Robust Hex-Dominant Mesh Generation using Field-Guided Polyhedral Agglomeration

XIFENG GAO, New York University
WENZEL JAKOB, École Polytechnique Fédérale de Lausanne (EPFL)
MARCO TARINI, Università degli Studi dell'Insubria and ISTI-CNR
DANIELE PANOZZO, New York University

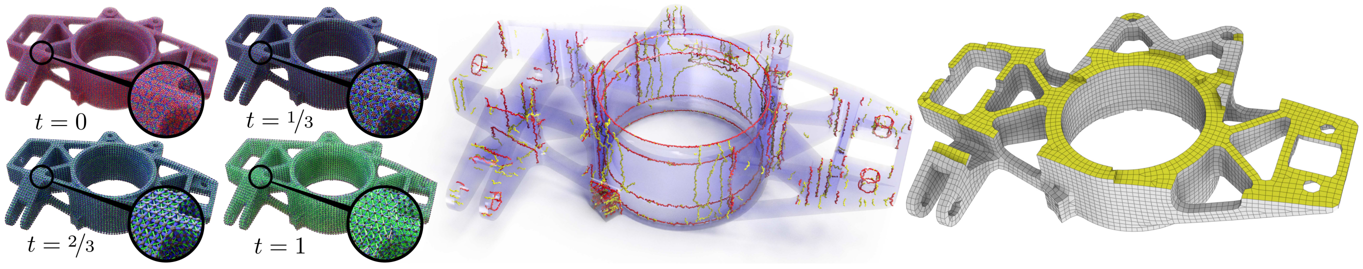


Fig. 1. The positions of the vertices of an input tetrahedral mesh (left, $t = 0$) are morphed ($t = 1$) into a hex-dominant mesh (right), guided by an orientation and a position field. The singularities of both fields are shown in the middle image, in red and yellow respectively.

We propose a robust and efficient field-aligned volumetric meshing algorithm that produces hex-dominant meshes, i.e. meshes that are predominantly composed of hexahedral elements while containing a small number of irregular polyhedra. The latter are placed according to the singularities of two optimized guiding fields, which allow our method to generate meshes with an exceptionally high amount of isotropy.

The field design phase of our method relies on a compact quaternionic representation of volumetric octa-fields and a corresponding optimization that explicitly models the discrete matchings between neighboring elements. This optimization naturally supports alignment constraints and scales to very large datasets. We also propose a novel extraction technique that uses field-guided mesh simplification to convert the optimized fields into a hex-dominant output mesh. Each simplification operation maintains topological validity as an invariant, ensuring manifold output. These steps easily generalize to other dimensions or representations, and we show how they can be an asset in existing 2D surface meshing techniques.

Our method can automatically and robustly convert any tetrahedral mesh into an isotropic hex-dominant mesh and (with minor modifications) can also convert any triangle mesh into a corresponding isotropic quad-dominant mesh, preserving its genus, number of holes, and manifoldness. We demonstrate the benefits of our algorithm on a large collection of shapes provided in the supplemental material along with all generated results.

CCS Concepts: • **Computing methodologies → Volumetric models**;

Additional Key Words and Phrases: 3D frame field, quaternionic representation, singularity graph, hexahedral dominant.

ACM Reference format:
Xifeng Gao, Wenzel Jakob, Marco Tarini, and Daniele Panozzo. 2017. Robust Hex-Dominant Mesh Generation using Field-Guided Polyhedral Agglomeration. *ACM Trans. Graph.* 36, 4, Article 114 (July 2017), 13 pages.
DOI: http://dx.doi.org/10.1145/3072959.3073676

This work was supported in part by the NSF CAREER award 1652515 and by the MIUR project DSurf.

1 INTRODUCTION

When solving non-linear partial differential equations on volumes, hexahedral meshes are generally preferred to tetrahedral ones, since they achieve the same accuracy with a drastically lower element count [Cifuentes and Kalbag 1992; Benzley et al. 1995; Tadepalli et al. 2010]. Unfortunately, the generation of high quality hexahedral meshes at coarse resolutions remains an elusive task involving a number of unsolved problems, thus automatic techniques capable of producing them robustly are still out of reach despite three decades of extensive research dedicated to this topic. Hexahedral-dominant meshes strike a good balance: they are easier to generate, since they can contain a small number of irregular elements, while offering good numerical properties [Owen and Saigal 2000; Martin et al. 2008; Reberol and Lévy 2016].

