Polycube Simplification for Coarse Layouts of Surfaces and Volumes

Gianmarco Cherchi
Università degli Studi di Cagliari, Italy

Marco Livesu
CNR IMATI, Genoa, Italy

Riccardo Scateni
Università degli Studi di Cagliari, Italy

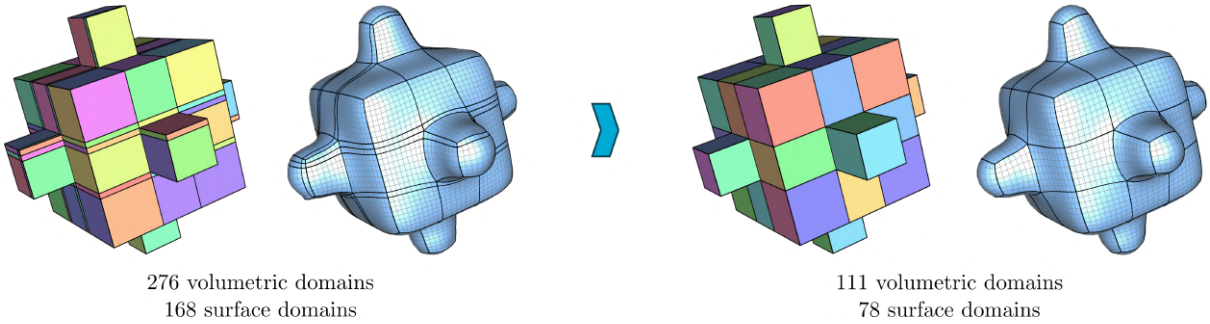


Figure 1: *State-of-the-art methods for polycube-based meshing do not consider the corner alignment problem, thus producing both surface and volumetric meshes with a poor structure (left). We propose an additional step in the pipeline: given a polycube we optimize for the position of its corners, maximizing the singularity alignment and producing well structured meshes with remarkably less domains (right). In this example we show how, with a few tiny adjustments on the position of the polycube corners, our method has been able to reduce the number of volumetric domains from 276 to 111 and the number of surface domains from 111 to 78.*

Abstract

Representing digital objects with structured meshes that embed a coarse block decomposition is a relevant problem in applications like computer animation, physically-based simulation and Computer Aided Design (CAD). One of the key ingredients to produce coarse block structures is to achieve a good alignment between the mesh singularities (i.e., the corners of each block). In this paper we improve on the polycube-based meshing pipeline to produce both surface and volumetric coarse block-structured meshes of general shapes. To this aim we add a new step in the pipeline. Our goal is to optimize the positions of the polycube corners to produce as coarse as possible base complexes. We rely on re-mapping the positions of the corners on an integer grid and then using integer numerical programming to reach the optimal. To the best of our knowledge this is the first attempt to solve the singularity misalignment problem directly in polycube space. Previous methods for polycube generation did not specifically address this issue. Our corner optimization strategy is efficient and requires a negligible extra running time for the meshing pipeline. In the paper we show that our optimized polycubes produce coarser block structured surface and volumetric meshes if compared with previous approaches. They also induce higher quality hexahedral meshes and are better suited for spline fitting because they reduce the number of splines necessary to cover the domain, thus improving both the efficiency and the overall level of smoothness throughout the volume.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

1. Introduction

A cuboid can be trivially turned into a structured mesh by gridding it. This simple statement has triggered the development of techniques that aim to transform a shape into a conformal collection of connected cuboids, also known as *polycubes* [THCM04].

The complex problem of generating a structured mesh out of a general shape becomes straighforward with a polycube at hand. The common pipeline starts with a morph that transforms the input shape into a polycube, generating a volumetric map; the cuboids of the polycube are then subdivided so as to achieve the desired scale and, as a last step, the inverse map is used to transfer the

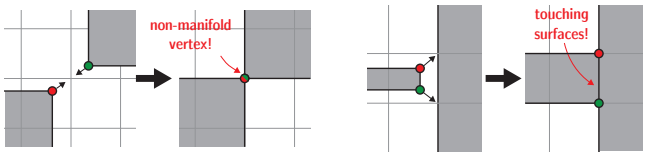


Figure 2: *Trivially snapping polycube corners to integer locations can generate a number of topological inconsistencies. Left: two corners (red and green) map to the same integer location, generating a non-manifold vertex (red/green). Right: two vertices snap to the closest iso-line, generating an overlap with another portion of the polycube. In our algorithm we use explicit constraints to avoid such cases.*

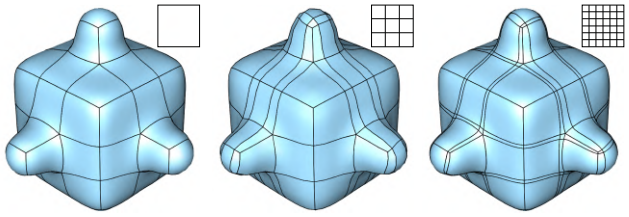


Figure 3: *State-of-the-art methods may generate structurally different meshes dependent from the density of the lattice used to sample the polycube. Our method consistently produces meshes with equivalent structure, regardless the density of the sampling.*

connectivity of the resulting mesh from the polycube to the input shape [GSZ11, LVS*13].

The shape of the polycube defines the structure of the resulting mesh, in the sense that polycube corners are singularities in the mesh and chains of edges connecting pairs of corners induce a coarse block-decomposition of the domain, also known as *base-complex* among practitioners [GDC15]. In a sense, the base-complex of a mesh $\mathcal{M}$ can be thought of as the coarsest mesh $\mathcal{M}'$ having exactly the same structure of $\mathcal{M}$. In the polycube case, the base-complex is the coarsest mesh that can be generated from a given polycube. Notice that the number of elements in the base-complex does not depend on the number of polycube corners. Depending on how well corners align (or misalign) to each other different base-complexes can be produced (Figure 1).

Having a good alignment between the singular elements of a mesh is a key ingredient in a number of applications. The singularity misalignment problem has been subject of extensive research in recent years both for surfaces and volumes [AFTR15, GDC15, VS15, BLP*13, MPKZ10]. In hexahedral meshing overly dense base-complexes tend to contain badly shaped cuboids that, when subdivided for the generation of the final hex-mesh, produce poor quality meshes with tiny chances of further optimization. Coarse base-complexes with well aligned singularities contain better shaped cuboids, therefore tend to produce higher quality hexahedral meshes [GDC15]. Furthermore, coarse base-complexes enable lower resolution meshing, with consequent benefits for applications both in terms of memory requirements and performance speedup. In higher order-meshing [LZLW15, WZLH13, LLWQ13, WHL*08] a spline basis is fit into each cuboid of the base-complex, with the resulting representation being $\mathcal{C}^2$ continuous within each cuboid and only $\mathcal{C}^0$ at the boundaries between adjacent cuboids. Coarse base-complexes minimize the extent of the $\mathcal{C}^0$ region, thus providing a higher smoothness throughout the whole domain, enabling both more accurate and more efficient simulations for applications like Isogeometric Analysis (IGA) [HCB05].

