Constructing an n-dimensional cell complex from a soup of (n–1)-dimensional faces

Ken Arroyo Ohori, Guillaume Damiand, Hugo Ledoux

Constructing an n-dimensional cell complex from a soup of $(n-1)$-dimensional faces*

Ken Arroyo Ohori[1], Guillaume Damiand[2], and Hugo Ledoux[1]

[1] Delft University of Technology, Delft, Netherlands
[2] Université de Lyon, LIRIS, UMR 5205 CNRS, 69622 Villeurbanne, France

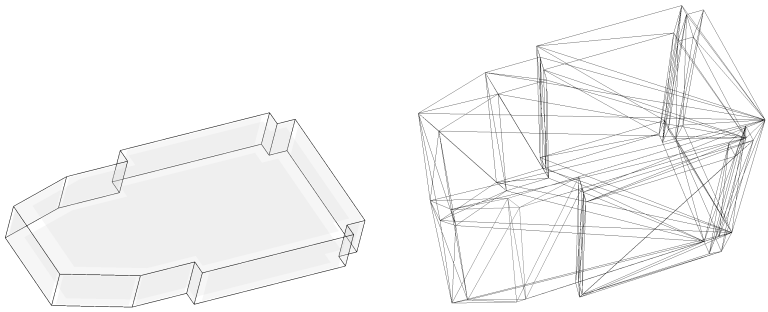
Abstract. There is substantial value in the use of higher-dimensional (>3D) digital objects in GIS that are built from complex real-world data. This use is however hampered by the difficulty of constructing such objects. In this paper, we present a dimension independent algorithm to build an n-dimensional cellular complex with linear geometries from its isolated $(n-1)$-dimensional faces represented as combinatorial maps. It does so by efficiently finding the common $(n-2)$-cells (ridges) along which they need to be linked. This process can then be iteratively applied in increasing dimension to construct objects of any dimension. We briefly describe combinatorial maps, present our algorithm using them as a base, and show an example using 2D, 3D and 4D objects which was verified to be correct, both manually and using automated methods.

1 Introduction

Higher-dimensional digital objects represent well-defined extents of space in arbitrary dimensions. In geographic information systems (GIS), these can be generated when 2D/3D space, time [12], scale [15], semantics [1], or others are all treated as independent axes of a coordinate system. An example of such an object is presented in Fig. 1. This form of representation offers interesting advantages compared to traditional ones where multiple representations of the same object are stored separately and linked in an ad hoc fashion, such as conceptual simplicity, immediate access to all existing topological relationships, the possibility to represent complex events like motion and the ease of maintaining consistency between objects [14].

However, creating higher dimensional digital objects from real-world data is inherently difficult on multiple levels. Since we are usually only familiar with up to 3D physical space, describing such objects might be straightforward mathematically, but it is nonetheless unintuitive. Higher dimensional data models are also complex, and thus realising even very simple objects requires a large number of operations on abstract elements. Finally, manipulating the related data

* Paper published in Proceedings of 1st International Conference on Applied Algorithms, LNCS 8321, pp. 37-48, January 2014. Thanks to Springer Berlin Heidelberg. The original publication is available at `http://dx.doi.org/10.1007/978-3-319-04126-1_4`



(a) 3D model (perspective projection of 0-, 1- and 2-cells)

(b) 4D model (double perspective projection of 0- and 1-cells only)

Fig. 1. The Aula Congress Centre in the TU Delft campus represented as extruded 3D and 4D models.

structures while ensuring that all its required references are correctly kept is already non-trivial in 3D [9], and increasingly difficult in higher dimensions.

Nonetheless, it is possible to use the concepts behind boundary representation to significantly reduce the difficulty of the problem. In boundary representation (b-rep or BREP), an n-dimensional closed object can be unambiguously described by the $(n-1)$-dimensional boundary that encloses it, as originally defined by Baumgart [2] and Braid [3] for a (3D) solid with a (2D) surface boundary composed of flat polygonal patches. This concept is valid in higher dimensions as well, and is related to the concept of a cell complex [7] in topology, where an n-dimensional cell (n-cell) in the complex has a number of $(n-1)$-cells (faces) as its boundary, and these faces are also part of the complex. An n-cell is an abstract object which is considered to be homeomorphic to an n-ball (e.g. point, segment, disk, ball, etc.). In this paper we use combinatorial maps to represent a cell complex, which are described in Sec. 2.

