Streaming Hyper-Cuboidal Meshes from Polytopal Complexes

A. Paoluzzi† C. Bajaj‡ V. Pascucci* G. Scorzelli*

†*Dipartimento Informatica e Automazione, Università "Roma Tre", Roma, Italy*
‡*Institute of Computational Engineering and Sciences & Department of Computer Science, University of Texas at Austin, USA*
**Scientific Computing and Imaging Institute & School of Computing, University of Utah, Salt Lake City, USA*

Abstract

We introduce a novel method for dynamic generation of decompositions of polytopal complexes with meshes of quads, hexahedra and higher dimensional cells having the hypercube topology. Both the algorithm and the mesh representation are multidimensional, and can be used with complexes of dimension two, three, four, and with discrete higher-dimensional spaces, both manifold and non manifold. The generation is very fast, since the refined meshes are derived by previously available cells using topological formulas and code templates, and can be easily parallelized. The proposed approach can be applied to any cell decomposition with convex cells, both solid and embedded, of any intrinsic dimension. Last but not least, the refined cells have the hypercube topology, as required by most simulation codes.

Key words: Computational geometry and topology, Geometric and topological representations, Geometry generation, processing, compression, and transmission

1. Introduction

The last open problem (P23) at the very end of Edelsbrunner's book "Geometry and Topology for Mesh Generation" [7], asks "What is the asymptotic order of the maximum number of hexahedral in a hexahedral mesh with n vertices?". In this paper we introduce a constructive method answering some related questions in a multidimensional setting.

First we give a definition of *hyper-cuboidal complex* (HCC), that generalizes the concept of hexahedral mesh to higher (and lower) dimensional spaces. Then we introduce a novel method to partitionate any finite-dimensional d-complex of polytopes, and to produce a HCC where *every k-cell has the topology of the k-hypercube* ($0 \leq k \leq d$). In 2D we obtain a mesh of quads from any mesh of convex polygons; in 3D we get an hexahedral mesh from any mesh of 3-polytopes such that all the vertices of every cell are trihedral. An interesting result is the following: a d-polytopal mesh E is subdivided into a HCC with n vertices, $n = |E^0| + \cdots + |E^d|$, where E^k denotes the input k-skeleton.

In particular, we use the abstract poset of a cell complex, completely characterized by its Hasse graph, as a representation of the complex. The Hasse graph $H = (N, A)$ of a d-complex is a d-partite graph, where nodes N and arcs A are partitioned into disjoint subsets N_k and $A_k \subset N_k \times N_{k+1}$ ($0 \leq k \leq d-1$), respectively.

The very simple generative algorithm introduced in this paper builds in a large extent over an amazing property relating the hypercube and simplex topologies: it is known [8] that the Hamming graph (see Section 2) is isomorphic to the Hasse diagram of the powerset lattice. In particular, the Hamming graph of the d-hypercube is isomorphic to the Hasse graph of the $(d-1)$-simplex. Of course, the topology of the simplex is combinatorially much easier to work with, since a d-simplex contains all $\binom{d}{k}$ subsets of k vertices as $(k-1)$-faces.

Strengths of this approach are: (a) an upper bound on the hexahedral decompositions of d-polytopal meshes with n-vertices, (b) a constructive characterization of the hexahedral mesh topology, and (c) a d-dimensional subdivision hierarchy enabling efficient hexahedral streaming, compression, and progressive computations on meshes. The main open questions left by the present work are: (a) the current decomposition is not (yet) provably optimal, and (b) no bounds are yet given on the quality of the hexahedral mesh elements, e.g. by guaranteeing that the Jacobian norm is positive for all hexahedra in the mesh in the 3D case.

Some motivating references related to hexahedral decompositions are: [5], [6], [11], [2], [3], [12].

The paper is organized as follows. Section 2 provides some background about polytopal complexes and related

concepts like the Hasse graph as the representation of a chain (cochain) complex and the operations of boundary and coboundary. Section 3 deals with some introductory concepts implied by the relationship between the Hasse diagram of a d-complex and the Hamming graphs of k-cuboids ($0 \leq k \leq d$), that is the main tool used in this paper. Section 4 discusses the HCC (Hyper-Cuboidal Complex) generative algorithm, giving both an informal preview and a formalized characterization. Section 5 quickly explores the range of validity of the algorithm. Section 6 provides a set of examples, showing the rate of growth of the number of k cells in a d-grid, and supplies the generation of decompositions of the circle, the sphere, and the simplest non-convex polyhedron. In the conclusion section we outline the open problems and the next research directions related to this approach.

2. Background

2.1. *Polytopal complex*

A polytope in $\mathbb{E}^d$ is a bounded convex subset of $\mathbb{E}^d$. A polytopal complex is a cell complex where all cells are polytopes. Simplicial complexes, quad meshes, hexahedral meshes, and in general every well-formed decomposition of finite spaces with convex subsets are polytopal complexes.

2.2. *Hasse graph representation*

A transitive reduction of a binary relation R on a set X is a minimal relation R' on X such that the transitive closure of R' is the same as the transitive closure of R. The Hasse diagram is a mathematical diagram used to represent a finite poset (partially ordered set) as a drawing of its transitive reduction. In practice, it is an implicitly oriented drawing of a relation of partial order. In our setting a Hasse diagram represents the (direct) containment between the faces of a set of polytopes.