Building upon the 2D instant meshing (IM) approach [Jakob et al. 2015], we propose a novel algorithm to efficiently, robustly, and automatically create field-aligned hex-dominant meshes.

Our first contribution is a quaternionic representation for a volumetric cross-field designed to efficiently support explicit encoding of the field matching. When paired with a hierarchical accelerations structure, this representation enables us to interpolate user-defined constraints, while naturally aligning to shape features. Our algorithm is extremely robust and it converges to a smooth field even with a random initialization.

Our second contribution is a robust extraction algorithm guaranteed to extract a compatible manifold mesh from any field-aligned parameterization — it is designed to work with local parameterizations that are characteristic of the output produced by the IM technique, but it can also be applied to any global parameterization generated by other means. The algorithm uses a sequence of local topological operations to collapse and split edges, faces, and polyhedra of the input mesh, eventually converting the input tetrahedral (or triangle) mesh into a hex-dominant (or quad-dominant) output mesh. Topological invariants are checked before each operation, and only those preserving the invariants are executed, which ensures that both genus and manifoldness of the input are preserved throughout this process.

While the two contributions are independently useful in existing meshing pipelines, they have been designed together to extend the IM pipeline to the volumetric cases. Combined, they lead to a simple, robust, automatic, and scalable pipeline that can automatically remesh the benchmark proposed in [Fu et al. 2016], which is composed of 106 meshes, with no user-interaction and no parameter tweaking. We attach all results in the additional material, including a 10 second long peeling animation for each one of them.

2 RELATED WORK

We review the literature for the creation of pure hexahedral and hex-dominant meshes. Many of these algorithms are based on similar corresponding methods previously developed for quadrilateral meshing — we restrict our survey to volumetric meshing techniques and we refer an interested reader to [Bommes et al. 2013] for an overview of quad meshing techniques.

Local Connectivity Editing. Two of the earliest attempts to achieve automatic hexahedral meshing are paving (i.e. inserting regular layers of cubes aligned with a boundary quad mesh) and sweeping (i.e. extruding a partial quad mesh) [Shepherd and Johnson 2008; Owen and Saigal 2000; Yamakawa and Shimada 2003]. Their implementation is extremely challenging due to the large number of special cases that could occur, and they introduce a large number of singularities in regions where the fronts meet. The special case of tubular models has been considered in Livesu et al. [2016], where a skeleton is used to sweep a regular hex-mesh in its interior. Our method optimizes guiding fields to determine the placement of singularities throughout the interior of the mesh. It can automatically mesh complex objects without requiring user input such as skeletal or cage-based decompositions.

Spatial Partitioning. Spatial partitioning methods can be used to discretize shapes in regular collection of cubes, which coarsely approximate the input shape. This approach is very popular [Su et al. 2004; Zhang and Bajaj 2006; Zhang et al. 2007], in particular combined with octrees [Maréchal 2009; Ito et al. 2009; Zhang et al. 2013]. These methods can represent only features that are well-aligned with the grid axes and place all singularities on the shape boundary, which is unfortunate since this is often the region of highest interest. These disadvantages are, however, compensated by the high robustness of these methods, making them the de facto standard for automatic hex mesh generation. Aside from the limitation to hex-dominant output, our algorithm shares the general robustness of these methods while adding two desirable properties: automatic alignment to shape features and improved placement of singularities.

Polycube Parametrization. Polycube methods [Gregson et al. 2011; Livesu et al. 2013; Huang et al. 2014; Fang et al. 2016; Fu et al. 2016; Li et al. 2013] parameterize the interior of a closed surface mesh into a polycube. The polycube is trivially subdivided in a hexahedral mesh, that is finally warped back into the input geometry. Similarly to spatial partitioning methods, all singularities are located on the surface boundary. However, in contrast to spatial partitioning schemes, polycube methods distribute them in a superior way to account for surface features, obtaining both higher quality elements and a lower total element count. These methods are unfortunately not guaranteed to produce a valid polycube and can fail on complex inputs, limiting their practical applicability. Our algorithm robustly generates hex-dominant meshes which similarly adapt to surface features.