The $(n-1)$-dimensional boundary of an n-cell is much easier to conceive than the original n-cell, since the $(n-1)$-cells that it is composed of can be themselves described individually. However, this requires the existence of an algorithm that is able to connect these separate $(n-1)$-cells to form the n-cell in an efficient manner, abstracting such issues as incompatible orientations in the model, the handling of duplicate cells and the identification of common boundaries. This operation, fully dimension independent, presented in Sec. 3 and the focus of the present paper, can be performed recursively in increasing dimension to generate arbitrary cell complexes in any dimension, and thus we refer to it as incremental construction.

Another possible use of incremental construction is to generate the topological information, i.e. incidence and adjacency, between a set of existing objects. This is in fact simply a subset of what the problem incremental construction is meant to solve—the identification of common boundaries—, and fits very well

within the frame of GIS, where data models tend to contain very limited topological information but topological queries are of great importance [5]. Some GIS models, like the OGC Simple Features Specification [10] have no topology in their structure, even repeating the coordinates of individual points when these appear in multiple line segments or polygons. Others, such as CityGML [11], only have implicit topological information (e.g. these surfaces should form a closed shell) which is often unenforced in their geometry.

We have implemented our algorithm based on the CGAL Combinatorial Maps and Linear Cell Complex packages, which are described in Sec. 4 together with the details of our implementation, including a discussion of the computational complexity of our approach. In Sec. 5, we present an example in 4D that shows this approach in practice, comparing its output with correct results which have been generated by the Linear Cell Complex package and verifying it analytically. We finish with conclusions and discussion in Sec. 6.

2 Combinatorial maps

Combinatorial maps (or simply maps) are an ordered topological model originally proposed by Edmonds [4] to describe the 2D surfaces of 3D objects. Their extension to arbitrary dimensions is described by Lienhardt [8] for objects without boundaries and extended by Poudret et al. [13] to objects with boundaries. They are able to describe subdivisions of orientable quasi-manifolds[3].

Intuitively, a combinatorial map is composed of two elements: *darts* and relations between them (β). The precise definition of a dart is related to an underlying simplicial decomposition of the object, each dart being equivalent to a simplex in it. However, intuitively it can be seen as an oriented edge on the boundary of a facet, which itself is on the boundary of a solid, which itself is on the boundary of a 4-cell, and so on. It is therefore equivalent to the half-edge data structure in 2D, but extends naturally to higher dimensions. Meanwhile, the relations are functions connecting darts that are related along a certain dimension. In this manner, β_1 joins consecutive oriented edges within a facet forming a loop, β_2 joins adjacent facets within a solid, β_3 joins adjacent solids within a 4-cell, and so on. As in other models based on directed elements, β_i-joined darts for $i > 1$ have opposite orientations.

More formally, an n-dimensional combinatorial map (or n-map) is defined by an $(n+1)$-tuple $M = (D, \beta_1, \ldots, \beta_n)$ where D is a finite and non-empty set of darts, β_1 is a partial permutation on D (a function $f : D \cup \{\varnothing\} \to D \cup \{\varnothing\}$ such that $\forall d_1 \in D$, $\forall d_2 \neq d_1 \in D$, $f(d_1) \neq \varnothing$ and $f(d_2) \neq \varnothing \Rightarrow f(d_1) \neq f(d_2)$), $\forall 2 \leq i \leq n$, β_i is a partial involution (a partial permutation f such that $\forall d \in D$, $f(d) \neq \varnothing \Rightarrow f(f(d)) = d$), and $\forall 1 \leq i < i+2 \leq j \leq n$, $\beta_i \circ \beta_j$ is also a partial involution (see an example in Fig. 2).

