

Article

Topologically Consistent Embedding of Manifold Cells into a Polyhedral Space Partition: A General Algorithm for Spatial Modeling of the Built Environment [†]

Maximilian Sternal * , Antonio Carlone  and Wolfgang Huhnt 

Chair of Computing in Civil Engineering, Technische Universität Berlin, 13355 Berlin, Germany; carlone@tu-berlin.de (A.C.); wolfgang.huhnt@tu-berlin.de (W.H.)

* Correspondence: maximilian.sternal@tu-berlin.de

[†] This paper is an extended version of our paper published in Sternal, M.; Oswald, S.; Huhnt, W. First Investigations of Advanced Operators for Consistent Spatial Modeling of the Built Environment. In Proceedings of the 20th International Conference on Computing in Civil and Building Engineering (ICCCBE 2024), Montréal, QC, Canada, 2024.

Abstract

Spatial models used in architecture, civil engineering, and geodesy require topological consistency to support downstream applications such as energy simulation, indoor navigation, and urban planning. Existing Boolean operation engines process individual mesh objects but do not guarantee the consistency of a complete space partition, particularly when unbounded space must be represented explicitly. This paper presents a general embedding algorithm that inserts an arbitrary polyhedral boundary representation B into an existing space partition A , which constitutes a manifold, gapless partition of Euclidean space including unbounded cells, through a sequence of topologically controlled split operations. Unbounded edges are treated as rays and intersected directly by the geometric kernel, so no separate virtualization of the geometry at infinity is required. The algorithm proceeds in three phases: face-by-face comparison with tolerance-based classification to detect all intersections, incremental collection of split data for edges, faces, and cells through a multi-pass traversal, and execution of split work steps on the topological kernel in ascending dimension order. By operating exclusively through split operations, validity is checked incrementally at each split rather than by a separate global repair pass, so A is kept a valid space partition throughout without any smoothing or repair of an invalid result. A systematic test case matrix covering nine categories of volumetric relationships and boundary contact types was developed. Selected representative scenarios from this matrix were implemented and checked for topological correctness on the resulting partition, including preservation of manifold connectedness and gap-free space coverage without post hoc repair. The algorithm generalizes the parametric design operators introduced in a preceding conference paper to the full three-dimensional case and provides a theoretical foundation for consistent spatial modeling of the built environment.



Academic Editor: Salvatore Antonio Biancardo

Received: 31 May 2026

Revised: 30 June 2026

Accepted: 1 July 2026

Published: 9 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: space partitioning; Boolean intersection; topological consistency; computational geometry; building information modeling; cellular complex; polyhedral modeling; embedding algorithm

1. Introduction

Digital spatial models are fundamental to the planning, construction, and operation of the built environment. In building information modeling (BIM), geographic information

systems (GIS), and urban digital twins, these models must capture not only the geometry of individual elements but also the topological relationships between them, such as adjacency, containment, and connectivity. Many tasks across the built environment depend on this topology: energy performance simulation needs gap-free thermal zones, indoor and outdoor navigation needs traversable spatial networks, and city modeling needs volumetric representations that satisfy the ISO 19107 spatial schema [1]. In all of these cases, space itself, including the open space between and around built objects, carries information that the application must use.

Most established modeling workflows are nevertheless *object-centric*. They describe a scene as a collection of separate solid objects, for example, single building elements stored in IFC [2], and treat the relationships between objects as secondary data that is added afterwards or recovered from geometry in a post-processing step. This view has a structural weakness. Because each object is modeled on its own, the model has no built-in guarantee that the objects together form a consistent whole: neighboring volumes may overlap, leave gaps, or meet in non-manifold ways, and the empty space around the objects is not represented at all. The consistency must then be checked and repaired after the fact, which is error-prone and rarely complete.

A *spatial-centric* approach starts from the opposite end. Instead of modeling objects and inferring their relations, it models space itself as the primary entity and treats every object as a region of that space. A *space partition* divides the entire Euclidean space, including the unbounded regions, into non-overlapping cells that together cover all of the space without gaps. Each object then corresponds to one or more cells, and adjacency between objects is read directly from the shared boundaries of the cells. This shift has three practical advantages for the built environment. First, *robustness*: the partition is modified only through controlled local operations, so the model cannot drift into an inconsistent state through accumulated numerical error. Second, *consistency by construction*: because every point of space belongs to exactly one cell, gaps and overlaps cannot occur, and the topology is correct at every stage rather than validated afterwards. Third, *interoperability*: a single consistent partition can serve many downstream uses, such as energy simulation, navigation, and 3D cadastre, without a separate repair step for each.

Space partitioning is a well-established idea in computational geometry. Binary space partitioning (BSP) trees and octrees recursively subdivide space and are widely used for robust Boolean operations on solids; for example, Nehring-Wirxel et al. [3] embed BSP trees in an octree with integer arithmetic to perform iterated constructive solid geometry with unconditional robustness. These subdivisions are powerful, but their cells are axis-aligned or half-space-bounded fragments that do not directly correspond to the meaningful regions of a building, such as rooms, components, or parcels. A *polyhedral* space partition keeps the central idea, a gap-free cover of all space, but uses general polyhedral cells whose faces coincide with the actual surfaces of the built environment. Huhnt et al. [4] introduced such a platform, in which nodes, edges, faces, and cells are explicit topological entities with orientation and adjacency, and in which all geometry is *manifold* and *topologically connected* into one complete space partition. The platform is modified through only two atomic work steps: *splitting* a domain into two sub-domains and *merging* two adjacent domains into one. The mathematical formulation has been developed in the PPSpace research project, sponsored by the German Research Foundation DFG. The complete kernel's operators for geometric and topologically consistent modifications of bounded and unbounded, singly and multiply connected as well as convex and non-convex domains were developed in the dissertation of Sternal [5], which is the reference implementation on which the present work is based. Related developments include robustness strategies for such partitions [6], their

use as a holistic alternative to conventional geometric workflows [7], and the transformation of boundary representation models into a partition [8,9].

In a spatial-centric workflow, a model is built through a sequence of *design steps*. Each design step introduces new geometry to the existing partition while keeping the partition consistent. In a preceding conference paper [10], it was shown how established parametric design functionalities, namely extrusion, slicing, and primitive placement, can each be expressed as such a design step by mapping them to the split and merge work steps of the partition platform. That work was limited to planar (two-dimensional) surface operations and identified the generalization to the full three-dimensional case as essential future work. The present paper addresses this gap by providing the general three-dimensional operation on which every design step is based.

This paper presents a *general embedding algorithm* that inserts an arbitrary polyhedral boundary representation B into an existing valid space partition A , such that A remains a valid, manifold, gapless space partition after the operation. The defining property of the algorithm is that it modifies A *exclusively through split operations*: no geometry is constructed from scratch, and the partition is manifold and connected after every individual split.

The input B must satisfy the following conditions:

- A watertight face or cell;
- No self-intersection and no self-contact.

It can have the following properties:

- Convex or non-convex;
- With holes (multiply connected) or without holes (simply connected);
- Bounded or unbounded.

Topological consistency is maintained incrementally: validity is checked at each individual split rather than by a separate post hoc repair pass, and unbounded cells are handled as first-class entities throughout. The contribution is primarily theoretical, supported by prototype-level validation: a systematic test case matrix with nine volumetric categories was developed, and selected representative cases were implemented in a Java prototype and checked for topological correctness of the resulting partition.

The remainder of this paper is organized as follows. Section 2 reviews related work on consistency challenges, topological data models, and Boolean operation engines. Section 3 summarizes the space partitioning platform and establishes notation. Section 4 presents the embedding algorithm in detail. Section 5 describes the prototype implementation and validation results. Section 6 discusses the approach in relation to existing methods and identifies limitations. Section 7 concludes the paper.

2. Background and Related Work

This section highlights related work on topological modeling and data structures and geometric boolean operation algorithms.

2.1. Consistency Challenges in Spatial Models for the Built Environment

Topological consistency, defined as the absence of gaps, overlaps, and non-manifold configurations, is a prerequisite for many analyses performed on spatial models of the built environment. A spatial model is consistent in this sense only if its geometry is *manifold* and its cells are *topologically connected* into a single, gapless complex. In building energy performance simulation, the geometry of thermal zones is typically derived from BIM models and exported in formats such as gbXML. Yang et al. [11] demonstrated that the reconstruction of second-level space boundaries from IFC models frequently produces gaps and overlapping surfaces that cause simulation failures. Massafra and Gulli [12] argued that

explicit topological models are necessary to achieve reliable BIM-to-building-performance-simulation interoperability, and later work by Massafra, Jabi, and Gulli [13] showed how a topological building representation supports building performance management across the life cycle. Similarly, Diakité and Zlatanova [14] showed that valid spatial descriptions are essential for BIM-based indoor navigation systems, where non-manifold geometry prevents the generation of correct navigation graphs.

These challenges are not limited to individual buildings. In city modeling, the OGC CityGML standard requires volumetric building representations that conform to ISO 19107 geometric and topological constraints [1,15]. Gröger and Plümer [15] formalized the consistency conditions for three-dimensional city models, demonstrating that most available datasets fail to satisfy them. Giovanella et al. [16] analyzed topological consistency in CityGML datasets and showed that inconsistencies between the stored geometry and the incidence graph are pervasive in practice. Ledoux [17] developed validation methods for solids represented under these standards and reported that a large share of real-world solids are invalid, often because of small gaps or non-manifold contacts that are not visible in a rendered view; earlier work by Ledoux and Meijers [18] showed that topologically consistent three-dimensional city models can be obtained directly by extrusion. At the interface between domains, Arroyo Ohori et al. [19] analyzed the integration of GIS and BIM data and identified the differing geometric and topological assumptions of the two worlds as a central obstacle. Such inconsistencies also arise upstream, when models are reconstructed from measured data: Carlone and Gabler [20] showed that simplifying room polygons from point-cloud data for automated floor-plan generation must preserve the essential geometric structure to avoid introducing such errors. The common thread across these application domains is that topological consistency is not an optional quality attribute but a functional requirement, and that current workflows, which model geometry first and validate or repair topology afterwards, systematically fail to guarantee it.

2.2. Topological Data Models

Several topological data models have been proposed to explicitly represent spatial relationships. Weiler's radial-edge structure [21] was among the earliest to represent non-manifold configurations in boundary representations, providing a rich adjacency structure for solid modeling. The radial-edge structure already provided a theory for radially connected cells in three-dimensional models. Building on the radial-edge, quad-edge, and half-edge representations, Boguslawski and Gold [22] introduced the dual half-edge data structure, which augments the half-edge representation with dual-graph information, enabling efficient navigation between primal and dual topological complexes. This structure supports the construction of Voronoi diagrams and Delaunay triangulations in three dimensions and has been applied to indoor spatial modeling. Boguslawski and Gold [22] also introduced specific Euler-operators for this data structure to create topologically connected cell complexes.

Jabi and Chatzivasileiadi [23] developed Topologic, a software library that represents topological entities (vertices, edges, wires, faces, shells, cells and cell complexes) and supports spatial reasoning through geometry, topology, and semantics. Topologic provides Boolean operations on cell complexes and has been integrated into visual programming environments for architectural design. The ISO 19107 standard [1] defines a comprehensive spatial schema for geographic information, specifying topological primitives and complexes up to three dimensions. Bradley [24] formalized the Euler–Poincaré operators for consistent manipulation of such complexes.

A second line of work builds spatial partitions specifically for the built environment. Suter [25] modeled multiple space views for schematic building design using space ontolo-

gies and layout transformation operations, and later outlined a feature-based procedure for decomposing architectural spaces into sub-spaces [26]. Gabler et al. [27] derived topological graphs from room polygons through a two-dimensional space partitioning approach, and in later work derived topological and geometric measures from floor polygons to classify spaces by function and form [28]. These approaches confirm the value of partition-based representations, but operate in either two dimensions or on bounded space only. The polyhedral space partitioning platform used in this work [4,5] differs from all of the above in three respects that are central to the present paper: it represents *unbounded* space through imaginary topological entities, it enforces a complete partition invariant in which every point of Euclidean space belongs to exactly one domain, and it restricts all modification to two atomic operations, split and merge, so that the manifold and connected structure of the space partition is preserved by construction rather than checked afterwards. The transformation of conventional boundary representation building models into such a partition was studied by Vetter [8] for the tetrahedral case.

