A dimension-independent extrusion algorithm using generalised maps

Ken Arroyo Ohori Hugo Ledoux Jantien Stoter

This is an author's version of the paper. The authoritative version is:

A dimension-independent extrusion algorithm using generalised maps. Ken Arroyo Ohori, Hugo Ledoux and Jantien Stoter. *International Journal of Geographical Information Science*, 2015.
DOI: 10.1080/13658816.2015.1010535

Related source code is available at
`https://github.com/kenohori/lcc-tools`

One solution to the integration of additional characteristics, e.g. time and scale, into GIS datasets is to model them as extra geometric dimensions perpendicular to the spatial ones, creating a higher-dimensional model. While this approach has been previously described and advocated, it is scarcely used in practice because of a lack of high-level construction algorithms and accompanying implementations. We present in this paper a dimension-independent extrusion algorithm permitting us to construct from any $(n-1)$-dimensional linear cell complex represented as a generalised map, an n-dimensional one by assigning to each $(n-1)$-cell one or more intervals where it exists along the n-th dimension. We have implemented the algorithm in C++11 using CGAL, made the source code publicly available, and tested it in experiments using real-world 2D GIS datasets which were extruded to construct up to 5D models.

1 Introduction

The integration into a GIS of additional parametrisable characteristics such as a third spatial dimension, time, and scale has been so far achieved mostly by extending and keeping several copies of existing 2D data structures rather than building new higher-dimensional ones. For instance, 3D systems often mimic the third dimension by using a so-called 2.5D structure, essentially treating the third dimension as an attribute; or they represent 3D objects only implicitly by their 2D boundary using a 2D data structure with no explicit 3D (volume to volume) topological relationships. Meanwhile, implementations of spatio-temporal GISs usually keep either multiple representations [Armstrong, 1988] or a list of changes per object [Worboys, 1992; Peuquet, 1994] using 2D structures.

It is known that incorporating these characteristics as extra geometric dimensions, orthogonal to the spatial ones, is an alternative that is theoretically sound and can bring practical advantages. Descartes [1637] already laid the foundations for nD geometry by putting coordinates to space, allowing the numerical description of geometric primitives and the use of algebraic methods on them, theories of nD geometry were developed by Riemann [1868] among others, and Poincaré [1895] developed algebraic topology directly with a dimension-independent formulation, stating that even if nD objects could not be [then] represented, they do have a precise topological definition, and consequently properties that can be studied. From an application point of view, Pigot and Hazelton [1992] argue that spatio-temporal processes can be analysed using the topological relationships between 4D objects, and van Oosterom and Stoter [2010] argue that the integration of space, time and scale into a 5D model for GIS can be used to ease data maintenance and improve consistency, as algorithms could detect if the 5D representation of an object is consistent and does not conflict with that of other objects.

However, higher-dimensional models at this moment are only described in theory, and implementations and practical uses of them are scarce. We believe that the main obstacle to their use is a lack of high-level algorithms to construct and manipulate higher-dimensional objects rather than a lack of appropriate data structures since various structures, e.g. generalised and combinatorial maps [Lienhardt, 1994], have been implemented in different programs and libraries.

We therefore investigate in this paper one construction technique for creating higher-dimensional datasets: extrusion. It is a commonly used technique in GIS to construct simple 3D city models [Ledoux and Meijers, 2011] that, as shown in Figure 1, 'lifts' a set of 2D building footprints, possibly first decomposed into parts, into a set of 3D polyhedra by assigning each one an interval along which it exists (from the ground to the height of the building, which is easily obtained with techniques like airborne laser technologies or photogrammetry). While not every possible 3D shape can be generated using extrusion—the 'top' and 'bottom' faces will always be horizontal, and the side faces connecting will always be vertical—, many shapes can still be constructed with this method and, compared to the construction of arbitrary objects, it is conceptually simpler to use and to ensure that its output is valid.

In this paper we present: 1. a generalisation of this technique to higher dimensions: any $(n-1)$-dimensional space partition is transformed into a set of n-dimensional prismatic polytopes, the higher-dimensional analogue of a set of prisms, and 2. an algorithm that computes it based on a generalised map representation of the input. It is an extension of Arroyo Ohori and Ledoux [2013] in which all objects were extruded along the same interval (e.g. a set of buildings extruded to the same height).

Our algorithm permits us to construct, for instance, a set of 5D objects—and the topological relationships between them— from a 2D planar partition representing buildings for a given area having as attributes: the height, the construction/destruction date, and an interval of scales at which this

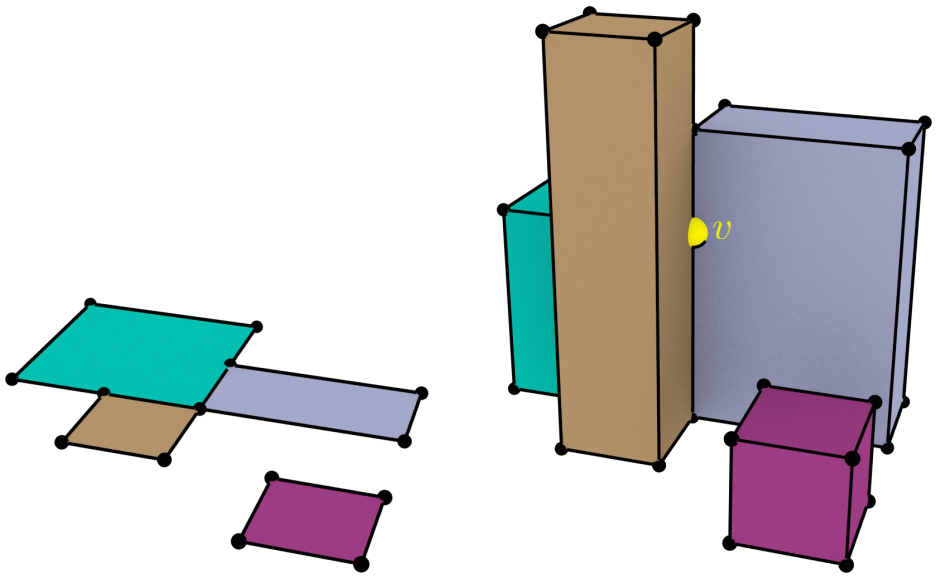


Figure 1: (a) A set of footprints with their heights, is extruded in (b) to generate a simple 3D city model. Note how the vertex v in (b), generated because of the extrusion of the back left polygon in (a), is used in the representation of the all 4 edges, 5 faces and 3 volumes that are incident to it, not only in those that are part of the back left polyhedron. This ensures that the cells in (b) are pairwise disjoint and have the expected correct topology.

representation is appropriate for a particular application. Section 2 covers the definition of a cell complex—a concept from topology that allows us to describe extrusion in a dimension-independent manner and from a high-level perspective—, and how to represent such a cell complex as a generalised map, which is the data structure used in our algorithm and implementation. The details of our algorithm are then described in Section 3 based on these concepts.

In the related literature, there are two algorithms that can be used for a similar purpose. The Cartesian product, a more general operation defined for generalised maps in Lienhardt et al. [2004], would permit us to generate a combinatorial structure equivalent to the extrusion of all the objects along a single interval by computing the Cartesian product of the original (unextruded) cell complex with an edge. However, since such an operation is limited to a single interval (and thus the same for all the input objects), the output cannot be directly applied for modelling real-world datasets. A related possibility, presented by Ferrucci [1993], is to first pre-process all the intervals (for all cells), splitting them into fragments at the other intervals' endpoints so that each fragment intersects another one if and only if they have the same endpoints, and then computing the output purely combinatorially for each possible combination of fragment and for each input cell. While this allows us to model real-world objects, the process of splitting the input intervals might greatly increase the number of output cells since two intersecting intervals are split even if their corresponding cells are not incident or adjacent to each other. In this paper we follow a similar approach, but instead process the input intervals for each input cell separately based on its incidence relationships with other cells, thus generating a smaller number of cells and a considerably smaller combinatorial structure.

We have implemented and tested our algorithm in C++11 based on the Computational Geometry Algorithms Library [CGAL, 2014], as described in Section 4, and have made our implementation publicly available under the open source MIT license [Open Source Initiative, 2014]. We have also made experiments generating consistent objects in up to 6D by combining publicly available GIS datasets having different attributes, as shown in Section 5. We finish with our conclusions and future work in Section 6.