Field-Aligned Methods. Field-aligned methods [Nieser et al. 2011; Huang et al. 2011; Li et al. 2012; Jiang et al. 2014] compute a hex-mesh in three stages. They first estimate the gradients of a volumetric parameterization using a directional field [Vaxman et al. 2016], compute a parameterization aligned with the estimated gradients, and finally trace the cubes edges in parametric space [Lyon et al. 2016]. Computing a parameterization that induces a pure hexahedral mesh remains an unsolved problem, and currently used heuristics tend to fail on complex inputs.

Our technique is based on a similar pipeline, but avoids computing a globally consistent parameterization in favor of local, pervertex parameterizations. Instead of explicitly tracing edges in the parameterization domain, we opt for a robust simplification-based algorithm that is guaranteed to produce a manifold hex-dominant mesh as output.

To our knowledge, the method by Sokolov et al. [2016] (henceforth referred to as PGP3D) is the only existing field-aligned parameterization method that also targets hex-dominant meshes. Our algorithm shares many similarities with this method, even if the produced meshes are different at a superficial level. PGP3D can robustly process complex CAD models with alignment to surface features, creating meshes composed of hexahedra, tetrahedra, triangle-based prisms, and quad-based pyramids, while our method may create arbitrary polyhedra. The price paid by PGP3D's approach is that their meshes are not conforming (i.e. there are interfaces where a quad is e.g. touching two triangles), and a layer of zero-volume elements needs to be introduced to convert them into conforming meshes. In our case, the meshes are conforming, but we cannot guarantee to have only a restricted set of elements. Converting between these two representations is straightforward (in the space of non-conforming meshes), and the favored representation depends on the application. For example, polyhedral meshes can be directly used for computer animation in the algorithm proposed in [Martin et al. 2008; Bishop 2014], while non-conforming meshes are ideal for [Reberol and Lévy 2016]. Quality-wise, the hexahedral elements of our meshes are more isotropic than those generated by PGP3D (figures 10 and 11). Our algorithm casts the optimization of both the orientation and of the directional field as simple local iterations, which are simple to implement and robust to random initializations (Figure 9). Additionally, our mesh extraction algorithm is general, can be used for surfaces and volumes, and it does not rely on a constrained Delaunay tetrahedralization step.

Existing Software. Robust software solutions exist for octree-based hexahedral meshing, producing cut cells and a high degree of refinement at the boundary [Car 2016; LBI 2016; Har 2016; HEX 2016; Bol 2016; Hex 2016; Mes 2016a; Kub 2016; XBX 2016]. Boundary-aligned