Here, $\varnothing$ is a special value used to indicate that a given dart d has no other dart in relation by a given β_i. In such a case we have $\beta_i(d) = \varnothing$ and that means

[3] A specific combinatorial interpretation of the concept of a manifold. See Lienhardt [8] for details.

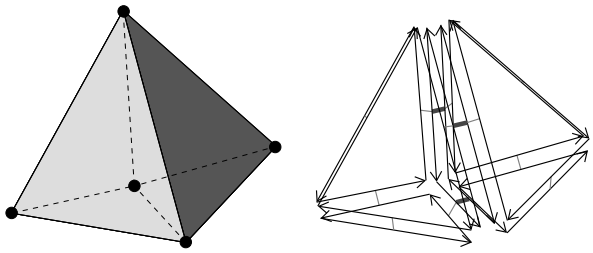


Fig. 2. (Left) A 3D cellular complex composed with two tetrahedra sharing a common face. There are 2 3-cells, 7 2-cells, 9 1-cells and 5 0-cells. (Right) The corresponding 3D combinatorial map having 48 darts.

that d belongs to the i-boundary of the described object (it belongs to only one i-cell). A dart d is said to be i-free when $\beta_i(d) = \varnothing$. Otherwise it is i-sewn with a second dart d' and we have $\beta_i(d) = d' \neq \varnothing$.

In order to traverse an n-map, the orbit operator $< A > (d) =< \beta_{a1}, \ldots, \beta_{ak} > (d)$ obtains all the darts that can be reached from dart d by successive applications of the links $\beta_{a1}, \ldots, \beta_{ak} \in A$. Certain orbits are particularly interesting: for any $1 \leq i \leq n$, $< \not\beta_i > (d) =< \beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \ldots, \beta_n > (d)$ contains all the darts in the i-cell of d, while $< \{\beta_i \circ \beta_j | \forall i, j\, 1 \leq i < j \leq n\} > (d)$ contains all the darts in the 0-cell (vertex) of d. As two darts linked by a β_i have opposite directions (for $2 \leq i \leq n$), they belong to the two vertices of a same edge. Thus by combining two β, we obtain a dart of the same vertex than d.

When the relations in an orbit are applied in a well-defined order, these orbits are generated in a consistent manner. This makes it possible to generate a canonical representation of (a subset of) the darts in a map, which combined with labelled darts can be used to test combinatorial map isomorphism in quadratic time, as shown by Gosselin et al. [6] and demonstrated by searching for patterns in images.

The incremental construction of cells in a combinatorial map is based on the sewing operator, which joins two i-cells along an $(i-1)$-cell which after the operation lies in their common boundary. Intuitively, given two i-free darts d_1 and d_2, the i-sew operation between d_1 and d_2 will pairwise link the orbits of these two darts by β_i so that we will obtain $\beta_i(d_1) = d_2$.

We can see in Fig. 3 an example of the 3-sew operation. Starting from a 3D combinatorial map describing two isolated tetrahedra, we identify two faces by using the 3-sew operations on darts d and d'. This operation puts in relation all the darts of the two initial faces by pairs so that we obtain a valid combinatorial map (i.e. the constraint that $\beta_1 \circ \beta_3$ is a partial involution is still satisfied).

A combinatorial map only describes the topological part of an object in term of a cell complex, i.e. the set of cells in all dimensions and all the incidence and adjacency relations. Applications often require adding information associated with specific cells (e.g. to associate a colour to each vertex, or a normal to each face). This is possible thanks to the attribute notion. An i-attribute is the

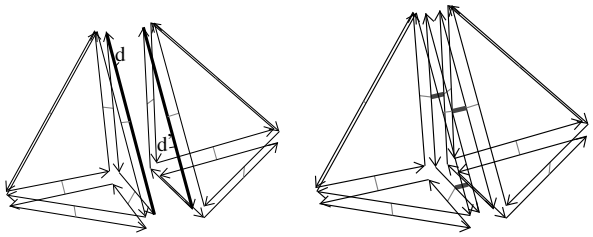


Fig. 3. (Left) A 3D combinatorial map describing 2 isolated tetrahedra. (Right) The 3D combinatorial map obtained after 3-sewing darts d and d'. All the darts of the two initial faces are 3-sewed by pairs.

information associated with i-cells. As cells are implicitly represented by sets of darts in combinatorial maps, links between i-attributes and i-cells are done through the darts of the i-cells: all the darts belonging to a same i-cell are linked to the same i-attribute.