2.3. Boolean and Intersection Operations

Boolean operations (union, intersection, and difference) are fundamental tools in computational geometry and solid modeling. The constructive solid geometry (CSG) paradigm represents objects as combinations of primitive solids through Boolean operations, stored as a tree of operations [29]. At the evaluation level, these operations must be resolved into explicit boundary representations, a process that requires robust intersection computation. Hoffmann [29] discussed the fundamental robustness challenges in geometric computations and the strategies (exact arithmetic, symbolic perturbation, interval arithmetic) employed to address them.

A range of robust Boolean engines has been developed. The CGAL library [30] provides Nef polyhedra, which support exact Boolean operations on three-dimensional polyhedral solids using exact arithmetic; the foundational data structure and algorithms for 3D selective Nef complexes, including handling of degeneracies and robustness of Boolean operations, were introduced by Granados et al. [31]. Nehring-Wirxel et al. [3] introduced octree-embedded binary space partitioning trees with plane-based geometry and integer arithmetic to perform iterated CSG with unconditionally robust operations. Cherchi et al. [32] presented an interactive and robust mesh-Boolean method for triangle meshes, and Zhou et al. [33] proposed a simple and robust scheme for Boolean operations on triangulated surfaces. OpenCascade [34] implements Boolean operations through its CAD modeling kernel, and the Manifold library [35] is a recent high-performance mesh Boolean engine designed for robustness and speed. These tools share a common paradigm: they compute the Boolean combination of two bounded solid objects and produce a new bounded solid as output. They do not maintain a space partition of the entire space, they do not track unbounded domains, and they do not enforce a partition invariant across a sequence of operations.

The most closely related approach is the topologically robust algorithm of Smith and Dodgson [36], which guarantees valid connectivity of the result even under numerical errors. The two approaches differ in two essential ways. First, the basic algorithm of Smith and Dodgson may generate geometric artifacts, principally gaps and slivers, and relies on a *data-smoothing post-process* that modifies the generated structure afterwards to remove these invalid regions. In contrast, the algorithm presented here never produces an invalid intermediate state: because the partition is modified only through split operations, every intermediate space partition is already manifold and connected, and no post hoc smoothing or repair step is required. Second, Smith and Dodgson operate on bounded triangle or polygon boundary meshes in the piecewise-linear domain; their method does not represent

or generalize to unbounded polyhedral cells. The algorithm presented here operates on a complete space partition that includes unbounded cells as first-class topological entities, so that the region outside all bounded objects is itself a valid part of the partition.

The embedding algorithm presented in this paper therefore differs from existing engines in several respects. First, only A is modified; B is read-only input. Second, modification occurs exclusively through split operations, which preserve the manifold, connected, gapless space partition by construction. Third, all new topological entities are explicitly named in split operations, providing full traceability. Fourth, ambiguous configurations, such as touching faces, coincident edges, or near-miss intersections, are detected and reported to the user rather than resolved through silent internal heuristics. Fifth, unbounded cells are treated as first-class entities throughout the computation. The topological kernel on which the algorithm operates is the consistent spatial modeling kernel developed in the dissertation of Sternal [5], which provides the arrow, facet, and voxel structures and the split and merge operators used throughout this paper.

3. Space Partitioning Platform

This section summarizes the space partitioning platform as introduced in [4] and further developed in [6–8,10]. The presentation establishes the notation and concepts required in Section 4.

3.1. Domain Types and Data Model

The platform represents Euclidean space as a *space partition*: a cellular complex that covers the entire Euclidean space without gaps or overlaps. Throughout this paper, the term *space partition* denotes such a complete, whole-space complex, while the term *cell* denotes a single three-dimensional domain of the partition, and the term *cellular complex* is reserved for a complex that need not cover all of space (for example the input domain B). The partition consists of topological domains of dimension 0 through 3: nodes, edges, faces, and cells. Each domain type except nodes exists in bounded and unbounded variants. Bounded domains are associated with finite point sets; unbounded domains extend to infinity in one or more directions. In addition, imaginary domains are defined purely topologically, without associated point sets, to represent the boundary structure at infinity. Table 1 summarizes the domain types.

Table 1. Domain types in the space partitioning platform.

Dimension	Symbol	Description
0	N_r	Real nodes with position coordinates
0	N_i	Imaginary nodes (direction vectors of unbounded edges)
1	E_b	Bounded edges (two real nodes)
1	E_u	Unbounded edges/rays (one real node, one imaginary node)
1	E_i	Imaginary edges (two imaginary nodes)
2	F_b	Bounded faces
2	F_u	Unbounded faces (boundary includes unbounded edges)
2	F_i	Imaginary faces (boundary consists of imaginary edges)
3	C_b	Bounded cells
3	C_u	Unbounded cells

The orientation of a domain boundary distinguishes a bounded interior from an unbounded one. For a bounded domain the directed boundary arrows enclose a finite interior, while reversing their orientation makes the surrounding unbounded region the interior (Figure 1). This is how the platform represents the open space around built objects as a first-class cell.

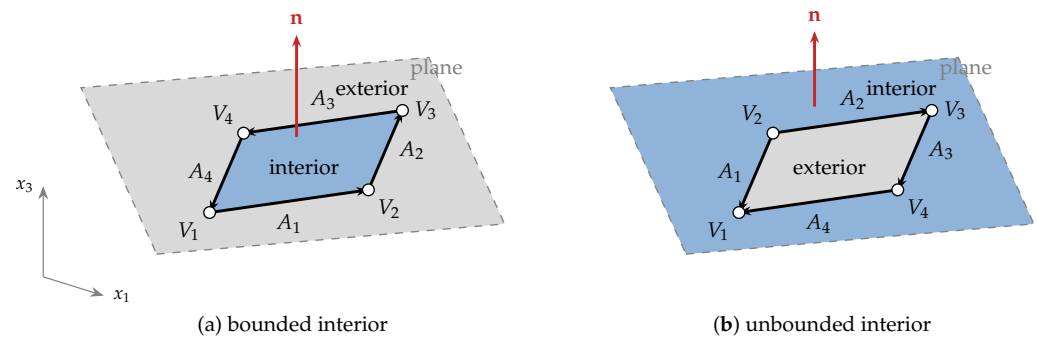


Figure 1. Orientation of a planar domain boundary. The red arrow shows the outward unit normal $\mathbf{n}$ of the supporting plane. (a) Directed boundary arrows A_1 – A_4 in counter-clockwise order (with respect to the outward normal $\mathbf{n}$) enclose a *bounded* interior. (b) Reversing the orientation makes the *unbounded* region outside the loop the interior, so that open space is represented explicitly as a domain of the partition.

The key invariant of the space partition is that every point in Euclidean space belongs to exactly one domain. There is no concept of space outside the partition; the unbounded cells cover all remaining space not occupied by bounded cells. The unbounded space surrounding a bounded object cannot be represented by a single unbounded cell: the boundary structure at infinity is itself subdivided, so that at least two unbounded cells are always required to cover the exterior [4,5]. Faces and cells may be simply connected or multiply connected (containing holes in their boundary).

3.2. User Model and Core Model

The partition platform distinguishes two complementary representations of the same topological structure. The *user model* provides an unoriented set-based boundary representation (BRep) through which users define and query geometry. Each cell stores a set of its bounding faces, each face stores a set of its bounding edges, and each edge stores its two endpoint nodes:

$$\text{Cell} \rightarrow \text{Set}\langle \text{Face} \rangle, \quad \text{Face} \rightarrow \text{Set}\langle \text{Edge} \rangle, \quad \text{Edge} \rightarrow \{N_{\text{start}}, N_{\text{end}}\}.$$

This representation is intuitive and sufficient for specifying input geometry, but it does not encode orientation or full adjacency: given a face, the user model does not directly provide the two cells on either side, nor does it encode the cyclic ordering of edges around a face.

The *core model* is implemented as a half-edge kernel [4,5] that stores the complete oriented topological adjacency structure. To distinguish the two representations, the core model uses its own terms: an edge corresponds to one or more *arrows*, a face to a *facet*, and a cell to a *voxel*. An arrow is an oriented child of an unoriented edge; it inherits the point set of its parent edge and acts as a vector pointing from the start node to the end node. For each face incident to an edge, two arrows of opposite orientation are defined; these are called *twin arrows*. An edge with m incident faces is therefore represented by $2m$ arrows. The m arrows that share the same direction form an *arrow bundle*, so every edge carries exactly two arrow bundles of opposite direction, with one twin-arrow pair contributed by each incident face. A bounded or unbounded edge is thus represented by at least four arrows, while an imaginary edge requires at least two. This bundle structure, rather than a single half-edge pair, is what allows the kernel to represent configurations in which more than two faces meet at one edge, which a single half-edge pair cannot.

Analogously, every bounded face is mapped to exactly two *facets* with antiparallel normal vectors, called *twin facets*. A facet is an oriented child of a face. Its boundary is a closed loop of arrows (the *arrow polygon*) whose direction is aligned to be positively oriented

with respect to the facet normal. The orientation of a facet, and therefore its normal vector, is thus derived directly from the order of its boundary polygon. Each voxel is bounded by facets whose normal vectors point outward toward the exterior of the voxel.

Navigation within a facet boundary follows the ordered cycle of arrows, shown in Figure 2; navigation between faces sharing an edge uses the *dihedral cycle*, which enumerates the arrows of one bundle in rotational order around the edge and thereby visits all faces meeting at that edge. This structure enables constant-time traversal of all adjacency relationships required by the embedding algorithm: given any edge, the incident faces and adjacent cells are immediately accessible through its arrow bundles; given any face, the complete bounding arrow polygon and both adjacent cells can be traversed. The core model is constructed automatically from the user model: given the set-based boundary specification of a partition, the kernel derives the arrows and facets, organizes them into arrow bundles and dihedral cycles, and assigns voxel adjacency for every facet. Users interact exclusively with the user model in terms of edges, faces, and cells; the arrow–facet–voxel core model is an internal representation on which the algorithm operates.

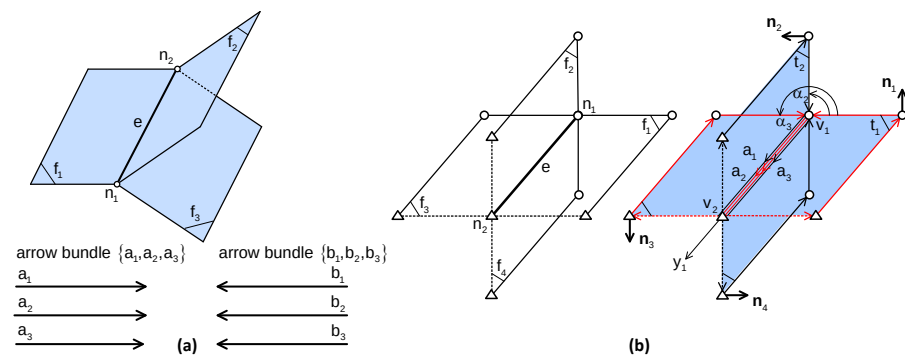


Figure 2. Dihedral cycle of arrows in a bundle. Arrows point in the same direction and start from the same node; each arrow belongs to a different polygon in 3D and the bundle is sorted counter-clockwise around the shared edge. (a) Three-face configuration with one arrow bundle per direction. (b) Multi-face configuration: each additional incident face contributes one twin-arrow pair to the bundle, rotationally sorted ccw.

Orientation is defined consistently across both models. The normal vector of a facet is fixed by the positive orientation of its boundary arrow polygon, and voxels are oriented such that their bounding facet normals point outward. During the initialization of a partition, the orientation of facets may be arbitrary; the kernel resolves this using the topological properties of the arrow bundles, together with the direction of unbounded edges defined by imaginary nodes, so that all facet normals become consistent. This orientation convention is the basis for the LEFT/RIGHT/ON classification used in the intersection detection phase of the embedding algorithm.

