

From the very beginning, solid modeling suffered from a dichotomy between "boundary" and "cellular" representations, that induced practitioners to separate the computation of the solid surface from that of its interior, and/or to introduce the so-called "non-manifold" representations, often requiring special and/or very complex data structures and algorithms. Conversely, our approach relies on standard mathematical methods, i.e., on basic linear algebra and algebraic topology, and allows for an unified evaluation of variadic Boolean expressions with solid models, by using cell decompositions of either the interior or the boundary. In particular, we use graded linear spaces of (co)chains of cells, as well as graded linear (co)boundary operators, and compute the operator matrices between such spaces.

The evaluation of a solid expression is therefore reduced to the computation of the *chain complex* of the $\mathbb{E}^d$ partition (arrangement) induced by the input, followed by the translation of the solid geometry formula to an equivalent binary formula of a finite algebra over the set of generators of the d-chain space. Finally, the bitwise evaluation of such binary expression, using native bitwise operators, is directly performed by the compiler. The output is the coordinate representation of the resulting d-chain in chain space, i.e., the matrix column generating the solid result, the boundary set of which is optionally produced as the product of the boundary matrix times this binary column vector.

Applied algebraic topology and linear algebra require the use of matrices as fundamental data structures, in accord with the current development of computational methods (Strang, 2019). Our (co)boundary matrices may be very large, but they are always very sparse, so yielding comparable space and time complexity with previously known methods.

⋆This work was partially supported from Sogei S.p.A. – the ICT company of the Italian Ministry of Economy and Finance, by grants 2016-17 and 2019-20.
⋆⋆V.S. is supported in part by National Science Foundation grant CMMI-1344205 and National Institute of Standards and Technology.
*Corresponding author
ORCID(s): 0000-0002-3958-8089 (A. Paoluzzi)

The advantages of formulating the evaluation of Boolean expressions between solid models in terms of (co)chain complexes and operations on sparse matrices are that: (1) the common and general algebraic topological nature of such operations is revealed; (2) implementation-specific low-level details and algorithms are hidden; (3) explicit connection to computing kernels (sparse matrix-vector and matrix-matrix multiplication) and to sparse numerical linear algebra systems on modern computational platforms is provided; (4) systematic development of correct-by-construction algorithms is supported ($\partial^2 = 0$ automatically satisfied); (5) the computational solution of every possible solid expression with union, intersection and complement, denoted as CSG, is reduced to an equivalent logical expression of a finite Boolean algebra, and natively solved by vectorized bitwise operators.

1.2. Related work

An up-to-date extensive survey of past and current methods and representations for solid modeling can be found in (Hoffmann and Shapiro, 2017). We discuss in the following a much smaller set of references, that either introduced the ideas discussed in our work or are directly linked to them. Note that such concepts were mostly published in the foundational decades of solid modeling technologies, i.e. in the 70's trough the 90's. Time is ripe to go beyond.

Some milestones are hence recalled in the following paragraphs, starting from Baumgart (1972), who introduced the data structure of Winged Edge Polyhedra for manifold representations at Stanford, and included the first formulation of "Euler operators" in the "Euclid" modeler, using primitive solids, operators for affine transforms, and imaging procedures for hidden surface removal. Braid (1975), starting from primitives like cubes, wedges, tetrahedrons, cylinders, sectors, and fillets, or from a planar primitive of straight lines, illustrated how to synthesize solids bounded by many faces, giving algorithms for addition (quasi-disjoint or intersecting union) and subtraction of solids.

The foundational Production Automation Project (PAP) at Rochester in the seventies described computational models of solid objects (Requicha, 1977) by using relevant results scattered throughout the mathematical literature, placed them in a coherent framework and presented them in a form accessible to engineers and computer scientists. Requicha and Voelcker (1977) provided also a mathematical foundation for *constructive solid geometry* by drawing on established results in modern axiomatic geometry and point-set topology. The term "constructive solid geometry" denotes a class of schemes for describing solid objects as compositions of primitive solid "building blocks".

Weiler introduced at RPI the first non-manifold representation (Weiler, 1986), called radial-edge data structure, and boundary graph operations for non-manifold geometric modeling topology. Since then, several similar data structures for non-manifold boundaries and interior structures (solid meshes) were introduced and implemented in commercial systems, with similar operations and performances. Hoffmann, Hopcroft, and Karasick at Cornell (Hoffmann et al., 1987) provided a reliable method for regularized intersection, union, difference, and complement of polyhedral solids described using a boundary representation and local cross-sectional graphs of any two intersecting surfaces. An *algebra of polyhedra* was devised by Paoluzzi and his students in Rome (Paoluzzi et al., 1989), using boundary triangulations, together with very simple algorithms for union, intersection, difference and complement. Their data structure, called winged-triangle, is space-optimal for piecewise-linear representations of polyhedra with curved boundaries.

The Selective Geometric Complex (SGC) by Rossignac and O'Connor (1989) at IBM Research provided a common framework for representing cellular decompositions of objects of mixed dimensions, having internal structures and possibly incomplete boundaries. 'Boundary-of' relations capture incidences between cells of various dimensions. Shapiro (1991) presented a *hierarchy of algebras* to define formally a family of Finite Set-theoretic Representations (FSR) of semi-algebraic subsets of $\mathbb{E}^d$, including many known representation schemes for solid and non-solid objects, such as boundary representations, Constructive Solid Geometry, cell decompositions, Selective Geometric Complexes, and others. Exemplary applications included B-rep → CSG and CSG → B-rep conversions.

Semi-dynamical algorithms for maintaining *arrangements* of polygons on the plane and the sphere were given by Goldwasser (1995). A recent work more related to the present paper is by Zhou et al. (2016). They compute mesh *arrangements* for *solid geometry*, taking as input any number of triangle meshes, iteratively resolving triangle intersections with previously subdivided 3D cells, and assigning winding number vectors to cells, in order to evaluate variadic Boolean expressions. Their approach applies only to boundary triangulations and uses standard geometric computing methods, while the present one applies to any cellular decomposition, either of the interior or of the boundary, computes intersections between line segments in 2D, and transforms every Boolean solid expression into a logical expression solved natively by the compiler with vectorized bitwise operators.

Our related work in geometrical and physical modeling with *chain and cochain complexes* was introduced in DiCarlo et al. (2009b) and DiCarlo et al. (2009a). The Linear Algebraic Representation (LAR), using sparse matrices, and its applications to the computation of (co)boundary matrices and other chain adjacencies is discussed in DiCarlo

et al. (2014). The computational pipeline and the detailed algorithms to compute the space decomposition induced by a collection of solid models is given in Paoluzzi et al. (2017a), Paoluzzi et al. (2017b). An open-source prototype implementation is available at `https://github.com/cvdlab/LinearAlgebraicRepresentation.jl`.

1.3. Overview

In Section 2 we provide a short introduction to the basic algebraic-topological concepts and notations used in this paper, including graded vector spaces, chain and cochain spaces, boundary and coboundary operators, complexes, arrangements, and finite Boolean algebras. Section 3 describes a hierarchy of algebras generated by decompositions of $\mathbb{E}^d$ within the frame of representation theory for solid modeling, and introduces a realization based on independent generators of chain spaces. Section 4 extends this hierarchy to algebras with topological operations. Section 5 analyzes the 2D/3D geometric and topological algorithms introduced. Some simple examples are scattered in the text. Section 6 explores the evaluation of some Boolean formulas with solid objects, both in 2D and in 3D, including the computation of the Euler characteristic of the boundary of a simple Boolean union. In Section 7 the main results of our approach are summarized and compared with the current state of the art, including (a) solid modeling via sparse matrices and linear algebra, (b) the computation of generators for the column space of boundary matrices, and (c) our new method for CSG evaluation. The closing section 8 presents a summary of contents, and outlines possible applications of the ideas introduced. Small chunks of Julia code are inserted in the examples.

2. Background

In this section we synthesize the main concepts used in the paper, and recall some notations and properties. We draw from (Arnold, 2018) and (Strang, 2019) for algebraic concepts.

2.1. Partitions of space

Let X be a topological space, and $\Lambda(X) = \bigcup_0^d \Lambda_p$ a partition of X, with Λ_p a set of (relatively) open, connected, and manifold p-cells. A *CW-structure* of X is a *filtration* $\emptyset = X_{-1} \subset X_0 \subset X_1 \subset \ldots \subset X_{d-1} \subset X = \bigcup X_p$, 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 $\Lambda_p = \Lambda_p(X)$.

A *stratification* of X with topology T is a filtration $F : \mathbb{N} \to T(X)$ where one imposes some additional regularity constraints on the complements $F(j)\backslash F(i)$ whenever $j \geq i$. In particular, they are either empty or a smooth submanifold of dimension i. If X is a smooth variety, then typically a stratification of X is a Whitney stratification. The connected components of the difference $F(i)\backslash F(i-1)$ are the *strata* of dimension i.

A *CW-complex* is a space X endowed with a CW-structure, which is also called a *cellular complex*. Every CW-complex is stratifiable (Borges, 1966). A cellular complex is *finite* when it contains a finite number of cells. A finite *cellular complex* Λ is a *partition* of a topological d-space X in a discrete set of p-cells ($0 \leq p \leq d$) such that: (a) $\sigma \in \Lambda$ implies $\tau \in \Lambda$ for all faces τ of σ; (b) the regularized intersection[1] of every pair of cells of Λ either is in Λ or is the empty set. A d-complex is *regular* if every closed d-cell equates the closure of its interior, and if 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* of a cell σ is its compact point-set.

2.2. Cells and Chains

A *p-manifold* is a topological space where each point has a neighborhood that is homeomorphic to $\mathbb{E}^p$. The *p-cell* σ ($0 \leq p \leq d$) of cellular complexes used in this paper is piecewise-linear, connected, possibly non convex, p-manifold, and not necessarily contractible[2]. An r-face τ of a p-cell σ ($0 \leq r \leq p$) is an r-cell contained in the frontier of σ.

A *p-chain* can be seen, with some abuse of language, as a collection of p-cells. The set $C = \oplus C_p$ of chains can be given the structure of a graded vector space (see the following Section 2.3) by defining sums of chains with the same dimension, and products times scalars in a field, with the usual properties.

A *basis* U_p is the set of *independent* (or *elementary*) chains $u_p \in C_p$, given by singletons of Λ_p elements. Every chain $c \in C_p$ is uniquely generated by a linear combination of the basis with field coefficients. Once the basis is fixed, the coordinate representation of each $\{\sigma_k\} = u_k \in C_p$ is unique. This is an ordered sequence of coefficients, either from $\{0, 1\}$ (unsigned representation) or from $\{-1, 0, +1\}$ (signed representation). With some abuse of language, we often call p-cells the independent generators of C_p, i.e. the elements of U_p.

[1] A regularized set is the closure of interior points of the set.

[2] The cells of CW-complexes are contractible to a point. With our LAR representation they may contain internal holes.