2 nD cell complexes and their representation as a generalised map

A cell complex, related to the concept of a CW complex in topology [Hatcher, 2002], is a decomposition of topological space into cells so that an i-dimensional cell (i-cell) is an object homeomorphic to an open i-ball (e.g. point, open arc, open disk, etc.). A 0-dimensional cell complex is composed of isolated vertices. For any integer $i > 0$, we can build an i-cell from an $(i-1)$-cell c by attaching i-dimensional faces to the $(i-1)$-dimensional faces of c. That is, an edge is built by linking the vertices on its boundary, a facet by linking the edges on its boundary, a volume by linking the facets on its boundary, and so on. Cell complexes are usually represented in a computer as a collection of cells and the boundary relationships that exist between them.

Many topological relationships can already be defined based on this non-geometric definition. Two i-cells are adjacent if they share at least one $(i-1)$-cell on their boundary. A cell c_1 and a cell c_2 are incident if c_1 is on the boundary of c_2 or c_2 is on the boundary of c_1.

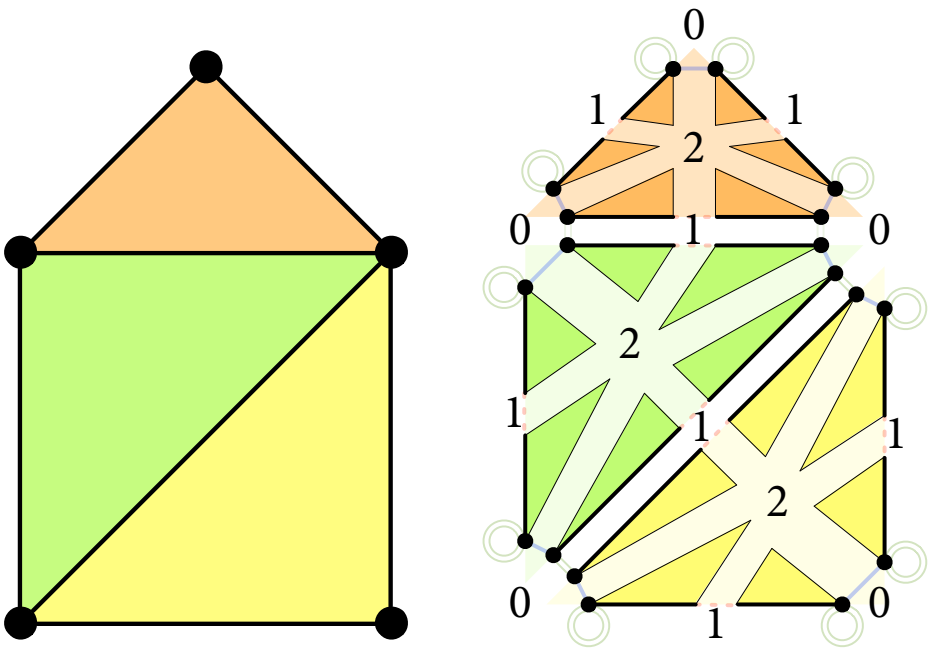
When a cell complex is embedded into Euclidean space and the cells are constrained to be pairwise disjoint geometrically (i.e. they do not intersect), this becomes a natural representation of the space partitions common in GIS data, with a set of non-overlapping i-dimensional objects being represented as an i-dimensional cell complex. Since GIS data generally consists of only linear geometries (i.e. points, straight line segments, polygons, polyhedra, etc.), it is sufficient to embed 0-cells as points by linking them to a tuple of coordinates. Higher-dimensional cells can have then implicit geometries based on interpolating the coordinates of the 0-cells on their boundary. A cell complex with these characteristics is known more formally as a linear cell complex, and together with its complement (i.e. the 'universe'), it forms a partition of n-dimensional space.

There are various data structures capable of storing a cell complex. In our approach we use a structure which is known as generalised maps. However, it is worth noting that our approach also works with other data structures that describe n-dimensional cell complexes (cf. Arroyo Ohori et al. [2015]). For instance, an implementation of a data structure based on an incidence graph [Rossignac and O'Connor, 1989; Masuda, 1993] would simply need to follow the first two parts of our algorithm, which generate all the required topological relations contained in them.

2.1 Generalised maps

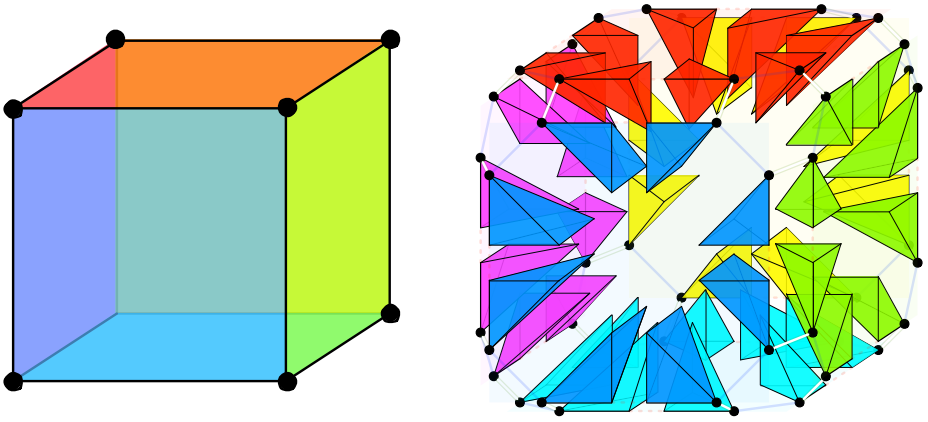
Generalised maps, also known as g-maps, were first described by Lienhardt [1994] and extended to objects with boundaries (i.e. allowing for the existence of the universe without modelling it explicitly) by Poudret et al. [2007]. Implementations of generalised maps are part of several software packages and libraries, including MOKA [Damiand, 2014], and Gocad [GOCAD Project, 2011], which gives them a practical advantage over other data structures.

A generalised map is a structure that represents a *purely combinatorial* simplicial decomposition (i.e. a combinatorial triangulation) of a cell complex. As shown in Figures 2 and 3, it is akin to a barycentric triangulation of the cell complex, built by creating a structure that has a node at each cell (of every dimension), and forming abstract



(a) (b)

Figure 2: (a) A set of three adjacent facets represented as a cell complex of 0- (dots), 1- (lines) and 2-cells (filled areas). (b) The same facets represented as a generalised map. The numbers show the dimension of the cell at each node of a simplex. The more compact visual representation of the map, which is used elsewhere in this paper and where simplices are only represented as edges with rounded endpoints, is also shown here.



(a) (b)

Figure 3: (a) A cube is represented as a cell complex of 0- (dots), 1- (lines), 2-cells (filled areas) and a 3-cell (implicit). (b) The same cube is represented as a generalised map. Notice that each node of every simplex still represents a cell of a different dimension.

simplices that have each node at a cell of different dimension, all of which are incident to each other. If the input is in the form of an incidence graph containing all the cells (of every dimension) as nodes, and the incidence relationships from each i-cell to the $(i-1)$-cells on its boundary as directed edges, each possible path from the highest-dimensional cells in the graph to the 0-cells yields one such simplex.

A generalised map is based on storing these abstract simplices, which are known as darts, together with the adjacency relationships between, which are known as involutions and denoted as α. Since a simplex is necessarily convex, a dart in an n-dimensional generalised map can be represented as an $(n+1)$-tuple $(c_0, c_1, \ldots, c_n)$ where c_i is a node that represents an i-dimensional cell. As an n-simplex in an n-dimensional simplicial complex has up to $n+1$ adjacent n-simplices, a dart can also be linked to $n+1$ neighbouring darts through a tuple of involutions $(\alpha_0, \alpha_1, \ldots, \alpha_n)$. The involution α_i connects two darts that represent almost the same n-tuple of cells; the only difference between these tuples is their i-element, which corresponds to a different i-cell.

Note that the node locations corresponding to the 0-cells can use the location of the vertices (i.e. the 0-th element in the tuple), but those of the cells of dimension one and higher are only a combinatorial construct and are not embedded into any specific location in space. This is a notion that lies between an abstract simplicial complex, where none of the nodes have an embedding, and a geometric simplicial complex, where all nodes have a specific embedding as points in space. In Figures 2 and 3 they have been drawn at the centroid of the cells they represent, but such a simple graphical representation is not always possible—in some (concave) polytopes it could lead to some simplices being drawn (partially) outside the cell. However, note also that unlike in a geometric nD triangulation [Shewchuk, 1998, 2000], in a generalised map constraints are not needed to make the decomposition conform to the shape of the object. In other words, generalised maps bring the strong algebraic properties of a geometric simplicial complex without the need to perform any geometric operations to triangulate the cells. While not all operations can be performed on the simplicial complex implicit in a generalised map, many can still be performed, including most construction methods and the computation of areas and volumes.