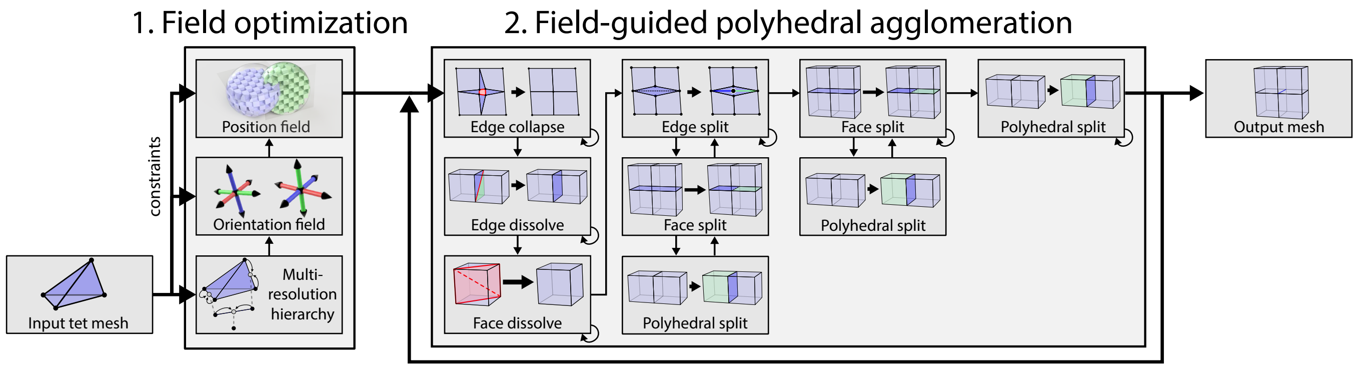


Fig. 2. Our method is composed of two high-level phases: **field optimization** takes an existing tetrahedral mesh as input and optimizes the smoothness of two guiding fields controlling the orientation and spatial placement of mesh elements. A multi-resolution hierarchy accelerates convergence during this process. The **field-guided agglomeration phase** converts the fields into a stream of local transformations applied to the input mesh, each preserving both manifold structure and genus. Self-loops indicate repeated application of an operation over the entire mesh until no more qualifying elements are left. Certain operations occur multiple times—once as a sweep over the mesh (with self-loop), and once as a local step triggered by another operation (an edge split e.g. triggers a face split, which in turn triggers a polyhedral split).

techniques exist to automatically meshing special geometries, such as cylinders, boxes and sweepable solids [PAM 2016; ANS 2016; Hyp 2016; Sie 2016]. Hexahedral meshes of arbitrary geometries are currently constructed by manually decomposing the shape into simpler pieces, that are then meshed while ensuring compatible interfaces are introduced [CUB 2016; Tre 2016; Ape 2016]. Mixed meshes, containing different types of elements, are easier to generate, and many commercial codes have been developed for this task [Aut 2016; Mes 2016b; AMP 2016; BET 2016; TEX 2016]; these tools are closer to our goals. However, to the best of our knowledge, no free or open-source implementation exists, and the limitations of these techniques, while informally known, are difficult to quantify precisely. In addition, none of these softwares can produce meshes aligned with a given volumetric orientation field.

Our reference implementation will be the first open-source, fully automatic, and unconditionally robust hex-dominant mesher. It is based on a field-aligned approach, ensuring alignment to features and even distribution of singularities in the interior, which ensures high element isotropy. We expect our contribution and reference implementation to have a large impact in the graphics and computer aided design research communities.

3 FIELD OPTIMIZATION

Seen from a high level, our algorithm consists of three major components (Figure 2) that follow a well-established structure used by numerous recent global parameterization methods: the first computes an *orientation field* guiding the orientation of edges in the output mesh; the second takes the orientation field as input and converts it into a *position field* guiding the spatial placement of mesh elements. The last step extracts the final mesh from the two fields.

Overview. The orientation and position optimizations are realized using a unified iterative smoothing algorithm, which accounts for inherent symmetries associated with the respective fields. The contents of the optimized output fields are then used to guide the mesh extraction phase, but the algorithm's correctness does not hinge upon their quality. This is a crucial property: although the fields will generally be in close agreement with the extracted mesh, they are not guaranteed to be free of degeneracies.

The final mesh extraction phase converts the information contained in the position field into a stream of local mesh operations that are sequentially applied to the input triangle or tet mesh, eventually producing the output mesh. Importantly, each iteration of this process preserves the manifoldness as an invariant; operations that would introduce non-manifold configurations are simply postponed and retried later on. During this process, our algorithm continues to update the position and orientation fields to account for changes resulting from the simplification steps. This is crucial: in certain situations, it may be necessary to locally subdivide (i.e. increase the element count rather than reduce it) in a part of the mesh to unlock a sequence of subsequent simplification operations. Having access to continually updated orientation and position fields leads to a straightforward criterion to decide whether and where such steps are necessary.

We now discuss each of these components in turn, focusing on the 3D case; the generalization to two dimensions is discussed later in Appendix.

Notation. The input to our method is an existing tetrahedral mesh with boundary $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{T})$ with vertex positions $\mathbf{x}_i$ for every vertex $i \in \mathcal{V}$. We define the neighborhood of a vertex as $\mathcal{N} = \{j \in \mathcal{V} \mid (i, j) \in \mathcal{E}\}$.

3.1 Orientation Field Optimization

The first phase of our method optimizes the smoothness of the orientation field which associates a 3D cross represented using a right handed coordinate frame $\mathbf{Q}_i \in \mathbb{R}^{3\times3}$ with every vertex $\mathbf{x}_i$ inside the volume. Each frame controls the alignment of a local parameterization encoded via the position field that is generated in a subsequent step and then used to guide the mesh extraction phase. Since our goal is to create isotropic output, we impose the constraint that the $\mathbf{Q}_i$ are rotation matrices.