2.3. Chain and cochain complexes

A *graded vector space* is a vector space V expressed as a direct sum of spaces V_k indexed by integers in $[0, d]$:

$$V = \oplus_{k=0}^{d} V_k, \qquad [0, d] := \{k \in \mathbb{N} \mid 0 \leq k \leq d\}.$$

A linear map $f : V \to W$ between graded vector spaces is called a *graded map* of degree p if $f(V_k) \subset W_{k+p}$.

A *chain complex* is a graded vector space V furnished with a graded linear map $\partial : V \to V$ of degree -1 which satisfies $\partial^2 = 0$, called *boundary operator*. In other words, a chain complex is a sequence of vector spaces C_k and linear maps $\partial_k : C_k \to C_{k-1}$, such that $\partial_{k-1} \circ \partial_k = 0$.

A *cochain complex* is a graded vector space V furnished with a graded linear map $\delta : V \to V$ of degree $+1$ which satisfies $\delta^2 = 0$, called *coboundary operator*. That is to say, a cochain complex is a sequence of vector spaces C^k and linear maps $\delta^k : C^k \to C^{k+1}$, such that $\delta^{k+1} \circ \delta^k = 0$.

Since any linear map $L : V \to W$ between linear spaces induces a dual map $L' : W' \to V'$ between their duals, any chain complex is associated with a dual cochain complex, and viceversa. Moreover,

$$(\delta^k \omega) g = \omega(\partial_{k+1} g), \qquad \omega \in C^k, g \in C_{k+1}.$$

2.4. Arrangements, (co)boundary matrices

The *arrangement* $\mathcal{A}(S)$ generated by a finite collection S of bounded geometric objects of dimension $d-1$ is a partition of $\mathbb{E}^d$ into a discrete set Λ of connected *open* p-cells, $(0 \leq p \leq d)$. We show in Section 5.1 that the topology of an arrangement of $\mathbb{E}^d$ is encoded in the (co)boundary maps $\partial_d, ..., \partial_1$ of the chain complex associated with the $\mathbb{E}^d$ partition. The input collection S may contain $(d\text{-}1)$-skeletons of cellular complexes, and/or $(d\text{-}1)$-complexes.

In order to represent both the $\mathbb{E}^d$ *partition* $\mathcal{A}$ into open cells, the generated *chain complex* bases U_p, and its *Boolean algebra* $\mathcal{A}$ (using the same symbol, via canonical identification of basis elements $u_k \in U_d$ of linear chain space with algebra atoms) we need to construct the whole chain of (co)boundary maps, i.e. their matrices.

Such matrices are very sparse, with sparsity growing linearly with the number m of cells (rows). Sparsity may be defined as one minus the ratio between non-zeros and the number of matrix elements. It is fair to consider that the non-zeros per row are bounded by a small constant in topological matrices, hence the number of non-zero elements grows linearly with m. With common data structures (Cimrman, 2015) for sparse matrices, the storage cost $O(m)$ is linear with the number of cells, with $O(1)$ small cost per cell that depends on the storage scheme.

3. Solid Geometry Algebras

In (Shapiro, 1991) a representation theory of solid modeling is based on the resolution of *functional forms* $\Phi(\mathcal{F})$ over *finite algebras* of semi-algebraic subsets of $\mathbb{E}^d$, generated by a set $\mathcal{F}$ of *polynomials*. In particular, Shapiro (1991) discusses a hierarchy of algebras over semi-algebraic subsets of $\mathbb{E}^d$, where algebra elements can be constructed using a finite number of FSR operations (standard set ops, closure, and connected component).

Such algebras differ either by the allowed subset of operations and/or by the type of elements, but possess a number of common characteristics. (a) Each algebra in the hierarchy is generated by a *decomposition* of $\mathbb{E}^d$ into join-irreducible elements, which is unique for a fixed set of polynomials F. (b) The *structure* of every element in an algebra is a unique union of join-irreducible elements in the corresponding $\mathbb{E}^d$ decomposition. (c) All algebras are contained in the algebra FSR($\mathcal{F}$) generated by the canonical Whitney regular sign-invariant stratification of $\mathbb{E}^d$. Hence, the elements of all algebras have a unique structure in FSR($\mathcal{F}$).

In this paper we represent (and implement for $d = 2, 3$) only the algebra of piecewise-linear CSG with closed regular cells, as generated by the decomposition of $\mathbb{E}^d$ induced by a collection of cellular complexes with polyhedral cells of dimension $d-1$ (Paoluzzi et al., 2017b). In particular, we show that the structure of each atom of this algebra is characterized by a single point, computed once and for all in its interior. A Set-Membership-Classification (SMC) test w.r.t. each characterizing point transforms the *structure* of any algebra element, and in particular the solid terms (input) of the formula, into a *logical array* of length m, equal to the number of atoms. The evaluation of any Boolean formula of a solid (polyhedral) algebra is hence computed by (a) the decomposition of $\mathbb{E}^d$ space induced by the input polyhedra, followed by (b) the computation of a sparse array of bits for each input solid, and (c) the native application of bitwise operators, according to the symbolic formula to be solved. A single $\mathbb{E}^d$ decomposition may be used to evaluate every functional form over the same algebra, including every subexpression between the same arguments.

Join-irreducible polyhedral 3-cells are computed by Paoluzzi et al. (2017b) as the basis of a linear 3-chain space. The basis generated by five cubes is mapped one-to-one to the the atoms of an algebra $\mathcal{A}(S)$ and is shown in Figure 3b.

3.1. Notations and definitions

This section introduces the notations and concepts related to algebras that are used throughout the manuscript. The definitions can be found in many introductory texts on discrete mathematics, abstract algebra, or data structures (e.g., see Berztiss (1975); Doerr and Levasseur (2019)).

Let $\mathcal{A}$ be a nonempty set, and operations $\otimes_i : \mathcal{A}_{n_i} \to \mathcal{A}$ be functions of n_i arguments. The system $\langle \mathcal{A}; \otimes_1, \dots, \otimes_k \rangle$ is called an *algebra*. Alternatively we say that $\mathcal{A}$ is a set with operations $\otimes_1, \dots, \otimes_k$. By definition, an algebra $\mathcal{A}$ is closed under its operations. If $\mathcal{A}$ has a finite number of elements it is called a *finite* algebra.

A set of elements $\mathcal{H} = \{h_1, ..., h_m\}$ generates the algebra $\mathcal{A}$ (under some operations) if $\mathcal{A}$ is the smallest set closed w.r.t the operations and containing $\mathcal{H}$. The elements $h_i \in \mathcal{H}$ are called *generators* of the algebra $\mathcal{A}$.

Let $\langle \mathcal{A}; \otimes_1, \dots, \otimes_k \rangle$ be an algebra generated by $\mathcal{H} = \{h_1, ..., h_m\}$. A syntactic expression constructed as a valid sequence of operations $\otimes_i$ on n variables x_i denoting elements of $\mathcal{A}$ is called a *functional form* Φ over the algebra $\mathcal{A}$. Note that $\Phi(x_1, ..., x_n)$ is not a function (is not a set of ordered pairs) but *defines* a function of n arguments $\phi : \mathcal{A}^n \to \mathcal{A}$. We use capital Greek letters such Φ, Π, etc. to denote forms, and denote functions over algebras by lower case Greek letters. The distinction between forms and functions is important. Forms and functions over an algebra are formally related by an *evaluation* process, assigning values to the variables in the form, and computing the resulting value of the expression. This relationship is expressed by writing

$$\phi(x_1, ..., x_n) = |\Phi(x_1, ..., x_n)|.$$

If $\mathcal{A}$ is a finite algebra, there is a finite number of distinct functions over $\mathcal{A}$, but an infinite number of distinct forms. If $S \in \mathcal{A}$ is an element of the algebra generated by $\mathcal{H}$, there exists a form Φ over $\mathcal{A}$ such that: $S = |\Phi(\mathcal{H})|$. We then say that S is *describable* in $\mathcal{A}$ by $\mathcal{H}$. In general, $\Phi(\mathcal{H})$ is not unique, but $|\Phi(\mathcal{H})|$ is unique by definition.

3.2. Finite Boolean Algebras

A *basis* B for a topological space X with topology T is a subset of open sets in T such that all other open sets can be written as union of elements of B. A set is said to be *closed* if its complement is *open*. When both a set and its complement are open, they are called *clopen*.

Let us consider both d-cells and d-chains as point-sets. The chains in a linear space C_d can be seen as open point-sets generated by *topological sum* of topological spaces. It can be proved that, if all connected components of a space are open, then a set is clopen if and only if it is the union of pairwise disjoint connected components. All chains in a C_d space are clopen. Therefore, the set C_d of all d-chains, including C_d itself and $\emptyset$, has the *discrete topology*, since all elements of the power set $X = \mathcal{P}(U_d)$ are clopen, where U_d is the C_d basis (see Section 2.2).

Hence, the set X of subsets of independent d-chains is a *discrete topological space*. Using the union and intersection as operations, the clopen subsets of a discrete topological space X form a *finite Boolean algebra*. It can be shown that every finite Boolean algebra is isomorphic to the Boolean algebra $\mathcal{B}$ of all subsets of a finite set (Halmos, 1963), and can be represented by the 2^N binary terms in $\mathcal{B}$. In our case $N = \dim C_d = \#\mathrm{U}_d$.

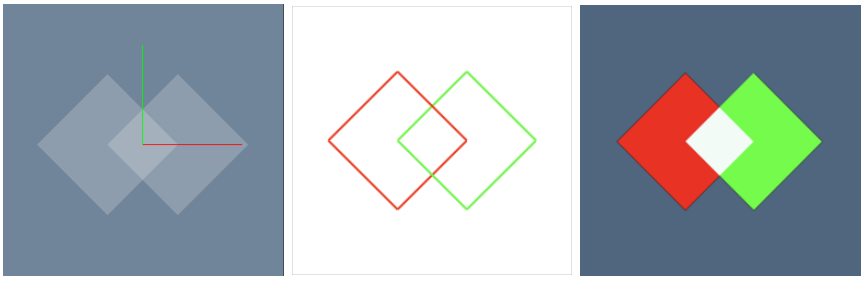
In the remainder of this manuscript, we (mostly) use the symbols $\mathcal{A}$, $\mathcal{L}$, and $\mathcal{B}$, for the arrangement $\mathcal{A}$ of $\mathbb{E}^d$, the finite algebra $\mathcal{L}$ (lattice) generated by a decomposition X of $\mathbb{E}^d$, and the Boolean algebra $\mathcal{B}$ over $\{0, 1\}^N$, with $N = \#X$, respectively. We maintain the notation $\mathcal{A}$ when the element $S \in \mathcal{A}$ is represented by binary d-chain coordinates.