The Hasse graph can be shown to be a direct implementation of the chain (cochain) complex defined by a decompositive representation scheme in solid modeling [4]. This representation is very general, in that it applies to most domains that can be characterized as cell complexes, without any restrictions on their type, dimension, codimension, orientability, manifoldness, and connectedness.

2.2.1. *Implementation*

The Hasse diagram implementation we make use in the XGE geometric kernel of the Python package pyplasm (see Section A), is locally a DAG but is globally cyclic, since every highest level node $n \in N_d$ (i.e. a d-face) is connected by a direct arc (n, m) to every lowest level node $m \in N_0$ (i.e. a 0-face) it is composed of. In other words, the arc subset $A_d \subset N_d \times N_0$ corresponds to the relation of convex combination between vertices and highest-dimensional cells of the complex; conversely, the subsets of arcs $A_k \subset N_{k-1} \times N_k$ ($1 \leq k \leq d$) represent the (direct) containment between faces of a polytopal complex.

2.3. *Hypercube topology*

The topology of hypercubes is amazingly rich and beautiful. Some aspect of this richness, useful in the remainder of this paper, are summarized in this section.

2.3.1. *Definitions*

The d-dimensional hypercube $Q^d := [0, 1]^d$ can be defined as the Cartesian product of d closed intervals $[0, 1] \subset \mathbb{R}$. It can be also defined either as the convex hull of its 2^d vertices, or as the point-set intersection of internal half-spaces bounded by the affine hulls of $2d$ (d-1)-facets. Let remember that a half-space is either of the two parts into which a hyperplane divides an affine space. Here the hyperplanes are the affine hulls of hypercube facets, i.e. of convex faces of dimension $d - 1$.

2.3.2. *Coordinate representation*

Each d-hypercube vertex can be identified by a d-tuple of binary coordinates. The 0-dimensional hypercube has only one vertex, with zero coordinates. Inductively, the d-hypercube can be generated by adding either one more 0 or one more 1, respectively, to two copies of the coordinates of vertices of the $(d - 1)$-dimensional hypercube.

2.3.3. *Hamming graph*

The Hamming graph $H(d, q)$ is the graph Cartesian product of d copies of the complete graph K_q. Therefore it is endowed with q^d vertices.

2.3.4. *Hypercube topology*

The d-hypercube topology is completely characterized by the Hamming graph $H(d, 2)$, i.e. by a graph (N, A) with $|N| = 2^d$, and such that there is an ard $(i, j) \in A$ if and only if the arcs n_i and n_j differ precisely by one coordinate. The Hamming graph of the 4-dimensional hypercube is shown in Figure 1.

2.3.5. *Hasse diagrams and Hamming graphs*

It is important to notice that the Hamming graph of the d-hypercube is isomorphic to the Hasse diagram of the $(d - 1)$-simplex. Looking at Figure 1, the reader may note that the drawing is isomorphic to the Hasse diagram of the tetrahedron σ_3, characterized by one 3-face, four 2-faces, six 1-faces, four 0-faces, and one (-1)-face (the empty set). We make use of this property in the mesh refinement algorithm introduced in this paper.

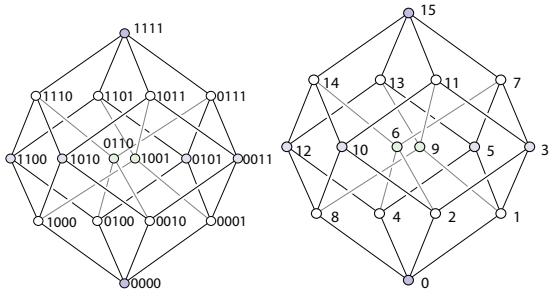

Fig. 1. The Hamming graph of the 4D hypercube. The binary strings in (a) are substituted in (b) by their decimal values.

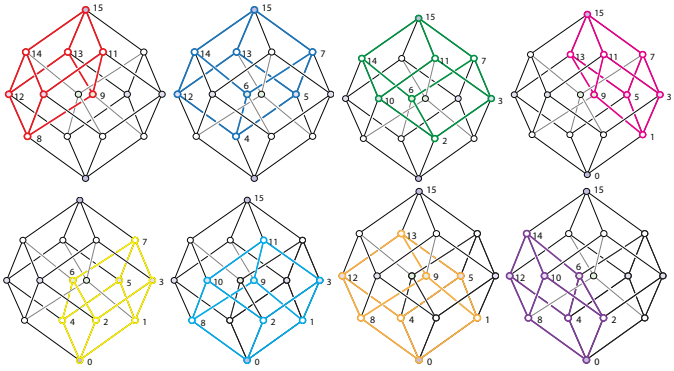

Fig. 2. (a) The four 3-cells incident on vertex $\mathbf{v}_{15}$ of the 4D hypercube; (b) the 3-cells incident on vertex $\mathbf{v}_0$.