3.3. Work Steps: Split and Merge

All modifications to the partition are performed through two atomic work steps [4,10]:

Definition 1 (Split). A split operation subdivides domain D of dimension d into two adjacent sub-domains D_1 and D_2 of the same dimension. The splitting boundary is a domain of dimension $d - 1$. After a split, all topological references in the kernel are updated to reflect the new adjacency structure.

Definition 2 (Merge). A merge operation is the inverse of a split: two adjacent domains D_1 and D_2 of the same dimension d sharing a common boundary domain of dimension $d - 1$ are merged into a single domain D . The shared boundary domain is removed.

These two operations are the only means by which the partition can be modified. Because a split always produces two valid sub-domains from a valid domain, and a merge combines two valid domains into a valid domain, the partition invariant is preserved after every work step. The embedding algorithm presented in Section 4 is expressed entirely in terms of split operations.

4. The Embedding Algorithm

This section presents the main contribution of this paper: a general algorithm for embedding a polyhedral boundary representation B into an existing space partition A through a sequence of topologically controlled split operations.

4.1. Domain Types Used in the Algorithm

The algorithm operates directly on the geometry of A and B without any prior virtualization of the elements at infinity. An unbounded edge is a *ray*: it has one real node with coordinates and a direction vector given by its imaginary node. The geometric kernel parametrizes such a ray as $P(t) = \mathbf{x}_{\text{real}} + t \mathbf{d}$ with $t \in [0, \infty)$, exactly as it parametrizes a bounded edge with $t \in [0, 1]$. The intersection routines (edge against plane, edge against edge) therefore treat bounded edges and unbounded edges in the same way, differing only in the upper bound of the valid parameter range. As a consequence, the distinction between bounded and unbounded elements is irrelevant during Phases 1 and 2, and no temporary coordinates need to be assigned or later discarded. Purely imaginary elements, namely imaginary edges (two imaginary nodes) and imaginary faces, have no real point and no associated direction; they represent only the topological closure of the partition at infinity. Such elements never participate in a geometric intersection test, because the finite and ray geometry of B can meet A only on its real (bounded or unbounded) elements. The imaginary closure is maintained purely topologically by the split operations of Phase 3. Table 2 lists the domain types that participate, grouped by origin.

Table 2. Domain types used in the embedding algorithm.

Symbol	Origin	Description
N_b	B	Nodes of the input domain B (original vertices)
E_b	B	Edges of B (segments between two nodes N_b)
F_b	B	Faces of B (planar polygons bounded by edges E_b)
N_a	A	Nodes of the space partition A
E_a	A	Edges of A (segments between two nodes N_a)
F_a	A	Faces of A (planar polygons bounded by edges E_a)
C_a	A	Cells of A (volumes bounded by faces F_a)
N_i	Algorithm	Intersection nodes created during Phase 1
E_i	Algorithm	Intersection edges created in Step 1.3

Every N_i and every N_b is mapped to exactly one element of A , whose dimension determines the split pass that processes it: a node in a cell drives a cell split, a node on a face drives a face split, a node on an edge drives an edge split, and a node coincident with N_a drives an identity.

4.2. Problem Statement and Input Requirements

Definition 3 (Embedding Problem). *Given a space partition A that constitutes a valid, manifold, gapless partition of Euclidean space (including unbounded cells), and an input domain B that is a watertight, oriented polyhedral boundary representation (face or cell, possibly multiply connected), compute a sequence of split operations on A such that the resulting space partition contains the geometry of B as an identifiable sub-complex, while keeping the partition invariant.*

The input requirements for B are: (i) B is watertight, which means that its boundary forms a closed manifold with no gaps or self-intersections; (ii) B is oriented, with outer normals defined consistently; (iii) B contains bounded elements and may contain unbounded elements; and (iv) B may be multiply connected (contains holes), and if so, each boundary component is processed individually.

Table 3 formalizes each condition, the method by which the algorithm checks it, and the behavior on violation.

Table 3. Input requirements for the embedding domain B : required condition, checking method, and algorithm behaviour on violation.

Condition	Checking Method	Behaviour on Violation
Watertight (closed, no gaps)	Each edge shared by exactly two faces (pre-processing)	Rejected; open-boundary location reported
Planar faces only	Faces planar within ϵ before Phase 1	Rejected; max planarity deviation reported
Holes/multiply connected faces	Each inner loop handled as a separate component	Supported; bridge path (Step 2.5)
Non-triangulated faces	General polygons; no triangulation assumed	Supported directly
Non-convex polygons	Convexity not assumed	Supported
No self-intersections	Face pairs of B tested in Phase 1	Rejected; location reported
Consistent orientation	Normals from arrow bundle topology	Normalized if unambiguous, else reported
Unbounded elements (rays)	Same routines as edges, $t \in [0, \infty)$	Supported; imaginary nodes give directions

The algorithm maintains five maps that track the relationship between elements of B and elements of A throughout the computation:

$$\mu_1 : E_b^B \rightarrow \text{List}\langle N_i \rangle, \quad (1)$$

$$\mu_2 : N_i \rightarrow \text{Element}(A), \quad (2)$$

$$\mu_3 : N_b^B \rightarrow \text{Element}(A), \quad (3)$$

$$\mu_4 : F_b^B \rightarrow \text{List}\langle E_i \rangle, \quad (4)$$

$$\mu_5 : E_i \rightarrow \text{Element}(A), \quad (5)$$

where N_i denotes intersection nodes, E_i denotes intersection edges, and $\text{Element}(A)$ refers to any topological element of A (node, edge, face, or cell). An intersection node N_i is any point at which the geometry of B touches or intersects the geometry of A . This includes not only newly computed crossing points, but also existing nodes that turn out to be coincident: when a node N_a of A lies on an edge E_b of B , or a face F_b of B , or coincides with a node N_b of B , that N_a is itself an intersection node N_i , because at that point, the two geometries intersect. The maps therefore treat newly created crossing points and coincident existing nodes uniformly as N_i . Map μ_1 stores intersection nodes per edge of B , ordered by parameter t along the edge. Map μ_2 records which element of A each intersection node was found on. Map μ_3 classifies the original nodes of B against the elements of A . Map μ_4 records intersection edges per face of B . Map μ_5 identifies the face of A on which each intersection edge lies.

The algorithm proceeds in three phases. The three phases play distinct conceptual roles. Phase 1 performs face-by-face comparisons to detect all intersections and populate the maps. Its building blocks, the edge–face and edge–edge intersection tests, are the same primitives used by established Boolean engines for triangle and polygon meshes,

such as the topology-oriented method of Smith and Dodgson [36] and the mesh-Boolean library of Lalish [35]. Algorithm 1 gives the overall control flow, and Algorithm 2 details the collection of split data in Phase 2. Individual steps are described in the following subsections.

Algorithm 1 Embedding of B into A (overall control flow).

Require: valid partition A ; watertight oriented domain B

Ensure: refined space partition A' containing B as a sub-complex

- 1: **Phase 1:** ▷ detect intersections, populate $\mu_1 \dots \mu_5$
 - 2: **for** each face pair (F_a, F_b) with $F_a \in A, F_b \in B$ **do**
 - 3: classify edges E_b against plane of F_a ▷ Step 1.1
 - 4: classify edges E_a against plane of F_b ▷ Step 1.2
 - 5: build intersection edges E_i from collected N_i ▷ Step 1.3
 - 6: **end for**
 - 7: assign residual nodes N_b to their containing cell C_a
 - 8: **Phase 2:** collect split data ▷ Steps 2.1–2.6, Algorithm 2
 - 9: **Phase 3:** ▷ execute on kernel, ascending dimension
 - 10: split affected edges E_a ▷ Step 3.1
 - 11: split affected faces F_a (inner boundaries first) ▷ Step 3.2
 - 12: split affected cells C_a ▷ Step 3.3
 - 13: assign new sub-domains to B ; **optional** merge ▷ Step 3.4
-

Algorithm 2 Phase 2: collecting split data in three passes.

- Pass 1—from node mappings μ_3*
- 1: **for all** nodes N_b with $\mu_3(N_b) = C_a$ (cell) **do**
 - 2: traverse B from N_b ; collect interior face fragments for split of C_a ▷ Step 2.1
 - 3: **end for**
 - 4: **for all** nodes N_b with $\mu_3(N_b) = F_a$ (face) **do**
 - 5: collect adjacent edges of B on F_a for face-split chains ▷ Step 2.2
 - 6: **end for**
 - 7: **for all** nodes N_b with $\mu_3(N_b) = E_a$ (edge) **do**
 - 8: add N_b and N_i on E_a ; sort by t ; define sub-edges ▷ Step 2.3
 - 9: **end for**
- Pass 2—from edge mappings*
- 10: **for all** edges E_b with both endpoints on faces of a common cell C_a **do**
 - 11: collect adjacent faces of B as extra split fragments ▷ Step 2.4
 - 12: **end for**
 - 13: **for all** edges E_b with both endpoints on edges of A **do**
 - 14: assemble face-split chains (bridge path if face has holes) ▷ Step 2.5
 - 15: **end for**
- Pass 3—remaining faces*
- 16: **for all** faces F_b not yet assigned to a cell split **do**
 - 17: build boundary-only splitting surface ▷ Step 2.6
 - 18: **end for**
-

Phase 1 reuses these standard primitives, but applies them to a complete space partition rather than to two bounded solids, and classifies every result against the elements of A instead of producing a new mesh. Phase 2 is specific to the topological kernel used in this work. It translates the intersection results from Phase 1 into *named, ranked split work steps*. The rank (dimension) of the element of A to which each element of B was mapped decides whether the element contributes to an edge, face, or cell split, and the same rank fixes the order in which the steps may be executed. Phase 2 is therefore the bridge between the generic intersection geometry of Phase 1 and the split operations of the partition kernel.

Phase 3 is the partition platform itself: it executes the split work steps on the kernel in ascending dimension order, with geometric and topological validity checks at every step and user feedback for ambiguous configurations, followed by an optional merge. Table 4 summarizes the input, output, and validity checks of the three phases.

Table 4. Three-phase data flow of the embedding algorithm: input, output with key data structures, and validity checks with their error handling.

Phase	Input → Output (Key Data Structures)	Validity Checks and Error Handling
1—Face-by-face comparison	A and $B \rightarrow$ populated maps $\mu_1\text{--}\mu_5$ (intersection nodes N_i , intersection edges E_i , node-to-element mappings via anchor–client localization)	Non-manifold detection and duplicate- N_i suppression (within ε); near-coincident geometry ($\varepsilon < d < \delta$) is flagged and reported to the user
2—Split data collection	Maps $\mu_1\text{--}\mu_5 \rightarrow$ named, dimensionally ranked split items per affected edge, face, and cell (sorted node lists, splitting chains, splitting surface fragments)	Chain closure and bridge-path validity for multiply connected faces; a non-completable chain or incomplete splitting surface is reported with its location
3—Kernel execution	Ranked split items $\rightarrow$ refined partition A' with B as an identifiable sub-complex (arrow / facet / voxel structures, sub-domain identifiers)	Per-split manifold and gap-free kernel invariants are checked after every split; a violation halts execution and reports the offending split

4.3. Tolerance Strategy

The algorithm employs a two-threshold tolerance strategy to handle the finite precision of floating-point arithmetic and the inherent ambiguity of near-coincident geometric configurations. A distance less than ε (the snap tolerance) indicates geometric identity: the intersection point is snapped to the nearest existing element. A distance between ε and δ (the warning threshold, with $\delta > \varepsilon$) indicates a suspicious configuration: the result is flagged and reported to the user for review. A distance greater than δ indicates clear separation. This strategy is applied consistently in every classification performed during Phases 1 and 2.

(i) Selection rules for ε and δ .

The snap tolerance ε is chosen as a small multiple of the machine epsilon scaled by the characteristic coordinate magnitude of the partition: $\varepsilon = \alpha \cdot \epsilon_{\text{mach}} \cdot \|\mathbf{x}\|_{\infty}$, where α is a small integer multiplier (typically in the range $10^2\text{--}10^4$, depending on the complexity of the computation chain), ϵ_{mach} is the floating-point machine epsilon for the working precision, and $\|\mathbf{x}\|_{\infty}$ is the maximum absolute coordinate value occurring in the partition. The warning threshold is set to $\delta = \beta \cdot \varepsilon$ with $\beta > 1$ (typically $\beta = 10\text{--}100$), giving a band of configurations that are neither snapped nor treated as cleanly separated and are instead surfaced to the user.