3.3. Finite algebras generated by decompositions of $\mathbb{E}^d$

The duality between decompositions of $\mathbb{E}^d$ and the finite algebras they generate is a powerful tool for constructing canonical formats in geometric modeling. An arbitrary finite collection of subsets $\{A_i\}$ of $\mathbb{E}^d$ satisfying $A_i \cap A_j = \emptyset$, $i \neq j$, and $\cup_i A_i = \mathbb{E}^d$, forms a *partition* (i.e., a disjunctive decomposition) of $\mathbb{E}^d$. Consider the set $\mathcal{L}$ of all subsets of $\mathbb{E}^d$ obtained by finite unions of sets A_j. It should be clear that $\mathcal{L}$ forms a finite Boolean algebra under the operations $\cup$, $\cap$, and complement. Complement is defined for any $S = \cup_{i \in I} A_i$ as $-S = \cup_{j \notin I} A_j$.

Let $\langle \mathcal{L}; +, \cdot \rangle$ be a *finite lattice* with a zero element 0. An element $a \neq 0$ of $\mathcal{L}$ is an *atom* of $\mathcal{L}$ if for all $x \in \mathcal{L}$, $x \cdot a = a$ implies $x = a$ or $x = 0$. For a finite lattice, the *join-irreducible* elements are those which cover precisely one element; further, every element is a join of join-irreducible elements. It is easy to see that every atom a of a lattice $\mathcal{L}$ is also a join-irreducible element of $\mathcal{L}$. If all join-irreducible elements are atoms, the lattice $\mathcal{L}$ is called *atomic*. Let $\mathcal{L}$ be an atomic lattice. Every element $x \in \mathcal{L}$ has a unique representation as the sum of atoms $a \in \mathcal{L}$, such that $x \cdot a = x$.

Consider the Boolean algebra $\langle \mathcal{L}; \cup, \cap, - \rangle$ generated by the partition $\{A_i\}$ of $\mathbb{E}^d$. The elements A_i are *atoms* of the atomic Boolean algebra $\mathcal{L}$. In particular, every finite Boolean algebra $\langle \mathcal{B}; +, \cdot \rangle$ is an atomic lattice. If $\mathcal{B}$ has N atoms A_i, then there are exactly 2^N distinct elements $S \in \mathcal{B}$.



(a) (b) (c) .

Figure 1: *Space arrangement* generated in $\mathbb{E}^2$ by: (a) two overlapping 2-cells A and B; (b) the 4+4 line segments (1-cells) in their skeletons, generating the *four* 2-cells in (c): blue (c_1), red (c_2), white (c_3), and green (c_4). The first (c_1) is the *outer* or *exterior* cell Ω, which is the complement of the union of the others.

Example 1. *Let $\mathcal{P}$ be a set of m hyperplanes that partition $\mathbb{E}^d$ into convex relatively open cells of dimension ranging from zero (points) to d. The collection of all such cells is called a* linear arrangement *$\mathcal{A}(\mathcal{P})$ and has been studied extensively in computational geometry (Edelsbrunner, 1987). The cells in $\mathcal{A}(\mathcal{P})$ are atoms of a Boolean algebra of subsets of $\mathbb{E}^d$ that can be formed by union of convex cells in the arrangement.*

3.4. Atoms from Chain Complexes

We assume that an $\mathbb{E}^d$ partition $\{A_i\} := \mathcal{A}(S) = |\Phi(\mathcal{H}(S_{d-1}))|$ is the result of a finite number of algebraic operations on a collection S_{d-1} of geometric shapes, i.e., of piecewise linear objects (polyhedra of $d-1$ dimension). The partition $\{A_i\}$ is here represented by the *chain complex* $C_\bullet = (C_p, \partial_p)$ associated with the $\mathcal{A}(S)$ *arrangement*. Depending on a choice of elements in $\{A_i\}$, various algebras can be built from primitives in $\mathcal{H}(S_{d-1})$.

The key to implementing this scenario is to determine the structure of S in $\mathcal{B}(\mathcal{H})$ *algebras* . If we restrict the set $\mathcal{H}(S)$ of allowed primitives, and/or allowed operations, we get various smaller algebras $\mathcal{B}(\mathcal{H}(S))$ for elements of $\mathcal{A}(S)$. The structure of S in these algebras allows the construction of canonical formats in various representation schemes.

Example 2. *The CSG format for S is just the structure of S in a finite Boolean algebra $\mathcal{B}$ of* closed regular sets. *To represent this algebra as $\langle \mathcal{A}; \cup, \cap, - \rangle$, we associate the U_d basis elements of the d-chain space C_d to the join-irreducible elements (of dimension d) of the arrangement $\mathcal{A}(\mathcal{H}) = \{A_i\}$, with $A_i \subset \mathbb{E}^d$.*

It is worthwhile to remark that the *B-rep format* of a solid expression, e.g., $A \oplus B = a \in C_d$, may be computed as $[b] = [\partial_d][a]$, the product of the matrix $[\partial_d]$ of boundary operator $\partial_d : C_d \to C_{d-1}$, times the coordinate vector $[a] = (0, 1, 0, 1)^t \in C_d$, given in Table 2. The resulting (d-1)-cycle $b \in C_{d-1}$ (here $d = 2$) may be translated into a geometric data structure, such as `LAR` (DiCarlo et al., 2014).

Example 3. *Consider the simple 2D example in Figure 1, which shows the overlapping 2-cells A and B, their 1-skeleton, and the four 2-cells c_1, c_2, c_3, c_4 of the space partition generated as 2D arrangement.*

Figure 1c shows a partition of $\mathbb{E}^2$ into four irreducible subsets of $\mathbb{E}^2$: the red region A, the green region B, the overlapping region $A \cup B$ and the outer region $\overline{A \cup B}$, i.e., the rest of the plane. The set of atoms of Boolean algebra $\mathcal{B}$ is the same: the colored regions are the four atoms of $\mathcal{B}$ algebra, and there are $2^4 = 16$ distinct elements $S \in \mathcal{B}$. The *structure* of each element $S \in \mathcal{B}$ is a *union* of $\mathcal{B}$ atoms; as a chain in C_2, it is a *sum* of basis elements.

Let consider the columns of Table 1. By columns, the table contains the coordinate representation of the 2-chains A, B, and $\Omega = \overline{A \cup B}$ in chain space C_2. In algebraic-topology language, we have

$$A = c_2 + c_3, \quad B = c_3 + c_4, \quad \Omega = c_1 = -(c_2 + c_3 + c_4).$$

The Boolean algebra of sets is represented here by the power set $\mathcal{A} = \mathcal{P}(U_2)$, whose $16 = 2^4$ binary terms of length $\#U_2 = 4$ are given in Table 2, together with their semantic interpretation in set algebra. Of course, the coordinate vector representing the universal set, i.e. the whole topological space $X := \mathbb{E}^2$ generated by $[\partial_2]$ columns is given by $[X] = (1, 1, 1, 1)^t$, including the exterior cell. Let us remember that the *complement* of A, denoted $-A$ or $\overline{A}$, is defined as $X \setminus A$ and that the *difference* operation $A \setminus B$ is defined as $A \cap \overline{B}$.

Table 1
Truth table associating 2-cells c_i (i=1,...,4) in U_2 (rows) and solid variables A, B and $\Omega = \overline{A \cup B}$ (columns). See Figure 1.

U_2	Ω	A	B
c_1	1	0	0
c_2	0	1	0
c_3	0	1	1
c_4	0	0	1

Table 2
Truth table providing the complete enumeration of elements S of the finite Boolean algebra $\mathcal{A}$ generated by two solid objects A, B and four atoms c_i in the basis U_2 of chain space C_2 associated with the arrangement $\mathcal{A}(S)$, with $S = \{\partial(A), \partial(B)\}$.

U_2	$X = \mathbb{E}^2$	A	B	$\overline{A \cup B}$	$A \cup B$	$A \cap B$	$A \setminus B$	$B \setminus A$	$A \oplus B$	$\overline{A \setminus B}$	$\overline{B}$	$\overline{B \setminus A}$	$\overline{A}$	$\overline{A \oplus B}$	$\overline{A \cap B}$	$\emptyset$
c_1	1	0	0	1	0	0	0	0	0	1	1	1	1	1	1	0
c_2	1	1	0	0	1	0	1	0	1	0	1	1	0	0	1	0
c_3	1	1	1	0	1	1	0	0	0	1	0	1	0	1	0	0
c_4	1	0	1	0	1	0	0	1	1	1	0	0	1	0	1	0

3.4.1. Complete and reduced structure

The $\mathcal{L}$ algebras of the decompositions of $\mathbb{E}^d$ by polynomials in $\mathcal{H}(S)$ are well-defined for semi-algebraic sets, and in particular for piecewise-linear polyhedra. An element S in an algebra $\mathcal{L}$ is an union of "cells", whose type (connected, closed, open, regular, etc.) is determined by the particular algebra. More formally, the *structure* of each element S in a Boolean algebra $\mathcal{L}$ is a unique finite union of atoms A_i.

The *Reduced structure* of an element S of a lattice $\mathcal{L}$ is a unique finite union of the *minimal* set of join-irreducible elements $J_k \in \mathcal{L}$ such that $J_k \subseteq S$. Our current implementation deals with the algebra $\mathcal{A} = \mathcal{P}(U_d)$, with $U_d = \{\mathbf{k}\, u_i\}$, where $\mathbf{k}$ is the topological *closure* operator, applied to all elements of U_d basis.

The *Complete structure* of an element S of a lattice $\mathcal{L}$ is a unique finite union of *all* join-irreducible elements $J_k \in \mathcal{L}$ such that $J_k \subseteq S$, and $J_k \in \mathcal{L}$. A closure algebra (see Section 4.1) will deal with $\mathcal{A} \equiv \mathcal{P}(U)$ where $U = \cup_{p=0}^{d} U_p$.

Example 4. *Consider the set of all simplices in a finite triangulation of a projective space. Every simplex of dimension k contains $k+1$ simplices of dimension $k-1$, and the intersection of any two simplices in the triangulation is either empty or a simplex. It follows that the set of all simplicial complexes in the triangulation is a lattice, and simplices of the triangulation are its join-irreducible elements. The* complete structure *of a simplicial complex is the union of all the simplices it contains. The* reduced structure *of a complex is given by the union of only those simplices that are not faces of another higher-dimensional simplex. Of course, this distinction is combinatorial and discrete, and produces different representations as coordinate vectors in different chain spaces.*

3.5. Canonical representation by chain generators

Every finite Boolean algebra is a distributive lattice $\mathcal{L}$ whose atoms are the join-irreducible elements of $\mathcal{L}$. It would be convenient to characterize lower-dimensional join-irreducible elements of a lattice by intersections between elements and their complements, but the complement operation is not defined in a lattice. Fortunately, we deal only with lattices that are obtained as sublattices of some Boolean algebra. For example, all closed (respectively open) elements of a Boolean algebra $\mathcal{L}$ form a sublattice of $\mathcal{L}$ (Birkoff, 1973). Representations of the join-irreducible elements of such lattices can be derived from representations of the corresponding atoms of the associated Boolean algebra, in turn provided by independent chains in linear chain spaces, described by their (sparse) binary coordinate vectors.