These attributes are very useful as they allow to associate any information to any cell, and given a dart, we have a direct access to all of its associated attributes. Moreover, these attributes can be used to describe the geometry of the objects. Indeed, we can associate to each vertex of a combinatorial map a point in $\mathbb{R}^{d2}$ by using 0-attributes. A combinatorial map with this type of embedding gives a linear cell complex. Indeed, in this case, the geometry of each edge is a segment, the geometry of each face is a planar polygon, and so on. $d2$ is the dimension of the geometry, i.e. the dimension of the ambient space. Generally we have $d2 \geq d$ (d being the combinatorial dimension). For example we can use $d = d2 = 2$ to describe a planar graph embedded in a plane or $d = 2$ and $d2 = 3$ to describe a polyhedral mesh embedded in $\mathbb{R}^3$.

Now given two i-cells in a linear cell complex, we can test if these two cells are identical, i.e. if they have both the same topology and the same embedding information. To test if they have the same topology, we use the technique introduced above for combinatorial map isomorphism, by considering only the two i-cells instead of two whole combinatorial maps. Testing if the two cells have the same embedding can be done during the isomorphism test, simply by testing if two darts d and d' considered by the isomorphism function are linked with two points with the same coordinates. Note that we have to consider the two possible orientations of one of the two i-cells in order to detect also if the two cells are identical but with reverse orientations. We make use of this technique to efficiently test for the identity of two cells in Sec. 3.

3 Incremental Construction

Since an n-cell in a cell complex can be described by the $(n-1)$-cells on its boundary, the same thinking can be applied in reverse: a yet unbuilt n-cell can be constructed based on a set of $(n-1)$-cells which are known to form its complete (closed) boundary.

The algorithm is applied object by object in increasing dimension, constructing isolated 0-cells first, and continuing through 2-cells, 3-cells and further. Our explanation follows the construction of *single cell* of a certain dimension, starting with unique isolated vertices in Sec. 3.1. 1-cells are skipped since 2-cells can be easily described as a succession of 0-cells. 2-cells are then built from an ordered sequence of 0-cells, as covered in Sec. 3.2. Finally, for n-cells ($n > 2$), the method receives an unordered set (soup) of $(n-1)$-cells, the geometries of which are used as faces of the finished n-cell that is returned, as explained in Sec. 3.3. These individual cell creation algorithms are applied object by object in order of increasing dimension, so that the lower dimensional cells generated as the output of earlier stages can be used as the input of latter stages.

The incremental construction algorithm as a whole keeps track of already built cells by maintaining a reference to one of their darts, making sure that no identical cells (with equal geometry and topology) are ever created. For efficiency reasons, it is convenient to use darts which are embedded into the lexicographically smallest points of each cell, which combined with smallest point indices for all existing n- and $(n-1)$-cells, and the $(n-2)$-cells on the boundary of the $(n-1)$-cells to be used, greatly accelerates this process.

3.1 Vertices (0D)

A single point, as defined by a unique tuple of coordinates, can be present in multiple higher dimensional cells and thus described multiple times. However, to ensure the correct generation of topology and enforce the orientable quasi-manifold criterion, it should only be created once. When doing so, a new point embedding at that location should be created, and a new free dart representing the 0-cell should be created and linked to it. Every new instance of the same point should then link to this original dart.

The output of this step is thus a map from each input point to a 0-cell (dart embedded into a point) at that location (see an example in Fig. 4(a)).