(ii) Relationship to model scale.

Because both ε and δ are defined relative to $\|\mathbf{x}\|_{\infty}$, the classification is invariant under uniform rescaling of the model: doubling all coordinates doubles both thresholds, so the same geometric configuration produces the same classification regardless of the unit system. When the model is re-centred (e.g., survey coordinates translated to a local origin), $\|\mathbf{x}\|_{\infty}$ decreases and the effective thresholds tighten correspondingly, improving numerical discrimination near the model interior.

(iii) Mixed-precision input.

When BIM-authored coordinates and survey coordinates of differing precision are combined in the same partition, the coarser of the two precisions governs the choice of ε :

the threshold must not be set finer than the precision of the least precise input source, or genuine coincidences at the BIM-coordinate level may be classified as clear separations. Near-coincident configurations below δ that originate from this precision mismatch are surfaced as warnings rather than resolved silently, prompting the user to inspect and if necessary adjust the input.

A full empirical sensitivity analysis—reporting the counts of snaps, warnings, and errors as ε and δ are varied across a range of model sizes and contact configurations—is a quantitative measurement study that requires optimized implementation. This sweep is deferred to the dedicated follow-up software publication, which will also provide the source code and test-data generator needed for independent reproduction.

The localization of a point relative to a domain uses the concept of *anchor* and *client zones* introduced for the partition platform [5,6]. The boundary of a domain is decomposed into open subdomains called anchor zones; each anchor zone has an associated *client zone* containing all points that are closer to it than to any other anchor zone, so the client zones partition the surrounding space. For a query point (a *client* C), the closest boundary point is its computed *anchor* H , and the segment $\overline{CH}$ is the *anchor line*, whose length is the distance to the boundary. For a planar face with unit normal $\mathbf{n}$ and a point X_0 in its plane, the vector from X_0 to the client X_m splits into an in-plane part $\mathbf{s}$ and a normal part, $\mathbf{x}_m - \mathbf{x}_0 = \mathbf{s} + d_m \mathbf{n}$ with $\mathbf{n}^\top \mathbf{n} = 1$ and $\mathbf{n}^\top \mathbf{s} = 0$. Left-multiplying by $\mathbf{n}^\top$ gives the signed distance $d_m = \mathbf{n}^\top (\mathbf{x}_m - \mathbf{x}_0)$, and the anchor location follows as $\mathbf{h}_m = \mathbf{x}_m - d_m \mathbf{n}$ (Figure 3a). The rank (dimension) of the anchor's domain determines whether the client lies on a node, an edge, or a face, and the orientation of the core model arrow or facet normal at the anchor determines whether the client is inner, outer, or on the boundary (Figure 3b). Throughout this operation, the phrase “the anchor of N_i is checked against the elements of F_a ” refers to this localization: the anchor of the intersection node is found on the boundary of F_a , and its rank fixes the element of A to which N_i is mapped (Figure 3).

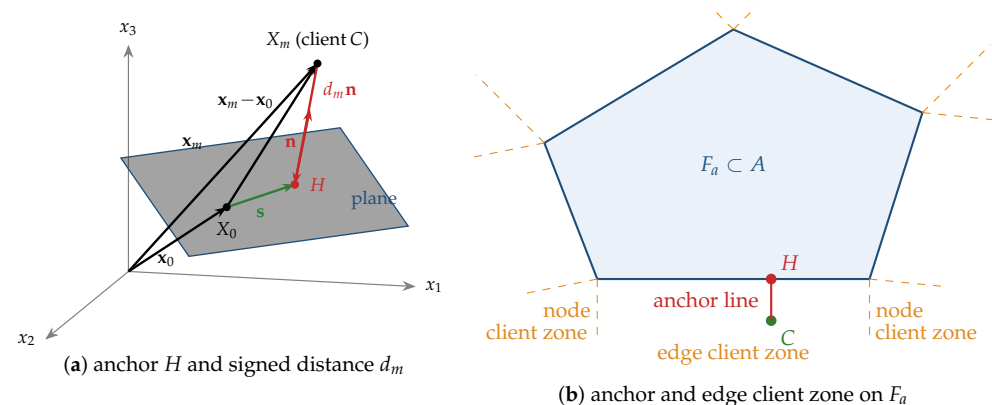


Figure 3. Localization of a query point against a domain of A [5,6]. Colors: the client point C is shown in green, the anchor H and the anchor line in red, and the node and edge client zones in orange. (a) The vector from a point X_0 in the supporting plane to the client X_m is decomposed as $\mathbf{x}_m - \mathbf{x}_0 = \mathbf{s} + d_m \mathbf{n}$, with $\mathbf{s}$ in the plane and $d_m \mathbf{n}$ along the unit normal $\mathbf{n}$; the signed distance is $d_m = \mathbf{n}^\top (\mathbf{x}_m - \mathbf{x}_0)$ and the anchor is $\mathbf{h}_m = \mathbf{x}_m - d_m \mathbf{n}$. (b) The boundary of a face F_a is decomposed into anchor zones, each with a client zone; the client C has an edge anchor H in the edge client zone, and the rank of the anchor together with the outward normal $\vec{n}$ classifies the client as inner, outer, or on the boundary.

4.4. Phase 1: Face-by-Face Comparison

The goal of Phase 1 is to populate all five maps by detecting every intersection between elements of B and elements of A . Each pair of faces, one from A and one from B , is compared, first through their bounding edges, then through their supporting planes.

The phase proceeds in three steps, summarized in Figure 4. The geometric basis of the classification is the position of a node relative to the supporting plane of a face, illustrated in Figure 5.

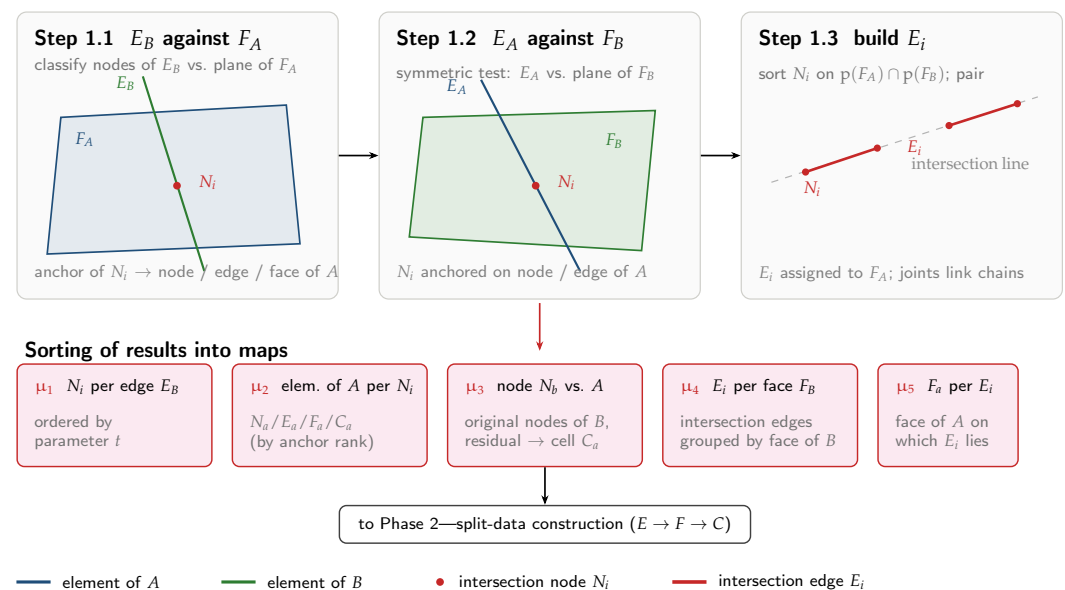


Figure 4. Detail of Phase 1 for a single face pair (F_A, F_B). Step 1.1 classifies every edge E_B of B against the plane of F_A ; Step 1.2 performs the symmetric test of every edge E_A against the plane of F_B . Each resulting intersection node N_i is anchored to the element of A on which it lies, its rank fixing the mapping. Step 1.3 sorts all nodes along the line $\text{plane}(F_A) \cap \text{plane}(F_B)$ and pairs consecutive nodes into intersection edges E_i . The classified results are stored in the five maps μ_1 – μ_5 , which drive the split data construction in Phase 2.

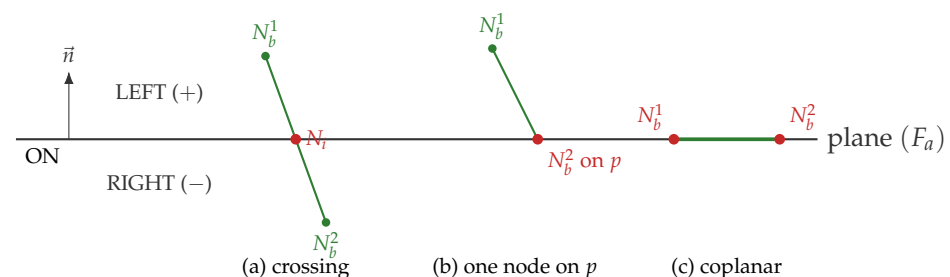


Figure 5. Classification of an edge E_b of B against the supporting plane of F_a . The facet normal $\vec{n}$ defines the LEFT (+), RIGHT (−), and ON half-spaces. (a) The two nodes lie on different sides; a single intersection node N_i (red) is created and anchored. (b) One node lies on the plane and is anchored directly. (c) Both nodes lie on the plane; the edge is coplanar and is intersected against the boundary edges of F_a .

Step 1.1 (Check E_b against F_a).

For each edge E_b of B and each face F_a of A , the two nodes of E_b are classified against the supporting plane p of F_a (Figure 5). Three cases arise. *Both nodes on different sides of p* : the edge crosses the plane; the intersection point N_i is computed and stored for E_b , and its anchor on F_a is determined via the client zone localization (Figure 3). The rank of the anchor's domain fixes the element of A to which N_i belongs: a node N_a , an edge E_a , or the face interior F_a . If the anchor line is shorter than ε , N_i is snapped to that element, and μ_1 , μ_2 are updated. *One node on p* : the node N_b lies on the plane of F_a ; its anchor is checked against elements of F_a and mapped via μ_3 . *Both nodes on p (coplanar edge)*: for each edge E_a of F_a , the segments E_b and E_a are tested, producing zero or one intersection point or an overlap segment; each N_i is stored

and anchored, and the endpoints N_b are anchored against F_a and mapped if they lie inside or on the boundary. All results are checked for non-manifold configurations.

When E_b is an unbounded edge (a ray), only its real node is classified against p ; the side toward which the ray continues is determined from the sign of $\mathbf{n}^\top \mathbf{d}$, where $\mathbf{n}$ is the plane normal and $\mathbf{d}$ the direction vector of the ray. A crossing exists when the resulting parameter t is non-negative, so the same plane-crossing logic applies as for a bounded edge (see Figure 6).

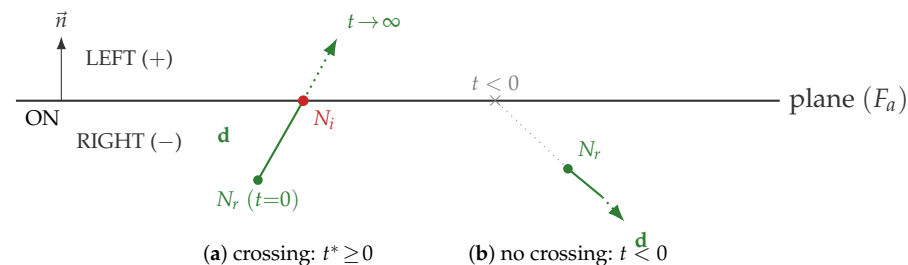


Figure 6. Intersection of an unbounded edge (ray) with the supporting plane of F_a . Colors: green marks the ray of B (origin N_r , direction $\mathbf{d}$), red the intersection node N_i , and gray the auxiliary backward supporting line; the black arrow $\vec{n}$ is the plane normal. The ray has a real origin N_r ($t = 0$) and direction $\mathbf{d}$, parametrized as $P(t) = \mathbf{x}_{N_r} + t \mathbf{d}$. The plane crossing is computed exactly as for a bounded edge, using the facet normal $\vec{n}$ and the LEFT (+), RIGHT (-), and ON regions. (a) The intersection parameter t^* is non-negative: the ray crosses the plane and an intersection node N_i (red) is created. (b) The supporting line meets the plane at $t < 0$, behind the origin: the crossing is rejected. The only difference from a bounded edge is the valid parameter range, $t \in [0, \infty)$ instead of $t \in [0, 1]$.