Despite the importance that singularity alignment covers for the aforementioned applications, previous methods for polycube computation do not consider this aspect, thus generating sub-optimal base-complexes with far too many cuboids.

Given a polycube map, previous methods generate the connectivity of the desired structured mesh by gridding the polycube with an integer lattice. To keep the map bijective, the corners of the polycube must be at integer locations. To ensure this property a naive snapping is usually performed, rounding each corner to its closest integer location [GSZ11]. Prior to gridding, the polycube is scaled by a factor s in order to control the mesh resolution.

Firstly, we observe that the naive snapping currently used in state-of-the-art approaches can introduce topological inconsistencies in the polycube structure. There are many pathological configurations possible. For example two disjoint corners c and c' may round to the same integer location (Figure 2, left); or a corner c may be projected on a polycube facet f if the distance $d(c, f) \leq 0.5$ (Figure 2, right). Secondly, we notice that the scaling factor s not only controls the mesh resolution but also has a non-intuitive, hard to control, effect on the mesh structure. In fact, small scaling factors will produce a better singularity alignment because polycube corners will be more likely to round to the same integer iso-lines. Conversely, big scaling factors will produce worse singularity alignments (and worse coarse layouts) because corners will be more likely to round to different integer iso-lines. In other words, the very same polycube may produce structurally different meshes depending on the sampling density (Figure 3).

In this paper we introduce, to date, the first method to optimize the structure of a polycube base-complex. Our method works on the topology of the polycube and, by carefully snapping corners to integer locations, minimizes the overall number of cuboids. We observe that the complexity of the problem we tackle is exponential in the number of corners. To render this problem tractable we therefore propose a greedy simplification strategy. We demonstrate our results on several polycubes produced with the most recent algorithms available in literature. The base-complexes produced by our method can be used both for spline fitting and hexahedral meshing. We also show that the boundary of our base-complexes can be used to generate well structured quad meshes that embed a coarse quad-layout [ULP*15].

Our **main contribution** to the meshing pipeline is an effective and robust numerical method to optimize the position of the polycube corners in the integer lattice. Specifically:

- we optimize the corner position in order to provide the best singularity alignment possible. For a given polycube, we consistently produce meshes with equivalent structure, regardless the scaling factor applied to the integer lattice to control the resolution. Our

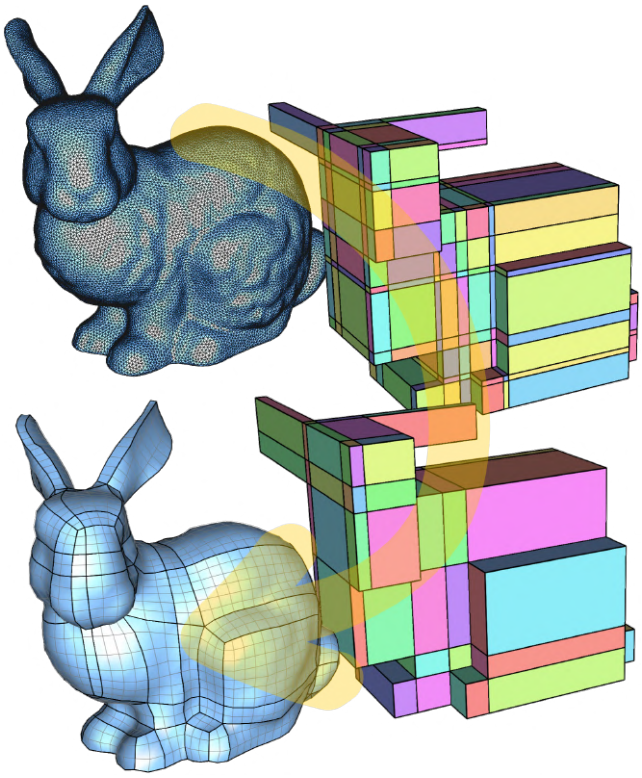


Figure 4: *A brief recap of the various stages of the remeshing pipeline: we start from a triangle or tetrahedral mesh (top left), we generate the polycube layout (top right), we simplify the base complex structure with our novel method (bottom right), and, finally, we generate a quad or an hexahedral mesh (bottom left).*

polycubes are on average 46% coarser than previous polycube based methods; they produce higher quality hexahedral meshes and are better suited for spline fitting.

- we propose a set of constraints to consistently generate structurally sound polycubes, avoiding the generation of collapsed edges, non-manifold corners and self-intersections. Our constraints are defined on the polycube corners and are easy to plug into existing solvers as they consist in linear equations and inequalities.

2. Related Work

Our work is related to research in several areas reviewed below.

Polycube maps. Bijections between a general shape and orthogonal polyhedra (or polycubes) were introduced in computer graphics as a mean to generate seamless texture maps [THCM04] and have received growing attention from the scientific community ever since, especially for volumetric applications like solid modeling [WHL*08] and hexahedral remeshing [LVS*13, HJS*14, YZWL14, GSZ11]. To asses the quality of a polycube the most important factors are the distortion induced by the map, and the number of corners, which will be singularities in the resulting splines/hehahedral meshes. The first attempts to automatically generate polycube maps were not sufficiently robust to process complex shapes and tended to produce either overly coarse [LJFW08] or overly complex [HWFQ09, GSZ11] polycubes, with the former suffering from high distortion and the latter producing unnecessary corners. This technology is now rather mature; the most recent algorithms are able to process complex shapes and consistently provide polycube maps with both low distortion and low corners count [LVS*13, HJS*14]. None of these methods takes into consideration how well corners align to each other, which is one of the key ingredients to derive a coarse block-decomposition of a shape. As a result, they are not suitable for the generation of coarse layouts. Our method tackles this very specific problem and is capable of working on top of any given polycube map, whether it is generated with one of the methods above or manually crafted, as in [WHL*08, THCM04].

Block-structured meshing. The generation of meshes that admit a coarse block decomposition is a problem with relevant applications in animation, FEM analysis and Computer Aided Design (CAD). For the surface case a block-structured mesh often comes in the form of a quadrilateral mesh obtained by gluing side to side a set of quadrilateral patches (or blocks) in a conforming way [BLP*13]. Each patch is a 2D array of quads. Depending on the applicative field these meshes are called *semi-regular* or *multi-block grids*, whereas the graph having as nodes the patches' corners and as arcs their edges is usually referred to as *coarse quad-layout*.