4. Algebras with topological operations

Given a semi-algebraic set S, it is well known that *closure* $\mathbf{k}S$, *boundary* ∂S, and *interior* $\mathbf{i}S$ are also semi-algebraic sets (e.g., Hironaka (1975), Gibson et al. (1976)). We note that algebras defined in Section 3 may not contain these sets for some of their elements. But even if all of them are in the same algebra, there is no way to relate them algebraically

using the standard set operations alone, or even allowing for some connected component operation. In other words, algebras of semi-algebraic sets in Section 3 do not provide a proper setting for many topological operations that are important in geometric modeling (Shapiro, 1991).

4.1. Closure algebras

Since algebras of semi-algebraic sets are not sufficient for defining many topological operations in geometric modeling, we need an "algebra of topology", such that topological properties of elements can be expressed by means of algebraic statements. Such an algebra is defined in Mckinsey and Tarski (1944) as a *closure algebra*. We may represent the atoms of a closure algebra by using the (graded) independent generators of chains spaces associated to an arrangement of $\mathbb{E}^d$ space. Closed and open elements of a Boolean algebra form a (sub)lattice. With respect to a closure algebra, such a lattice is called a *Brouwerian algebra* (resp. lattice) (Mckinsey and Tarski, 1946). Every Brouwerian algebra contains a Boolean algebra of regular elements.

Definition 1. *A set of elements $\mathcal{K}$ is a* closure algebra *with respect to the operations $+, \cdot, -$, and closure $\mathbf{k}$, if is closed under the closure operation or, in other words:*

1. *$\mathcal{K}$ is a Boolean algebra with respect to abstract operations $+, \cdot$, and $-$;*
2. *if $S \in \mathcal{K}$, then $\mathbf{k}S \in \mathcal{K}$.*

As a consequence of this definition, the family of all subsets of any topological space is a *closure algebra*. This statement also applies to the set of all semi-algebraic subsets of $\mathbb{E}^d$. Here, we consider closure algebras of piecewise-linear sets generated by a finite set of primitives $\mathcal{H}(S)$. In our current implementation, we deal only with closure algebras. With the addition of the closure **k** operation all the important topological operations can be derived in a closure algebra:

1. **interior :** $\mathbf{i}S = -\mathbf{k}(-S)$;
2. **exterior :** $\mathbf{e}S = -\mathbf{k}S$;
3. **boundary :** $\partial S = \mathbf{k}(S) \cap \mathbf{k}(-S)$;
4. **regularization :** $\mathbf{r}S = \mathbf{ki}S$.

4.2. Brouwerian algebras

The discussion in this section is based on Mckinsey and Tarski (1946), and applies to abstract algebras with operations $+, \cdot, -$, and **k**. We already know that all closed elements and all open elements of a Boolean algebra form distributive lattices. As sublattices of a closure algebra, these lattices are also known as *Brouwerian lattices*, or *Brouwerian algebras* (Mckinsey and Tarski, 1946), and have many additional properties. In our applications of geometric modeling in the piecewise-linear domain, we are going to implement such algebras in $\mathbb{E}^d$ simply as the set of all subsets $\mathcal{P}(U)$ of the graded chain base $U = \cup_{i=0}^{d} U_i$.

4.3. Point representation of join-irreducible elements

The following proposition suggests a natural representation as single points for join-irreducible elements, which is convenient for computational purposes. Such a representation is the foundation of the geometric evaluation algorithm introduced in this manuscript. The aim is to reduce an algebraic formula $S = |\Phi(\mathcal{H})|$ to the form of the coordinate representation of its solution, i.e.: $S \in \mathcal{A}(\mathcal{H}) \mapsto [S] \in C_d$, as a vector in chain space. This binary vector will be finally mapped to the canonical format of a particular representation scheme, normally a B-rep.

Proposition 1. *Let $\{A_j\}$ be the set of join-irreducible elements of a finite algebra $\mathcal{L}(\mathcal{H})$. Every join-irreducible element $A_j \in \mathcal{L}(\mathcal{H})$ can be represented by a single interior point $p \in A_j$.*

The assertion is easy to prove, since by construction the irreducible elements of the finite algebra are mapped to the independent chains of the $\mathbb{E}^d$ partition, and these are the minimal building blocks of the partition induced by the input. Hence, a single point p in the (relative) interior of $A_i \in U$ belongs to one and only one atom of the algebra. Remember that generators are disjoint or quasi-disjoint (cells are either open or closed). In case of closure algebras, the generators of lower dimension are disjoint or quasi-disjoint from the cells of the same dimension. For our purposes, the mapping of atoms to single interior points provides a simple powerful computational tool. In Section 5.2.2 it will be used to translate the structure of terms in a solid formula into the structure of terms of a set algebra.

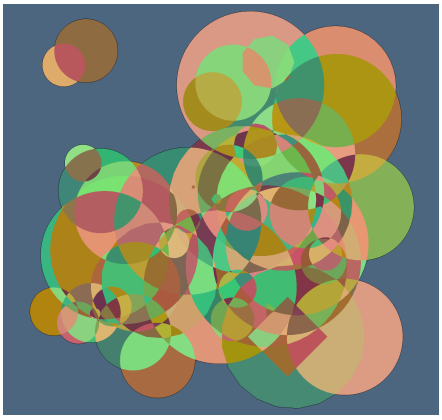


Figure 2: The join-irreducible atoms of the $\mathbb{E}^2$ partition generated by random polygonal approximations of the 2D circle.

5. Algorithms

Let us remark that a chain complex gives a *complete* representation of the topology of any space. Our method for computing solid Boolean Algebras generated by a collection of solid models is sequentially subdivided into two main steps: (a) the computation of the *chain complex* of the partition of $\mathbb{E}^d$ induced by the input; (b) the assessment of a *finite algebra of sets* generated by such partition. The mapping of terms to their coordinate representation in chain space allows for native binary resolution of every Boolean (CSG) expression between the input terms.

In order to evaluate the small scripts in this and the following sections, the reader should download the open-source `LAR` package from `https://github.com/cvdlab/LinearAlgebraicRepresentation.jl`. The sources of the examples discussed in this paper may be found in `./examples/booleanpaper/`.

5.1. Chain complex of space partition

In this section, we summarize the computational process for discovering the topology that fully characterizes the space partition generated by a collection of geometric complexes. A detailed presentation and discussion, including pseudocodes of algorithms, can be found in (Paoluzzi et al., 2017b). The input to this computational pipeline is a collection S_{d-1} of geometric shapes, given as cell complexes. The output is the *chain complex* $C_\bullet = (C_p, \partial_p)$ $(0 \leq p \leq d)$ associated with the $\mathcal{A}(S_{d-1})$ arrangement, i.e., the topology of the $\mathbb{E}^d$ partition. Specifically, the sparse matrices of the unknown graded operator $\partial := (\partial_d, \cdots, \partial_1)$ are computed. For the sake of clarity, we assume in the following that $d = 3$. The same pipeline works for $d = 2$, and is actually used for the 2D steps of the 3D pipeline.

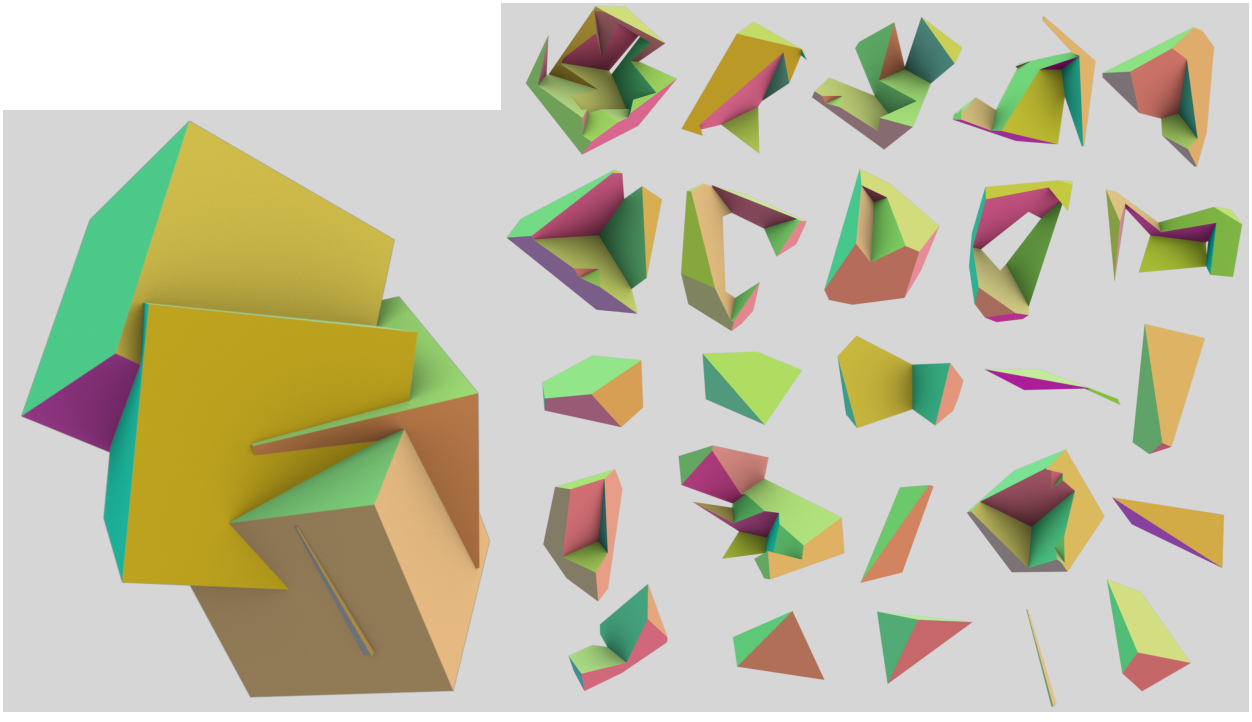
5.1.1. 2-cell partition

Each 2-cell σ in the input S_2 is *independently* intersected with the others, producing its own chain complex $C_\bullet(\sigma)$ of $\mathbb{E}^2$ space. Each set $\mathcal{I}(\sigma)$, of 2-cells of possible intersection with σ, is efficiently computed by combinatorial intersection of query results on d different (one for each coordinate) interval-trees (Preparata and Shamos, 1985). Every $\mathcal{I}(\sigma)$ set, for $\sigma \in S_2$, is affinely mapped in $\mathbb{E}^3$, leading σ to the $z = 0$ subspace. The arrangement $\mathcal{A}(\sigma)$ is firstly computed in $\mathbb{E}^2$, and then mapped back into $\mathbb{E}^3$. The actual intersection is computed between line segments in 2D, retaining only the maximal 2-connected subgraph of results, so obtaining a regular 2-complex.

To compute the 2-cells as 1-cycles and the corresponding $[\partial_2]$ matrices, the *Topological Gift Wrapping* (TGW) algorithm (Paoluzzi et al., 2017b) is applied in 2D. This computation is executed *independently* for each 2-cell σ in the input cellular complexes, by local intersection of piecewise-linear polynomials $\mathcal{H}(S_{d-1})$. In our current prototype implementation, it is currently computed through Julia's `channels`, able to take advantage of both local and remote compute nodes, making use of an *embarrassingly parallel* data-driven approach.