The darts and involutions represent the combinatorial structure of a generalised map. However, in order to represent the geometry and other characteristics of the model, additional *embedding* structures are needed. Each of these structures stores the information of one cell of a certain dimension, so that an i-embedding contains the necessary information of an i-cell. Since only linear geometries are required, only the 0- (point) embeddings are strictly necessary, which store the coordinates of a vertex. Nonetheless, it is practical to allow for higher-dimensional embedding structures, since these can be used to store the attributes of a specific higher-dimensional cell—as would be expected in a GIS application.

3 Extruding an $(n-1)$-dimensional space partition to an n-dimensional cell complex

We consider here the extension of extrusion as used in GIS—from a planar partition of 2D polygons to a set of 3D prisms—to higher dimensions. This operation thus 'lifts' an $(n-1)$-dimensional space partition, to an n-dimensional cell complex by assigning one or more intervals to each $(n-1)$-cell, i.e. a subset of 1D Euclidean space[1], along which this cell is defined. For instance, a building footprint can be extruded by an interval $(0, \mathit{height})$, or a building can be extruded along the time intervals $(1700, 1950) \cup (2010, \infty)$. Note that this contrasts with the use of the term 'extrusion' in other fields. For instance, in geometric

[1] This is equivalent to an edge in Lienhardt et al. [2004] or a 1-dimensional polyhedron in Ferrucci [1993].

modelling, extrusion is a well-known operation in which all of the objects in the model are 'dragged' along a predefined path given by a curve, more akin to an actual (physical) extrusion process, but whose output topology can be computed without any geometric computations.

Our dimension-independent extrusion algorithm requires two input arguments: an $(n-1)$-dimensional space partition of $(n-1)$-polytopes embedded into $(n-1)$-dimensional space, stored as an $(n-1)$-dimensional generalised map M; and a map of extrusion intervals ρ that links each $(n-1)$-cell c in M to a set R of 1-dimensional intervals, where every interval r in R is represented as a pair of values $(r_{\min}, r_{\max})$ where c is extruded along the n-th dimension. The intervals in ρ for the cells of lower dimension do not need to be given, since they can be computed by the algorithm based on their incidence relationships to the $(n-1)$-cells. Note that, as Figure 4 shows, multiple intervals for the same cell are possible. There are two cases in which this can happen. For an $(n-1)$-dimensional cell, multiple intervals may be explicitly provided in the input (e.g. if the input complex represents a set of buildings, and the buildings should be extruded along the time dimension, there could be a building that existed in two separate periods of time). For a lower-dimensional cell, multiple extrusion intervals may be passed to this cell from several adjacent higher-dimensional cells. This is a continuation of our previous work, which only considered a single interval for all objects. Its result is an n-dimensional generalised map M' representing a n-dimensional cell complex containing a set of prismatic n-polytopes, the n-dimensional analogue of a set of prisms, which also do not intersect geometrically. The output map M' creates entirely new structures, i.e. it does not reuse the darts or embeddings of M, since the $(n-1)$-simplices in M are similar but not identical to the n-simplices in the base of M'.

As Figures 5 and 6 show, the cells in the new n-dimensional cell complex in M' have a direct relation to and can be expressed in terms of the $(n-1)$-cells in M and their extrusion intervals in ρ. This property is used in order to define the cells of the output cell complex, which are equivalent to the embeddings in M'. These consist of:

- 'Base' and 'top' $(n-1)$-cells (i.e. faces), which are constructed from every $(n-1)$-cell c in M at the $r_{\min}$ and $r_{\max}$ end points of every interval in $\rho(c)$.
- A series of prismatic faces linking corresponding $(n-2)$-cells (i.e. ridges) c of the above mentioned $r_{\min}$ and $r_{\max}$ faces for every interval, such that every one of these faces corresponds to the extrusion of the ridge along an interval in $\rho(c)$.

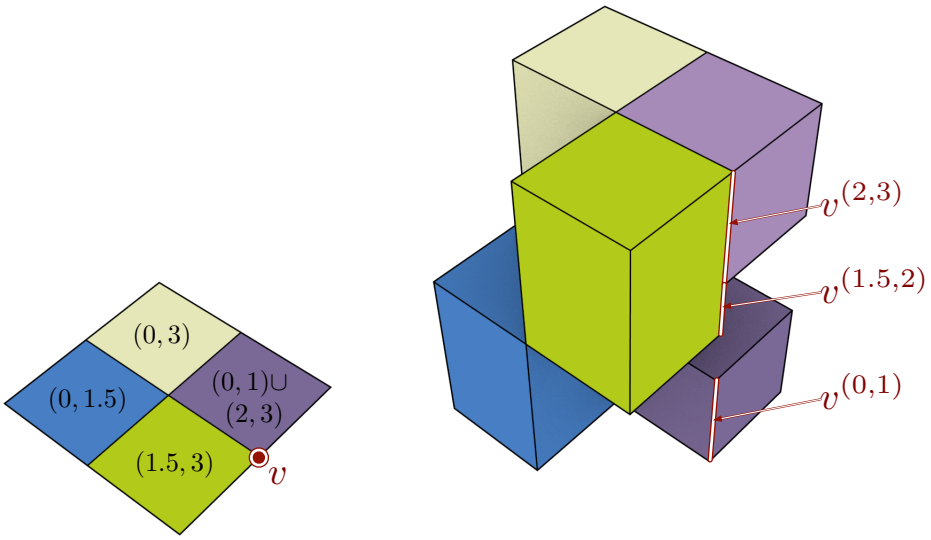
For the combinatorial structure, our algorithm takes advantage of the fact that the darts in the generalised map M can be extruded largely independently based on querying the combinatorial structure of M and simple 1D geometric queries along the n-th dimension. Intuitively, the extrusion of a single $(n-1)$-simplex in M consists of layers of n-simplices that are 'stacked'[2] so as to form one part of the prism-shaped output. Each new layer corresponds to a new n-simplex that shares all but one of the nodes with the n-simplex below it.

The extrusion algorithm is thus divided into three parts that are performed sequentially and explained in detail in the following sections: (1) propagating the input intervals to all the cells in the input cell complex, (2) generating the new embeddings (i.e. the attributes and geometry) for each input cell, and (3) generating the combinatorial structure (i.e. the darts and involutions) and linking each dart to its correct embeddings for every dimension.

3.1 Propagating the extrusion intervals to all cells

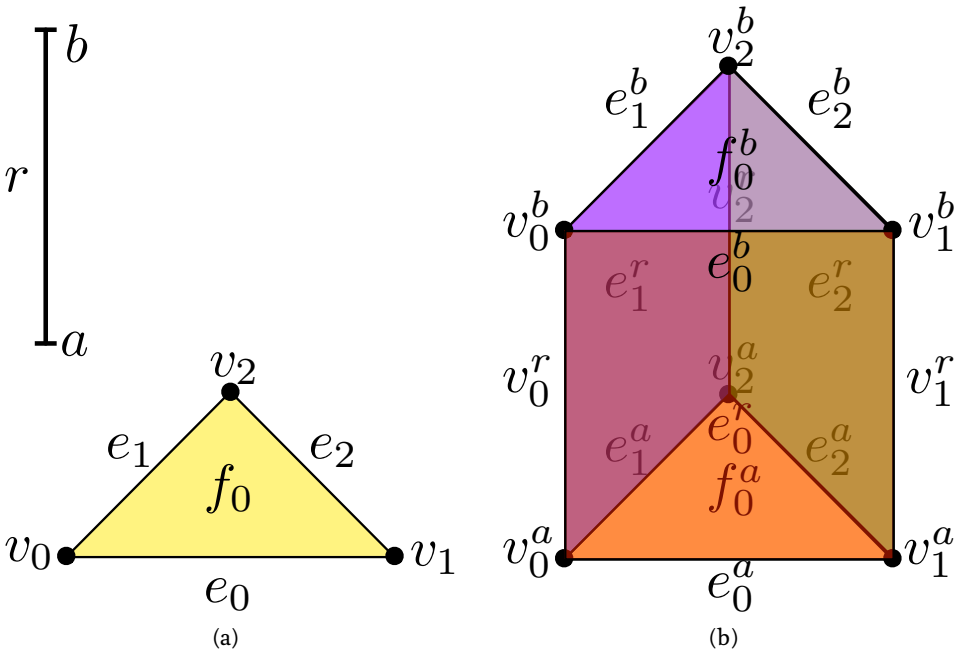
While the extrusion intervals are defined only for the $(n-1)$-cells, the cells of every dimension need to be extruded, and therefore the extrusion intervals need to be propagated from the $(n-1)$-cells to the cells

[2] Not in the sense of a stack as a data structure, but as a pile of simplices arranged vertically along the n-th dimension.