Step 1.2 (Check E_a against F_b): The symmetric operation: for each edge E_a of A and each face F_b of B , the two nodes N_a are classified against the supporting plane (F_b). *Both nodes on the same side:* no intersection. *Both nodes on different sides:* the intersection point N_i is computed and stored for F_b ; its anchor is checked against N_a and E_a of A (the intersected element belongs to A , so anchoring determines which element of A the new node lies on), and μ_2 is updated. *One node on p :* the node N_a is stored for F_b , recording that this node of A lies on a face of B . Intersection points geometrically identical (within ϵ) to points already found in Step 1.1 are skipped to avoid duplicates.

Step 1.3 (Build intersection edges and assign residual nodes). For each face pair (F_a, F_b) where intersection nodes have been found on both F_b and E_b , the intersection edges E_i are constructed, as shown in Figure 7. All N_i of the face pair are sorted along the intersection line $\text{plane}(F_b) \cap \text{plane}(F_a)$, consecutive sorted nodes are paired to create intersection edges E_i assigned to F_a , and μ_4, μ_5 are updated. Nodes mapped to an edge E_a act as *joints* connecting E_i segments across adjacent face groups, since E_a is shared between two faces of A ; nodes mapped to a node N_a serve as joints for all face groups meeting at N_a . These multi-valent joints are essential for constructing continuous intersection edge chains across face boundaries. Finally, every node N_b of B that remains unmapped is assigned to the cell of A that contains it; since A is a complete space partition, each residual node lies in exactly one cell (bounded or unbounded), and μ_3 is updated accordingly.

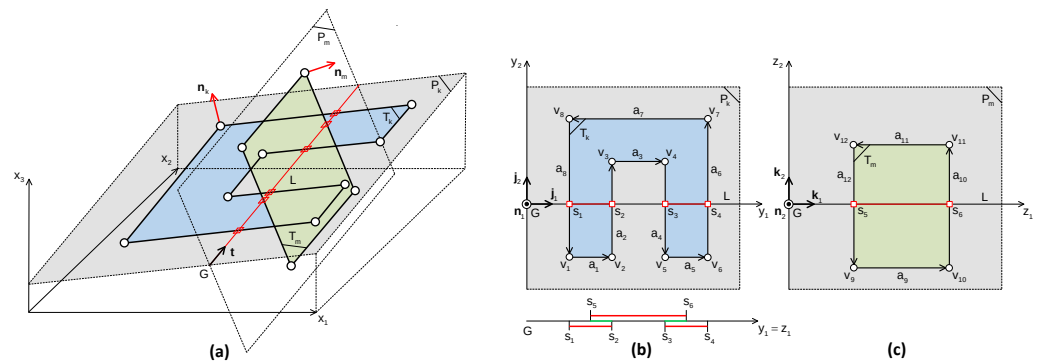


Figure 7. Intersection of two planar faces and the 1D sorting step. The colored arrows are the plane normals $\mathbf{n}_k$ and $\mathbf{n}_m$ of the two faces; red marks the intersection nodes and segments on the common line L . (a) 3D view: faces T_k and T_m lie in planes P_k and P_m with normals $\mathbf{n}_k$ and $\mathbf{n}_m$; their common line L carries the intersection segments. (b) In-plane view of T_k . (c) 1D sorting strip: nodes along L are sorted to build intersection edges $s_1 \dots s_6$. This local two-dimensional reduction is the geometric basis of the face-by-face comparison in Phase 1.

4.5. Phase 2: Collecting Split Data

With all maps populated, Phase 2 assembles the split data required to execute modifications on the kernel. The geometric basis of the comparison is the intersection of two planar faces, which reduces to a one-dimensional problem along their common line. This phase is the bridge between the generic intersection geometry, shown in Figure 7 for two faces of A and B , and the split operations of the kernel: it routes every element of B to a split of the correct rank, according to the element of A on which it was found, and assigns the names and identifiers that make each split traceable. The split data is collected incrementally in six steps through three passes (Algorithm 2). The first pass starts from node mappings (where nodes of B landed in A) and works upward; the second pass uses edge-level information; the third pass handles remaining faces. This multi-pass structure ensures that the split data for each domain is complete before execution.

Step 2.1 (Cell splits: nodes in cells) For every node N_b of B with $\mu_3(N_b) = C_a$, the topology of B is traversed from N_b : adjacent edges E_b are followed and adjacent faces F_b collected, until elements mapped to the boundary of C_a are reached. The collected face fragments of B and the intersection edges E_i on the boundary faces of C_a together form the interior portion of the splitting surface for C_a .

Figure 8 summarizes how each element of B is routed, by the rank of the element of A on which it was found, into an edge, face, or cell split.

Step 2.2 (Face splits: nodes in faces). For every node N_b with $\mu_3(N_b) = F_a$, the adjacent edges of B emanating from N_b that lie on F_a are collected. Together with intersection edges E_i mapped to F_a (via μ_5), they contribute to the splitting chains that divide F_a into sub-faces.

Step 2.3 (Edge splits: nodes in edges): For every node N_b with $\mu_3(N_b) = E_a$, the edge split is constructed. Intersection nodes N_i mapped to E_a (via μ_2) are included. All such nodes are sorted by their parameter t along E_a ($t = 0$ at the start node, $t = 1$ at the end node), and sub-edges are defined between consecutive nodes: $[N_{\text{start}} \rightarrow N_1]$, $[N_1 \rightarrow N_2]$, $\dots$, $[N_k \rightarrow N_{\text{end}}]$, each assigned a unique identifier. This is the only pass required for edge splits, as all relevant nodes are known after Phase 1.

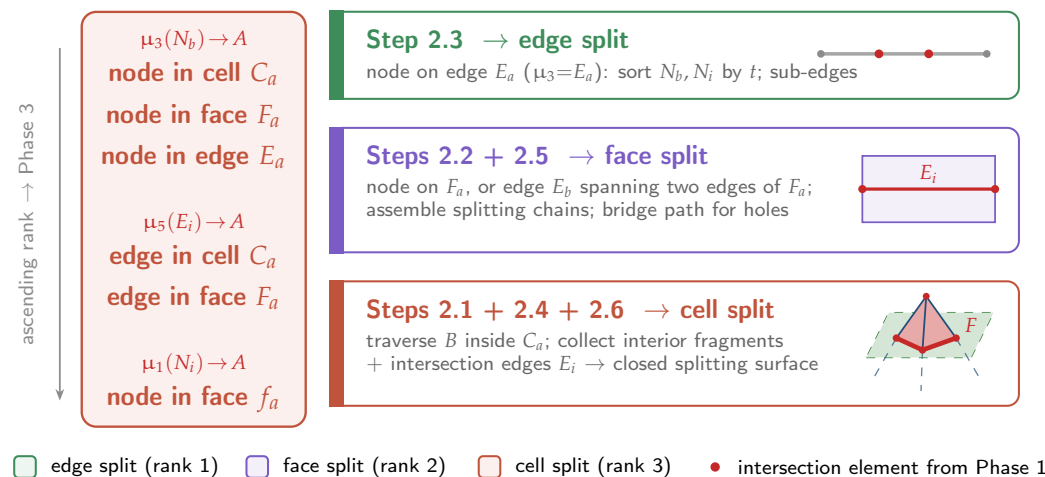


Figure 8. Detail of Phase 2: routing mapped elements to split types. The rank of the element of A to which each node of B was mapped in Phase 1 decides which kind of split it contributes to. **(green)** Nodes on an edge feed the edge split (Step 2.3). **(purple)** Nodes on a face, and edges spanning two intersections of a face, feed the face split (Steps 2.2 and 2.5, with a bridge path for faces that have holes). **(red)** Nodes inside a cell, its adjacent edges spanning faces, and boundary-only faces feed the cell split (Steps 2.1, 2.4, and 2.6). The split data is assembled iteratively so that it can be executed in ascending rank order in Phase 3.

Step 2.4 (Cell splits: edges with both nodes on faces): For every edge E_b of B whose two endpoints are both mapped to faces of A that bound a common cell C_a , the adjacent faces of B along E_b are collected. These faces contribute additional splitting surface fragments for the cells adjacent to those faces. This pass captures geometry from edges of B that span between two boundary faces of a cell without passing through its interior (and thus were not captured in Step 2.1).

Step 2.5 (Face splits: edges with both nodes on edges): For every edge E_b of B whose two endpoints are both mapped to edges of A , the face split is constructed. Together with intersection edges E_i and the data from Step 2.2, the complete splitting chains for each affected face F_a are assembled. For a simply connected face, a single chain crossing from one boundary edge to another divides F_a into two sub-faces. For a multiply connected face (with inner boundary loops representing holes), connecting the outer boundary to an inner boundary with a single chain would produce a non-manifold configuration: at the connection point, four boundary edges would meet at one node. Instead, two directed chain segments are required, forming a closed *bridge path* that opens the hole boundary into the outer boundary, converting the multiply connected face into two simply connected sub-faces. Inner boundary intersections are always resolved before outer boundary intersections, so that the sub-edge identifiers required for the bridge path are available. Each new sub-face and boundary edge is assigned a unique identifier.

Step 2.6 (Cell splits: remaining faces in cells): For every face F_b of B not yet assigned to a cell split in Steps 2.1 or 2.4, the cell split is constructed. This pass handles the boundary-only case: no node N_b lies inside C_a , but faces of B lying on the boundary of C_a , together with intersection edges and coincident elements, form a closed splitting surface. This occurs in touching or coplanar configurations where the surface of B lies entirely on the boundary of A .

After all six steps, the complete split data for every affected edge, face, and cell of A has been assembled. The watertightness of B guarantees that every cell splitting surface is closed.

4.6. Phase 3: Execute Split Work Steps

Phase 3 is where the partition platform performs the actual modification of A . The split work steps assembled in Phase 2 are executed on the topological kernel in strictly ascending dimension order: edges first, then faces, then cells. This ordering is mandatory because face splits reference sub-edge identifiers created during edge splits, and cell splits reference sub-face and sub-edge identifiers created in earlier steps. Because the kernel modifies the partition only through split operations, and each split produces valid sub-domains from a valid domain, the partition stays manifold and gapless after every individual step; the geometric and topological validity checks of the kernel confirm this and report any ambiguous configuration such as non-manifold edges (see Figure 9) to the user.

Step 3.1 (Edge splits): For each affected edge E_a , the edge split work step is executed using the sorted intersection points from Step 2.3. The kernel creates new sub-edges with the assigned identifiers and updates all arrow and arrow bundle references in the adjacent facets and voxels.

Step 3.2 (Face splits): For each affected face F_a , the face split work step is executed using the splitting chains assembled in Steps 2.2 and 2.5, with sub-edge identifiers from Step 3.1. Inner boundary splits are executed before outer boundary splits. The kernel creates new sub-faces and updates topology.

Step 3.3 (Cell splits): For each affected cell C_a , the cell split work step is executed using the splitting surface assembled in Steps 2.1, 2.4, and 2.6, combined with the face and edge identifiers from Steps 3.1 and 3.2 and the topology elements of B . The kernel creates two new sub-cells and updates all adjacency information.

After all splits, an assignment procedure identifies which newly created sub-domains belong to B . The split operations alone only subdivide existing domains of A ; they do not record which resulting sub-domains correspond to the interior or boundary of B . Starting from surface elements directly attributable to B (intersection edges, face fragments from B 's boundary), a breadth-first search propagates the assignment to adjacent domains, yielding a complete identification of the cells, faces, edges, and nodes that constitute the embedded representation of B . This assignment is a prerequisite for the optional merge step and for any subsequent query that refers to B as an entity within the partition.