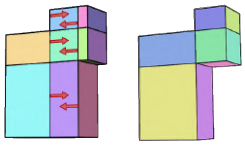
A variety of methods for the automatic computation of coarse block-structured meshes have been proposed in literature [BLK11, TPP*11]. However, most of these methods are either too demanding from a computational point of view [BCE*13, CBK12] or focus on a specific class of shapes and do not scale well on general models [ULP*15]. Our method builds upon a given polycube map and is therefore general enough to be applied to any class of shapes. Another body of research deals with the generation of user interfaces for the manual generation of quad-meshes and quad-layouts. These methods can produce extremely high quality layouts [MTP*15, CK14, TPSHSH13], but, in order to achieve the maximum result, they need to be controlled by an experienced user.

For the volumetric case, in [GMD*15] a sweeping method that subdivides the volume enclosed by a triangular mesh into a set of cuboids is proposed. This method can produce extremely coarse layouts, though at the expense of a major deviation from the target shape. It also requires some manual intervention to set up the harmonic field that guides the meshing process. Notice that this manual step may have a dramatic impact on the final result and needs experienced users. In [GDC15] Gao and colleagues showed that by coarsening the singularity structure of a given hexahedral mesh the per-element quality of the mesh can be improved without affecting the deviation from the target shape. This method is quite general as it works with any hexahedral mesh, regardless the meshing technology used to generate it. It is, however, very demanding from the computational point of view (it may require up to 30 minutes of computation). Our method amounts to just an additional, lightweight, step in the polycube meshing pipeline, and is capable of producing comparable results (Section 7).

3. Overview

We present here our polycube simplification strategy. This is the optimization step of the pipeline as described in Figure 4.

We observe that the blocks of a base-complex are glued face-by-face. Each block separation (i.e., a face shared between two adjacent blocks) defines a separation plane that propagates throughout the whole complex. Therefore, keeping the faces on a limited set of planes reduces the overall number of blocks.



We *align* the block faces on the smallest possible set of planes in each direction to produce a base-complex with the lowest possible number of blocks. In the example aside the initial base complex has 26 surface patches and 33 volumetric domains (included the inner padding). Simply by aligning two faces we reduce to 18 surface patches and 22 volumetric domains.

For ease of formulation and implementation we reduce the problem of aligning faces to the problem of aligning corners. Whenever the position of a corner changes, we ensure that the position of all the incident edges and faces is coherently updated so as to preserve the axis aligned structure of the polycube. We achieve this with a set of explicit constraints, detailed in Section 5.1. During the simplification we can, therefore, focus only on the relations between polycube corners.

The core of our simplification strategy is a concise iterative method: at each iteration we first identify a set of pairs of corners to be aligned (Section 4) and then solve an integer numerical program to align them in the integer lattice (Section 5). The alignment process is cumulative, meaning that at each iteration we preserve all the corner alignments produced at the previous iterations. This ensures that the number of aligned pairs grows monotonically. The algorithm converges when no further alignments are possible. Since the set of possible corner pairs is finite, convergence is guaranteed.

We motivate the use of an iterative method by observing two things: (i) a corner may want to align with many other corners, so it needs to be paired more than once; (ii) not all the alignments can be discovered *right away*. We empirically observed that iteratively aligning corners and looking for new pairs allows us to produce coarser base-complexes than trying to align all the corner pairs in a single global solve.

The few lines of pseudo-code in Algorithm 1 summarize the main steps of our method. We start with an input polycube (either computed with off-the-shelf algorithms or manually crafted). We support both surface and volumetric polycubes that may come in the form of either a triangular mesh or a tetrahedral mesh. In the first step we extract the polycube structure (i.e., the set of corners and their connectivity). We then move to the iterative alignment, which is the core of the method. Details regarding this part are given in Sections 4 and 5. At the end of the alignment, for each corner in the polycube we have new (integer) coordinates. In the final step we morph the input polycube into its new, optimized, structure. Details about this part are given in Section 6. The result is a simplified polycube embedding a coarse base-complex that can be used for surface and volumetric meshing, or for spline fitting.

Procedure Polycube Simplification

input : a polycube $\mathcal{P}$
output : a simplified polycube $\mathcal{P}'$

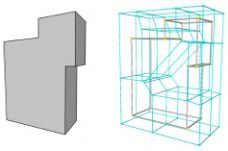
repeat
  Compute corner pairs (Section 4)
  Align corner pairs (Section 5)
until *convergence*;
Morph $\mathcal{P}$ onto the simplified polycube structure (Section 6)
return $\mathcal{P}$

Algorithm 1: Our simplification algorithm in a nutshell. We iteratively interleave corner pairing and alignment until convergence (i.e. until no further alignment can be performed). We eventually morph the input polycube in its optimized integer structure.

4. Corner pairing

We detail here how to compute $A = \{A_x, A_y, A_z\}$, the sets of corner pairs we want to align along x, y and z respectively.

To find pairs of neighbour corners we employ a heuristic based on the three-dimensional Voronoi diagram of the polycbue corners. Specifically, we consider the dual graph of the Voronoi partitioning as the adjacency graph, labeling corners belonging to adjacent cells as neighbours. In Figure 5 we show a simple 2D example of how we use the Voronoi diagram to pair non-adjacent corners.



Once we have defined the complete list of neighbour pairs we can select, among them, the candidates for alignment along each coordinate. We, thus, prune the graph of adjacencies discarding the arcs (pairs of vertices) according to the following rules:

- We remove from A the pairs which are end-points of the same edge since they are already aligned along one coordinate and it is not possible to align them along another one without changing the edge orientation or without collapsing it.
- We remove from A the pairs of vertices which are end-points of edges incident on the same vertex. In a polycube, if the corners (c, c') and the corners (c, c'') are already aligned along one coordinate, it is not possible to align the pair (c', c'') without losing the axis-align property or collapsing an edge.

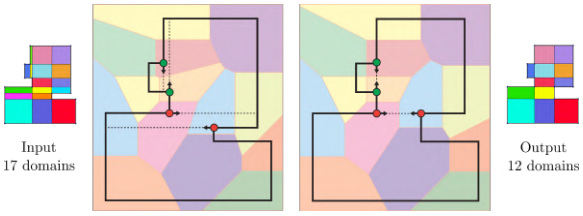


Figure 5: *A 2D example that explain how we need to use the Voronoi diagram to find the candidate pairs of corners to align; the Voronoi diagram in background is computed with the original positions of the vertices (left side); after the alignment we reduce the number of domains from 17 to 12 (right side).*

- We finally remove from A external adjacent corners since it is useless to try to align them since their alignment does not produce any reduction in the number of domain of the base complex.

After the pruning, we have in A only the candidates for alignment. To obtain the sub-sets A_x, A_y and A_z we now just need to determine, for each pair in $(c,c') \in A$,the coordinate along which the alignment is possible. If we found more than one possible alignment for a corner c along the same coordinate, we select as candidate for the alignment the corner c' closest along the considered coordinate.