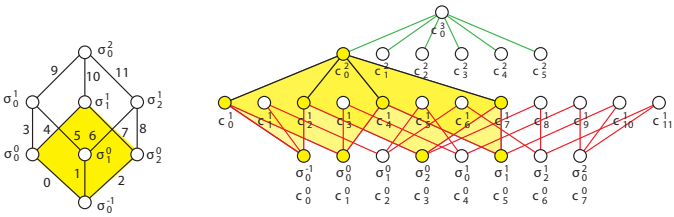

Fig. 3. The relationship between the Hamming graph of the 3-cube (i.e. the Hasse graph $H(\sigma_2)$ of the 2-simplex) and the Hasse graph $H(c_3)$ of the 3-cube. Just a A_1 subset, corresponding to the arcs on the boundary of the yellow face on the left diagram, is shown in the right diagram. In yellow are highlighted the 4 vertex-nodes, the 4 edge-nodes and their face-node.

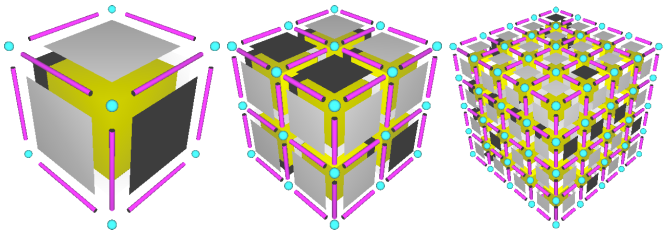

Fig. 4. The hexahedral complexes generated by the decomposition of the 3D cube complex after one and two iterations.

3. Cardinality of Hasse graphs

We can build the Hasse graph $H(c_3)$ of the 3-cube starting from the Hasse graph $H(\sigma_2)$ of the 2-simplex. In particular, we may get the k-level nodes (k-cells) of the output graph $H(c_3)$ from the set of sublattices of depth k in the input graph $H(\sigma_2)$. Therefore, according to the above and looking at Figure 3, we get that the number of 0 cells in $H(c_3)$ is given by the number of nodes $1+3+3+1=8$ in $H(\sigma_2)$; the number of 1-cells in $H(c_3)$ is the number of edges $3+6+3=12$ in $H(\sigma_2)$; the number of 2-cells in $H(c_3)$ is the number of square subgraphs $3+3=6$ in $H(\sigma_2)$.

4. HCC generative algorithm

In abstract terms, we can see the generation of a HCC as a morphism $\gamma : \mathcal{H}^d \to \mathcal{H}^d$ between Hasse graphs of d-complexes, that takes into account the topology of $(k-1)$-simplices ($1 \leq k \leq d$). Let consider the Hasse graphs $H := (N, A)$, and $H' = \gamma(H) := (\gamma(N), \gamma(A)) = (N', A')$. Accordingly, nodes in N'_k are one-to-one with the H subgraphs that are isomorph to the Hasse graph of the simplex σ_{k-1}; and the arc (u, w), with $u, w \in N'$, exists in A' if and only if $\gamma^{-1}(u)$ is a subgraph of $\gamma^{-1}(w)$.

4.1. *Algorithm preview*

The first step of the generative algorithm takes the Hasse graph $H = (N, A)$ of the input complex and starts from the fact that the node set N is one-to-one with the k-faces of the complex ($0 \leq k \leq d$). Therefore, the Hasse graph $H' = (N', A')$ of the output HCC is built bottom-up, starting from the subset N'_0 of nodes representing the 0-faces of the ouput complex. In particular, in each node $n' \in N'$ is stored the coordinate position of the centroid of one face of the complex. By definition, the set N'_0 is isomorphic to the set N via the identity map:

$$N'_0 = \gamma(N) := \mathrm{id}(N) = N. \tag{1}$$

The geometric relationship between the two node sets is elucidated in Figures 3a and 3b, where an exploded view of the HCC of the tetrahedral complex inscribed in the unit 3D sphere is shown.

Subsequently, the subset N'_1 of H' nodes corresponding to the 1-cells of the HCC is built. We therefore look in H for all the subgraphs isomorph to the Hasse graph of the 0-simplex. The latter is an algebraic lattice represented by only two nodes (the zero and the unity) and only one arc. The set of all such subgraphs, to be mapped one-to-one with the node subset N'_1, is therefore constituted by the set A of arcs of $H = (N, A)$. Therefore we can easily build N'_1 from the graph H by considering the downwards arcs leaving from each $n \in N_k$ ($1 \leq k \leq d$), i.e.:

$$N'_1 = \gamma(A) = \bigcup_{k=1}^{d} \gamma(A_k)$$

The arc subset $A'_1 \subset N'_1 \times N'_0$ contains two elements for each node $n' \in N'_1$. In particular, for each $n' = \gamma(a)$, with $a = (n_h, n_k) \in A$, we place in A'_1 the pair of arcs (n', n'_h), (n', n'_k). According to (1), $n'_h = \gamma(n_h) = n_h$, and $n'_k =$