Step 3.4 (Merge, optional). Optionally, a merge procedure consolidates multiple adjacent domains belonging to B into single domains, restoring a compact representation. Merges are executed in descending dimension order (cells first, then faces, then edges) to maintain consistency. Whether to merge is a user decision: a merged representation is more compact, while an unmerged representation retains the fine-grained traceability of every split operation.

4.7. Existence of Solution and Non-Manifold Detection

A valid input configuration does not guarantee that B can be embedded into A while maintaining the manifold property of all resulting domains. A non-manifold result can arise when a split would produce a domain whose boundary touches itself. For example, a node contact of a face B , where no edges connected to that node are in contact, leads to a non-manifold configuration as shown in Figure 9. The same situation occurs with edge contacts, where no faces around that edge are adjacent.

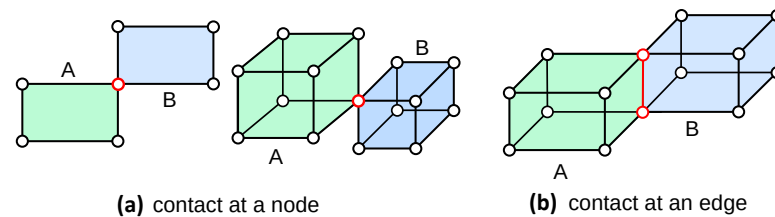


Figure 9. Non-manifold configurations arising from lower-rank boundary contact. A valid face of domain *B* (green) can create non-manifold configurations when it coincides with another boundary object only at a lower-rank domain. The label *B* marks the input domain (green); blue marks the elements of the existing partition *A* that are touched. (a) Node contact: *B* touches a node of *A* but no edge connected to that node is in contact, producing a non-manifold node. (b) Edge contact (red): *B* touches an edge of *A* but no face around that edge is adjacent, producing a non-manifold edge. Both configurations are detected during Phase 2 and reported to the user.

The algorithm detects such configurations during Phase 2 when constructing the splitting chains for faces. If a splitting chain cannot be completed without producing a non-manifold domain, the algorithm reports the condition to the user together with the geometric location of the problematic configuration. The user must then modify the input, for instance, by adjusting *B*, before the embedding can proceed. This design choice prioritizes correctness and user control over silent geometric heuristics.

5. Prototype Implementation and Validation

5.1. Implementation Overview

The embedding algorithm was implemented as part of a prototype space partitioning application developed in Java. The topological kernel is the consistent spatial modeling kernel of Sternal [5], which provides the arrow, facet, and voxel structures and the split and merge work steps used throughout this paper; it was extended to support the five maps and the split data structures required by the embedding algorithm. The original kernel was built on Java 17 LTS [5]. For the present work, the implementation was upgraded to Java 25 LTS and JMonkeyEngine 3.9 for visualization.

The upgrade serves two purposes. First, the prototype must respond to user interactions and, at the same time, run the geometric and topological computations of the embedding algorithm and update the three-dimensional view. To keep these concerns separate without manual thread management, the implementation uses the structured concurrency model, which is available as a preview feature in Java 25 through the `StructuredTaskScope` API [37]. Each user-triggered embedding runs as a structured task scope: the intersection detection of Phase 1, the split data construction of Phase 2, and the view update are launched as child tasks of one scope, so that they either all complete or are all cancelled together, and an error in any one of them is propagated back to the user interface in a controlled way. Second, modern Java language features (records for the immutable split data items, sealed types for the domain hierarchy, and pattern matching for the rank-based routing of Phase 2) make the kernel code shorter and closer to the formal description of the algorithm.

It must be stressed that the prototype is a research testbed, not a production tool. Its purpose is to test the topological algorithms, to support graph analysis on the resulting partition, and to study geometrically robust modeling; performance optimization and a production-quality user interface were not objectives of this work. Three-dimensional visualization is provided through JMonkeyEngine, which renders both the user model and the internal core model and supports the inspection of adjacency relationships [5].

5.2. Test Configuration

To validate the algorithm systematically, a standardized space partition A was defined as the modeling environment for the input domain B . The configuration models a simple but topologically rich building scene: a wall with a door and a window, a room behind the wall, a second cell that hovers inside the room, and the exterior split into air above the ground and soil below it. It consists of five bounded cells and two unbounded cells, listed below and shown in Figure 10.

The bounded cells are:

- A_w : a wall slab standing in front of the room. Two openings are cut through its thickness, so the wall is non-convex and its front face is multiply connected.
- A_o : the window cell, a box of the same thickness as the wall that fills the upper opening. It is enclosed by wall material on all four sides, so it represents a multiply connected (hole) configuration; its back face touches the room A_r .
- A_d : the door cell, a box of the same thickness as the wall that fills the lower opening. It is open at the floor edge, so it represents a non-convex notch in the wall boundary; its back face touches the room A_r .
- A_r : the room, a bounded cell directly behind the wall. The back faces of A_w , A_o , and A_d all touch A_r .
- A_i : an inner cell that hovers fully inside A_r without touching any of its faces. The boundary of A_i therefore forms an internal cavity in A_r , which makes A_r multiply connected in three dimensions.

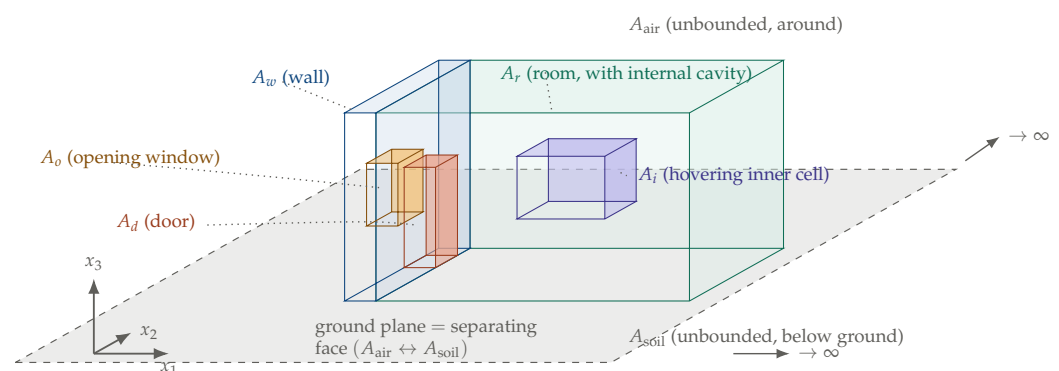


Figure 10. Standard test space partition A (isometric view). Each cell is drawn in its own color, and each label uses the color of its cell: blue wall A_w , green room A_r , purple inner cell A_i , orange window A_o , red-orange door A_d ; gray marks the ground plane, the coordinate axes, and the unbounded cells. A wall A_w stands in front of the room A_r ; the window cell A_o is enclosed on all sides by wall material (a multiply connected hole), while the door cell A_d is open at the floor edge (a non-convex notch). Both touch the room with their back faces. The inner cell A_i hovers fully inside A_r , forming an internal cavity. The exterior is split by the unbounded ground plane face into the unbounded air cell A_{air} above and the unbounded soil cell A_{soil} below.

The space outside the building is unbounded. As stated in Section 3.1, the unbounded space surrounding a bounded object cannot be represented by a single unbounded cell; at least two unbounded cells are always required to cover the exterior [4,5]. The exterior is therefore divided by the ground plane, which acts as an unbounded separating face, into two unbounded cells:

- A_{air} : the air cell above the ground plane, surrounding the building and extending to infinity.
- A_{soil} : the soil cell below the ground plane, extending to infinity downward.

Together, the seven cells form a complete and gap-free space partition of Euclidean space.

The input domain B is a polyhedron placed at parameterized positions and orientations relative to A . This single configuration was chosen because it exercises, in one model, every class of relationship that the embedding algorithm must handle. Table 5 maps each class to the specific pair of cells or the specific feature that tests it.

Table 5. Topological coverage of the standard test configuration. Each class of relationship handled by the embedding algorithm is exercised by a specific pair of cells or a specific feature of A .

Class of Relationship	Where it Occurs in the Test Configuration
Bounded $\leftrightarrow$ bounded	wall–room ($A_w \leftrightarrow A_r$), wall–door ($A_w \leftrightarrow A_d$), wall–window ($A_w \leftrightarrow A_o$)
Bounded $\leftrightarrow$ unbounded	wall–air ($A_w \leftrightarrow A_{\text{air}}$), room–soil contact at the ground plane ($A_r \leftrightarrow A_{\text{soil}}$)
Unbounded $\leftrightarrow$ unbounded	air–soil across the ground plane ($A_{\text{air}} \leftrightarrow A_{\text{soil}}$)
Non-convex domain	wall A_w with the door notch open at the floor edge
Multiply connected face	front face of the wall A_w around the window hole A_o
Multiply connected cell (cavity)	room A_r with the hovering inner cell A_i as internal cavity
Fully enclosed bounded cell	inner cell A_i , whose entire boundary touches only A_r

5.3. Test Case Matrix

The test cases were organized along three classification dimensions: the volumetric relationship between B and the cells of A (nine categories 0–8), the boundary contact type (seven types: no contact, face-on-face, edge-on-edge, node-on-node, face-on-edge, face-on-node, edge-on-node), and which boundary of A is affected (the outer wall–air boundary, the room–inner-cavity boundary, the window hole boundary, and the ground plane). Table 6 summarizes the nine volumetric categories. Where a category refers to the unbounded cell, B is placed in the air cell A_{air} adjacent to the wall.

Table 6. Volumetric relationship categories in the test case matrix, with prototype implementation status. Categories 0, 3, 4, and 7 were implemented and verified; the remaining categories are defined in the generator and scheduled for the regression suite.

Cat.	Description	Key Challenge	Status
0	B entirely in A_{air}	Cell split of one unbounded cell	Implemented
1	B identical to a bounded cell of A	Boundary-only cell splits	Pending
2	B touching but not crossing a boundary	ON cases, point contact	Pending
3	B inside a bounded cell	Cavity creation	Implemented
4	B crossing the wall–air boundary ($A_w \leftrightarrow A_{\text{air}}$)	Edge/face/cell splits across boundary	Implemented
5	B crossing the room–cavity boundary ($A_r \leftrightarrow A_i$)	Splits on two adjacent cells	Pending
6	B crossing the window hole boundary	Multiply connected face handling	Pending
7	B crossing wall, room, and cavity	Several adjacent cells split	Implemented
8	B contains a bounded cell of A	Cell split of A_{air}	Pending

The full matrix comprises over 40 individual test cases spanning nine volumetric categories (0–8). To create these cases reproducibly, a dedicated test case generator was used. The generator places the input domain B at parameterized positions and orientations relative to the standard configuration of A , so that each category of the matrix can be populated systematically and the same case can be reproduced for regression testing. In the current prototype, selected representative cases from categories 0, 3, 4, and 7 were implemented and the topological correctness of the resulting partition was verified. Full coverage of the matrix is ongoing work and is being incorporated into the broader regression suite of the partition platform.

The selection of categories 0, 3, 4, and 7 for the initial implementation was deliberate. Together, they exercise the four structurally distinct mechanisms of the algorithm in increasing difficulty: Category 0 is the minimal case (a single cell split of an unbounded cell with no edge or face splits); Category 3 introduces cavity creation and exercises face and cell splits without any boundary crossing; Category 4 is the first true boundary-crossing case, exercising coordinated edge, face, and cell splits across two adjacent cells; and Category 7 is the hardest validated case, in which several adjacent cells are split in a single operation spanning bounded and unbounded domains. This subset spans the algorithm's core behaviours from the trivial split to multi-cell boundary crossing, providing the strongest evidence of generality for the least redundant verification effort. The remaining categories (1, 2, 5, 6, 8) exercise the same split mechanisms in different boundary-contact configurations and are being added to the platform's automated regression suite; their complete, automated validation will be reported—together with the source code and test case generator—in a dedicated follow-up publication.

Validation was not based solely on visualization. For each implemented case, the resulting space partition was examined against the kernel invariants targeted by the algorithm, namely, the preservation of a connected manifold space partition, gap-free coverage of space, and consistent adjacency relations after the execution of the embedding steps. In all validated cases, these checks were satisfied without post hoc geometric repair. The selected results are described in the following subsection.