(a) (b)

Figure 4: All the cells are given extrusion intervals based on the intervals for the $(n-1)$-cells. Note how it is possible to have multiple intervals per cell both in the input, e.g. the right square is extruded along two intervals, and in the output, e.g. the two edges $v^{(0,1)}$, $v^{(1.5,2)}$ and $v^{(2,3)}$ are the result of the extrusion of a single vertex v whose extrusion interval was not given as input.



(a) (b)

Figure 5: Extruding the embeddings of an i-cell c along a single interval $r = (a, b)$ such that $a, b \in \mathbb{R}, a < b$, generates the embeddings of three cells: two i-cells c^a and c^b, and an $(i+1)$-cell $c^r = c^{(a,b)}$ lying between them.

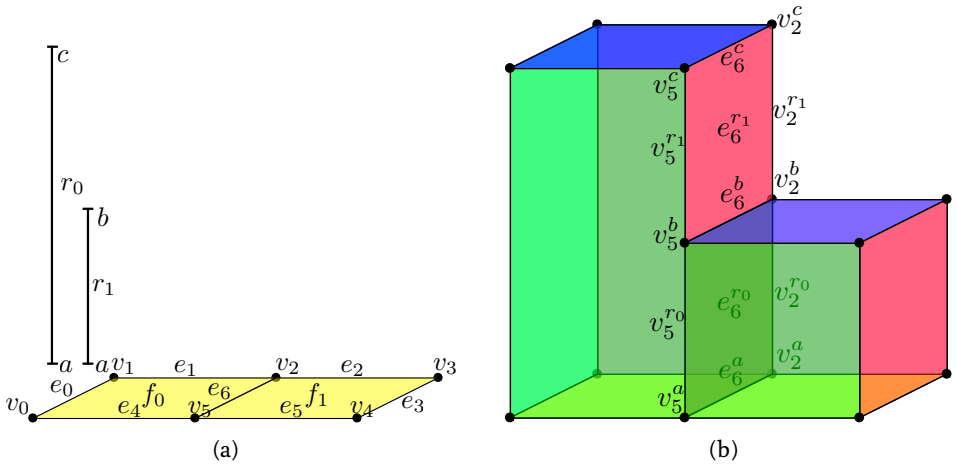


Figure 6: Extruding a 2D cell complex using an interval that is defined per 2-cell. The facet f_0 is to be extruded along the interval $r_0 = (a, c)$ and f_1 along $r_1 = (a, b)$. Note how the vertices and edges incident to multiple facets in the input are extruded along the intervals of these facets, generating a series of cells connecting the end points of their intervals.

of lower dimension. This is done recursively in decreasing dimension, using the incidence relationships between every i-cell and the $(i-1)$-cells on its boundary to pass the intervals of the former to the latter, using the same map of extrusion intervals ρ. Because incidence is a transitive relation, an interval attached to a particular lower dimensional cell thus indicates that it is incident to an $(n-1)$-cell that will be extruded along that interval.

As $(i-1)$-cells can be on the boundary of multiple i-cells, several intersecting intervals can be passed to the map of extrusion intervals of a lower dimensional cell. Since we want to generate non-intersecting cells, these need to be split into a set of non-intersecting intervals as shown in Figure 7, each of which will represent the incidence of this cell to an equal set of higher-dimensional cells along the entirety of the interval. A sketch of a simple process to do this is shown in Algorithm 1, which assumes that the sets of intervals are kept sorted. Note that this is essentially a one dimensional analogue to other operations used in GIS, e.g. time-composites in spatio-temporal modelling [Langran and Chrisman, 1988].

Considering what happens when a single

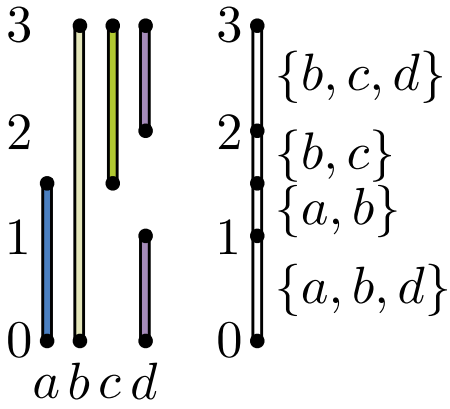


Figure 7: The extrusion intervals from the cells incident to the middle vertex of Figure 4 in (a), are passed on to it, resulting in a new set of non-intersecting intervals in (b).

new interval is passed to a cell in the incremental process shown in Algorithm 1, the total number of intervals of that cell can increase from k to $2k + 1$ in the worst case, which happens when the new interval starts before and ends after all other intervals and these are all not contiguous (i.e. they are all separated by empty intervals). However, when taking into account r intervals passed to a cell, the number of intervals can only increase to $2r - 1$. As each extrusion inter-

Algorithm 1: PROPAGATERANGES

Input : generalised map M of the input cell complex,
map ρ with the extrusion intervals of the $(n-1)$-cells in M
Output: map ρ with the extrusion intervals of all the cells in M

```
for i ← n − 1 to 0 do
    foreach i-cell c in M do
        foreach (i − 1)-cell b on the boundary of c do
            foreach extrusion interval r = (r_min, r_max) ∈ ρ(c) do
                Find the intervals R' ⊆ ρ(b) whose interiors intersect with r
                if r_min is in the interior of an interval r' = (r'_min, r'_max) ∈ R' then
                    Remove r' from ρ(b)
                    Add (r'_min, r_min) to ρ(b)
                    Add (r_min, r'_max) to ρ(b)
                if r_min is outside all intervals in R' then
                    Add a new interval in R' from r_min to the minimum of the lowest
                    interval in R'
                if r_max is in the interior of an interval r' = (r'_min, r'_max) ∈ R' then
                    Remove r' from ρ(b)
                    Add (r'_min, r_max) to ρ(b)
                    Add (r_max, r'_max) to ρ(b)
                if r_max is outside all intervals in R' then
                    Add a new interval in R' from the maximum of the highest interval
                    in R' to r_max
                foreach empty interval (r'_min, r'_max) between consecutive intervals in R'
                do
                    Add (r'_min, r'_max) to ρ(b)
```

val is bounded by two values, r intervals lead to at most $2r$ boundary values and thus the number of intervals cannot be higher than $2r-1$.

If a new interval can be added to a cell in $O(\log r)$ time (e.g. by maintaining an augmented red-black tree which stores the endpoints of the intervals), then the total time to construct the ordered set of all intervals for a cell c is $O(r \log r)$, where r is the number of intervals that are passed onto c. The overall computational complexity of this step depends on the incidence relationships between the cells in the complex.

3.2 Generating the new embeddings

When a single i-cell is extruded along a single interval $r = (r_{\min}, r_{\max})$, three new cells are generated: two i-cells that correspond to the end points of the interval $r_{\min}$ and $r_{\max}$, and a prismatic $(i+1)$-cell lying between them. Therefore, extruding an i-dimensional embedding (i-embedding) results in two i-embeddings and one $(i+1)$-embedding. In the case of linear geometries, this means that in order to generate the geometry of a model, extruding a point entails the creation of two additional points, one with an appended $r_{\min}$ coordinate, and one with an $r_{\max}$ one.

With multiple intervals, this same procedure can then be used by applying it per embedding and per interval. Notice that since some embeddings are shared by multiple intervals (when the minimum of an interval is equal to the maximum of another), the generation of some of these can be omitted.