3.2 Facets (2D)

In order to create a 2-cell from a sequence of 0-cells, three general steps are needed:

1. The unique 0-cells, as resulting from the evaluation of the 0-cells' point embeddings in the map obtained when processing all 0-cells, are obtained. The result of this evaluation might be the same 0-cell as provided, or an already existing 0-cell at that location. Each of these 0-cells might be a single 1-free dart, in which case it can be used directly (see an example in Fig. 4(b)), or it can be a non 1-free group of darts, in which case it has already been used as part of a different 1-cell (and possibly other higher dimensional cells). These darts used in 1-cells can be reused only when they would become part of the 2-cell that will be built and are not part of any 2-cell. If the darts cannot be reused, a copy of them *with opposite orientation*

has to be created, linking it to the same embedding as the original (see examples in Fig. 4(c) to (h). In all these cases, there is at least one dart which is duplicated). The opposite orientation ensures that the two (old and new) can then be 2-sewn when constructing a 3-cell in a subsequent step.
2. The darts obtained in the previous step are 1-sewn sequentially, and the last is 1-sewn to the first, forming a closed loop.
3. Just as in the creation of 0-cells, a 2-cell can be on the boundary of multiple higher dimensional cells (or simply described multiple times in the input), and as such, it is necessary to check if the 2-cell has already been created. If any comparison returns that the 2-cell already exists, the newly created darts are deleted[4] and the existing one is returned, otherwise the new 2-cell is returned.

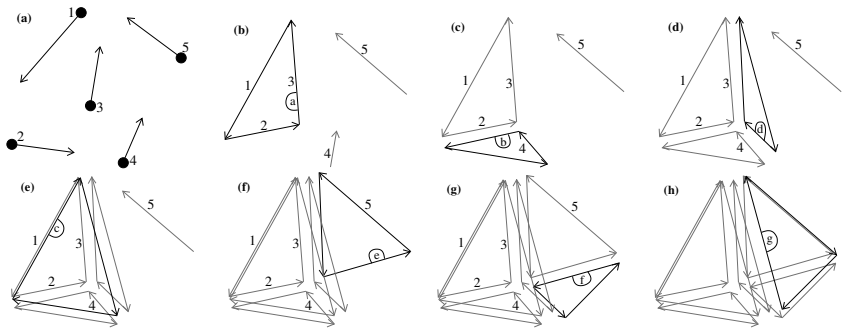


Fig. 4. Illustration of the different steps of the reconstruction of 2-cells. (a) Initial configuration: one dart per vertex. (b) After a = `make_2_cell(1, 2, 3)`. (c) After b = `make_2_cell(2, 4, 3)`. (d) After d = `make_2_cell(1, 4, 3)`. (e) After c = `make_2_cell(1, 4, 2)`. (f) After e = `make_2_cell(1, 3, 5)`. (g) After f = `make_2_cell(5, 3, 4)`. (h) Final result, after g = `make_2_cell(4, 5, 1)`.

3.3 3-cells and higher (dimension independent)

The method to create n-cells from their $(n-1)$-cell boundaries is identical for all $n > 2$, allowing a dimension-independent function to be created. As with the creation of 2-cells from 0-cells, it consists of three general steps:

1. First of all, whether each $(n-1)$-cell has already been created beforehand is checked. This is meant so that multiple identical $(n-1)$-cells are never created in the final cell complex, even when they are given as input. If an $(n-1)$-cell already exists and it is $(n-1)$-free, it is reused as part of the n-cell

[4] Since the 2-cell already exists, all the used darts are copies of existing ones. This deletion thus does not erase any unique instance of a 0- or 1-cell.

(see an example in Fig. 5(a)). If it exists but is already part of a different n-cell, it is duplicated with reverse orientation (see an example in Fig. 5(b) for face labeled d). As in the case of 2-cells, this is done so that it has the same geometric embeddings and attributes, but its opposite orientation ensures that the two (old and new) can be directly n-sewn together.

2. The $(n-1)$-cells (faces) are $(n-1)$-sewn along their common $(n-2)$-dimensional boundaries (ridges). If two groups of connected $(n-1)$-cells with incompatible orientations would be joined by this operation (i.e. the pair of two corresponding ridges have the same orientation), the orientation of one of the groups is reversed before the link is created. If more than one match for a ridge is found, the object being represented is not a quasi-cellular manifold, and thus cannot be represented using combinatorial maps.
3. The newly constructed n-cell is finally compared to other n-cells to check if it already exists. If an n-cell is found to exist, the algorithm should delete the darts that are part of the newly created n-cell and instead return the existing cell. This ensures that only a single instance of an n-cell is created.