5.4. Selected Results

Results for four representative test cases that exercise progressively more complex aspects of the algorithm are presented below.

5.4.1. Category 0: B Entirely in the Unbounded Cell (T0.1)

The simplest case: B is a box entirely within the air cell A_{air} , with no contact with any bounded cell. Phase 1 maps all nodes of B to A_{air} . No edge or face splits are required in A . Phase 3 executes a single cell split of A_{air} using the surface of B as the splitting surface. The result is a new bounded cell (the interior of B) and a modified unbounded cell. Figure 11 shows the input and the result of this worked example.

5.4.2. Category 3: B Inside the Room A_r (T3.1)

B is positioned entirely inside the room A_r with no contact with any boundary of A . Phase 1 detects intersections between edges of B and faces of A_r , producing intersection nodes and edges on the boundary faces of A_r . Phase 2 constructs face split data for the affected faces and cell split data for A_r . Phase 3 first splits the affected edges and faces, then splits A_r into two cells: one corresponding to the interior of B and one to the remainder.

5.4.3. Category 4: B Crossing the Wall–Air Boundary (T4.1)

B straddles the wall–air boundary, partially inside the wall A_w and partially in the air cell A_{air} . Phase 1 produces intersection nodes on the edges of B that cross faces of the wall boundary, and on edges of A_w that cross faces of B . Intersection edges are built on the affected faces. Phase 2 constructs edge split data, face split data (the wall face through which B passes is split into sub-faces), and cell split data for both A_w and A_{air} . Phase 3 executes all splits. The result is that both A_w and A_{air} are each split into two cells, and the resulting space partition preserves the intended topological embedding of B under the implemented consistency checks.

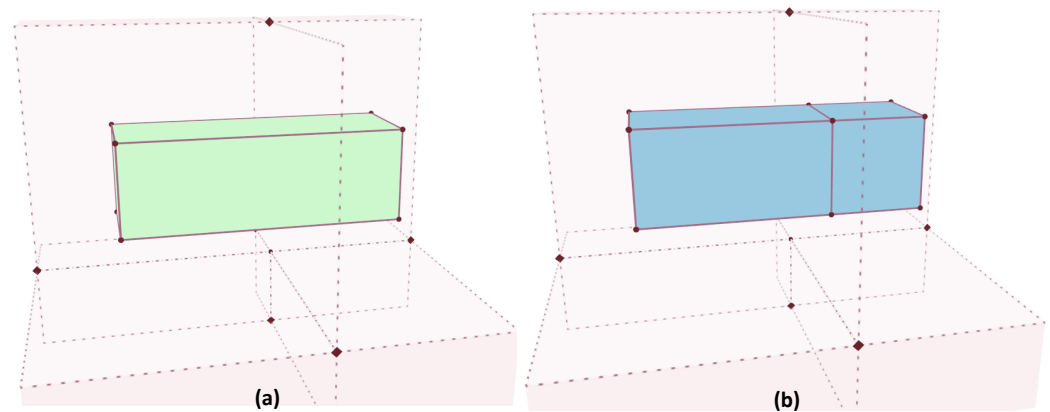


Figure 11. Worked example of test case T0.1: embedding B into the unbounded cell A_{air} . (a) Input: the box B (green) placed inside A_{air} (light red), before embedding. (b) Result: after embedding, A_{air} has been split through the boundary surface of B into two cells (blue)—one corresponding to the interior of B and one to the remaining unbounded cell. The resulting space partition satisfies the topological consistency checks used in the prototype.

5.4.4. Category 7: B Crossing Wall, Room, and Cavity (T7.1)

The most complex validated case: B spans the air cell A_{air} , the wall A_w , and the room A_r , crossing the wall–air boundary (between A_{air} and A_w) and the room boundary (between A_w and A_r). Phase 1 produces intersection nodes and edges on multiple faces of A . Phase 2 constructs split data for edges and faces on both boundaries, and cell split data for the three cells A_{air} , A_w , and A_r . Phase 3 executes the splits in order: edges, then faces on the wall–air boundary, faces on the room boundary, then the three cell splits. The result is a refined space partition in which each affected cell is subdivided and the sub-domains corresponding to B are consistently identified across all three cells under the implemented topological checks.

6. Discussion

6.1. General Approach

This research aims to bridge the gap between design functionalities—as they are currently used in state-of-the-art applications—and topological operators, which ensure a consistent model. Based on the results presented, it can be concluded that this goal is achievable. The proof that the chosen approach covers all cases is currently being developed. To this end, all necessary split operations that may result from intersecting any domains are compiled in parallel. All of these split operations must appear somewhere in the selected approach so that completeness can be verified.

6.2. Comparison to Existing Boolean Engines

The embedding algorithm differs from classical Boolean engines in several fundamental ways. First, it operates on a complete space partition rather than individual bounded objects. The partition invariant (every point belongs to exactly one cell) is maintained throughout the computation and is never temporarily violated. Second, the modification of A occurs exclusively through split operations. Unlike mesh Boolean libraries that construct new geometry from intersection curves and retriangulate affected regions, the algorithm names every new topological entity within a split operation, providing full traceability from input to result. Third, B is treated as a read-only input; it is not consumed or modified by the operation. Fourth, ambiguous configurations, including near-coincident geometry and non-manifold potential, are detected and reported to the user rather than resolved through internal heuristics. This design philosophy trades full automation for transparency

and correctness, which is appropriate for applications in the built environment where errors in spatial models can propagate into physical construction.

Compared to CGAL Nef polyhedra, which guarantee exact arithmetic for individual Boolean operations, the embedding algorithm operates with floating-point coordinates and a tolerance-based classification. This is a deliberate choice for the target domain, where input geometry originates from BIM models and surveying data with inherent measurement uncertainty. The two-threshold strategy (ϵ for snap, δ for warning) provides a pragmatic middle ground between exact computation and robustness in the presence of imprecise input.

6.3. Distinction from the 2D Design Step Prototype

The parametric design operators introduced in the conference paper [10], namely extrusion, slicing, and primitive placement, can be understood as specific instantiations of the general embedding algorithm. An extrusion generates a polyhedral body B from a profile curve and a direction vector; a primitive placement creates a predefined polyhedral B at a specified location; a slicing operation generates a planar face B that cuts through existing cells. Each of these generates a specific B and invokes the embedding algorithm to integrate it into the partition. The conference paper demonstrated this mapping for the two-dimensional case; the present algorithm generalizes it to three dimensions and handles the full complexity of boundary contact types, multiply connected faces, and unbounded elements. This three-dimensional generalization is what makes every parametric design step usable on real volumetric building and city models: the operation handles the complete set of boundary-contact types (face-on-face, edge-on-face, node-on-face, edge-on-edge, and node-on-edge contacts), multiply connected faces and cells (including holes and internal cavities), and unbounded cells as first-class entities throughout. Consequently, it provides the single general operation on which extrusion, slicing, primitive placement, and any future design-step operator can be built in true 3D, while guaranteeing topological consistency by construction at every intermediate step. This is the central contribution of the present work: not merely a lifted version of the 2D algorithm, but the foundation for a consistent three-dimensional spatial modeling pipeline for the built environment.

6.4. Processing Status and Future Work

The paper describes the current status of the investigation. First, the test case matrix has been only partially validated in the prototype implementation. While the tested categories cover the most important structural cases (no contact, cavity creation, boundary crossing, both-boundary crossing), the boundary-only and degenerate-contact cases (categories 1 and 2), the inner-boundary crossing (category 5), the hole-boundary crossing (category 6), and the enclosing case (category 8) require further testing. Second, the computational performance of the algorithm has not been characterized. The current naïve implementation of Phase 1 performs a pairwise face comparison between all faces of A and all faces of B , which has quadratic cost $O(|F_A| \cdot |F_B|)$ in the number of faces; for large BIM or GIS partitions with many hundreds or thousands of faces, this brute-force approach becomes the primary bottleneck.

Concrete acceleration strategies that are suitable for this data model are: (a) *axis-aligned bounding-box (AABB) pre-filtering*: the axis-aligned bounding boxes of each face of A and each face of B are computed once, and only face pairs whose bounding boxes overlap are forwarded to the exact intersection test, eliminating the large majority of pairs at negligible cost; (b) *bounding-volume hierarchies (BVHs)*: a BVH is built over the faces of A , exploiting the explicit adjacency structure already stored in the partition kernel, so that all faces of A whose bounding volume does not overlap the bounding volume of a face of B are pruned in

logarithmic time; (c) *spatial hashing*: the faces of A are hashed into a regular grid of cells, and each face of B is tested only against faces of A in the same or adjacent grid cells; (d) *R-tree indexing*: a persistent R-tree over the faces of A supports range queries for each face of B , and is particularly efficient when the same partition A is reused across many sequential embedding operations. Among these, the bounding-box pre-filter combined with a BVH is the most promising first step, because the partition already maintains explicit adjacency between faces (through the arrow bundle structure), which allows a BVH to be constructed incrementally and updated locally after each split operation without a full rebuild. Full computational performance characterization—including face counts, comparison counts, per-phase timing, memory usage, and scaling behaviour—is deferred to a dedicated follow-up software publication, which will also release the optimized implementation and test-data generator.

Third, the algorithm is restricted to polyhedral (flat-faced) geometry. Curved surfaces, which are common in architectural design, are not currently supported. Fourth, semantic information, such as material properties, functional classifications, and identifiers from BIM models, is not addressed by the algorithm. Connecting the topological partition to semantic data models such as IFC or BOT [38] remains future work.

Further development targets include: proof that the chosen approach covers all possible cases, completion of the test case matrix with automated verification, integration of the embedding algorithm with the parametric design layer to create a full three-dimensional modeling pipeline, investigation of novel design functionalities that exploit the contact-face topology of the partition (as discussed in [10]), and export interfaces to standard formats including IFC, CityGML, and IndoorGML for interoperability with existing AEC workflows.

7. Conclusions

This paper presented a general embedding algorithm for Boolean intersection in polyhedral space partitions. The algorithm inserts an arbitrary watertight polyhedral boundary representation B into an existing space partition A , which constitutes a manifold, gapless partition of Euclidean space including unbounded cells, through a structured sequence of split operations organized in three phases: intersection detection with tolerance-based classification (Phase 1), construction of split data (Phase 2), and kernel execution in ascending dimension order (Phase 3). Unbounded edges are treated as rays and intersected directly by the geometric kernel, so the elements at infinity require no separate virtualization step.

The key property of the algorithm is that it guarantees topological consistency by construction. Because the space partition is modified exclusively through split operations, and each split produces valid sub-domains from a valid domain, the partition invariant is preserved after every operation. No separate post hoc geometric repair pass is required; instead, validity is verified incrementally at each split. Ambiguous or non-manifold configurations are detected during the algorithm and reported to the user rather than resolved silently.

The algorithm generalizes the parametric design operators presented in the preceding conference paper to the full three-dimensional case and provides the theoretical foundation for a consistent spatial modeling approach for the built environment. This work benefits practitioners and tool developers across the built environment—BIM and GIS modelers, digital-twin and city-information-model developers, and the builders of energy simulation, indoor and outdoor navigation, and 3D cadastre applications—who require spatial models that are guaranteed gap-free and manifold rather than repaired after the fact. A systematic test case matrix with nine volumetric categories was developed and the selected cases were validated using a prototype implementation in Java. The results confirm that the algorithm

correctly handles configurations ranging from simple cavity creation to multi-boundary crossing scenarios.

Future work will focus on finishing the proof that the approach covers all configurations, completing the test case validation, characterizing computational performance, and connecting the topological partition to semantic data models for interoperability with BIM and GIS standards. Oswald and Calliku [39], who explored embedding GIS terrain data into a three-dimensional space partition to extend architectural modeling towards the parcel and city scale, took a first step toward GIS integration on this platform.

Author Contributions: Conceptualization, M.S.; methodology, M.S. and W.H.; software, M.S.; validation, A.C. and M.S.; writing—original draft preparation, M.S.; writing—review and editing, A.C. and W.H.; supervision, W.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant numbers 429900376 and 578449518. The theoretical background and the implementation of the space partitioning platform were developed in the project “Polyhedral Space Partition for the Digital Modeling of Topology and Geometry” (PPSpace, DFG project number 429900376), funded from 2020 to 2024. The research presented in this paper is carried out as preparation of the follow-up project “Consistent Construction Functionalities with Multiply Linked Semantics on the Basis of a Polyhedral Space Partition” (PPSpace-Design, DFG project number 578449518), funded by the DFG since 2026.