The extrusion algorithm for the embeddings receives the set of input embeddings E and the map of extruded intervals ρ which, being the output of PROPAGATERANGES (Algorithm 1), now contains a set

of non-intersecting intervals for the cells of every dimension, and it returns an entirely new set of extruded embeddings E', as well a function $ex(e, v) \rightarrow E'$ linking an input embedding $e \in E$ and an interval or end point of an interval v to an extruded (output) embedding $e' \in E'$. For instance, given an interval $r = (r_{\min}, r_{\max}) \in \rho(e)$, v can be r, $r_{\min}$ or $r_{\max}$, reflecting the fact that extruding an i-embedding results in two i-embeddings (respectively for $r_{\min}$ and $r_{\max}$) and one $(i+1)$-embedding (for r).

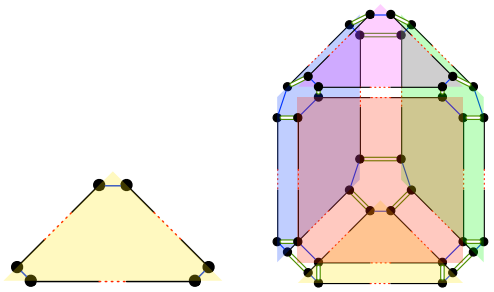
The general procedure to generate the new embeddings and their relation to the old embeddings and the intervals is shown in Algorithm 2. Note that a practical implementation of this algorithm has to deal with the desired attributes for each of the extruded cells rather than simply making a copy of the input ones (lines 4, 7 and 10) and appending one more coordinate to the 0-embeddings (lines 6 and 12).

Since this part of the algorithm iterates through all the cells in the input cell complex, and it generates at most three new embeddings per interval for each cell, the computational complexity can be $O(rn)$ per cell, as long as the map ex can be queried and the new embeddings can be created in time that is linear on the dimension (which in practice is a relatively small constant).

3.3 Generating the new combinatorial structure and linking it to its correct embeddings

The extrusion of a *single* dart in the input map M along a single interval $r = (r_{\min}, r_{\max})$ generates a series of connected darts in M' connected by a sequence of involutions $\alpha_{n-1}, \alpha_{n-2}, \ldots, \alpha_1, \alpha_0, \alpha_1, \ldots, \alpha_{n-2}, \alpha_{n-1}$, as shown in Figure 8 for the cell complex from Figure 5. Intuitively, these are equivalent to stacked n-simplices that together form a prism[3]. Since two darts connected by an α_i involution share all but the i-th node, and

[3]This is always true combinatorially, but it might not be true geometrically if the $(n-1)$-simplex is not embedded so as to lie in the interior of the cell.



(a) (b)

Figure 8: The darts in the cell complexes in Figure 5. Note that one can obtain a dart's representation as a simplex by considering additional nodes at its corresponding facet and one in the interior of the volume.

the i-th node in a generalised map means that a dart belongs to a certain i-cell, this series of darts represents a succession of simplices that progressively change from the cells at the 'base' ($r_{\min}$) to those at the 'side' ($(r_{\min}, r_{\max})$) to those at the 'top' ($r_{\max}$). As the pattern that is followed in each stack of darts in M' is the same for all the darts in M, assuming that their extrusion intervals are the same, they can also be expressed in terms of the dart in M being extruded and their position in the stack. We will therefore refer to the darts at a certain level in the stack as a layer, and use various functions that map an input dart in M and a layer to the equivalent dart at a specific layer in M'.

An intuitive justification for this is that if we consider the darts in the base and top cells in the extrusion of a cell f along a single interval $r = (r_{\min}, r_{\max})$, these belong to different cells of every dimension except for their n-cell (which is $ex(f, r)$). For every dimension $i < n$ the base darts belong to the extruded $ex(e_i, r_{\min})$ cells, while the top darts belong to the extruded $ex(e_i, r_{\max})$ cells. The darts of the faces on the sides, which connect corresponding ridges of the base and top, belong instead to a mixture of cells in $ex(e_i, r_{\min})$, $ex(e_i, r_{\max})$ and $ex(e_i, r)$. The darts closer to the base face belong to cells in $ex(e_i, r_{\min})$ and $ex(e_i, r)$, while those closer to the top belong to cells in $ex(e_i, r_{\max})$

Algorithm 2: EMBEDDINGSEXTRUSION

```
Input  : set E of the embeddings in the input cell complex,
         map ρ of the extrusion intervals for all the cells of the input cell complex
Output: set E' of the embeddings for the output cell complex,
         map ex that links an input embedding to its extruded output embeddings
1  foreach e ∈ E do
2      foreach r = (r_min, r_max) ∈ ρ(e) do
3          if ex(e, r_min) = ∅ then
4              ex(e, r_min) ← e
5              if e.dimension = 0 then
6                  Append r_min to the coordinates of ex(e, r_min)
7          ex(e, r) ← e
8          ex(e, r).dimension ← ex(e, r).dimension + 1
9          if ex(e, r_max) = ∅ then
10             ex(e, r_max) ← e
11             if e.dimension = 0 then
12                 Append r_max to the coordinates of ex(e, r_max)
13         Put ex(e, r_min), ex(e, r) and ex(e, r_max) in E'
```

and $ex(e_i, r)$. This is natural when we consider that α_i-linked darts differ only in one of their cells (i.e. the i-cells), all other cells being the same for both darts. This means that there must be a natural progression of layers of darts: starting from the base, they change their cells one by one from $ex(e, r_{\min})$ to $ex(e, r)$ from the highest dimension down, and then change their cells one by one from $ex(e, r)$ to $ex(e, r_{\max})$ from the lowest dimension up until reaching the top.

The algorithm to generate the extruded combinatorial structure therefore works by generating layers of darts patterned on those in the input cell complex, starting from those for the base face, moving on to the $2n - 2$ layers for the side faces, and finishing with the top face. The output darts and the involutions between them are expressed in terms of the input generalised map M. For this, we use a function $cur : M \to M'$ that maps the darts of M to the current layer of the output map M'. Note that this means that we only need to keep track of (and maintain in memory) two layers of darts at a time, one in M and one in M', each of which has at most the number of darts in the input map. This bounds the memory usage of this part of the algorithm, which is therefore on the order of $O(d)$, with d the number of darts in the input space partition.

As shown by Ferrucci [1993], when multiple intervals are involved this procedure can simply be repeated for all intervals, assuming that these have all been subdivided so as not to intersect one another. However, it is possible to greatly reduce the number of darts generated by skipping the creation of some of the darts. This is possible because we have propagated the extrusion intervals to each cell of every dimension independently. Based on our algorithm, the lower-dimensional cells in the complex have received all the intervals from their incident higher-dimensional cells so that the intervals in the lower-dimensional cells contain all the endpoints of the intervals of their incidences. However, the same is not true in the opposite direction: the intervals for the higher-dimensional cells have not been subdivided so as to exactly match the ones of their incident lower-dimensional cells. Nevertheless, as Figure 9 shows, even when an extrusion interval that would be used by the pattern described above is not in a cell, it is possible to map it to a *bigger* interval that contains it. If we consider the darts that would be generated using the above mentioned approach for all the non-intersecting intervals in the cell complex, but mapping the extrusion intervals to bigger containing ones when the smaller ones do not exist, this can result in many darts that are

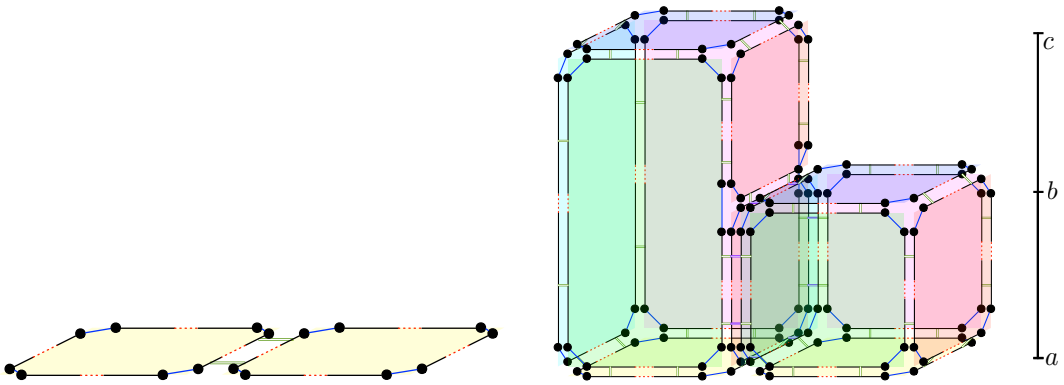


Figure 9: The darts in the cell complexes in Figure 6. Note that the darts of the right cube in (b), when viewed as individual stacks, are similar to the ones in Figure 8. The darts of the left box are however different: those on the left all involve an extrusion along the interval (a, c) due to the fact that there is no vertex, edge or facet extruded to b. On the other hand, those in the right do have a vertex and an edge at b, but not a facet and still belong to the same volume.

equivalent (i.e. they have nodes at the same cells), and thus can be skipped during the process.