5.1.2. 2-skeleton in 3D

The set of $C_\bullet(\sigma)$ complexes $(\sigma \in S_{d-1})$ generated in $\mathbb{E}^2$ is properly embedded in $\mathbb{E}^3$ by inverse affine transformations. Here all the geometric congruences between 0-, 1-, and 2- fragmented cells are discovered, and are used to represent each equivalence class of congruent (sub)cells with a single member. Take into consideration that two



(a) (b)

Figure 3: (a) A collection $\mathcal{S}$ of five random cubes in $\mathbb{E}^3$; (b) the display of 3-cells of the generated $\mathbb{E}^3$ arrangement $\mathcal{A}(\mathcal{S})$. 3-cells are here not in scale, and are suitably rotated to better exhibit their complex structure. Their assembly gives the union of the five cubes. Each *3-cell is generated* as *a column* of the *sparse matrix* of *chain map* $\partial_3 : C_3 \to C_2$, with values in $\{-1, 0, 1\}$. They are the join-irreducible *atoms* of the CSG algebra with closed regular cells.

geometries are *congruent* when they differ by a rigid transformation, and that congruence is an *equivalence* relation. Congruent 0-cells (vertices) are efficiently discovered using kd-trees (Bentley, 1975). Once congruent vertices have been identified via a suitable step of round-off, congruent 1-cells and 2-cells are detected simply by sorting their canonical representations (subsets of ordered indices of vertices), in order to eliminate duplicates. The results of this stage are the signed sparse matrices $[\partial_1] = [\delta_0]^t$ and $[\partial_2] = [\delta_1]^t$ of chain complex of the $\mathbb{E}^3$ partition.

5.1.3. 3-boundary operator

The TGW algorithm is applied in 3D to the sparse matrix $[\partial_2] : C_2 \to C_1$. The unknown sparse matrix $[\partial_3] : C_3 \to C_2$ codifying the unknown 3-cells in U_3 (see Figure 3b for a visual example) is built in this stage.

Let us remember that the matrix of a linear operator from a linear space C_3 to a linear space C_2 contains by column the coordinates (drawn from $\{-1, 0, 1\}$) of the elements of the U_3 basis, written as linear combinations of the U_2 basis. In other words, by column in $[\partial_3]$ we build the C_2 representation of the maximal set of independent generators for C_3, written as signed 2-cycles, i.e., as closed 2-chains of oriented 2-cells.

TGW works by iteratively building (one by one) the independent 2-cycles, using ∂_2 and δ_1 for the stepwise constructions of (partial) 2-cycles and their boundaries, until to get them closed. During the construction of linearly independent 3-cells as 2-cycles, TGW satisfies the property that, in a (d-1)-manifold, the (d-2)-cells are shared by at most *two* (d-1)-cells. Actually, the TGW algorithm produces the $[\partial_3^+]$ matrix, augmented by the column of the *outer cell*, which is a linear combination of the others. The generation of 2-cycles gives a partition of $\mathbb{E}^3$ into a set of manifold spaces with closed boundaries. The possible presence of holes in the solid input terms must be taken into account, by removing their columns from the base, and adding them either to the exterior cell or to their enclosing cell, whose boundary become disconnected (Paoluzzi et al., 2017b).

1: **procedure** GENERATE.AND.TEST(p_k, $\mathcal{I}(p_k)$)(X) ▷ Set Membership Computations
2: structure(X) := $\emptyset$; $[X]$:= [0] ▷ Initialization of term X in algebra of sets $\mathcal{A}(S) \cong \mathcal{B}$
3: **for all** join-irreducible $A_i \in \mathcal{I}(p_k)$ **do**
4: **if** SetMembershipClassification(p_k, A_i) **then** ▷ if and only if $p_k \in \mathbf{i}A_i$
5: structure(X) := structure(X) $\cup$ A_i; $[X][i]$:= 1 ▷ put 1 on i-th element of array $[X]$
6: **end if**
7: **end for**
8: **end procedure** ▷ Return the binary array $[X]$, coordinate vector for $X \in C_d$

Figure 4: Set Membership Classification algorithm implemented through a point-polyhedron-containment test.

5.2. Chains as atoms of Boolean Algebras

The second step of our approach to constructive solid geometry consists in generating a representation of solid arguments as linear combinations of independent 3-chains, i.e., of 3-cells of the space partition generated by the input. The U_3 basis of 3-chains is represented by the columns of the ∂_3 matrix. This set of 2-cycles can be seen as a collection of point-sets, including the whole space and the empty set. In this sense, it generates both a discrete topology of $\mathbb{E}^3$ and, via the Stone Representation Theorem (Stone, 1936), a finite Boolean algebra $\mathcal{B}$ over C_3 elements.

This second stage of the evaluation process of a functional form including solid models and set operators is much simpler then the first stage, consisting in executing a series of set-membership-classification (SMC) tests, computable in parallel, to build a representation of the input terms into the set algebra $\mathcal{B}$, using sparse arrays of bits.

5.2.1. Algebra of sets

Any finite collection of sets closed under set-theoretic operations forms a Boolean algebra, i.e. a complemented distributive lattice, the *join* and *meet* operators being union and intersection, and the *complement* operator being set complement. The bottom element is $\emptyset$ and the top element is the universal set under consideration. By Stone (1936), every Boolean algebra of N atoms—and hence the Boolean algebra of solid objects closed under regularized union and intersection—is isomorphic to the Boolean algebra of sets over $\{0,1\}^N$. Here, we look for the binary representation of solid objects terms in a solid geometry formula, i.e., for the coefficients of their components in the C_d basis. If all chains in the basis U_d are equioriented, then such coefficients are drawn from $\{0,1\}$, and every coordinate vector for C_d elements is a binary sequence in $\{0,1\}^N$, with $N = \#U_d$. In Julia we compute this representation of algebraic terms using arrays of `BitArray` type, or sparse arrays of type `Int8`, consuming few bytes per non-zero element.

5.2.2. Generate-and-test algorithm

The representation of join-irreducible elements of $\mathcal{L}(S) \cong \mathcal{A}(S_{d-1})$ as discrete point-sets (see Section 4.3), is used to map the *structure* of each term $X \in \mathcal{L}(S)$ to the *set algebra* $\mathcal{B} \cong \mathcal{A}(S)$. Assume that: (a) $X \in \mathcal{L}(S)$; (b) a partition of $\mathbb{E}^d$ into join-irreducible subsets $A_k \in \mathcal{L}(S)$ is known; (c) a one-to-one mapping $A_k \mapsto p_k$ between atoms and internal points is given; (d) a SMC oracle, i.e., a set-membership classification (Tilove, 1980) test is available.

A naive approach to SMC, where each single point p_k (in the interior of an atom $A_k \in \mathcal{L}(\mathcal{H})$) is tested against *all* input solid terms $X \in \mathcal{H}(S)$ may be computed in quadratic time $O(NM)$, where N is the number of atoms, and M is the number of input solid terms X. An efficient $O(N \log M)$ procedure is established here by using two (i.e., $d-1$) one-dimensional interval-trees for the decomposition $\{A_k\}$ of $\mathbb{E}^3$, in order to execute the SMC test only against the terms in the subset $\mathcal{I}(p_k) \subseteq U_3$, whose containment boxes intersect a ray from the test point.

Therefore, the structure of term X in the finite algebra $\mathcal{L}(S)$ can be computed using the *generate-and-test* procedure in Figure 4. Such a SMC test is simple and does not involve the resolution of "on-on ambiguities" Tilove (1980), because of the choice of an *internal* point p_k in each algebra atom A_k.

In our current implementation in 3D the SMC test is executed by intersecting a ray from p_k with the planes containing the 2-cells of atoms in $\mathcal{I}(p_k)$, and testing for point-polygon-containment in these planes (via maps to the $z = 0$ subspace). In summary, we decompose $\mathbb{E}^d$ into join-irreducible elements A_k of algebra $\mathcal{L}(S)$, and represent each A_k with a point p_k. By the Jordan curve theorem, an odd intersection number of the ray for p_k with boundary 2-cells of X produces an oracle answer about the query statement $p_k \in X$, and hence $A_k \subseteq X$. An even number gives the converse.

5.2.3. Binary representation of Boolean terms

We contruct a representation of each term X of a solid Boolean expression, as a subset of U_3 (basis of 3-chains). Remember that, by construction, U_3 partitions both $\mathbb{E}^3$ and the input solid objects. To translate a solid algebraic formula to machine language it is sufficient, for each 3-cell $u_k \in U_d$, to test for SMC a single internal point $p_k \in u_k$ by checking if $p_k \in X$. In the affirmative case the k-th bit of coordinate vector $[X] \in \{0,1\}^N \cong \mathcal{P}(U_d)$ is set to `true`.

Example 5. Space arrangement from `assembly` tree. *Let us consider the* `assembly` *constructed by putting together three instances of a unit cube, suitably rotated and translated. The semantics of* `Lar.Struct()` *is similar to that of PHIGS structures (Kasper and Arns, 1993; Paoluzzi, 2003):*

```
julia> m,n,p = 1,1,1;
    Lar = LinearAlgebraicRepresentation;
    V,(VV,EV,FV,CV) = Lar.cuboidGrid([m,n,p],true);
    cube = V,FV,EV;
julia> assembly = Lar.Struct([ cube,
    Lar.t(.3,.4,.25), Lar.r(pi/5,0,0), Lar.r(0,0,pi/12), cube,
    Lar.t(-.2,.4,-.2), Lar.r(0,pi/5,0), Lar.r(0,pi/12,0), cube ]);
julia> W, EV, FE, CF, boolmatrix = Lar.bool3d(assembly);
```

The application of function `Lar.bool3d()` *to* `assembly` *returns the* `(geometry,topology)` *of 3D space partition generated by it. Here* `geometry` *is given by the embedding matrix* `W` *of vertices (0-cells), and* `topology` *is given by the three sparse matrices* `CF, FE, EV`*, i.e.,* $\delta_3, \delta_2, \delta_1$*, of chain complex describing the* $\mathcal{A}($`assembly`$)$ *arrangement.*

Example 6. Boolean matrix. *The array value of type* `Bool` *returned in the variable* `boolmatrix` *contains by column the results of efficient point-solid containment tests (SMC) for atomic 3-cells (rows), w.r.t. the terms of* $\mathbb{E}^3$ *partition: the outer 3-cell* Ω *and each* `cube` *(columns).*

```
julia> Matrix(boolmatrix)
8x4 Array{Bool,2}:
 true  false  false  false
false  false  false   true
false   true   true  false
false   true   true   true
false   true  false  false
false  false   true  false
false   true  false   true
false  false   true   true
```