5. Corner alignment

We pose the corner alignment problem as an integer optimization problem enriched with a set of linear constraints aimed to preserve both the corner alignments achieved at the previous iterations and the topological structure of the polycube.

Let A be the set of corner pairs we want to align at the current iteration, and let A_x^*,A_y^* and A_z^* the sets of corner pairs for which an alignment has already been achieved at previous steps (along the x,y and z coordinates, respectively). We formulate our optimization problem as follows

$$\begin{array}{l} \min E = E_{align}(A) + \lambda \cdot E_{shape} \\ \quad \text{s.t.} \\ c_y = c'_y \quad c_z = c'_z \qquad \forall (c,c') \in A_x^* \\ c_x = c'_x \quad c_z = c'_z \qquad \forall (c,c') \in A_y^* \\ c_x = c'_x \quad c_y = c'_y \qquad \forall (c,c') \in A_z^* \\ \text{polycube structural constraints.} \end{array} \tag{1}$$

The first term of our energy (E_{align}) aims to snap each corner pair $(c,c') \in A$ on the same iso-line of the integer lattice. We perform this operation independently on each dimension. Let us imagine to split A into three sub-sets, A_x, A_y and A_z, representing the sets of corner pairs to be aligned along the x, y and z coordinate respectively. We can express E_{align} as follows

$$E_{align}(A) = \sum_{(c,c') \in A_x} (c_x - c'_x)^2 + \sum_{(c,c') \in A_y} (c_y - c'_y)^2 + \sum_{(c,c') \in A_z} (c_z - c'_z)^2$$

In our experiments we noticed that the alignment term alone is not capable of producing extremely coarse base-complexes. We learned that the algorithm tended to align the furthest pairs $(c,c') \in A$ first, leaving all the *"easy"* alignments for the subsequent iterations. This *"complex first, easy after"* behaviour is well explained by the fact that E_{align} is quadratic, and thus aligning the furthest pairs first is the best way to rapidly minimize the energy. The problem with this behavior is that performing the most difficult alignments first may heavily change the shape of the polycube, generating deadlock configurations in which the (in the beginning) closest corner pairs are no longer possible to align because of the constraints the numerical program is subject to.

We compensate this behaviour by adding an additional regularization term to the energy: E_{shape}. This energy is a simple corner-wise attraction to the input polycube, that is

$$E_{shape} = \sum_c \|c - \tilde{c}\|^2$$

with c being the current corner position and $\tilde{c}$ the original corner position. The regularization term is particularly useful in the first iterations because prevents dramatic changes in the polycube shape, thus favoring the alignment between the closest corner pairs first. It is however limiting towards the end of the optimization, when these easy alignments are no longer available, and to align the furthest corner pairs would be necessary. In Equation 1 we therefore multiply E_{shape} by a scaling factor λ. We experimentally found that starting with $\lambda = 1$ at the first iteration and halving it after each iteration provides the wanted behavior; all the results produced in this paper have been produced using this scaling strategy. Notice that different weighting schemes may accommodate better results for certain shapes. In Section 7 we discuss how to use λ to control the trade-off between polycube simplification and mapping distortion.

5.1. Structural constraints

We impose a series of constraints to preserve the axis aligned structure of the input polycube, also avoiding edge collapses and self-intersections. This reduces to a set of linear constraints, as detailed below.

Collinearity of the end-points. We constraint polycube edges in order to keep them axis-aligned. We have seen before that in the E_{align} portion of the energy function we try to align corner pairs along one coordinate. We want to avoid that this attempt will move the edge off the integer lattice. Let $e(c,c')$ be a polycube edge connecting the corners c and c', and let us suppose that e is aligned with the x axis. In order to keep its original orientation when we solve for the c and c' coordinates we impose the following two linear constraints

$$\begin{cases} c_y = c'_y \\ c_z = c'_z \end{cases} \tag{2}$$

These linear constraints prevent e to move off the x axis. In a similar fashion edges aligned with the y and z axis can be forced to maintain their original alignment.

Minimum length of edges. In our formulation the smallest edge length is fixed to 1 to avoid edge collapses. We ensure this by combining (2) with one more linear constraint per edge $e(c,c')$

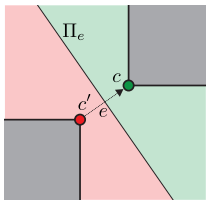
$$(c - c') \cdot u \geq 1 \tag{3}$$

Here $u = \frac{c-c'}{\|c-c'\|}$ is a pre-computed unit length vector aligned with e. We remind the reader that the polycube edges cannot change orientation during the optimization, therefore we can compute the vectors u once using the original corner coordinates and use them throughout the whole iterative simplification. Given the axis aligned structure of a polycube u can be either $(\pm 1,0,0)$ or $(0,\pm 1,0)$ or $(0,0,\pm 1)$. Furthermore, notice that this constraint not only prevents the edge to be shorter than 1, but also preserves its original orientation, avoiding an edge flip.

Corners collapse. All the constraints described before are quite natural to impose. One condition more subtle to check is the avoidance of the collapse of corners that are not end-points of the same

edge. In particular we want to avoid that corners pairs $(c,c') \in A$ reach the alignment by occupying the same position in the integer lattice. In this case we do not have a simple criterion like the edge collapse to avoid the move. Recall that, in such a case, the resultant polycube would be not manifold and would lose its original topology (Figure 2, left).

To tackle this problem we add to our model a particular constraint that creates a separation plane between the two corners that are attracting each other. Let $e(c,c')$ be an invisible edge connecting the non adjacent corners c and c'. We define Π_e as the plane passing through the middle point of e and having $\frac{c-c'}{\|c-c'\|}$ as normal orientation. This defines a partition of the space, with c belonging to the positive half-space and c' belonging to the negative half-space of Π_e. The planes Π_e are computed at each iteration according to the current coordinates of the polycube. When we solve for the new polycube coordinates we then add two constraints to keep c and c' in the half-space they belong to, specifically:



$$\begin{cases} \Pi_e(c) > 0 \\ \Pi_e(c') < 0 \end{cases} \tag{4}$$

We do this for each corner pair $(c,c') \in A$.

Dummy vertices and edges. The last condition to control is even more subtle than the previous. We want to avoid that: (i) parts of the polycube which are not adjacent compenetrate; (ii) the outside boundary and any of the holes of a non-simple face touch, thus changing the face topology. To this extent we use *dummy vertices* and *dummy edges*. A dummy vertex is defined as the intersection between the supporting line of a polycube edge and the polycube faces closest to it on each side (if any). A dummy edge, on the other hand, connects an end-point of a polycube edge to its corresponding dummy vertex. We compute all the dummy vertices and edges in our polycube and treat them as if they were real polycube edges, imposing that their length must be equal or bigger than 1 (as in 3). With this simple new set of constraints we avoid collapses between vertices and edge, vertices and faces, edges and edges or inner and outer boundaries of the same face. For consistency, we also impose special constraints to ensure that dummy vertices will stay within the edge or face they belong to. In Figure 6 we emphasize the importance of our consistency constraints, showing an example of polycube optimization with and without dummy edges. As can be noticed, dummy edges ensure topological consistency, without penalizing the quality of the alignment.