Therefore, in order to generate the darts for all intervals of all cells, we repeat this procedure for all the intervals in ρ, after making them non-intersecting using the same procedure delineated in Section 3.1, and skipping all the darts that would be equivalent to those that have already been created. This is done using a sweep-plane-like algorithm that generates up to n layers of darts at the events at the beginning or end of an interval, creating darts only when the sweep-plane[4] passes by the beginning or end of their extrusion intervals (i.e. not while it is in their interior). The darts are linked to their appropriate embedding according to the same pattern. The complete procedure to generate the combinatorial structure is presented in Algorithm 3, which uses the Algorithm 4 when the sweep plane passes by the beginning of an interval and Algorithm 5 by the end of one. If we denote an input dart as d, we will denote its corresponding dart in the current layer of the output as $cur(d)$, the dart linked to d by an i-involution as $\alpha_i(d)$, and the i-embedding of d as $e_i(d)$.

For the latter two algorithms, lines 1–8 show the generation of a new layer of darts and its linking to the previous one, lines 12–17 show how the darts within the layer are linked based on the pattern of the input map, and lines 18–21 show how the darts within the layer are related to their correct embeddings, which are also patterned after the embeddings in the input.

Notice that we are generating layers of darts in a grid-like fashion, each layer containing at most the number of darts in the input map, and calling GMapLayerBegin and GMapLayerEnd to create at most $2n$ layers per non-intersecting interval. The time complexity of computing the set of non-intersecting intervals in Algorithm 1 is $O(r \log r)$ as before, while the number of darts in the output map is bounded by $O(ndr)$, where n is the extrusion dimension, d is the total number of darts in the input map and r is the total number of intervals in the input.

[4] More precisely, the sweeping shape used in extruding a $(n-1)$-dimensional space partition is a shape that is unbounded along $n-1$ dimensions in nD space, e.g. a line in $\mathbb{R}^2$ or a plane in $\mathbb{R}^3$.

Algorithm 3: GMapExtrusion

Input : generalised map M of the input cell complex,
set E of the embeddings in the input cell complex,
set E' of the embeddings for the output cell complex,
map ρ of the extrusion intervals for all the cells of the input cell complex,
map ex that links an input embedding to its extruded output embeddings
Output: generalised map M' of the output cell complex

Compute an ordered set of non-intersecting intervals r_{all} using all the intervals for all cells in ρ
Consider a sweep plane that passes through all the intervals r_{all} in increasing order along dimension n
if *the sweep plane passes by the beginning of an interval* $r = (r_{\min}, r_{\max}) \in r_{all}$ **then**
  for $i \leftarrow n$ **to** 0 **do**
    GMapLayerBegin$(M, M', i, E, E', \rho, r, ex)$
if *the sweep plane passes by the end of an interval* $r = (r_{\min}, r_{\max}) \in r_{all}$ **then**
  for $i \leftarrow 0$ **to** n **do**
    GMapLayerEnd$(M, M', i, E, E', \rho, r, ex)$

Algorithm 4: GMapLayerBegin

Input : generalised map M of the input cell complex,
generalised map M' of the output cell complex,
dimension i of the current layer,
set E of the embeddings in the input cell complex,
set E' of the embeddings of the output cell complex,
map ρ of the extrusion intervals of all the cells of the input cell complex,
current interval r,
map ex that links an input embedding to its extruded output embeddings
Output: generalised map M' of the output cell complex

foreach *dart* $d \in M$ **do**
  if $\exists r' \in \rho(e_{n-1}(d)) \mid r \subseteq r'$ **then**
    if $\exists r'' \in \rho(e_i(d)) \mid r''_{\min} = r_{\min}$ **then**
      $last \leftarrow cur(d)$
      $cur(d) \leftarrow$ new dart
      Put $cur(d)$ in M'
      $\alpha_{i+1}(cur(d)) \leftarrow last$
      $\alpha_{i+1}(last) \leftarrow cur(d)$
foreach *dart* $d \in M$ **do**
  if $\exists r' \in \rho(e_{n-1}(d)) \mid r \subseteq r'$ **then**
    if $\exists r'' \in \rho(e_i(d)) \mid r''_{\min} = r_{\min}$ **then**
      for $inv \leftarrow 0$ **to** $i - 1$ **do**
        $\alpha'_{inv}(cur(d)) \leftarrow cur(\alpha_{inv}(d))$
        $\alpha'_{inv}(cur(\alpha_{inv}(d))) \leftarrow cur(d)$
      for $inv \leftarrow i + 2$ **to** n **do**
        $\alpha'_{inv}(cur(d)) \leftarrow cur(\alpha_{inv-1}(d))$
        $\alpha'_{inv}(cur(\alpha_{inv-1}(d))) \leftarrow cur(d)$
      for $emb \leftarrow 0$ **to** i **do**
        $e'_{emb}(cur(d)) \leftarrow ex(e_{emb}(d), r_{\min})$
      for $emb \leftarrow i + 1$ **to** n **do**
        $e'_{emb}(cur(d)) \leftarrow ex(e_{emb-1}(d), r)$

Algorithm 5: GMapLayerEnd

Input : generalised map M of the input cell complex,
generalised map M' of the output cell complex,
dimension i of the current layer,
set E of the embeddings in the input cell complex,
set E' of the embeddings of the output cell complex,
map ρ of the extrusion intervals of all the cells of the input cell complex,
current interval r,
map ex that links an input embedding to its extruded output embeddings
Output: generalised map M' of the output cell complex

```
1  foreach dart d ∈ M do
2      if ∃r' ∈ ρ(e_{n-1}(d)) | r ⊆ r' then
3          if ∃r'' ∈ ρ(e_i(d)) | r''_max = r_max then
4              last ← cur(d)
5              cur(d) ← new dart
6              Put cur(d) in M'
7              α_i(cur(d)) ← last
8              α_i(last) ← cur(d)
9  foreach dart d ∈ M do
10     if ∃r' ∈ ρ(e_{n-1}(d)) | r ⊆ r' then
11         if ∃r'' ∈ ρ(e_i(d)) | r''_max = r_max then
12             for inv ← 0 to i − 1 do
13                 α'_inv(cur(d)) ← cur(α_inv(d))
14                 α'_inv(cur(α_inv(d))) ← cur(d)
15             for inv ← i + 2 to n do
16                 α'_inv(cur(d)) ← cur(α_{inv-1}(d))
17                 α'_inv(cur(α_{inv-1}(d))) ← cur(d)
18             for emb ← 0 to i do
19                 e'_emb(cur(d)) ← ex(e_emb(d), r_max)
20             for emb ← i + 1 to n do
21                 e'_emb(cur(d)) ← ex(e_{emb-1}(d), r)
```

4 Implementation

We have implemented our extrusion algorithm in C++11 and made it available under the open source MIT license at `https://github.com/kenohori/lcc-tools`. It requires and builds upon the CGAL packages Combinatorial Maps and Linear Cell Complex, among others. The first package provides data structures and algorithms to store and to efficiently iterate over the darts of a combinatorial map, and the second links the 0-embeddings to other CGAL types in order to store the geometry of a model. A combinatorial map is a structure very similar to a generalised map, but a combinatorial map dart is equivalent to two generalised map darts. The reader is referred to Damiand and Lienhardt [2014] for the details of the differences between the two.