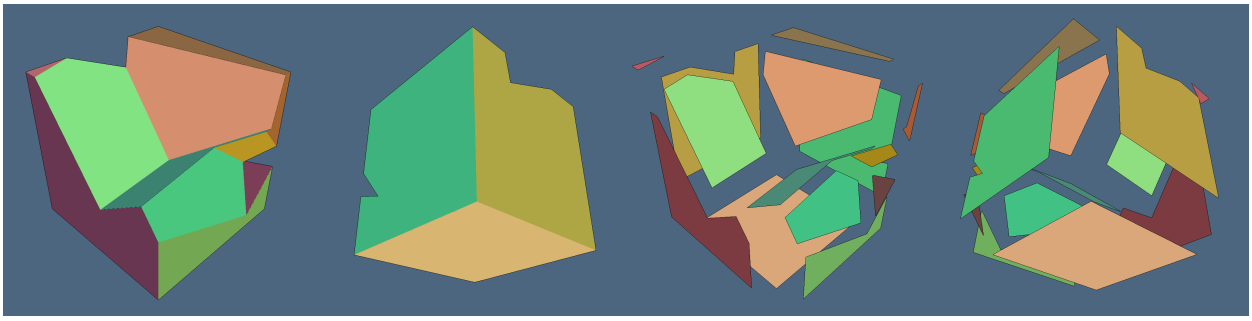
Our variables Ω, A, B, C *are extracted from* `boolmatrix` *columns, so describing how each one is partitioned by ordered 3-cells in* U_3*. The whole space is* $X = \Omega \cup A \cup B \cup C$*:*

```
julia> A,B,C = boolmatrix[:,2],boolmatrix[:,3],boolmatrix[:,4]
(Bool[false, false, true, true, true, false, true, false],
 Bool[false, false, true, true, false, true, false, true],
 Bool[false, true, false, true, false, false, true, true])
```

5.2.4. Bitwise resolution of set algebra expressions

When the input values of solid variables have been mapped to arrays of Boolean values, any expression of their finite Boolean algebra can be evaluated by using logical operators, that operate by comparing corresponding bits of variables. In particular, the Julia language offers bitwise logical operators *and* (`&`), *or* (`|`), *xor* ($\veebar$), and *complement* (`!`), as well as a dot mechanism for applying elementwise any function to arrays. Hence, we can write expressions like `(A .& B)` or `(A .| B)` that are bitwise evaluated and return the result in a new `BitArray` vector. We remark that these operators can be used also in *prefix* and *variadic* form. Hence, bitwise operators can be applied at the same time to any finite number of variable elements.

Example 7. Boolean formulas. *Some simple examples follow. The last expression is the intersection of the first term* A *with complement of union of others terms* $B \cup C$*, with* A, B, C *of Example 6, giving the set difference* $(A \setminus B) \setminus C$*.* `AminBminC` *is the variable holding the result of the set difference denoted* $\&(A, \overline{B}, \overline{C})$*, and then mapped into the model in Figure 5.*



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

Figure 5: Boolean difference $(A \setminus B) \setminus C$ of three cubes, with 2-cells in different colors: (a) view from the front; (b) view from the back; (c) front with exploded 2-cells; (d) back with exploded 2-cells. Note that 2-cells of the resulting boundary may be non-convex.

```
julia> AorB = A .| B;
julia> AandB = A .& B;
julia> AxorB = A .? B;
julia> AorBorC = A .| B .| C;
julia> AorBorC = .|(A, B, C);
julia> AandBandC = A .& B .& C;
julia> AandBandC = .&(A, B, C);

julia> AminBminC = .&(A, .!B, .!C)
8-element BitArray{1}:
[false, false, false, false, true, false, false, false]
```

5.2.5. Boundary computation

In most cases, the target geometric computational environment is able to display—more in general to handle—a solid model only by using some boundary representation, typically a triangulation. It is easy to get such a representation by multiplying the matrix of 3-boundary operator $\partial_3 : C_3 \to C_2$ times the coordinate vector in C_3 space of the solid expression, which is given by a binary term of our set algebra.

It is wortwhile to remark that, in order to display a triangulation of boundary faces in their proper position in space, the whole information required is contained in the *geometric chain complex* (GCC):

$$\mu : C_0 \to \mathbb{E}^3,\ (\delta_0, \delta_1, \delta_2) \qquad \equiv \qquad \texttt{(geometry, topology) = (W, (EV, FE, CF))}$$

A GCC allows to transform the (possibly non connected) boundary 2-cycle of a Boolean result (see the example below) into a complete B-rep of the solid result. Note that ordered pairs of letters from `V,E,F,C`, correspond to *Vertices*→*Edges*→*Faces*→*Cells* into the *Column*→*Row* order of matrix maps of operators.

Example 8. *Let us remark that* `CF` *(i.e.,* `Faces` → `Cells`*) is the sparse matrix of coboundary operator* $\delta_2 : C_2 \to C_3$*, so that we have* $[\partial_3] = [\delta_2]^t$*, which in Julia is* `CF'`*. Note that the value of* `AminBminC = .&(A,.!B,.!C)` *given in Example 9, is the* C_2 *representation of the oriented boundary 2-cycle of solid difference, displayed in Figure 5.*

Of course, the 14 non-zero elements in the `boundary` array (given below) of the `AminBminC` variable, correspond to the oriented boundary 2-cells of the solid result (see Example 9). Each of them is transformed into a possibly non connected 1-cycle by the `FE'` sparse matrix, i.e., by the 2-boundary matrix $[\partial_2] = [\delta_1]^t$. Finally, every 1-cycle is transformed into one or more cyclic sequences of 0-cells, using the `EV'` matrix, i.e., by using $[\partial_1] = [\delta_0]^t$. The indices of 0-cells are cyclically ordered, and used to generate sequences of 3D points via the embedding matrix `W`, which provides the vertex coordinates by column. This last step gives the ordered input for face triangulation using a CDT (Constrained Delaunay Triangulation) algorithm (Shewchuk, 2002) in 2D.

Example 9. Boundary of solid expression. *The variable* `AminBminC` *contains the logical representation of the Boolean expression* $(A \setminus B) \setminus C$*. It is converted to a binary array of type* `Int8` *by vectorized constructor* "`Int8.`"*, in order to compute the* `AminBminC` *boundary (see Figure 5) via multiplication times* $[\partial_3] \equiv$ `CF'`*:*

```
julia> difference = Int8.(AminBminC)
8-element Array{Int8,1}:
[0,  0,  0,  0,  1,  0,  0,  0]

julia> boundary = CF' * difference
47-element Array{Int8,1}:
[1, 0, -1, 0, 0, 0, -1, 0, 1, 0, 1, 0, 0, 0, 1, -1, 0, -1, 0, -1, 0, 0, 1, 0, 0, 0, -1,
0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 1, 0, 0, 0, 0, -1, 0, 0, 0, 0]
```

Finally, we note that in chain notation it is possible to write the following expression for the oriented boundary of the ternary solid Boolean difference. The actual reduction to triangulated boundary is obtained using `FE'` and `EV'` sparse matrices, and a constrained Delaunay triangulation (CDT) algorithm.

$$\texttt{boundary} \mapsto f_{A\setminus B\setminus C} = f_1 - f_3 - f_7 + f_9 + f_{11} + f_{15} - f_{16} - f_{18} - f_{20} + f_{23} - f_{27} - f_{33} + f_{38} - f_{43}$$

6. Computational examples

This section provides the reader with several examples of *solid modeling programming style* with Julia and its sparse and non-sparse arrays. We think that taking a look at a few simple concrete examples of this style is necessary for a good understanding of our computational approach and its possible developments. Sections 6.1 and 6.2 aim at showing the sequence of spaces and transformations that finally produce a solid model when evaluating a Boolean expression through a function application `Lar.bool3d(assembly)`. We show in Section 6.3 that the exactness property of the chain complex can be used to check the accuracy of calculations. The *Euler characteristic* of the `union` solid model generated in Example 5 is stepwise computed in Section 6.4, where we use the chain maps $\partial_3, \partial_2, \partial_1$ to obtain the sets (and numbers) of *faces*, *edges* and *vertices* belonging to the boundary of the `union` solid, which is a closed 2-manifold of genus zero (see Figure 7). A simple API and a mini DSL (Domain Specific Language) for CSG expressions are being developed.

6.1. Variadic union

In order to compute the `union` of three affinely transformed instances of the unit `cube`, we consider the `assembly` expression given in Example 5. First we get the $\mathbb{E}^3$ space partition generated by `assembly` through the function `Lar.bool3d`; then we combine the logical arrays `A`, `B`, and `C`, building the value of `BitArray` type for the `union` variable, that stores the logical representation of the specific 3-chain. Let us remark that the bitwise "`or`" operator ("|") is applied in a vectorized way to arrays, by inserting a dot character:

```
julia> W, (EV, FE, CF), boolmatrix = Lar.bool3d(assembly);
julia> A,B,C = boolmatrix[:,2],boolmatrix[:,3],boolmatrix[:,4]
julia> union = .|(A, B, C);
julia> @show union;
union = Bool[false, true, true, true, true, true, true, true]
```

Finally, the boundary 2-cycle `faces` is generated by multiplication of the sparse matrix $[\partial_3]$ (i.e., `CF'`) times the binary converted `union`. The mapping `Bool` $\rightarrow \{0, 1\}$ is applied via a vectorized application of the `Int8` constructor:

```
julia> faces = CF' * Int8.(union);
julia> @show faces;
faces = [1, 0, -1, 0, 0, 0, -1, 0, 1, 0, 1, 0, 0, 0, 1, -1, 0, -1, 0, 0, 1, -1, 0, -1, 1,
0, 0, 0, 1, 1, 0, -1, 0, 0, 1, 0, -1, 0, 0, 0, -1, 1, 0, 1, 0, 0, -1]
```

With $f_k \in U_2$, where U_2 is the basis of chain space C_2 generated by $\mathcal{A}$ (`assembly`), we may write in chain notation:

$$\begin{aligned}\texttt{faces} \mapsto f_{A\cup B\cup C} = {} & f_1 - f_3 - f_7 + f_9 + f_{11} + f_{15} - f_{16} - f_{18} + f_{21} - f_{22} - f_{24} \\ & + f_{25} + f_{29} + f_{30} - f_{32} + f_{35} - f_{37} - f_{41} + f_{42} + f_{44} - f_{47}\end{aligned} \tag{1}$$

6.2. Meaning of ∂ matrices

By definition, the matrix $[\partial_2]$ = `FE'` contains by columns the U_2 basis expressed as an ordered sequence of 1-cycle vectors. Signed values provide circular orderings of edge nodes, extendable to higher dimensions. With $e_h \in U_1$, we have:

$$\begin{aligned}
\texttt{FE'[:,1]} &\mapsto f_1 = e_1 - e_2 + e_4 - e_5 + e_6 - e_7 + e_8 \\
\texttt{FE'[:,2]} &\mapsto f_2 = e_3 + e_7 - e_8 \\
&\ldots \quad \ldots \quad \ldots \\
\texttt{FE'[:,47]} &\mapsto f_{47} = e_{64} - e_{69} + e_{77} - e_{81} - e_{86} + e_{88}
\end{aligned}$$

Of course, each 1-cell e_h is mapped to an ordered pair of 0-cells $v_j \in U_0$, via the boundary matrix $[\partial_1] = [\delta_0]^t$ = `EV'`:

$$\begin{aligned}
\texttt{EV'[:,1]} &\mapsto e_1 = v_2 - v_1 \\
\texttt{EV'[:,2]} &\mapsto e_2 = v_6 - v_3 \\
&\ldots \quad \ldots \quad \ldots \\
\texttt{EV'[:,49]} &\mapsto e_{49} = v_{29} - v_{26}
\end{aligned}$$

The geometric embedding in $\mathbb{E}^3$ of the $A \cup B \cup C$ model generated in Section 6.1 is provided by the coordinate array `W` $\in \mathbb{R}^3_{49}$, with 3 rows and 43 columns. The coordinate array `V` embedding the initial `assembly` model, consisting of three non (yet) intersected cubes, has instead dimension 3×24.