Dummy edges computation. For the computation of dummy edges we work on each dimension separately. We explain our procedure for the x axis; everything applies also to the y and z axis. Let $f_x^0 \leq f_x^1 \leq ... \leq f_x^n$ be the list of polycube facets having the x axis as normal orientation, ordered according to their x coordinate. Let $e(c,c')$ be a polycube edge aligned with the x axis, such that $c_x < c'_x$. We first extend e from the c side, finding the first non-empty intersection with the closest facet f_x having x coordinate lower than c_x.

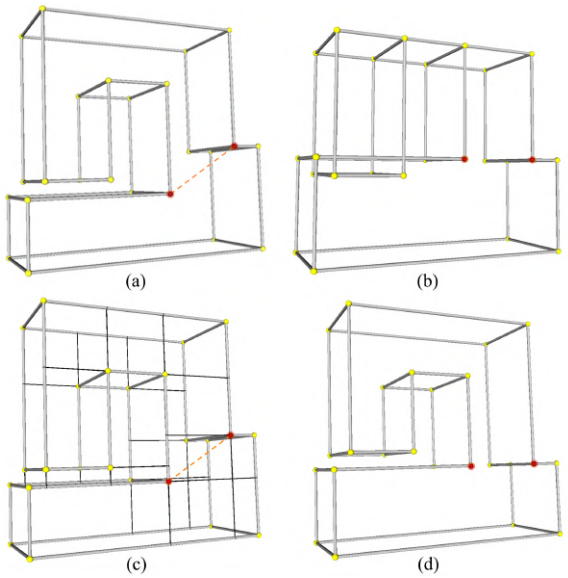


Figure 6: *A simple example of a pair of corners that we want to align (a). In (b) we can see the resultant alignment without using dummy constraints. In (c) the black edges represent the dummy edges and in (d) the final polycube after the optimization with the dummy constraints.*

If such facet exists, we consider the intersection point as a dummy vertex and we add a dummy edge connecting such point with c. We, than, repeat it for c'. We do this for any edge in the polycube, using similar lists for the y and z axis as well. Notice that given the coarseness of polcyubes this procedure is very fast (i.e., a fraction of a second).

6. Finalization

At the end of the corner optimization we have a new polycube structure, optimized in the sense that it has the least number of blocks. This polycube may be dramatically different from the input one because the alignment process produces a lot of compression and stretching of the block volumes, with bad consequences for the distortion of the associated polycube map. In order to preserve the quality of the original polycube map as much as possible, we solve the problem formulated in Equation 1 once more, without the E_{align} part of the energy and with $\lambda = 1$. In this final optimization we constrain *all* the corner pairs alignments found at the previous stage of the algorithm. This generates a polycube structure that is as close as possible to the input polycube but at the same time has integer coordinates and optimal structure. The resulting polycube is so similar to the input one that the distortion induced by our optimization in the polycube map is often negligible (Figure 7, right).

We then finalize our output by fitting the input polycube (which is either a triangle or a tetrahedral mesh) into this structure. Let $\mathcal{P}$ be the input polycube: to fit the polycube in the optimized structure we solve a simple laplacian problem $\nabla\mathcal{P} = 0$. We constrain, for each vertex $p \in \mathcal{P}$: (i) all its three coordinates if it is a polycube corner;

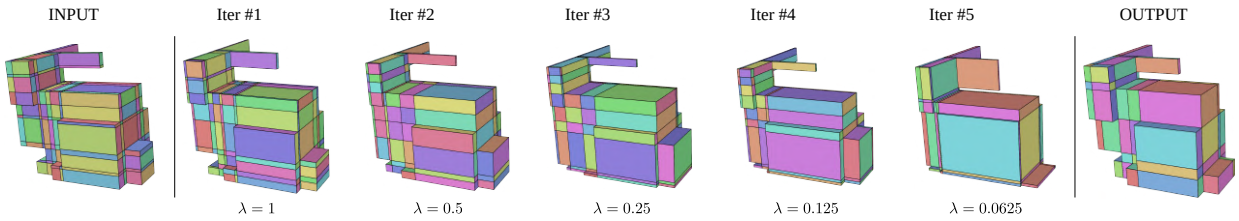


Figure 7: *From left to right: the input polycube, the five iterations necessary to complete the simplification process, and the output polycube. As can be noticed the iterative corner alignment dramatically changes the polycube appearance, introducing a lot of compression and stretching of the domains. In the last step of our algorithm we solve a constrained problem to restore the original aspect of the polycube while preserving its optimal structure. The result (rightmost bunny) is so similar to the input that the distortion induced by our simplification is often negligible. Should this not be the case, the user can control the simplification process by setting a lower bound for* λ*, making the algorithm quit before its natural convergence and thus producing a less simplified polycube that better preserves the quality of the input polycube map.*

(ii) two, if p lies on a polycube edge; (iii) one if p is onto a polycube facet; (iv) no coordinates at all if p is inside the polycube. The last condition applies only if $\mathcal{P}$ is a tetrahedral mesh. For surface meshes we implement the ∇ operator using the cotangent weights whereas for volumetric meshes we use the 3D mean value coordinates introduced by Floater and colleagues in [FKR05]. The result of this process is a polycube with the same connectivity of the input but optimized structure. This polycube may contain flipped or inverted elements and also has overlaps at concave features. Depending on the applications, an optimization strategy may be used to improve the mesh quality (e.g., [AL13]).

7. Results

We implemented our algorithm on a MacBook Pro equipped with an Intel Core i5 2,6 GHz processor and 8 GB of RAM. We used Voro++ [Ryc09] for the computation of the Voronoi partitioning and Gurobi [Gur] as numerical solver. For the generation of the meshes we implemented the standard polycube-based meshing pipeline as described in [GSZ11]. We then relied on the Edge-Cone Rectification method [LSVT15] to optimize the resulting hexahedral meshes and remove all the inverted elements possibly present. A gallery of results achieved with our method is depicted in Figures 8 (only base-complexes) and 9 (base-complexes and meshes).