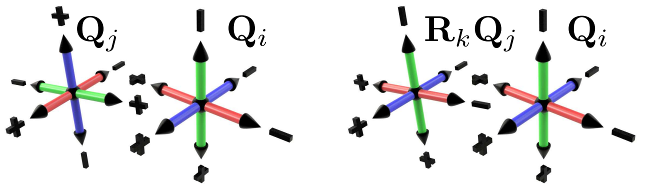


Fig. 3. **Left**: Incompatible pair of orientations. **Right**: Matched orientations using one of the 24 canonical rotations from the field's symmetry group.

Smoothness. The orientation field is subject to symmetries that must be considered when reasoning about its smoothness. Analogous to cross fields on surfaces, any 3D frame that can be reached by transforming an existing one using arbitrary sequences of 90°-rotations around any of its axes is considered part of the same equivalence class, leading to a 24-fold symmetry (Figure 3).

We define the smoothness of the orientation field using an energy function that measures the dissimilarity of appropriately rotated neighboring frames, i.e.

$$E_Q(\mathbf{Q}, \boldsymbol{\kappa}) := \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}(i)} d_Q(\mathbf{R}_{\kappa_{ij}} \mathbf{Q}_i, \mathbf{Q}_j)^2 \tag{1}$$

where $d_Q(\mathbf{Q}_i, \mathbf{Q}_j) = \frac{1}{2} \| \log(\mathbf{Q}_i \mathbf{Q}_j^T) \|$ is a metric denoting geodesic distance on the rotation group $SO(3)$ and $\boldsymbol{\kappa}$ denotes a matching on edges. Each matching variable κ_{ij} selects one of the 24 possible rotation matrices $\mathbf{R}_1, \ldots, \mathbf{R}_{24}$ from the frame field's symmetry group. Optimizing the smoothness thus entails finding both the set of matchings and orientations so that the energy is minimized, i.e.

$$(\mathbf{Q}^*, \boldsymbol{\kappa}^*) = \operatorname{argmin} E_Q(\mathbf{Q}, \boldsymbol{\kappa}). \tag{2}$$

Similar to surface-based cross field smoothness energies that incorporate integer variables [Bommes et al. 2009], the discrete nature of the matchings leads to a challenging non-convex optimization problem whose combinatorial nature defies techniques that exhaustively search the solution space for the global optimum. As in the 2D case, we observed that the energy landscape is characterized by a large number of local minima that generally don't correspond to satisfactory solutions, hence purely local iterative solvers cannot be used.

Multi-Resolution Hierarchy. Our optimization approach is inspired by the Instant Field-Aligned Meshes (IM) [Jakob et al. 2015] technique, specifically the observation that a relatively naive iterative algorithm can produce excellent results when it is combined with a coarse-to-fine optimization scheme based on an unstructured multiresolution hierarchy. As in IM, we build this hierarchy level by level starting from the input tet mesh, with each progressively coarser level containing approximately half the number of vertices. The coarser levels only encode edge connectivity, which suffices to solve diffusion-type smoothing problems.

To create a coarser layer from an existing one in the hierarchy, we visit each edge (sorted from shortest to longest) and attempt to collapse it into a single vertex located at its barycenter (Figure 4). A vertex can only be part of a single operation in each phase, and the coarsening step finishes when no more edges can be merged. We repeat this process until only a single vertex remains and keep track

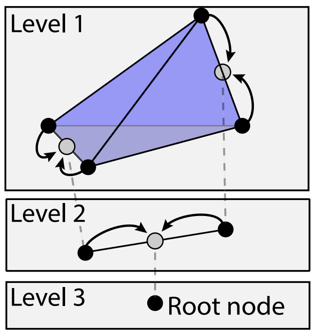
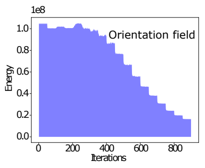


Fig. 4. **Left**: Our method's two field optimizations propagate course solutions through an unstructured multiresolution hierarchy that is constructed by greedily collapsing adjacent vertices. The above illustration shows a tiny hierarchy constructed from a single tetrahedron.

of all intermediate layers as well as the relations between finer and coarser-level vertices that can be used to propagate solutions up and down in the resulting tree.