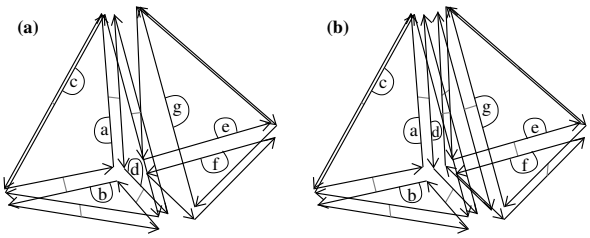


Fig. 5. Illustration of the two steps of the reconstruction of 3-cells. We start from the combinatorial map given in Fig. 4(h) which is the result of the reconstruction of 2-cells. (a) After `make_3_cell(`a, b, c, d`)`. (b) Final result, after `make_3_cell(`d, e, f, g`)`.

4 Implementation and Complexity

We have implemented the incremental construction algorithm in C++ using the Combinatorial Maps and Linear Cell Complex packages in CGAL[5]. In order to improve the performance of the incremental construction algorithm, we use some indices that map the lexicographically smallest point embedding of some cells of a given dimension to a dart embedded at that location. These indices are implemented as C++ Standard Library[6] `map`s with point embeddings as keys and `list`s of darts as values, using a custom compare function so that the points are internally sorted in lexicographical order. Because `std::map` is

[5] The Computational Geometry Algorithms Library: `http://www.cgal.org`
[6] For instance, the GNU Standard C++ Library: `http://gcc.gnu.org/libstdc++/`

normally implemented as a self-balancing binary search tree, $O(\log n)$ search, insertion and deletion times and $O(n)$ space can be expected.

Since we create objects dimension by dimension, it is not necessary to maintain indices for all the cells of all dimensions at the same time. The only ones used are: all n- and $(n-1)$-cells, and the $(n-2)$-cells on the boundary of the $(n-1)$-cells for that step. Most of these can be built incrementally, adding new cells as they are created in $O(\log c)$, with c the number of cells of that dimension, assuming that the smallest vertex and a dart embedded there are kept during its construction. The complexity of building any index of cells of any dimension is thus $O(c \log c)$ and it uses $O(c)$ space. Note that this also gives the computational complexity of creating a map of all unique 0-cells in the cell complex.

Checking whether a given cell already exists in the cell complex is more complex. Finding a list of cells that have a certain smallest vertex is done in $O(\log c)$. Theoretically, all existing cells in the complex could have the same smallest vertex, leading to up to c quadratic time cell-to-cell comparisons just to find whether one cell exists. However, every dart is only part of *a single* cell of any given dimension, so while every dart could conceivably be a starting point for the identity comparison, a single dart cannot be used as a starting point in more than one comparison, and thus a maximum of d_{complex} identity comparisons will be made for *all* cells, with d_{complex} the total number of darts in the cell complex. From these d_{complex} darts, two identity comparisons are started, one assuming that the two cells (new and existing) have the same orientation, and one assuming opposite orientations. Each of these involves a number of dart-to-dart comparisons in the canonical representations that *cannot* be higher than the number of darts in the smallest of the two cells. The number of darts in the existing cell is unknown, but starting from the number of darts in the newly created cell (d_{cell}), it is safe to say that no more than d_{cell} dart-to-dart comparisons will be made in each identity test, leading to a worst-case time complexity of $O(d_{\text{complex}} d_{\text{cell}})$. Note that this is similar to an isomorphism test starting at every dart of the complex.