We compare the standard polycube meshing pipeline with a modified pipeline in which we added our simplification system prior to the hexahedral mesh generation step. We relied on Polycut [LVS*13] to generate all the polycubes we used in our tests. In Table 1 we compare our numerical results with the standard meshing pipeline. For each model we show: elements count, per-element quality (minimum and average Scaled Jacobian), and number of domains in the base-complex. For the sake of a fair comparison we always tried to generate hexahedral meshes with similar elements count and we always run the hexmesh optimizer with standard parameters. In *all* cases our optimized polycubes produced higher quality hexahedral meshes compared to the meshes produced from the non-optimized counterparts. This confirms what Gao and collegaues had already shown in their recent work [GDC15]. Overall, we have been able to reduce the complexity of the initial polycube with factors ranging from 25% to 70%. In the last column of the table we report the time necessary to perform our polycube simplification. Even in the most complex cases a few seconds are enough for our algorithm to converge.

Unfortunately, we could not perform precise and extensive com-

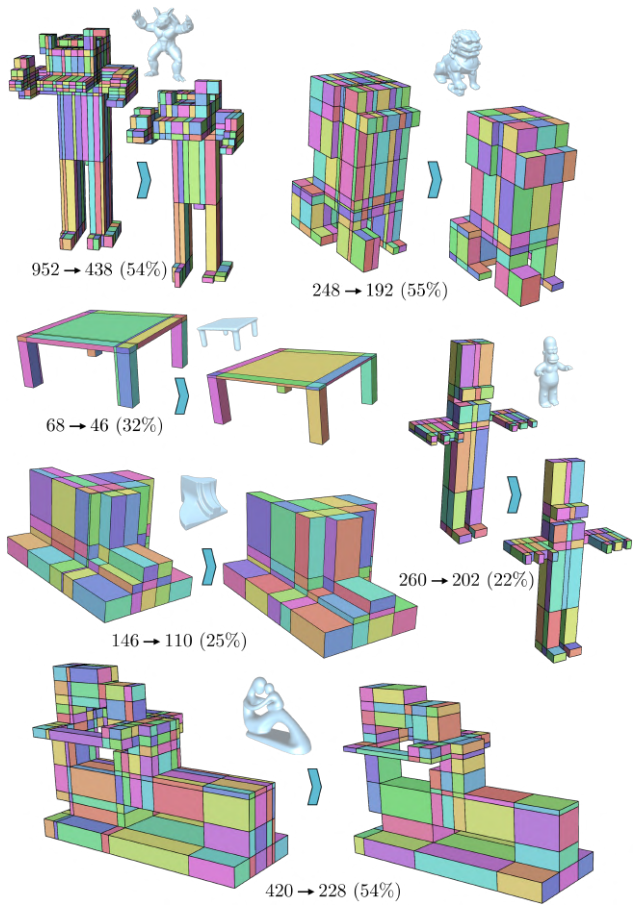
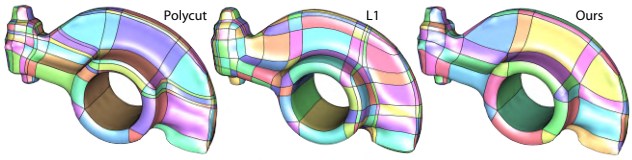


Figure 8: *A gallery of optimized polycubes, shown with the color coded quad layout. For every polycube we report the number of surface blocks in the input, in the output and the gain ratio.*

Model	Without simplification			With simplification			Gain	Time
	# Hexa	min / avg SJ	Domains	# Hexa	min / avg SJ	Domains		
RockerArm	22K	.224 / .943	648	22K	.266 / .944	332	49%	0.82s
Bunny	48K	.180 / .966	636	8K	.205 / .935	197	70%	2.43s
ASM	30K	.224 / .970	184	30K	.286 / .971	114	38%	1.33s
CubeSpikes	32K	.658 / .974	276	23K	.690 / .978	111	60%	1.36s
Block	18K	.178 / .954	158	18K	.179 / .955	100	37%	1.96s
Femur	15K	.503 / .957	145	15K	.544 / .959	110	24%	0.43s
Hand	32K	.462 / .967	172	5K	.486 / .935	107	38%	1.33s
Table	8K	.304 / .938	195	10K	.592 / .940	149	24%	1.20s
Teapot	35K	.450 / .978	323	34K	.570 / .979	193	40%	4.28s

Table 1: *We show here the improvements in terms of number of domains obtained with our simplification. The decreased number of domains has a positive impact on hexahedral remeshing. In all our experiments the hexahedral meshes computed on top of optimized polycubes had both higher minimum and average Scaled Jacobian.*

parisons against our closest competitor [GDC15] since we did not have the polycubes they used to generate the hexahedral meshes shown in their paper. However, to give the reader an idea of the performances of the two algorithms we report here two comparisons. For the RockerArm model they started with a polycube-generated hexahedral mesh having 664 domains, and they simplified its structure reducing it to 335 domains (50% gain). We passed from 648 to 335 domains (49% gain). For the Bunny model they passed from 580 to 194 domains (67% gain). We passed from 636 to 197 domains (gain 67%). These numbers suggest that our algorithm produces comparable results both in terms of reduction factor and minimum number of domains in the complex. Our algorithm converges in a few seconds while the method presented in [GDC15] requires a much heavier computational effort, up to 30 minutes.



Model	[LVS*13]		[HJS*14]		Ours	
	#sv	#dom	#sv	#dom	#sv	#dom
Cube Spikes	56	168	–	–	56	78
Bunny	64	352	76	176	64	136
Block	48	112	–	–	48	76
Rocker Arm	62	352	64	426	62	208

Table 2: *We report here both visual and numerical results for the generation of coarse quad-layouts. We compare with the two most recent polycube-based methods. For each algorithm we report both the number of singular vertices (#sv) in the layout and the number of surface domains (#dom). As can be noticed our smart simplification strategy allows us to consistently outperform the quad layouts produced from other polycubes.*

Coarse Quad-Layouts. Our method can also be used to generate quadrilateral meshes embedding a coarse quad-layout. To do so, at the meshing stage we sample only the surface of the base-complex, ignoring the vertices of the integer lattice located in the interior of the polycube. In Table 2 we compare our quad layouts with the ones produced by [HJS*14, LVS*13]. We consistently outperform previous methods generating coarser layouts. For some of the shapes we tested, we have been able to match the performances of some of the best algorithms specifically designed to generate coarse quad-layouts. In particular, our layout for the block model has 48 singular vertices and 76 blocks and is equivalent to the ones generated from [BCE*13, CBK12]; our layout for the Cube Spikes model has 56 singularities and 78 domains and is equivalent to the one generated from [TPP*11]. However, since our optimization works in the polycube space, it may not be flexible enough to match the performances of these algorithms for shapes whose features fail to align to the XYZ frame. For example, the method proposed in [ULP*15] is capable of producing a layout with 24 singularities and 28 domains for the RockerArm, whereas our best result on that model contains 62 singularities and 208 domains.