Algorithm 1. Hyper-Cuboidal Complex (HCC) Splitting

Data: graph $g = (N, A) = \cup_k (N_k, A_k), \quad 0 \le k \le d$
Result: *new* graph $g' = (N', A') = (\cup_k N'_k, \cup_k A'_k), \quad 0 \le k \le d$
begin

```
1  for n ∈ N do                                                  // create N'_0
2      new n' ∈ N'_0
3      n' ⟵ Centroid(n)
4  for k ∈ {1, d−1} do                          // create N'_k, A'_{k−1},  1 ≤ k ≤ d
5      for h ∈ {k, d} do
6          for n ∈ N_h do
7              H_n ⟵ DownTraverse(g, n, k)
8              F ⟵ {UpTraverse(g', H_k, k−1) | N'_0 ⊃ H_k ∈ H_n}
9              for F'_{k−1} ∈ F do                                // update N'_k
10                 new n' ∈ N'_k
11                 for f' ∈ F'_{k−1} ⊂ N'_{k−1} do                // update A'_{k−1}
12                     new a' ∈ A'_{k−1}
13                     a' ⟵ (f', n')
14 for n ∈ N_d do                          // create N'_d, A'_{d−1}, A'_d
15     H_n ⟵ DownTraverse(g, n, d)
16     F ⟵ {UpTraverse(g', H_d, d−1) | N'_0 ⊃ H_d ∈ H_n}
17     V ⟵ {RemoveDups(H_d) | N'_0 ⊃ H_d ∈ H_n}
18     for (F'_{d−1}, V'_0) ∈ F × V do                            // create N'_d
19         new n' ∈ N'_d
20         for f' ∈ F'_{d−1} ⊂ N'_{d−1} do                        // update A'_{d−1}
21             new a' ∈ A'_{d−1}
22             a' ⟵ (f', n')
23         for v' ∈ V'_0 ⊂ N'_0 do                                // update A'_d ≡ A'_{−1}
24             new a' ∈ A'_d
25             a' ⟵ (n', v')
26
```

$\gamma(n_k) = n_k$. An exploded picture of the partial construction, up to this point, of the HCC complex represented by the Hasse graph H', is given in Figure **??**c.

The same construction applies to the upper levels N'_k and A'_k $(2 \le k \le d)$ of the H' graph. Specifically, nodes $n' \in N'_k$ are the images, under the morphism γ, of the Hasse graph of the $(k-1)$-simplex, that we denote $H(\sigma_{k-1})$. The next problem hence concerns the extraction from H of all the subgraphs isomorphic to $H(\sigma_{k-1})$. Luckily, the structure of this graph is very simple. We discuss the algorithm for this extraction in the next two subsections.

4.2. *Lattice character of* $H(\sigma_k)$

The special character of Hasse graphs of simplices is inherent to their lattice nature. Therefore they have a lowest element (the zero of the lattice) and a highest element (the lattice's unit). Also, the longest graph path in $H(\sigma_k)$ has length k. Last but not least, $H(\sigma_k)$ is isomorphic to the lattice of the power set $\mathcal{P}(K)$, the set of all the subsets of $K = \{0, 1, \ldots, k\}$. This lattice is built by assuming K as the graph root (the lattice's unit), then by connetting it to the nodes that represent the $k+1$ subsets of k elements extracted from K, and then by repeating ricorsively this construction for each new node, until the $k+1$ singletons $\{0\}, \{1\}, \ldots, \{k\}$ are connected to the empty set, i.e. to the lattice's zero.

4.3. *Enumeration of* H *subgraphs isomorphic to* $H(\sigma_k)$

According to the lattice properties recalled above, the enumeration of the H subgraphs isomorphic to $H(\sigma_k)$ can be generated by considering each node

$$n \in \bigcup_{h=d+1-k}^{d} N_h$$

as the root of a set of lattice subgraphs of H of length k. Let us denote as $\gamma^k(n, m)$ the (unique) lattice subgraph with lenght k, unit n and zero m. In order to generate the nodes belonging to each $\gamma^k(n, m)$ subgraph rooted in n, it is sufficient (a) to compute all the paths of length k leaving downwards from n; (b) sort them with respect to the last element; (c) group the paths in equivalence groups with respect to the same last element m. Finally, the union set of nodes in each group of paths is computed, to remove the repeated nodes, since paths with same unit and zero may share also other nodes.

Of course, the search for subgraph enumeration occurs in H, so that the nodes in every group refer to faces of the input polytopal complex. By means of the canonical identification of N with N'_0, the very same group of nodes $\gamma^k(n, m)$ also refers to nodes of graph H' corresponding to 0-cells of the output HCC.

4.4. *Morphism* $\gamma : H \to H'$

For each H subgraph isomorphic to $H(\sigma_{k-1})$ and rooted in $n \in N$, a new node $\eta(n) =: n' \in N'_k$ is created $(1 \le k \le d)$. The arcs in A'_k are generated bottom-up by traversing upwards H' from the elements in N'_0 of each group $\gamma^k(n, m)$ described in the above subsection, until to reach a subset $S'_{k-1}(n, m) \subset N'_{k-1}$ of nodes of H' corresponding to the $(k-1)$-faces of the output HCC, that get at this point connected to the new node $n' \in N'_k$. Therefore we have, for each layer E'_k of edges in the output ECC:

$$E'_k = \bigcup_{n' \in N'_k} S'_{k-1}(n, m) \times \{n'\}, \qquad 1 \le k \le d$$

4.5. *Comments to HCC splitting algoritm*

Here we discuss the HCC splitting, whose pseudocode is given in Algorithm 1. The formulation is dimension-independent, and can be applied to any-dimensional polytopal complex. It is composed by three stages, denoted below as algorithm initialization, main body and finalization, to be executed in sequence. The process, in bottom-up style, starts by generating the 0 cells of the Hasse graph of the output complex, continues by generating the k-layers of the Hasse graph, and terminates by generating the top-level layer of d-cells, with links to subsets of 0-cells they are convex combination of.