6.3. Correctness checks

A computational approach based on chain complexes offers unique tools for checking the accuracy of calculations, that are correct by construction, since both $\partial^2 = 0$ and $\delta^2 = 0$ hold, when applied to any chain. In words, *every boundary is a cycle,* or equivalently: the *chain complex is exact*. Also, we know that the boundary of every solid is a possibly non connected closed surface, hence a 2-cycle. This holds if and only if the construction of $[\partial]$ matrices is done correctly. In the following example, we start from the chain of boundary `faces` of Section 6.1.

```
julia> pairs = [(f,sign) for (f,sign) in enumerate(copCF' * Int8.(union)) if sign != 0];
julia> faces = map(prod, pairs);
julia> @show faces;
faces = [1,-3,-7,9,11,15,-16,-18,21,-22,-24,25,29,30,-32,35,-37,-41,42,44,-47]
```

We obtain the following 2-chain as boundary 2-cycle. The cardinality of the `faces` array provides $\chi_2 = 21$.

$$\texttt{faces} \mapsto f_{A\cup B\cup C} = f_1 - f_3 - f_7 + f_9 + f_{11} + f_{15} - f_{16} - f_{18} + f_{21} - f_{22} - f_{24} + f_{25} + f_{29} + f_{30} - f_{32} + f_{35} - f_{37} - f_{41} + f_{42} + f_{44} - f_{47}$$

The edge subset on the boundary of the `union` solid is computed by: (a) transforming the `faces` array of signed indices of 2-cells into the `COORD` representation (`rows,cols,vals`) of a sparse matrix (Cimrman, 2015); (b) holding a single boundary face (2-cell) per column in `facemat`; (c) multiplying this matrix times the $[\partial_2]$ operator, thus obtaining the new sparse matrix `edges4face`, holding a face 1-cycle per column; and, finally, (d) extracting only the positive instance of boundary edges belonging to the boundary faces. The number of boundary edges $\chi_1 = 57$ is one half of the non-zero terms in the sparse vector `edges4face`. The other half has the opposite sign, so that the total sum is zero:

```
julia> nonzeros = hcat([[abs.(face),k,sign(face)] for (k,face) in enumerate(faces)]...);
julia> facemat = sparse([nonzeros[k,:] for k=1:size(nonzeros,1)]...);
julia> edges4face = copFE' * facemat;
julia> rows,cols,vals = findnz(edges4face);

julia> edges = [e*sign for (e,sign) in zip(rows,vals) if sign==1];
julia> @show edges;
edges = [1,4,6,8,10,14,18,20,9,23,26,27,2,34,30,36,38,5,29,24,43,15,42,28,47,51,53,54,21,52,57,
48,61,62,64,50,59,66,55,58,40,63,68,77,79,80,35,78,83,72,88,7,74,87,69,81,86]
```

We remark again that, without filtering out the terms of negative `sign`, we would get an `edges` array of signed indices summing to zero, according to the constraint $\partial^2 = 0$. This attests to the exactness of calculations.

6.4. Euler characteristic

The Euler characteristic of a solid of genus g in $\mathbb{E}^3$ is defined as

$$\chi(g) = \chi_0 - \chi_1 + \chi_2 = 2 - 2g,$$

where χ_0, χ_1, χ_2 are, respectively, equal to the number of vertices, edges and faces on the boundary of the solid. In our case, the number of boundary faces is $\chi_2 = 21$, according to Eq. (1). The `edges` indices given in Section 6.3 determine the 1-chain $e_{A\cup B\cup C}$ (or, more precisely, the non-independent 1-cycle generated by the independent 2-cycles of boundary faces). The cardinality of the `edges` array provides $\chi_1 = 57$. Note that, in order to counting the edges, we consider only the positive instances of 1-cells, since they appear in pairs (positive and negative) in a closed and coherently oriented cellular 2-complex.

$$\begin{aligned}\texttt{edges} \mapsto e_{A\cup B\cup C} = {} & e_1 + e_4 + e_6 + e_8 + e_{10} + e_{14} + e_{18} + e_{20} + e_9 + e_{23} + e_{26} + e_{27} + e_2 + e_{34} + e_{30} + e_{36} + e_{38} + e_5 + e_{29} + e_{24} \\ & + e_{43} + e_{15} + e_{42} + e_{28} + e_{47} + e_{51} + e_{53} + e_{54} + e_{21} + e_{52} + e_{57} + e_{48} + e_{61} + e_{62} + e_{64} + e_{50} + e_{59} + e_{66} + e_{55} \\ & + e_{68} + e_{77} + e_{79} + e_{58} + e_{40} + e_{63} + e_{80} + e_{35} + e_{78} + e_{83} + e_{72} + e_{88} + e_7 + e_{74} + e_{87} + e_{69} + e_{81} + e_{86}\end{aligned} \tag{2}$$

The final script computes the 0-chain $v_{A\cup B\cup C}$ of vertices on the boundary of `union` solid, with cardinality $\chi_0 = 38$.

```
julia> nonzeros = hcat([[abs(e),k,sign(e)] for (k,e) in enumerate(edges)]...);
julia> edgemat = sparse([nonzeros[k,:] for k=1:size(nonzeros,1)]...);
julia> verts = sort(collect(Set(findnz(copEV' * edgemat)[1])));
julia> @show verts;
verts = [1,2,3,4,5,6,7,8,9,10,12,15,17,18,19,20,21,24,25,26,27,30,31,32,33,34,35,36,37,38,39,41,
44,45,46,47,48,49]
```

$$\begin{aligned}\texttt{verts} \mapsto v_{A\cup B\cup C} = {} & v_1 + v_2 + v_3 + v_4 + v_5 + v_6 + v_7 + v_8 + v_9 + v_{10} + v_{12} + v_{15} + v_{17} + v_{18} + v_{19} + v_{20} + v_{21} + v_{24} + v_{25} + v_{26} \\ & + v_{27} + v_{30} + v_{31} + v_{32} + v_{33} + v_{34} + v_{35} + v_{36} + v_{37} + v_{38} + v_{39} + v_{41} + v_{44} + v_{45} + v_{46} + v_{47} + v_{48} + v_{49}\end{aligned} \tag{3}$$

The generated `union` solid model has topological genus $g = 0$ (see Figure 7). Therefore, the test of correctness provided by checking the Euler characteristic via Eqs. (1), (2), and (3), gives the correct answer:

$$\chi(\texttt{union}) = \chi_0 - \chi_1 + \chi_2 = 38 - 57 + 21 = 2$$

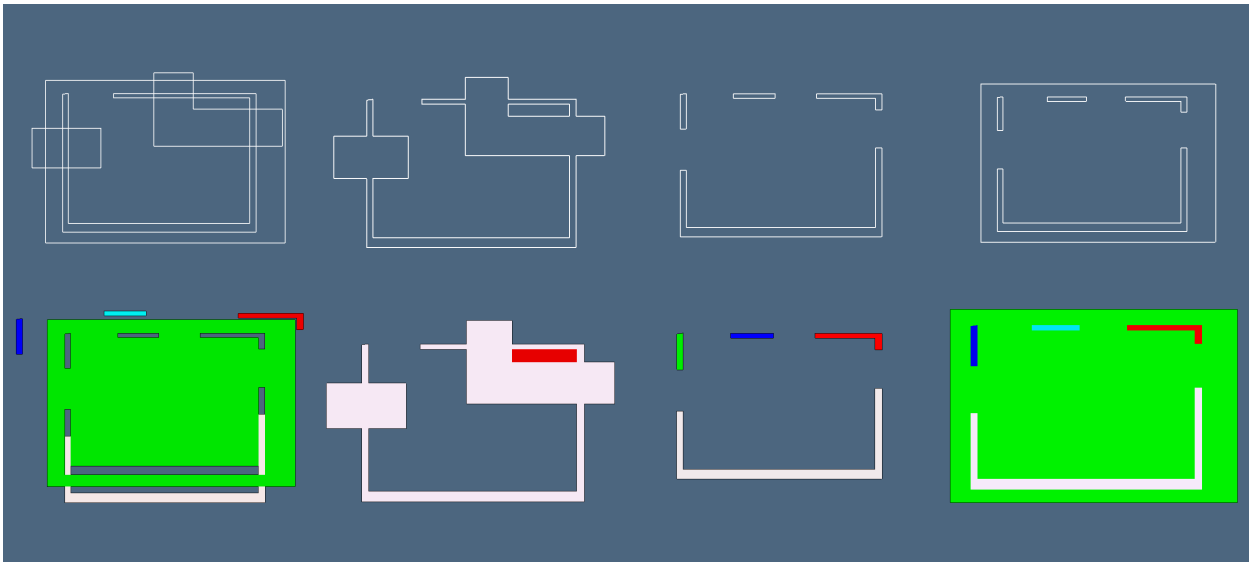
7. Results

The approach put forth in this paper is novel in several ways. First of all, it does not use traditional methods of computational geometry—in particular, of the solid modeling subfield—which are based on specialized data structures, that are often very complex and require the implementation of sophisticated algorithms.

We use instead basic tools and methods of linear algebra and algebraic topology, i.e., (sparse) matrices of linear operators and matrix multiplication, plus filtering. Hence, accuracy of topological computations is guaranteed by contruction, since operator matrices and chain bases, as well as any chain, satisfy the (graded) constraints $\partial^2 = 0$ and $\delta^2 = 0$, which are easy to check. Furthermore, we have shown that space and time complexities are comparable with those of traditional methods, and it is reasonable to expect that our approach might be extended to generic dimensions and/or implemented on highly parallel computational engines, using standard computing kernels. In this last case a good speed-up is expected, given the high level of possible parallelization of our algorithms.