Simplicity vs distortion. Optimizing the structure of the a polycube base-complex is always a matter of finding the right balance between simplification and mapping distortion. Too aggressive simplifications may result in distorted polycubes that degrade the quality of the associated map. In Figure 7 we show all the iterations (and associated λ values) for the simplification of the Bunny's polycube. We observe that by properly setting a lower bound for the λ parameter the user can control the simplification process, making the iterative simplification quit before the natural convergence, thus obtaining a partially simplified polycube with lower distortion map. We believe this is an easy and intuitive way, for the user, to control the tradeoff between simplicity and distortion.

8. Limitations

The method proposed in this paper is a heuristic and, as such, is subject to some limitations. Here we recap the major shortcomings of our approach.

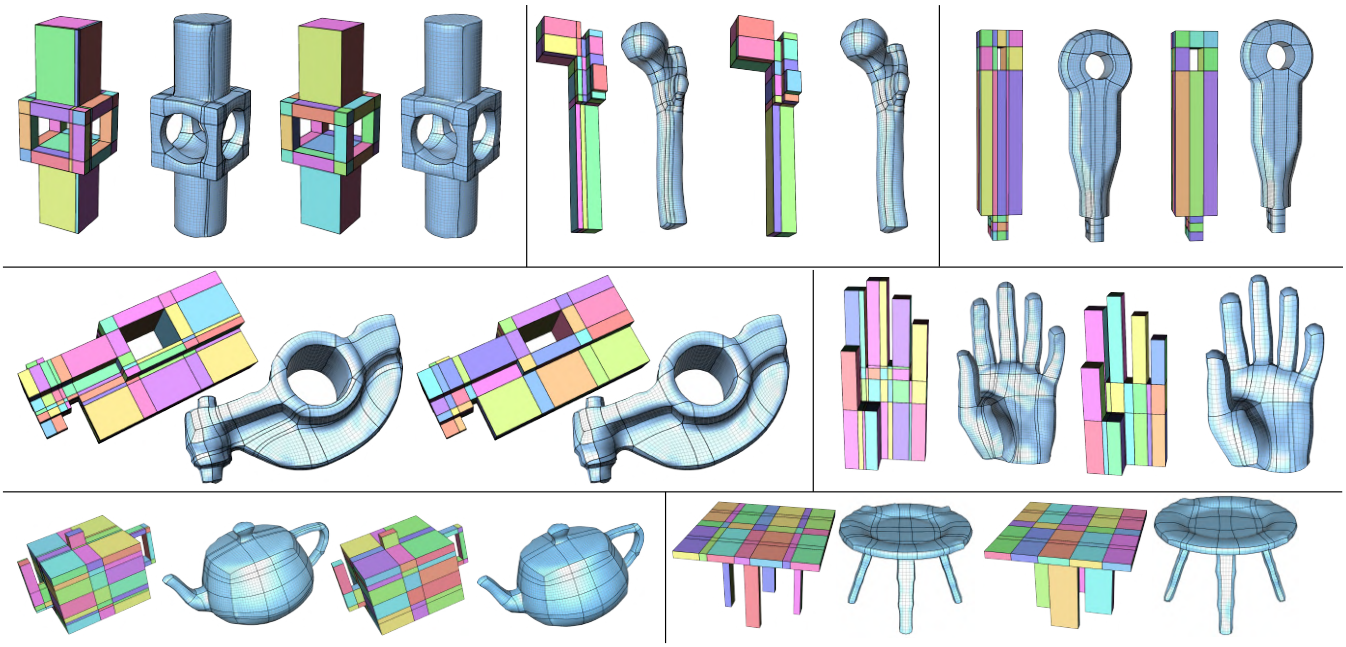
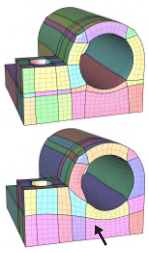


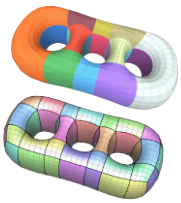
Figure 9: *A gallery of results obtained with our method. For every model: the original polycube, computed with [LVS*13] and the block layout derived from it (left); our optimized polycube and the block layout derived from it (right).*

Corner pairing. The Voronoi-based corner pairing strategy described in Section 4 is not guaranteed to generate all possible corner couples. Furthermore, when there are multiple possible pairings for a given corner, we arbitrarily select the closest one along the considered coordinate, which may not be the optimal choice in some cases, as depicted in Figure 10.

Map distortion. In the finalization step (Section 6) we maximize the similarity between the input and output polycubes, assuming that this is a good proxy to bound the distortion of the polycube map. Although this heuristic produces good results for most of the models we tested, there might be pathological cases in which this assumption is not true. An example of this is given in the inset aside, where some of the domains underwent severe stretching after simplification, thus producing a low quality coarse layout. In this case the user can trade simplicity for a lower map distortion with the mechanism described in Section 7.



Domain. We optimize the mesh structure in the polycube space, therefore we inherit the limitations of the domain we live in. In the image aside we show two different structures for the triple torus. The one above was obtained with [TPP*11] and the one below is our result. The curved domains at the extremities of the shape cannot be represented in the polycube space, therefore we split them into three sub-domains each, thus generating a higher number of cuboids.



9. Concluding remarks

We have presented here a novel method for polycube simplification. Our method is very fast, simple to be inserted in a re-meshing pipeline and needs a single parameter to be set. It is thought to be plugged into the overall pipeline able to re-mesh shapes described by triangle meshes into quad meshes (in case of surfaces), or shapes described by tetrahedral meshes into hexahedral meshes (in case of

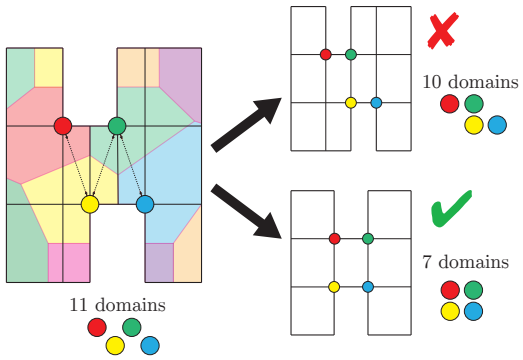


Figure 10: *When a polycube corner can align to more, nearly equidistant corners, our alignment scheme may take the wrong decision, generating sub-optimal results. Here the yellow corner can align with both the green and red corners (left). Chosing the yellow/green pair we produce a layout with 10 domains (top right); chosing the yellow/red pair we go down to 7 domains (bottom right), thus achieving the optimum.*

volumes). The simplification is performed in the polycube space, optimally aligning vertices in the integer lattice. Our approach overcomes previous limitations in polycube-based meshing, making the process independent from the sampling resolution and generating structured meshes with lower number of domains.