Initialization (lines 1–3)

For every node n in the input Hasse graph $g = (N, A)$, i.e. for every k-face ($0 \leq k \leq d$) in the input polytopal complex, create a 0-level node n' in the output node set N', and set-up its position to the centroid of the face n. Of course, the centroid of a 0-face (vertex) coincides with the vertex itself.

Main body (lines 4–14)

For every intermediate level k of the multipartite graph g, with $1 \leq k \leq d-1$, the algorithm creates the corresponding level sets N'_k, A'_{k-1} in the output graph g'.

This task is accomplished by looking in g for all the sublattices of depth k. In order to be endowed with depth k, such subgraphs must just be rooted in level subsets $N_h \subset N$, with $k \leq h \leq d$. Since we look for sublattices isomorphic to the Hasse graph of the $(k-1)$-simplex, denoted here $H(\sigma_{k-1})$, it suffices to extract only the sets of subgraph nodes, all rooted on n, and denoted as $\mathcal{H}_n$. Actually, the function call $DownTraverse(g, n, k)$ (a) computes all the down-paths of g with root n and length k, (b) sorts them on the last element, (c) groups the paths with the same last element (the zero of each sublattice), finally (d) eliminating the duplicated nodes.

In row (12) of Algorithm 1 each element H_k of the set $\mathcal{H}_n$ is transformed into the set F'_{k-1} of $(k-1)$-faces of the new node $n' \in N'_k$ uniquely associated with the subgraph H_k of g. In particular, a new arc in A'_{k-1} is created for each element in F'_{k-1}.

Finalization (lines 15–26)

This last section of the algorithm is dedicated to the construction of the last pair N'_d, A'_{d-1} of nodes and arcs of the output graph. At the same time, a further set of arcs $A'_d \equiv A'_{-1}$ is created, linking each d-dimensional face (i.e. each node $n' \in N'_d$) to its vertices (nodes $n' \in N'_0$).

5. Range of validity

It is not difficult to understand, looking at the mapping between sublattices of Hasse diagrams and node cells of the generated complex, why Algorithm 1 generates full (hyper)-cuboidal decompositions only when starting from polytopal d-complexes where all d-cells have all vertices on the extremal point of a d-hedral angle, i.e. at the intersection of d internal halfspaces bounded by the affine hulls of the $(d-1)$-facets of the d-cell. In particular, in 3D we need that each internal angle of every 3-cell is a trihedron, where a triple of three affinely independent vectors share a common vertex [1]. When this condition is not satisfied, if the goal it to generate a complex with only cuboidal cells, then some cutting of 3-cells that do not satisfy this requirement is needed. In particular, in order to generate a full hexahedral decomposition of the sphere of Example 6.3, a decomposition of "basket" polytopes of each sphere layer is required, as shown by Figure 8.

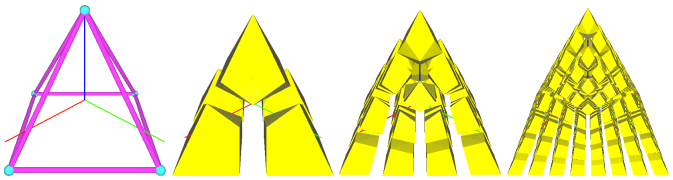

Fig. 5. (a-d) Polytopal decomposition of the pyramid; the top vertex has degree 4, and the cell it belongs to is a "Morse crystal".

Table 1
Cardinalities of k-skeletons of d-grids

d		k_0	k_1	k_2	k_3	k_4	$\sum_i k_i$
1	n	3	2				5
2	n^2	9	12	4			25
3	n^3	27	54	36	8		125
4	n^4	81	216	216	96	16	625
1	n	5	4				9
2	n^2	25	40	16			81
3	n^3	125	300	240	64		729
4	n^4	625	2000	2400	1280	256	6561
1	n	7	6				13
2	n^2	49	84	36			169
3	n^3	343	882	756	216		2197
4	n^4	2401	8232	10584	6048	1296	28561
1	n	9	8				17
2	n^2	81	144	64			289
3	n^3	729	1944	1728	512		4913
4	n^4	6561	23328	31104	18432	4096	83521

The simplest example of non-trihedral polytope is given by the 3D pyramid with squared basis. The iterated application to it of Algorithm 1 is shown in Figure 5. It is possible to notice that all other 3-cells have 8 vertices but are not hexahedrons, because of a non-planarity, that is being smoothed with the distance of cell from the Morse crystal.