Both these packages and our implementation make heavy use of the traits programming technique [Myers, 1995] and recursive templates (TMP or template metaprogramming) in order to produce efficient code. The dimensions of the map and the embedding space, the presence and types of attributes for each dimension and most other arguments are passed as template parameters. This entails that the actual structures and the algorithms used are generated at compilation time, and thus are dimension independent (the dimension is a template parameter), efficient (no variable-length structures are used) and fast (many iterations can be done in constant time). Since the recursive code is unrolled during compilation, its execution is also not lim-

```
// Abstracts a map of cell->ranges for a particular dimension
template <class LCC, unsigned int dimension>
struct Extrusion_ranges_map_of_dimension {
public:
  typedef std::map<typename LCC::template Attribute_const_handle<dimension>::type,
    Extrusion_ranges<LCC> > type;
  type ranges_map;
};

// Abstracts a tuple of maps of cell->ranges, each element containing the map of a particular dimension
template <class LCC, unsigned int dimension = LCC::dimension, class Result = CGAL::cpp11::tuple<> >
struct Extrusion_ranges_tuple_per_dimension_up_to;

template <class LCC, class ... Result>
struct Extrusion_ranges_tuple_per_dimension_up_to<LCC, 0, CGAL::cpp11::tuple<Result ...> > {
  typedef CGAL::cpp11::tuple<Extrusion_ranges_map_of_dimension<LCC, 0>, Result ...> type;
};

template <class LCC, unsigned int dimension, class ... Result>
struct Extrusion_ranges_tuple_per_dimension_up_to<LCC, dimension, CGAL::cpp11::tuple<Result ...> > {
  typedef typename Extrusion_ranges_tuple_per_dimension_up_to<LCC,dimension - 1,
    CGAL::cpp11::tuple<Extrusion_ranges_map_of_dimension<LCC, dimension>, Result ...> >::type type;
};

// Abstracts a tuple of maps of cell->ranges, each element containing the map of a particular dimension
template <class LCC_>
struct Extrusion_ranges_tuple_per_dimension {
public:
  typedef LCC_ LCC;
  typedef typename Extrusion_ranges_tuple_per_dimension_up_to<LCC>::type type;
  type ranges;
};
```

Figure 10: Recursive templates can be used to generate dimension independent code. Note that `std::map` is used for simplicity, but this information could also be encoded in the embeddings of the linear cell complex directly.

ited by the size of the stack. A small excerpt of this type of code, used to store the map of extrusion intervals ρ, is shown in Figure 10.

One shortcoming of our current prototype implementation is that it uses the C++ type `std::map` in order to link cells to their extrusion intervals, which offers only logarithmic time access rather than the constant time that would be possible by integrating this into the templated structures. This would involve storing the set of extrusion intervals of a cell directly in the data structure that is used for its attributes.

In order to input and output data, as well as to visualise our results, we use the OGR Simple Feature Library [GDAL, 2014] to read standard GIS data formats, and we wrote a small Wavefront OBJ [Reddy, 1994] writer that is capable of outputting the faces or darts (as triangles) in 2D and 3D linear cell complexes.

5 Experiments with real-world datasets

We have tested our algorithms by extruding various 2D datasets to higher dimensions. For this, we have made a few tests using several free and open datasets in the area of Delft, the Netherlands, matching the geometries in some with the attributes present in others so as to obtain new attributes and appropriate extrusion intervals for each geometry. The tests were performed on a Mac OS X computer with a 2.7 GHz Intel Core 2 Duo processor and 12 GB of RAM. The main characteristics of the datasets tested are shown in Table 1.

Note however that since we use an `std::map` to access the extrusion intervals of each cell, the running time of our code is dominated by the many times this query is performed (which is several times per interval and per cell). The times provided in Table 1 are therefore not indicative of the theoretical complexity of our algorithm, which is instead dominated by the generation and linking of the darts in the extruded dataset (Section 3.3). The generation of the non-intersecting intervals for all cells (Section 3.1) and the generation of the extruded

Table 1: Characteristics of the tested datasets. The cell counts represent the total number of cells for all dimensions.

Test	Input		Output		time
	darts	cells	darts	cells	
One GBKN building to 5D	14	29	80 640	783	3s
370 buildings to 4D	3 268	6 065	123 184	42 552	12s
TOP10NL to 3D	30 098	48 562	181 640	148 102	2m39s

embeddings (Section 3.2) always ran in under a second, even for very large datasets.

One test involved, the Aula Congress Centre in Delft, a building represented by a single polygon with 14 vertices extracted from the GBKN dataset [GBKN, 2014], as shown in Figure 11. It was extruded from 2D up to 5D using some manually added attributes: its height, construction date and a specified level of detail for the model. The end result was a generalised map with 80 640 darts, 112 vertices, 280 edges, 260 facets, 110 volumes, 20 4-cells and 1 5-cell. It was generated in 3 seconds using 15 MB of RAM.

Another test, shown in 2D and 3D in Figure 12, involved a previously generated dataset of the Delft University of Technology (TU Delft) campus (see [Ledoux and Meijers, 2011]), consisting of 2 749 vertices, 2 946 edges, and 370 facets. This dataset, covering 2.3 km^2 with 370 buildings, was originally built from the GBKN dataset by manually forming footprint polygons from the lines in the dataset. Building heights for each polygon were obtained from the AHN [Het Waterschapshuis, 2014] dataset (airborne laser altimetry), while building dates were obtained from the BAG [Kadaster, 2014a] dataset. This dataset, including the added attributes, is available together with the source code of the program. It was therefore extruded from its original 2D representation to 4D using building heights for the third dimension and dates for the fourth dimension. The result was a generalised map with 123 184 darts, 8 613 vertices, 17 919 edges, 12 471 facets, 3 310 volumes and 239 4-cells. It was generated in 12 seconds using 46 MB of RAM.

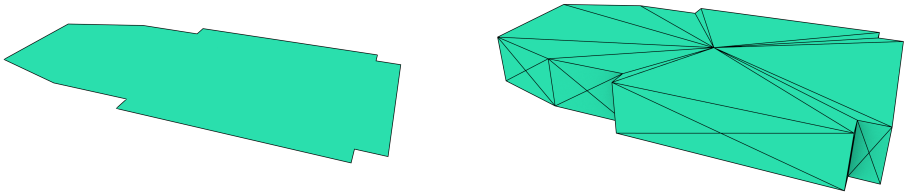
One more test used 1 836 buildings from the TOP10NL dataset [Kadaster, 2014b], which was extruded to 3D using intervals from building dates also obtained from the BAG dataset. The result was a generalised map with 181 640 darts, 46 464 vertices, 71 826 edges, 27 975 facets and 1 837 volumes.

An algorithm was used to verify that the constructed datasets conform to the definition of a generalised map[5]. Additional tests were made to ensure that all the darts of an i-cell correctly point to it in their i-embeddings, and to verify that all the darts of an n-cell are linked to point embeddings within the extrusion interval given for the n-cell, among other tests. The extruded datasets were also inspected visually in 2D and 3D by exporting 2-cells as polygons and verifying that they form a valid cell complex, i.e. that 2-cells intersect only at their common boundaries, forming 1-cells that are also in the complex. In 2D and 3D, individual darts were also exported as triangles to visually verify that they form a valid generalised map (as shown in Figures 11(b) and 12(c)).

6 Conclusions and future work

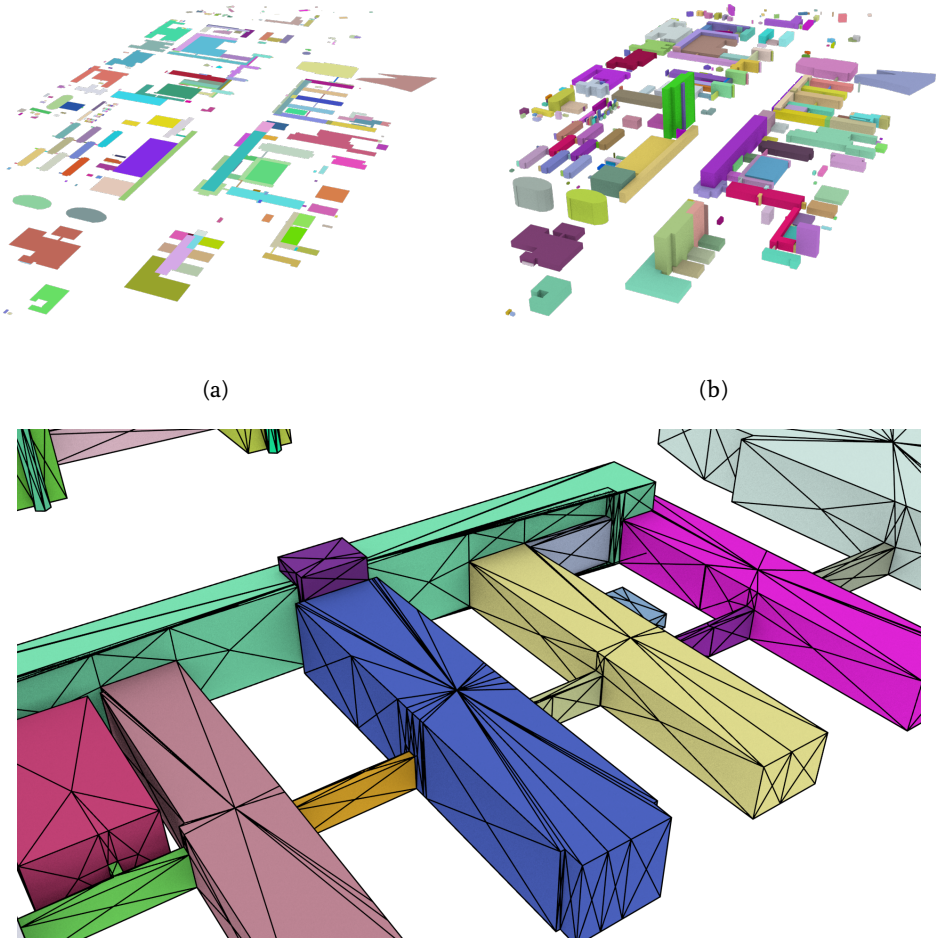
Extrusion, in the GIS sense, has a natural extension to arbitrary dimensions, and we believe that the conceptual simplicity of using this operation makes it very suitable for the generation of certain types of higher-dimensional data. Our method supports