Acknowledgements

The authors wish to thank the anonymous reviewers for the insightful comments that helped in improving the quality of the paper.

References

[AFTR15] ARMSTRONG C. G., FOGG H. J., TIERNEY C. M., ROBINSON T. T.: Common themes in multi-block structured quad/hex mesh generation. *Procedia Engineering 124* (2015), 70–82. 2

[AL13] AIGERMAN N., LIPMAN Y.: Injective and Bounded Distortion Mappings in 3D. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 106:1–106:14. 7

[BCE*13] BOMMES D., CAMPEN M., EBKE H.-C., ALLIEZ P., KOBBELT L.: Integer-grid maps for reliable quad meshing. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 98. 3, 8

[BLK11] BOMMES D., LEMPFER T., KOBBELT L.: Global structure optimization of quadrilateral meshes. *Computer Graphics Forum 30*, 2 (2011), 375–384. 3

[BLP*13] BOMMES D., LÉVY B., PIETRONI N., PUPPO E., SILVA C., TARINI M., ZORIN D.: Quad-Mesh Generation and Processing: A Survey. *Computer Graphics Forum 32*, 6 (2013), 51–76. 2, 3

[CBK12] CAMPEN M., BOMMES D., KOBBELT L.: Dual loops meshing: quality quad layouts on manifolds. *ACM Transactions on Graphics (TOG) 31*, 4 (2012), 110. 3, 8

[CK14] CAMPEN M., KOBBELT L.: Dual strip weaving: Interactive design of quad layouts using elastica strips. *ACM Transactions on Graphics (TOG) 33*, 6 (2014), 183. 3

[FKR05] FLOATER M. S., KÓS G., REIMERS M.: Mean value coordinates in 3D. *Computer Aided Geometric Design 22*, 7 (2005), 623–631. 7

[GDC15] GAO X., DENG Z., CHEN G.: Hexahedral Mesh Re-parameterization from Aligned Base-complex. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 142:1–142:10. 2, 3, 7, 8

[GMD*15] GAO X., MARTIN T., DENG S., COHEN E., DENG Z., CHEN G.: Structured Volume Decomposition via Generalized Sweeping. *Visualization and Computer Graphics, IEEE Transactions on PP*, 99 (2015), 1–1. 3

[GSZ11] GREGSON J., SHEFFER A., ZHANG E.: All-Hex Mesh Generation via Volumetric PolyCube Deformation. *Computer Graphics Forum* (2011). 2, 3, 7

[Gur] GUROBI: Optimizer 6.5. http://www.gurobi.com/. 7

[HCB05] HUGHES T. J., COTTRELL J. A., BAZILEVS Y.: Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering 194*, 39 (2005), 4135–4195. 2

[HJS*14] HUANG J., JIANG T., SHI Z., TONG Y., BAO H., DESBRUN M.: ℓ1-based Construction of Polycube Maps from Complex Shapes. *ACM Transactions on Graphics (TOG) 33*, 3 (2014), 25:1–25:11. 3, 8

[HWFQ09] HE Y., WANG H., FU C.-W., QIN H.: A divide-and-conquer approach for automatic polycube map construction. *Computers & Graphics 33*, 3 (2009), 369–380. 3

[LJFW08] LIN J., JIN X., FAN Z., WANG C. C.: Automatic polycube-maps. In *Advances in Geometric Modeling and Processing*. Springer, 2008, pp. 3–16. 3

[LLWQ13] LI B., LI X., WANG K., QIN H.: Surface mesh to volumetric spline conversion with generalized polycubes. *Visualization and Computer Graphics, IEEE Transactions on 19*, 9 (2013), 1539–1551. 2

[LSVT15] LIVESU M., SHEFFER A., VINING N., TARINI M.: Practical Hex-Mesh Optimization via Edge-Cone Rectification. *ACM Transactions on Graphics (TOG) 34*, 4 (2015). 7

[LVS*13] LIVESU M., VINING N., SHEFFER A., GREGSON J., SCATENI R.: PolyCut: Monotone Graph-Cuts for PolyCube Base-Complex Construction. *ACM Transactions on Graphics (TOG) 32*, 6 (2013). 2, 3, 7, 8, 9

[LZLW15] LIU L., ZHANG Y., LIU Y., WANG W.: Feature-preserving T-mesh construction using skeleton-based polycubes. *Computer-Aided Design 58* (2015), 162–172. 2

[MPKZ10] MYLES A., PIETRONI N., KOVACS D., ZORIN D.: Feature-aligned T-meshes. *ACM Transactions on Graphics (TOG) 29*, 4 (2010), 117. 2

[MTP*15] MARCIAS G., TAKAYAMA K., PIETRONI N., PANOZZO D., SORKINE-HORNUNG O., PUPPO E., CIGNONI P.: Data-driven interactive quadrangulation. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 65. 3

[Ryc09] RYCROFT C.: Voro++: A three-dimensional Voronoi cell library in C++. *Lawrence Berkeley National Laboratory* (2009). 7

[THCM04] TARINI M., HORMANN K., CIGNONI P., MONTANI C.: Polycube-maps. *ACM transactions on graphics (TOG) 23*, 3 (2004), 853–860. 1, 3

[TPP*11] TARINI M., PUPPO E., PANOZZO D., PIETRONI N., CIGNONI P.: Simple quad domains for field aligned mesh parametrization. *ACM Transactions on Graphics (TOG) 30*, 6 (2011), 142. 3, 8, 9

[TPSHSH13] TAKAYAMA K., PANOZZO D., SORKINE-HORNUNG A., SORKINE-HORNUNG O.: Sketch-based generation and editing of quad meshes. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 97. 3

[ULP*15] USAI F., LIVESU M., PUPPO E., TARINI M., SCATENI R.: Extraction of the Quad Layout of a Triangle Mesh Guided by Its Curve Skeleton. *ACM Transactions on Graphics (TOG) 35*, 1 (2015), 6:1–6:13. 2, 3, 8

[VS15] VERMA C. S., SURESH K.: A Robust Combinatorial Approach to Reduce Singularities in Quadrilateral Meshes. *Procedia Engineering 124* (2015), 252 – 264. 24th International Meshing Roundtable. 2

[WHL*08] WANG H., HE Y., LI X., GU X., QIN H.: Polycube Splines. *Computer-Aided Design 40* (2008), 721–733. 2, 3

[WZLH13] WANG W., ZHANG Y., LIU L., HUGHES T. J.: Trivariate solid T-spline construction from boundary triangulations with arbitrary genus topology. *Computer-Aided Design 45*, 2 (2013), 351–360. 2

[YZWL14] YU W., ZHANG K., WAN S., LI X.: Optimizing polycube domain construction for hexahedral remeshing. *Computer-Aided Design 46* (2014), 58 – 68. 3