Data Availability Statement: The prototype implementation, test case generator, and validated test case data presented in this study are being prepared for public release as part of the broader space-partition platform. They will be published separately in a dedicated follow-up software publication. Until then, these materials are available from the corresponding author upon reasonable request.

Acknowledgments: The authors would like to thank Peter Jan Pahl for his dedicated work in developing the theory of the space partitioning approach as part of the PPSpace project. The authors acknowledge the use of generative AI tools (Claude, Anthropic) for assistance with manuscript drafting and language editing using DeepL for translations. All scientific content, algorithmic design, and implementation were performed by the authors.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. ISO/TC 211; ISO 19107:2019. Geographic Information—Spatial Schema. International Organization for Standardization: Geneva, Switzerland, 2019.
2. ISO 16739-1:2018; Industry Foundation Classes (IFC) for Data Sharing in the Construction and Facility Management Industries—Part 1: Data Schema. International Organization for Standardization: Geneva, Switzerland, 2018.
3. Nehring-Wirxel, J.; Trettner, P.; Kobbelt, L. Fast Exact Booleans for Iterated CSG using Octree-Embedded BSPs. *Comput.-Aided Des.* **2021**, *135*, 103015. <https://doi.org/10.1016/j.cad.2021.103015>.
4. Huhnt, W.; Sternal, M.; Pahl, P.J. Modeling bounded and unbounded space with polyhedra: Topology and operators for manifold cell complexes. *Adv. Eng. Inform.* **2022**, *54*, 101790. <https://doi.org/10.1016/j.aei.2022.101790>.
5. Sternal, M. Polyhedral Partition of the Euclidean Space for Consistent Spatial Modeling of the Built Environment. Ph.D. Thesis, Technische Universität Berlin, Berlin, Germany, 2025. <https://doi.org/10.14279/depositonce-23869>.
6. Sternal, M.; Huhnt, W. Robust Modeling of Polyhedral Space Partitions. In *Advances in Information Technology in Civil and Building Engineering*; Skatulla, S., Beushausen, H., Eds.; Lecture Notes in Civil Engineering; Springer: Cham, Switzerland, 2023; Volume 358, pp. 427–442. https://doi.org/10.1007/978-3-031-32515-1_30.
7. Huhnt, W.; Vetter, J.Z.; Sternal, M. Space Partitioning as a Holistic Alternative to Traditional Geometric Modeling Workflows in the AEC Industry. In *Advances in Information Technology in Civil and Building Engineering*; Skatulla, S., Beushausen, H., Eds.; Lecture Notes in Civil Engineering; Springer: Cham, Switzerland, 2023; Volume 358, pp. 415–426. https://doi.org/10.1007/978-3-031-32515-1_29.
8. Vetter, J.Z. Transformation of Digital Building Models from Boundary Representation to Tetrahedral Space Partitioning. Ph.D. Thesis, Technische Universität Berlin, Berlin, Germany, 2025. <https://doi.org/10.14279/depositonce-24035>.

9. Vetter, J.Z.; Huhnt, W. Transforming Building Components into a Space Partition to Identify Indoor and Outdoor Spaces in Digital Building Models. In *Advances in Information Technology in Civil and Building Engineering*; Skatulla, S., Beushausen, H., Eds.; Lecture Notes in Civil Engineering; Springer: Cham, Switzerland, 2023; Volume 358, pp. 403–413. https://doi.org/10.1007/978-3-031-32515-1_28.
10. Sternal, M.; Oswald, S.; Huhnt, W. First Investigations of Advanced Operators for Consistent Spatial Modeling of the Built Environment. In *Advances in Information Technology in Civil and Building Engineering*; Francis, A., Miresco, E., Melhado, S., Eds.; Lecture Notes in Civil Engineering; Springer: Cham, Switzerland, 2025; Volume 628, pp. 54–67. https://doi.org/10.1007/978-3-031-84208-5_5.
11. Yang, X.; Koehl, M.; Grussenmeyer, P. Mesh-to-BEM: Parametric gbXML reconstruction for BIM-based building energy modeling. *Autom. Constr.* **2022**, *140*, 104372. <https://doi.org/10.1016/j.autcon.2022.104372>.
12. Massafra, A.; Gulli, R. Enabling Bidirectional Interoperability between BIM and BPS through Lightweight Topological Models. In Proceedings of the 41st Conference on Education and Research in Computer Aided Architectural Design in Europe (eCAADe 2023), Graz, Austria, 20–23 September 2023; Volume 2, pp. 187–196. <https://doi.org/10.52842/conf.ecaade.2023.2.187>.
13. Massafra, A.; Jabi, W.; Gulli, R. Topological BIM for building performance management. *Autom. Constr.* **2024**, *166*, 105628. <https://doi.org/10.1016/j.autcon.2024.105628>.
14. Diakit , A.A.; Zlatanova, S. Valid Space Description in BIM for 3D Indoor Navigation. *Int. J. 3-D Inf. Model.* **2016**, *5*, 1–17. <https://doi.org/10.4018/IJ3DIM.2016070101>.
15. Gr ger, G.; Pl mer, L. How to achieve consistency for 3D city models. *GeoInformatica* **2011**, *15*, 137–165. <https://doi.org/10.1007/s10707-009-0091-6>.
16. Giovanella, A.; Bradley, P.E.; Wursthorn, S. Evaluation of Topological Consistency in CityGML. *ISPRS Int. J. Geo-Inf.* **2019**, *8*, 278. <https://doi.org/10.3390/ijgi8060278>.
17. Ledoux, H. On the Validation of Solids Represented with the International Standards for Geographic Information. *Comput.-Aided Civ. Infrastruct. Eng.* **2013**, *28*, 693–706. <https://doi.org/10.1111/mice.12043>.
18. Ledoux, H.; Meijers, M. Topologically consistent 3D city models obtained by extrusion. *Int. J. Geogr. Inf. Sci.* **2011**, *25*, 557–574. <https://doi.org/10.1080/13658811003623277>.
19. Arroyo Otori, K.; Biljecki, F.; Diakit , A.; Krijnen, T.; Ledoux, H.; Stoter, J. Towards an Integration of GIS and BIM Data: What Are the Geometric and Topological Issues? *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.* **2017**, *IV-4/W5*, 1–8. <https://doi.org/10.5194/isprs-annals-IV-4-W5-1-2017>.
20. Carlone, A.; Gabler, F. *Simplification of Polygons From Point Cloud Data for Automated Floorplan Generation*; Technische Universit t Hamburg, Institut f r Digitales und Autonomes Bauen: Hamburg, Germany, 2024. <https://doi.org/10.15480/882.13546>.
21. Weiler, K. The Radial Edge Structure: A Topological Representation for Non-Manifold Geometric Boundary Modeling. In *Geometric Modeling for CAD Applications*; Wozny, M.J., McLaughlin, H.W., Encarn  o, J.L., Eds.; North-Holland: Amsterdam, The Netherlands, 1988; pp. 3–36.
22. Boguslawski, P.; Gold, C. The Dual Half-Edge—A Topological Primal/Dual Data Structure and Construction Operators for Modelling and Manipulating Cell Complexes. *ISPRS Int. J. Geo-Inf.* **2016**, *5*, 19. <https://doi.org/10.3390/ijgi5020019>.
23. Jabi, W.; Chatzivasileiadi, A. Topologic: Exploring Spatial Reasoning Through Geometry, Topology, and Semantics. In *Formal Methods in Architecture*; Eloy, S., Leite Viana, D., Morais, F., Vieira Vaz, J., Eds.; Advances in Science, Technology & Innovation; Springer: Cham, Switzerland, 2021; pp. 277–285. https://doi.org/10.1007/978-3-030-57509-0_25.
24. Bradley, P.E. *Euler–Poincar  Operators for Architectural Complexes*; Manuscript, Karlsruhe Institute of Technology: Karlsruhe, Germany, 2008. Available online: <https://www2.ipf.kit.edu/Personen/bradley/Arch/euler-poincare.pdf> (accessed on 25 June 2026).
25. Suter, G. Modeling multiple space views for schematic building design using space ontologies and layout transformation operations. *Autom. Constr.* **2022**, *134*, 104041. <https://doi.org/10.1016/j.autcon.2021.104041>.
26. Suter, G. Feature-Based Decomposition of Architectural Spaces: Outline of a Procedure and Research Challenges. In *Advances in Information Technology in Civil and Building Engineering*; Skatulla, S., Beushausen, H., Eds.; Lecture Notes in Civil Engineering; Springer: Cham, Switzerland, 2023; Volume 358, pp. 443–456. https://doi.org/10.1007/978-3-031-32515-1_31.
27. Gabler, F.; Ziaee, A.; Huhnt, W.; Suter, G. Extracting Topological Features from Room Polygons Based on a 2-Dimensional Space Partitioning Approach. In Proceedings of the 30th International Workshop on Intelligent Computing in Engineering (EG-ICE 2023), London, UK, 4–7 July 2023.
28. Gabler, F.; Ziaee, A.; Huhnt, W.; Suter, G. Topological and geometric measures derived from space floor polygons for classification of spaces in buildings by function and form. *J. Build. Eng.* **2025**, *114*, 114455. <https://doi.org/10.1016/j.jobbe.2025.114455>.
29. Hoffmann, C.M. Robustness in Geometric Computations. *J. Comput. Inf. Sci. Eng.* **2001**, *1*, 143–155.
30. The CGAL Project. *CGAL User and Reference Manual*, 5th ed.; CGAL Editorial Board: 2024. Available online: <https://www.cgal.org> (accessed on 25 June 2026).

31. Granados, M.; Hachenberger, P.; Hert, S.; Kettner, L.; Mehlhorn, K.; Seel, M. Boolean Operations on 3D Selective Nef Complexes: Data Structure, Algorithms, and Implementation. In *Algorithms—ESA 2003*; Di Battista, G., Zwick, U., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2832, pp. 654–666. https://doi.org/10.1007/978-3-540-39658-1_59.
32. Cherchi, G.; Pellacini, F.; Attene, M.; Livesu, M. Interactive and Robust Mesh Booleans. *ACM Trans. Graph.* **2022**, *41*, 1–14. <https://doi.org/10.1145/3550454.3555460>.
33. Zhou, M.; Qin, J.; Mei, G.; Tipper, J.C. Simple and Robust Boolean Operations for Triangulated Surfaces. *Mathematics* **2023**, *11*, 2713. <https://doi.org/10.3390/math11122713>.
34. Open Cascade S.A.S. Open CASCADE Technology. 2024. Available online: <https://www.opencascade.com> (accessed on 25 June 2026).
35. Lalish-Menagh, E. Manifold: A Provably Robust Library for Geometric Operations. 2022. Available online: <https://github.com/elalish/manifold> (accessed on 25 June 2026).
36. Smith, J.M.; Dodgson, N.A. A topologically robust algorithm for Boolean operations on polyhedral shapes using approximate arithmetic. *Comput.-Aided Des.* **2007**, *39*, 149–163. <https://doi.org/10.1016/j.cad.2006.11.003>.
37. Pressler, R. JEP 505: Structured Concurrency (Fifth Preview). OpenJDK Enhancement Proposal. 2025. Available online: <https://openjdk.org/jeps/505> (accessed on 30 May 2026).
38. Rasmussen, M.H.; Pauwels, P.; Hviid, C.A.; Karlshøj, J. Proposing a Central AEC Ontology That Allows for Domain Specific Extensions. In *Proceedings of the Joint Conference on Computing in Construction (JC3)*; Lean Construction Institute: Heraklion, Greece, 2017; pp. 237–244.
39. Oswald, S.; Calliku, G. *Integration of GIS Data into 3D Space Partitioning to Enhance Architectural Modeling*; Technische Universität Hamburg, Institut für Digitales und Autonomes Bauen: Hamburg, Germany, 2024. <https://doi.org/10.15480/882.13548>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.