6. Examples

In this section we discuss some examples of generation of polytopal and hyper-cuboidal cell complexes. In particular we show the rate of growth of the number of k cells in a d-grid, the generation of decompositions of the circle and the sphere, and the decomposition of the simplest non-convex polyhedron.

6.1. *Cell complexes of multidimensional grids*

Let quickly analyze some numeric relation between the number of cells of the k-skeletons ($0 \leq k \leq d$) of d-dimensional grids, generated by Cartesian product of d instances of a 1-dimensional grid with n vertices (0-cells). In particular, we show in Table 1 that the total number of k-cells ($0 \leq k \leq d$) of the d-complex associated to a d-grid grows with the same order of magnitude than the number

k_0 of vertices of the grid, usually called nodes in numerical analysis and related software systems.

6.2. *Polytopal decomposition of circle*

In this section we show the quad decomposition of polytopal complexes that partition the 2D circle. Such polytopal decompositions are generated by starting from a n-gon centered in the origin, and attaching an irregular pentagon — with three vertices on the next circle boundary — to each 1-cell of the previous boundary, therefore allowing for doubling the number of the boundary cells at each step of refinement. Of course, the construction may be repeated more times, in order to further improve the generated circle approximation. The last picture in Figure 6 shows the refinement into a quadrilateral cell complex generated by the algorithm presented in this paper.

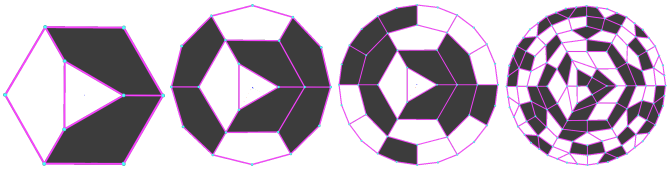

Fig. 6. (a-c) Polytopal decomposition of the circle; (d) quad decomposition of the previous complex.

6.3. *Polytopal decomposition of sphere*

A polytopal decomposition of the 3D sphere may be generated by starting from the unit tetrahedron with centroid in the origin of a Cartesian system in $\mathbb{E}^3$, and with vertices lying on the surface of the unit sphere. To each triangular cell of this "initial" tetrahedron's boundary we may attach a "basket" polytope generated by projecting the vertices of the face from the origin (center of the sphere) to the sphere surface with radius 2. Also, we project the centroids of the face edges to the same circle, and connect the dots, as shown in Figure 7a, where the 1-skeleton of the bottom of such "basket" polytopes is shown. In Figures 7b and 7c the full 1-skeleton of the generated decomposition and the boundary 2-cells of the basket adjacent to the tetrahedral face are shown, respectively. In Figure 7d an exploded view of the generated solid polytopes is finally given. It may be of interest to note that the sphere decomposition so generated is a *geodetic sphere*, since all cell faces belong to a great circle. The construction of the sphere given here is a direct generalization of the circle decomposition given in Example 6.2 and shown in Figure 6, generated by layers with double number of polytopal cells with respect to the previous layer. in this case the number of cells for each layer increases times a factor 6, since on each triangle on the surface of a layer is attached a polytope with superior boundary made by 3×2 triangles, as shown in Figure 7.

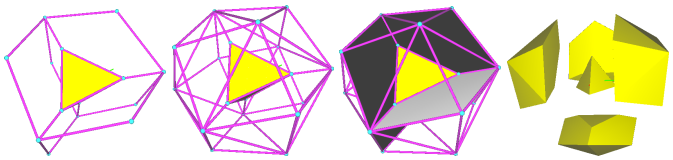

Fig. 7. Some constructive steps in the process of building a polytopal decomposition of the sphere, starting from a tetrahedron centered in the origin.

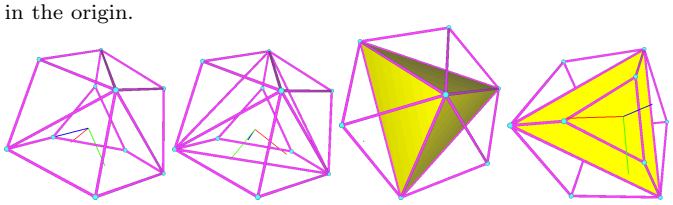

Fig. 8. Decompositions of the non-trihedral basket polytope into 5 trihedral polytopes without adding new vertices.

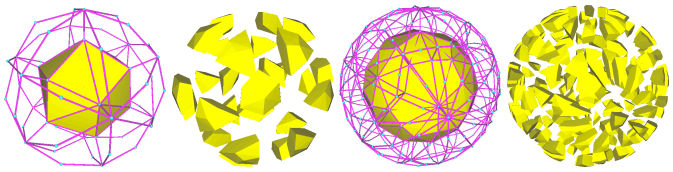

Fig. 9. Polytopal decompositions of the sphere: (a,b) with 2 layers of polytopes; (c,d) with 3 layers of polytopes.