As with other iterative solution techniques for nonconvex energies, our method cannot guarantee convergence to a global minimum. The discrete matchings in the energy lead to a solution space that is characterized by many local minima that correspond to low-quality solutions, hence an algorithm that "merely" converges to any local minimum is not usable for mesh generation. Our use of a hierarchy was driven by the empirically motivated hypothesis that the global minima of the energy function for two adjacent hierarchy levels are closely related so that a converged solution at one level provides an excellent warm start for optimization at the next finer level. The inset shows a plot of the smoothness energy for the fertility model in Figure 9.



Nonlinear Gauss-Seidel. We now focus on the iterative part of the algorithm performed on each level, which consists of simple sequence nonlinear Gauss-Seidel steps where each step replaces the rotation $\mathbf{Q}_i$ at a vertex i with an average of the rotations of its neighbors $\mathcal{N}(i)$. The averaging proceeds vertex by vertex using a conceptual sequence of updates of the following form (initialized with $\mathbf{Q}_i' \leftarrow 0$) followed by re-normalization:

$$\mathbf{Q}_i' \leftarrow \mathbf{Q}_i' + \mathbf{R}_{\kappa_{ij}} \mathbf{Q}_j \tag{3}$$

Since the averaging only considers pairs of elements at a time, the best matching κ_{ij} between $\mathbf{Q}_i'$ and $\mathbf{Q}_j$ can be determined using an exhaustive search. At this point, we note that a representing rotations as 3×3 matrices leads to an unnecessarily inefficient implementation: in addition to the obvious redundancy, a considerable amount of arithmetic is spent evaluating the rotation group metric $d_Q(\cdot, \cdot)$ to determine the best matching κ_{ij} in each smoothing iteration. Finally, the resulting average $\mathbf{Q}_i'$ is generally not orthogonal and will require re-normalization using Gram-Schmidt or a similar algorithm.

Quaternionic Representation. These drawbacks motivate our reliance on unit quaternions $\mathbf{q}_i \in \mathbb{Q}$, which admit highly efficient implementations of the key operations that are needed by the orientation field optimization. In this representation, the rotation group metric takes on the simple form $d_Q(\mathbf{q}_i, \mathbf{q}_j) = \arccos\langle \mathbf{q}_i, \mathbf{q}_j \rangle$ and the canonical rotations $P_1, \ldots, P_{24}$ from the symmetry group are elements of

$$\mathcal{R} := \left\{ \mathbf{r}/\|\mathbf{r}\| \;\middle|\; \mathbf{r}_i \in \left\{ -1, 0, 1 \right\}^4 \wedge \sum |p_i| = 1, 2, \text{or } 4 \right\}, \tag{4}$$

i.e. the set of unit quaternions with 1, 2, or 4 nonzero entries of the same magnitude. Note that $\mathcal{R}$ has 48 elements as every rotation can be represented by two equivalent quaternions of opposite sign.

Finding the best matching κ_{ij} now entails minimizing $d_q(\mathbf{q}_i, \mathbf{q}_j \mathbf{r}_k)$ over $\mathbf{r}_k \in \mathcal{R}$, and the form of d_Q implies that the minimizer is simply the quaternion $\mathbf{r}^*$ maximizing the inner product $\langle \mathbf{q}_i, \mathbf{q}_j \mathbf{r}^* \rangle$. Let

$$\alpha_k := \langle \mathbf{q}_i, \mathbf{q}_j \mathbf{e}_k \rangle, \qquad (k = 1, \ldots, 4) \tag{5}$$

where $\mathbf{e}_k$ are the standard basis vectors in $\mathbb{Q}$. Due to linearity, the inner product β^* associated with the best matching $\mathbf{r}^*$ can be expressed as

$$\beta^* := \langle \mathbf{q}_i, \mathbf{q}_j \mathbf{r}^* \rangle = \sum_{k=1}^{4} \alpha_k p_k^*. \tag{6}$$