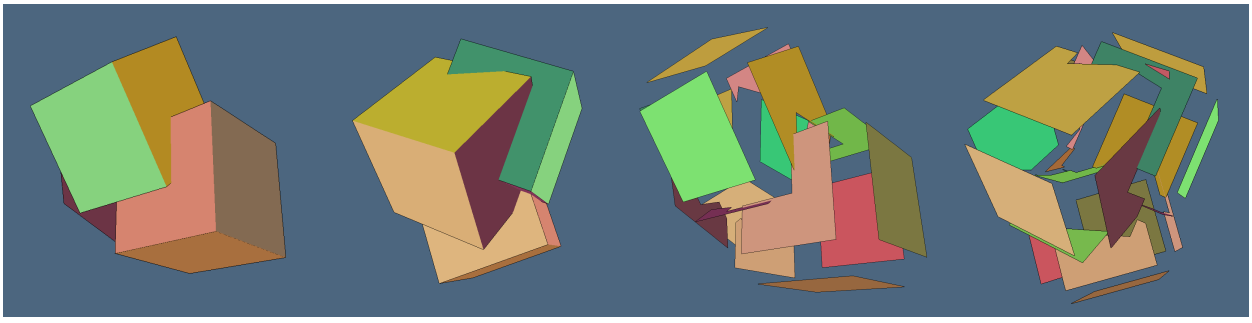
In this paper we have also given a new characterization of the computational process for evaluating CSG expressions of any depth and complexity. Traditional evaluation methods require a post-order DFS traversal of the expression tree, and the sequential computation of each Boolean operation encountered on nodes, until the root operation is evaluated. It is well known that such a process lacks in robustness and accumulates numerical errors, that ultimately modify the local topology and make the applications stop in error. Hence, intermediate boundary representations need to be generated and carefully curated, before continuing the traversal.

With our approach, the evaluation of a CSG expression of any complexity is done with a different computational process. The tree itself is only used to apply affine transformations to solid primitives, in order to scale, rotate and translate them in their final ("world") positions and attitudes. All their 2-cells are thus accumulated in a single collection, and each of them is independently operated, generating a collection of local topologies that are merged by boundary congruence, using a *single* round-off operation on vertices. The global space partition is so generated, and



(a,b) (c,d) (e,f) (h,g) .

Figure 6: Some examples of variadic Booleans, obtained by applying the Boolean operators "`.|`", "`.-`", "`.!`", and "`.&`" to the `assembly` variable, with 1-cells (B-reps) imported from `SVG` files. (a,b) start-to-end: from polygon input via `SVG` to (exploded) difference of outer box and interior walls; (c,d) boundary of union and union of some shapes; (e,f) difference 2-complex and its boundary; (g,h) boundary of outer box minus the previous 2-complex, and the corresponding 2-complex.



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

Figure 7: Boolean union $A \cup B \cup C$ of three cubes (from `assembly` of Example 5), with 2-cells in different colors: (a) view from the front; (b) view from the back; (c) front with exploded 2-cells; (d) back with exploded 2-cells. For clarity sake, only the boundary 2-cells are displayed. The space partition generated other 2-cells in the interior, of course.

all elementary solids are classified w.r.t. 3-cells with a single point-set containment test. Finally, *any Boolean form* of arbitrary complexity, with the same variables, can be immediately evaluated by bitwise vectorized logical operations.

Last but not least, all distinction is removed between manifold and non-manifold geometries, both in 2D and 3D, allowing for mixing B-reps and cellular decompositions of the interior of elementary solids. B-reps may be always generated for efficiency purposes—via boundary operator multiplication—in order to strongly reduce the number of 2-cells to be decomposed (in parallel of course), but this is not strictly required. This could be useful, in particular, for combining outer surfaces with regular solid grids, and is consistent with the generation of internal structures.

Acknowledgements

We would like to acknowledge the good initial implementation work with arrangement computations done by Giulio Martella and Francesco Furiani.

8. Conclusions and future prospects

We have introduced a novel approach to computation of Boolean operations between solid models. In particular, we have shown that any Boolean expression between solid models may be evaluated using the finite algebra associated with the set of independent generators of the chain space produced by the arrangement of Euclidean space generated by a collection of geometric objects. The computational approach to geometric design introduced by this paper is quite different from traditional methods of geometric modeling and computational geometry. A prototype open-source implementation was written in Julia language (Bezanson et al., 2017), mostly using sparse arrays. At the present time, we can only evaluate CSG formulas with closed regular cells. A new implementation for closure algebras is on the way, using the parallel features of the language, that provide best-in-class support to linear algebra and matrix computations. We think that, by introducing linear methods in solid geometry, we have in some sense only scratched the surface of the new body of knowledge being currently investigated by machine learning methods for image understanding. New ML methods need formal and abstract techniques for merging solid models derived from partial images. We have already done some early applications of our topological algebraic method to 3D medical imaging, by combining boundary and coboundary operators with filters, aiming at discovering and tracing complex interior structures. We hope that the techniques introduced by this paper will provide the basics for many applications in this and other fields.

References

Arnold, D.N., 2018. Finite Element Exterior Calculus. volume 93 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

Baumgart, B.G., 1972. Winged edge polyhedron representation. Technical Report Stan-CS-320. Stanford, CA, USA.

Bentley, J.L., 1975. Multidimensional binary search trees used for associative searching. Commun. ACM 18, 509–517. URL: http://doi.acm.org/10.1145/361002.361007, doi:10.1145/361002.361007.

Berztiss, A.T., 1975. Data Structures: Theory and Practice. 2nd ed., Academic Press, Inc., Orlando, FL, USA.

Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B., 2017. Julia: A fresh approach to numerical computing. SIAM Review 59, 65–98. URL: http://julialang.org/publications/julia-fresh-approach-BEKS.pdf, doi:10.1137/141000671.

Birkoff, G., 1973. Lattice Theory. third edition ed., merican Mathematical Society, Providence, Rhode Island.

Borges, C.J.R., 1966. On stratifiable spaces. Pacific Journal Of Mathematics 17, 1–16.

Braid, I.C., 1975. The synthesis of solids bounded by many faces. Commun. ACM 18, 209–216. URL: http://doi.acm.org/10.1145/360715.360727, doi:10.1145/360715.360727.

Cimrman, R., 2015. Sparse matrices in scipy, in: Varoquaux, G., Gouillart, E., Vahtras, O., deBuyl, P. (Eds.), Scipy lecture notes. Zenodo. URL: https://scipy-lectures.org/advanced/scipy_sparse/index.html, doi:10.5281/zenodo.594102.

DiCarlo, A., Milicchio, F., Paoluzzi, A., Shapiro, V., 2009a. Chain-based representations for solid and physical modeling. Automation Science and Engineering, IEEE Transactions on 6, 454 –467. doi:10.1109/TASE.2009.2021342.

DiCarlo, A., Milicchio, F., Paoluzzi, A., Shapiro, V., 2009b. Discrete physics using metrized chains, in: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling, Acm, New York, NY, USA. pp. 135–145. doi:10.1145/1629255.1629273.

DiCarlo, A., Paoluzzi, A., Shapiro, V., 2014. Linear algebraic representation for topological structures. Comput. Aided Des. 46, 269–274. URL: http://dx.doi.org/10.1016/j.cad.2013.08.044, doi:10.1016/j.cad.2013.08.044.

Doerr, A., Levasseur, K., 2019. Applied Discrete Structures. 3rd ed., Creative Commons 3.0. URL: https://open.umn.edu/opentextbooks/textbooks/applied-discrete-structures.

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

Gibson, C., Wirthmüller, K., Plessis, A.D., Looijenga, E., 1976. Topological Stability of Smooth Mappings. Springer-Verlag, Berlin Heidelberg.

Goldwasser, M., 1995. An implementation for maintaining arrangements of polygons, in: Proceedings of the Eleventh Annual Symposium on Computational Geometry, Acm, New York, NY, USA. pp. 432–433. URL: http://doi.acm.org/10.1145/220279.220337.

Halmos, P., 1963. Lectures on Boolean Algebras. Van Nostrand.

Hironaka, H., 1975. Triangulation of algebraic sets, in: In Proceedings of Symposia in Pure Mathematics, Algebraic Geometry, pp. 165–185.

Hoffmann, C.M., Hopcroft, J.E., Karasick, M.S., 1987. Robust Set Operations on Polyhedral Solids. Technical Report. Ithaca, NY, USA.

Hoffmann, C.M., Shapiro, V., 2017. Solid modeling, in: Toth, C.D., O'Rourke, J., Goodman, J.E. (Eds.), Handbook of Discrete and Computational Geometry. Chapman and Hall/CRC. chapter 57. URL: https://doi.org/10.1201/9781315119601.

Kasper, J.E., Arns, D., 1993. Graphics Programming with PHIGS and PHIGS PLUS. Hewlett-Packard/Addison-Wesley, Inc., Reading, MA, USA.

Mckinsey, J.C.C., Tarski, A., 1944. The algebra of topology. Annals of Mathematics 45, 141 191.

Mckinsey, J.C.C., Tarski, A., 1946. On closed elements in closure algebras. Annals of Mathematics 47, 122—162.

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

Paoluzzi, A., Ramella, M., Santarelli, A., 1989. Boolean algebra over linear polyhedra. Comput. Aided Des. 21, 474–484. URL: http://dl.acm.org/citation.cfm?id=70248.70249.

Paoluzzi, A., Shapiro, V., DiCarlo, A., 2017a. Arrangements of cellular complexes. CoRR abs/1704.00142. URL: http://arxiv.org/abs/1704.00142, arXiv:1704.00142.

Paoluzzi, A., Shapiro, V., DiCarlo, A., Furiani, F., Martella, G., Scorzelli, G., 2017b. Topological computing of arrangements with (co)chains. ACM Transactions on Spatial Algorithms and Systems (Submitted).

Preparata, F.P., Shamos, M.I., 1985. Computational Geometry: An Introduction. Springer-Verlag New York, Inc., New York, NY, USA.

Requicha, A., 1977. Mathematical models of rigid solids, in: Tech. Memo28, Production Automation Project. University of Rochester.

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

Rossignac, J.R., O'Connor, M.A., 1989. A dimension-Independent Model for Pointsets with Internal Structures and Incomplete Boundaries. Technical Report Research Report RC 14340. IBM Research Division. Yorktown Heights, N.Y. 10598.

Shapiro, V., 1991. Representations of Semi-algebraic Sets in Finite Algebras Generated by Space Decompositions. Ph.D. thesis. Ithaca, NY, USA.

Shewchuk, J.R., 2002. Delaunay refinement algorithms for triangular mesh generation. Computational Geometry 22, 21 – 74. URL: http://www.sciencedirect.com/science/article/pii/S0925772101000475, doi:https://doi.org/10.1016/S0925-7721(01)00047-5.

Stone, M.H., 1936. The theory of representations for Boolean algebras. Trans. Am. Math. Soc. 40, 37–111. doi:10.2307/1989664.

Strang, G., 2019. Linear Algebra and Learning from Data. Wellesley–Cambridge Press. URL: http://math.mit.edu/~gs/learningfromdata/.

Tilove, R.B., 1980. Set membership classification: A unified approach to geometric intersection problems. IEEE Trans. on Computer , 874–883.

Weiler, K.J., 1986. Topological structures for geometric modeling. Ph.D. thesis. Rensselaer Polytechnic Institute.

Zhou, Q., Grinspun, E., Zorin, D., Jacobson, A., 2016. Mesh arrangements for solid geometry. ACM Trans. Graph. 35, 39:1–39:15. URL: http://doi.acm.org/10.1145/2897824.2925901, doi:10.1145/2897824.2925901.