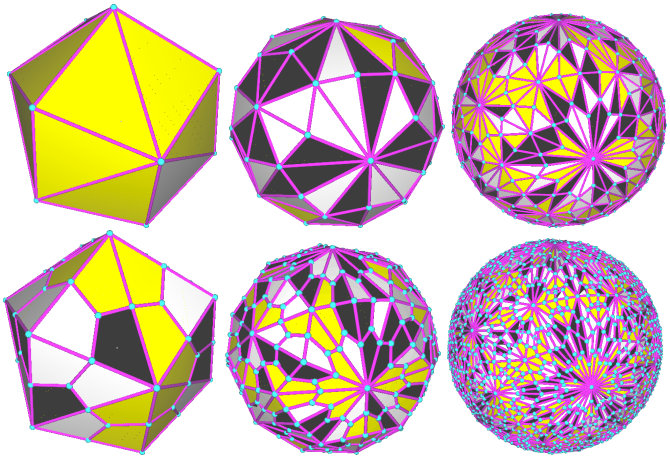

Fig. 10. Spheres with one, two and three polytopal layers. Polytopal decompositions are on the top row. The corresponding hexahedral decompositions are on the bottom row.

6.4. *Polytopal decomposition of Schönhardt polyhedron*

The Schönhardt polyhedron is the simplest non-convex polyhedron that cannot be triangulated into tetrahedra without adding new vertices. It can be generated by rotating in opposite directions the two triangular faces of a wedge (the extruded triangle) about the normal axis. The polytopal complex shown in Figure 11 is produced by adding 8 vertices at the extremal points of the visibility kernel.

It is easy to prove that the Schönhardt polyhedron is a star-shaped polyhedron. It can in fact be shown that each intersection of the lateral boundary with a 1-parameter family of planes, parallel to the top and bottom faces, is a

star-shaped polygon [13]. In particular, the generic intersection polygon has six edges and degenerates linearly into the two extreme triangles, whereas at the middle of the family all the sides have equal length.

The visibility kernel is obtained as the common intersection of the internal half-spaces bounded by the affine hulls of the 6 lateral faces of the solid. The polytopal complex resulting from the construction shown in Figure 11 contains 1 hexahedron, 6 square pyramids, 9+2=11 tetrahedra, for a total of 18 3-cells produced by adding 8 internal vertices to the initial 6 vertices of the Schönhardt polyhedron.

Unfortunately, the 6 pyramids have one non-trihedral vertex, so that a full decomposition into hexahedra is not possible, as shown by Figure 12. A good hexahedral decomposition can be conversely obtained starting for a polytopal decomposition with all trihedrals angles, obtained by adding 9 more vertices—one for each tetrahedron connecting a pair of adjacent star branches.

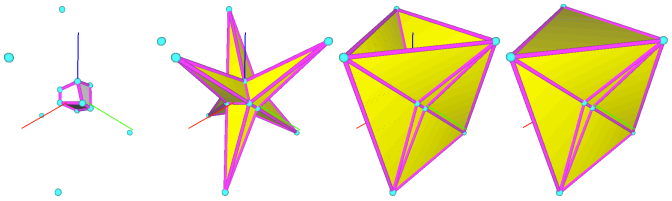

Fig. 11. The construction of the Schönhardt polyhedron step-by-step: (a) visibility kernel of such star-shaped polyhedron; (b) adding a pyramid per kernel's face; (c) adding a tetrahedron per kernel's edge (but the ones on the lateral surface); (d) adding a top and a bottom tetrahedron.

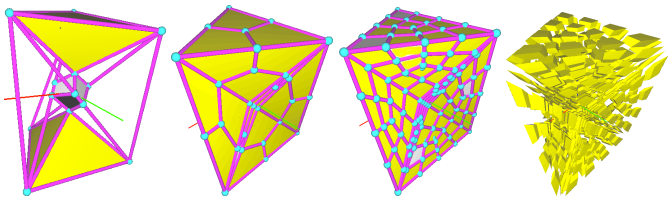

Fig. 12. Schönhardt polytopal complex transformed into a hexahedral complex: (a) the logic of the initial polytopal decomposition) (b) after one decomposition step; (c) after two decomposition steps; (d) exploded view of the 3-skeleton of the complex.

7. Conclusion

In this paper we have introduced a novel multidimensional algorithm for hierarchical splitting of polytopal complexes into hyper-cuboidal complexes, a dimension-independent generalization of hexahedral meshes and meshes of quads. The given approach is robust and fast. Furthermore, it is not difficult to set-up in a distributed computational environment, where it can be executed as a streaming of k-cells, for k growing from zero to the dimension d of the solid cells of the mesh. Further improvements are possible, making even faster the reconstruction of the topology of the $(k-1)$-facets of k-cells, that are currently generated in a purely combinatorial manner.

Acknowledgement

The algorithms and the cell complexes discussed in this paper have been implemented in Python using the pyplasm library, a recent port of the geometric language Plasm [9,10], that includes the C++ geometry kernel XGE and its fast implementation of Hasse graphs. Pyplasm provides efficient support for dimension-independent geometric programming, including Boolean ops, Cartesian products, Minkowsky sums, Schlegel diagrams, and much more. Ports to other modern languages (Cython, Javascript, Clojure, Erlang) and to mobile devices are on the way.

References

[1] Altshiller-Court, N. The trihedral angle. In *Modern Pure Solid Geometry*. Chelsea, New York, NY, USA, 1979, ch. 2, pp. 27–41.