Finally, creating an n-cell from a set of $(n-1)$-cells on its boundary is more expensive, since the $(n-2)$-cell (ridge) index needs to be computed for every n-cell. Following the same reasoning as above, it can be created in $O(r \log r)$ with r the number of ridges in the n-cell, and uses $O(r)$ space. Checking whether a single ridge has a corresponding match in the index is done in $O(d_{\text{cell}} d_{\text{ridge}})$, with d_{cell} the number of darts in the n-cell and d_{ridge} the number of darts in the ridge to be tested. Since this is done for all the ridges in an n-cell, the total complexity of this step, which dominates the running time of the algorithm, is

$$\sum_{\text{ridges}} O(d_{\text{cell}} d_{\text{ridge}}) = O(d_{\text{cell}}^2).$$

The analyses given above give an indication of the computational and space complexity of the incremental algorithm as a whole. However, it is worth noting that in realistic cases the algorithm fares far better than in these worst-case scenarios: the number of cells that have a certain smallest vertex is normally far

lower than the total number of cells in the complex, most of their darts are not embedded at the smallest vertex, and from these darts most identity comparisons will fail long before reaching the end of their canonical representation.

Finally, one more nuance can affect the performance of this approach. We have discussed that when two groups of darts with incompatible orientations have to be joined, the orientation of one of these has to be reversed. This is easily done by obtaining all the connected darts of one of the groups, preferably the one that is expected to be smaller, and reversing their orientation 2-cell by 2-cell. Every dart d in a 2-cell is then 1-sewn to the previous dart in the polygonal curve of the 2-cell ($\beta_1^{-1}(d)$). A group of n darts can then have its orientation reversed in $O(n)$ time. This is not a problem in practice since GIS datasets generally store nearby objects close together, but if a cell complex is incrementally constructed in the worst possible way, i.e. creating as many disconnected groups as possible, this could have to be repeated for every cell of every dimension.

5 Example

The CGAL Linear Cell Complex package provides functions to generate a series of primitives which are known to be created with correct geometry and topology, and can then be sewn together to generate more complex models. We have therefore created various 2D, 3D and 4D cell complexes using both these functions and our approach. In this manner it was possible to test the validity of our models using the identity comparison described in Sec. 2, as well as manually verifying all β-links.

In the following we show an interesting case, how a tesseract (see Fig. 6) can be generated using our approach. A tesseract is the 4D analogue of a cube, and is a 4-cell bounded by 8 cubical 3-cells, each of which is bounded by 6 square 2-cells. It thus consists of one 4-cell, 8 3-cells, 24 2-cells, 32 1-cells and 16 0-cells.

Using our approach, an empty 0-cell index is first created. Then, the 16 vertices of the tesseract, each vertex p_i described by a tuple of coordinates $(x_i, y_i, z_i, w_i$, can be created as $p_i =$ `make_0_cell`(x_i, y_i, z_i, w_i), which returns a unique dart embedded at each location, and added to the 0-cell index. At this point, the algorithm would have built an unconnected cell complex consisting solely of 16 completely free darts.

An empty index of 2-cells is then created. Each of its 24 square facets can be built based on its vertices as $f_i =$ `make_2_cell`(p_j, p_k, p_l, p_m), which 1-sews (copies of) these darts in a loop and returns the dart embedded at the smallest vertex of the facet. These are added to the index of 2-cells. Since every vertex is used in 6 different 2-cells, each dart would be copied 5 times. The cell complex at this point thus consists of 24 disconnected groups of 4 darts each.

Next, an empty index of 3-cells is created and the index of 0-cells can be deleted. For each of the 8 cubical 3-cells, a function call of the form $v_i =$ `make_3_cell`$(f_j, f_k, f_l, f_m, f_n, f_o)$ is made. At this point, an index of the 1D ridges of each face is built, which is used to find the 12 pairs of corresponding ridges that are then be 2-sewn together. When a 3-cell is created, it is added