[5] i.e. checking that the links between darts correctly form involutions, and all the darts in the orbit of an i-cell are linked to its correct i-attribute and vice versa. See Lienhardt [1994] for the exact definition of what this means.



(a) (b)

Figure 11: Extruding a dataset of the footprint of the Aula Congress Centre in Delft to 3D.



(a) (b)

(c)

Figure 12: Extruding a dataset of TU Delft to 3D.

multiple intervals per input object and it is memory efficient—only three layers of darts need to be kept in main memory at a time. It is also relatively fast, with a worst case complexity of $O(ndr)$ in the main algorithm, where n is the extrusion dimension, d is the total number of darts in the input map and r is the total number of intervals in the input, but offers better complexity in practice.

We have implemented it and made our source code publicly available under a permissive license and tested it with publicly available datasets commonly used in GIS. Among other applications, it can be used in order to load existing 2D/3D datasets into higher dimensional models for nD GIS, both directly and used as a base for other operations that modify the extruded output. For instance, an object moving in time can be modelled by first extruding it and then shifting the coordinates of one of its end faces to match their new positions. Moreover, since this technique can be applied cell by cell, it can also be combined with other operations, such as building nD cell complexes incrementally [Arroyo Ohori et al., 2014], where more complex models are required for a subset of the objects in a model.

While our implementation performs well enough for typical GIS datasets, it could be improved upon by integrating the extrusion intervals into the embeddings of the cells. This would make it possible to support much larger datasets and match the theoretical complexity of our algorithm. Another possibility would be implementing our algorithm in a more compact representation of a simplicial complex, e.g. Boissonnat and Maria [2012].

Acknowledgements

We are grateful to the anonymous reviewers who provided us with very thorough reviews that helped to greatly improve this article. This research is supported by the Dutch Technology Foundation STW, which is part of the Netherlands Organisation for Scientific Research (NWO), and which is partly funded by the Ministry of Economic Affairs (Project code: 11300).

References

Marc P. Armstrong. Temporality in spatial databases. In *Proceedings of GIS/LIS '88*, pages 880–889, 1988.

Ken Arroyo Ohori and Hugo Ledoux. Using extrusion to generate higher-dimensional GIS datasets. In Craig Knoblock, Peer Kröger, John Krumm, Markus Schneider, and Peter Widmayer, editors, *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 418–421, Orlando, United States, November 2013. The Association for Computing Machinery.

Ken Arroyo Ohori, Guillaume Damiand, and Hugo Ledoux. Constructing an n-dimensional cell complex from a soup of (n-1)-dimensional faces. In Prosenjit Gupta and Christos Zaroliagis, editors, *ICAA 2014*, volume 8321 of *Lecture Notes in Computer Science*, pages 36–47. Springer International Publishing Switzerland, Kolkata, India, January 2014.

Ken Arroyo Ohori, Hugo Ledoux, and Jantien Stoter. An evaluation and classification of nD topological data structures for the representation of objects in a higher-dimensional GIS. *International Journal of Geographical Information Science*, 2015.

Jean-Daniel Boissonnat and Clément Maria. The simplex tree: An efficient data structure for general simplicial complexes. In *Algorithms – ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 731–742. Springer Berlin Heidelberg, 2012.

CGAL. Cgal - computational geometry algorithms library, 2014. Available from: `http://www.cgal.org`.

Guillaume Damiand. Moka modeler, 2014. Available from: http://moka-modeller.sourceforge.net.

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

René Descartes. *Discours de la méthode*. Jan Maire, Leyde, 1637.

Vincenzo Ferrucci. Generalised extrusion of polyhedra. In *2nd ACM Solid Modelling '93*, pages 35–42. ACM, 1993.

GBKN. Gbkn, de standaard basiskaart van nederland, 2014. Available from: http://www.gbkn.nl.

GDAL. Ogr: Ogr simple features library, 2014. Available from: http://gdal.org/1.11/ogr/index.html.

GOCAD Project. gocad - home, 2011. Available from: http://www.gocad.org/w4/.

Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

Het Waterschapshuis. Ahn - website - homepage, 2014. Available from: http://www.ahn.nl.

Kadaster. Bag, 2014a. Available from: http://www.kadaster.nl/bag.

Kadaster. Top10nl, 2014b. Available from: http://www.kadaster.nl/web/Themas/Registraties/BRT/TOP10NL.htm.

Gail Langran and Nicholas R. Chrisman. A framework for temporal geographic informations. *Cartographica*, 25(3):1–14, 1988.

Hugo Ledoux and Martijn Meijers. Topologically consistent 3D city models obtained by extrusion. *International Journal of Geographical Information Science*, 25(4):557–574, 2011.

Pascal Lienhardt. n-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry and Applications*, 4(3):275–324, 1994.

Pascal Lienhardt, Xavier Skapin, and Antoine Bergey. Cartesian product of simplicial and cellular structures. *International Journal of Computational Geometry and Applications*, 14(3):115–159, 2004.

Hiroshi Masuda. Topological operators and boolean operations for complex-based non-manifold geometric models. *Computer-Aided Design*, 25(2), 1993.

Nathan C. Myers. Traits: A new and useful template technique. *C++ Report*, June 1995.

Open Source Initiative. The mit license (mit) | open source initiative, 2014. Available from: http://opensource.org/licenses/MIT.

Donna J. Peuquet. It's about time: A conceptual framework for the representation of temporal dynamics in geographic information systems. *Annals of the Association of American Geographers*, 84(3):441–461, 1994.

Simon Pigot and N. W. J. Hazelton. The fundamentals of a topological model for a four-dimensional GIS. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, pages 580–591, 1992.

M.H. Poincaré. Analysis situs. *Journal de l'École polytechnique*, 2(1):1–123, 1895.

M. Poudret, A. Arnould, Y. Bertrand, and P. Lienhardt. Cartes combinatoires ouvertes. Technical Report 2007-01, Laboratoire SIC, UFR SFA, Université de Poitiers, October 2007.

Martin Reddy. Obj - wavefront object files, 1994. Available from: http://www.martinreddy.net/gfx/3d/OBJ.spec.

B. Riemann. *Ueber die Hypothesen, welche der Geometrie zu Grunde liegen*. PhD thesis, Abhandlungen der Königlichen Gesellschaft der Wissenschaften zu Göttingen, 1868.

J. Rossignac and M. O'Connor. SGC: A dimension-independent model for pointsets with internal structures and incomplete boundaries. In M. Wosny, J. Turner, and K. Preiss, editors, *Proceedings of the IFIP Workshop on CAD/CAM*, pages 145–180, 1989.

Jonathan Richard Shewchuk. A condition guaranteeing the existence of higher-dimensional constrained delaunay triangulations. In *Proceedings of the 14th Annual Symposium on Computational Geometry*, pages 76–85. ACM, 1998.

Jonathan Richard Shewchuk. Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations. In *Proceedings of the 16th Annual Symposium on Computational Geometry*, pages 350–359. ACM, 2000.

Peter van Oosterom and Jantien Stoter. 5D data modelling: Full integration of 2D/3D space, time and scale dimensions. In *Proceedings of the 6th International Conference GIScience 2010*, pages 311–324. Springer Berlin / Heidelberg, 2010.

M.F. Worboys. A model for spatio-temporal information. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, pages 602–611, 1992.