[2] Bajaj, C., Schaefer, S., Warren, J., and Xu, G. A subdivision scheme for hexahedral meshes. *The Visual Computer 18*, 5–6 (2002), 451–460.

[3] Bajaj, C. L., Warren, J., and Xu, G. A smooth subdivision scheme for hexahedral meshes. Tech. rep., Austin, TX, USA, 2001.

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

[5] Dong, S., Bremer, P.-T., Garland, M., Pascucci, V., and Hart, J. C. Spectral surface quadrangulation. *ACM Trans. Graph. 25* (July 2006), 1057–1066.

[6] Dong, S., Bremer, P.-T., Garland, M., Pascucci, V., and Hart, J. C. Spectral surface quadrangulation. In *ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), SIGGRAPH '06, Acm, pp. 1057–1066.

[7] Edelsbrunner, H. *Geometry and Topology for Mesh Generation (Cambridge Monographs on Applied and Computational Mathematics)*. Cambridge University Press, New York, NY, USA, 2001.

[8] Elsenhans, A.-S., Kohnert, A., and Wassermann, A. Construction of codes for network coding. In *Proceedings of the 19th International Symposium on Mathematical Theory of Networks and Systems (MTNS 2010)* (Budapest, Hungary, 2010).

[9] Paoluzzi, A. *Geometric Programming for Computer Aided Design*. John Wiley & Sons, Chichester, UK, 2003.

[10] Paoluzzi, A., Pascucci, V., and Vicentino, M. Geometric programming: A programming approach to geometric design. *ACM Transactions on Graphics 14*, 3 (1995), 266–306.

[11] Pascucci, V. Slow growing subdivision (SGS) in any dimension: Towards removing the curse of dimensionality. *Computer Graphics Forum 21*, 3 (2002).

[12] Pascucci, V. Slow growing volumetric subdivision. In *ACM SIGGRAPH 2002 conference abstracts and applications* (New York, NY, USA, 2002), SIGGRAPH '02, Acm, pp. 251–251.

[13] Preparata, F. P., and Shamos, M. I. *Computational geometry: an introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

Appendix A. Algorithm's source in Python

A pseudo-code description of the HCC splitting is given in Algorithm 1. The Python implementation of the function

`hccmesh`, closely resembling the pseudocode formulation, is given by Listing 1 in Appendix A. The implementation strongly relies on the internals of the *Graph* class of the XGE (eXtreme Geometric Environment) C++ kernel of the Python package pyplasm.

```
def hccmesh(g):
    d = g.getMaxDimCells()
    g1 = Graph(d)
    for node in range(1,g.getNumNode()+1):  # create the 0-layer of HCC
        newnode = g1.addNode(0)
        g1.setVecf(newnode,CENTROID(g)(node))
    for k in range(1,d):                    # create N_k, A_{k-1} (1<=k<=d-1)
        for h in range(k,d+1):
            for root in CELLSPERLEVEL(g)(h):
                subgraphs = DOWNTRAVERSE(g,k,root)  # search g for subgraphs
                faces = [UPTRAVERSE(sg,k) for sg in subgraphs] # backtrack g1
                for face in faces:          # update N_k
                    newnode = g1.addNode(k)
                    for node in face:       # update A_{k-1}
                        g1.addArch(node,newnode)
    for root in CELLSPERLEVEL(g)(d):        # create the d-layer of HCC
        subgraphs = DOWNTRAVERSE(g,d,root)
        facets = [UPTRAVERSE(sg,d) for sg in subgraphs]
        vertices = [list(set(CAT(sg))) for sg in subgraphs]
        for facet,verts in zip(facets,vertices):
            newnode = g1.addNode(d)         # update N_d
            for node in facet:              # update A_{d-1}
                g1.addArch(node,newnode)
            for vert in verts:              # update A_d = A_{-1}
                g1.addArch(newnode,vert)
    return g1                              # return the output HCC graph
```

Listing 1: The HCC algorithm coded in Python

The library functions invoked by `hccmesh` have the semantics discussed in the following:

`Graph` constructor of the objects of the `Graph` class, data structure used for representing the Hasse diagrams of cell complexes.

`Graph.getMaxDimCells` method of the `Graph` class, used to extract the maximal dimension of cells in the represented complex.

`Graph.getNumNode` method of the `Graph` class, used to get the current number of nodes in the Hasse diagram.

`Graph.addNode` method of the `Graph` class, used to add a new node (cell) of given dimension to the Hasse diagram of the complex.

`Graph.addArch` method of the `Graph` class, used to add a new arc (direct containment relation) to the Hasse diagram of the complex.

`Graph.setVecf` method of the `Graph` class, used to store a either a position vector or a hyperplane covector to a node of the complex.

`CENTROID` pyplasm function to compute the centroid of a cell.

`CELLSPERLEVEL` pyplasm function to return the list of IDs of cells of given dimension.

`DOWNTRAVERSE` pyplasm function. Return all the subgraphs of a Hasse graph that are also sublattices of given depth.

`UPTRAVERSE` pyplasm function. Return the set of nodes (cells) that are sons of the root of a sublattice starting from its nodes of level zero.

`CAT` pyplasm function. Concatenates a list of lists.