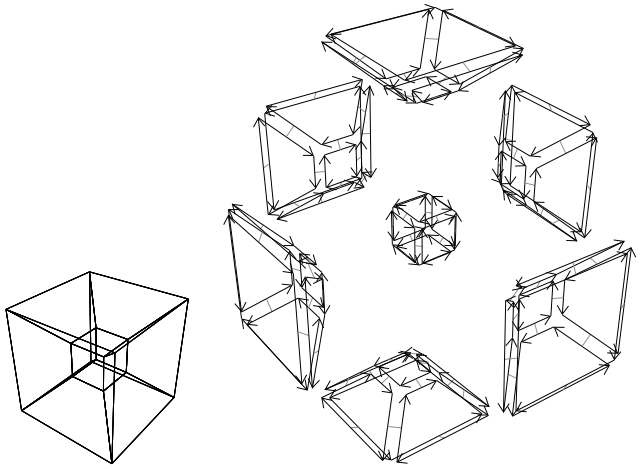


Fig. 6. (Left) A tesseract (edges only). (Right) A combinatorial maps representation of a tesseract, β_3 and the "external" cube are omitted for clarity.

to the index. Since every facet bounds two 3-cells, each dart is duplicated once again, resulting in a cell complex of 8 disconnected groups of 24 darts each.

Finally, the tesseract is created with the function $t =$ `make_4_cell(`$v_1, v_2, \ldots, v_8$`)`. This can use the index of 2-cells to find the 24 corresponding pairs of facets that are then 3-sewn to generate the final cell complex.

We tested the validity of this object by performing a series of tests on the structure (complete and symmetric sewing), testing whether each cube was identical to the expected outcome, and manually verified the β-links of its 192 darts.

6 Conclusions and future work

We have shown that it is possible to apply the fundamental concept of boundary representation, describing an n-cell by its $(n-1)$-dimensional faces, to incrementally construct cell complexes of any dimension. To the best of our knowledge, this technique is the only one that has been described and/or implemented for 4D cell complexes or higher. Using a variety of indices is efficient, generating an n-cell in $O(d^2)$ in the worst case, with d the total number of darts in the cell, and our algorithm should fare markedly better in realistic datasets.

We intend to use this approach, supplemented with techniques under development that generate the required $(n-1)$-dimensional faces, to generate objects for higher dimensional geographic information systems, as well as other applications. This will allow us to take real-world 3D city models and incorporate additional dimensions to them, such as time and scale, to create 4D/5D objects to which higher dimensional analyses can be performed.

7 Acknowledgments

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).

Bibliography

[1] Alkyoni Baglatzi and Werner Kuhn. On the formulation of conceptual spaces for land cover classification systems. In Danny Vandenbroucke, Bénédicte Bucher, and Joep Crompvoets, editors, *Geographic Information Science at the Heart of Europe*, Lecture Notes in Geoinformation and Cartography, pages 173–188. Springer, 2013.

[2] Bruce G. Baumgart. A polyhedron representation for computer vision. In *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*, pages 589–596, 1975.

[3] I.C. Braid. The synthesis of solids bounded by many faces. *Communications of the ACM*, 18(4):209–216, 1975.

[4] J. Edmonds. A combinatorial representation of polyhedral surfaces. *Notices of the American Mathematical Society*, 7, 1960.

[5] Max J. Egenhofer and R. D. Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2):161–174, 1991.

[6] Stéphane Gosselin, Guillaume Damiand, and Christine Solnon. Efficient search of combinatorial maps using signatures. *Theoretical Computer Science*, 412(15):1392–1405, March 2011.

[7] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

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

[9] Martti Mäntylä. *An introduction to solid modeling*. Computer Science Press, New York, USA, 1988.

[10] OGC. *OpenGIS Implementation Specification for Geographic Information - Simple Feature Access - Part 1: Common Architecture*. Open Geospatial Consortium, 1.2.1 edition, May 2011.

[11] Open Geospatial Consortium. *OGC City Geography Markup Language (CityGML) Encoding Standard*, 2.0.0 edition, April 2012.

[12] Donna J. Peuquet. *Representations of Space and Time*. Guilford Press, 2002.

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

[14] Jantien Stoter, Hugo Ledoux, Martijn Meijers, Ken Arroyo Ohori, and Peter van Oosterom. 5D modeling - applications and advantages. In *Proceedings of the Geospatial World Forum 2012*, page 9, April 2012.

[15] Peter van Oosterom and Martijn Meijers. Towards a true vario-scale structure supporting smooth-zoom. In *Proceedings of the 14th ICA/ISPRS Workshop on Generalisation and Multiple Representation, Paris*, 2011.