Interestingly, due to the simple structure of $\mathcal{R}$, the inner products α_k already contain sufficient information to distinguish how many nonzero entries the best matching $\mathbf{r}^*$ must have. Specifically, let

$$\gamma_1 := \max_i |\alpha_i|, \quad \gamma_2 := \max_{i,j} |\alpha_i| + |\alpha_j|, \quad \gamma_4 := \sum_i |\alpha_i|. \tag{7}$$

If both $\gamma_4 > \gamma_1$ and $\gamma_4 > \gamma_2$ then the largest possible inner product $\beta^* = \gamma_4$ necessarily involves a quaternion of the form $\mathbf{r} = (\pm 1, \pm 1, \pm 1, \pm 1)/2$ whose component signs match those of the inner products α_i. A similar logic can be used to determine $\mathbf{r}^*$ for the other two cases, hence finding a matching reduces to a simple computation involving four dot products, determining which of three cases applies and finally constructing a unit quaternion with the corresponding set of nonzero entries and signs. A detailed discussion of these steps is provided in the supplemental material. Needless to say, this is dramatically more efficient than the brute force matrix-based approach.

Using the quaternionic representation, one full sweep of the nonlinear Gauss-Seidel update then takes on the form

```
function OPTIMIZE-ORIENTATIONS(q)
    for $i = 1, \ldots, n$ do
        $\mathbf{q}_i' \leftarrow \mathbf{0}$
        for each $j \in \mathcal{N}(i)$ do
            $\mathbf{q}_i' \leftarrow \mathbf{q}_i' + \mathbf{q}_j\, \mathbf{r}^*(\mathbf{q}_i, \mathbf{q}_j)$
            $\mathbf{q}_i \leftarrow \mathbf{q}_i' / \|\mathbf{q}_i'\|$
```

This iteration visits every vertex in the mesh, replacing the local orientation field value with an average while using the previously discussed matchings $\mathbf{r}^*$ to account for symmetries. We run the optimization for 200 iterations at every level and then copy the solution to the next finer level. While experimenting with different versions of this approach, we found that a judicious amount of randomness can help the individual smoothing iterations move out of local minima, which further improves convergence in conjunction with the hierarchy. Specifically, we modify the traversal in line 4 to visit the neighboring vertices $\mathcal{N}(i)$ using a different order in each iteration. The order of the vertices in the outer loop (line 2) is not randomized.

Constraints. Without added constraints, the above algorithm will simply converge to a constant orientation field; however, to be useful for remeshing, the field should smoothly interpolate boundary constraints and possibly internal alignment constraints if specified.

In standard iterative solvers for linear systems, such point-wise constraints are easily applied by replacing certain variables with constants. However, this is too severe of a constraint in our case, since the orientation field values should only be fixed up to transformations by the underlying symmetry group. Furthermore, the orientation field frames on the boundary should still have a degree of freedom to permit rotation perpendicular to the surface normal.

A simple extension of Algorithm 1 then corrects the averaged orientations $\mathbf{q}_i$ by applying the smallest rotation that will make one of the frame's axes align with the normal vector $\mathbf{n}_i$ at boundary vertices. As in the case of the matchings, this rotation is easily and efficiently obtained by reasoning about magnitudes and signs of the quaternion entries. We provide a detailed specification of this step in the supplemental material.

Symmetrized Orientations. Before continuing, we establish two additional definitions that are required by the subsequent phases. The field

$$\mathbf{q}_{ij} := \frac{\mathbf{q}_i + \mathbf{q}_j \mathbf{r}^*(\mathbf{q}_i, \mathbf{q}_j)}{\|\mathbf{q}_i + \mathbf{q}_j \mathbf{r}^*(\mathbf{q}_i, \mathbf{q}_j)\|}, \tag{8}$$

defines a symmetrized set of orientations satisfying $\mathbf{q}_{ij} \equiv \mathbf{q}_{ji}$ (i.e. equality up to symmetries). We furthermore define $\mathbf{Q}(\mathbf{q})$ as the matrix representation of a rotation quaternion $\mathbf{q}$.

3.2 Position Field Optimization

The position field optimization is a generalization of the corresponding step in the two-dimensional IM technique, which we discuss here for completeness.

The position field is easiest to visualize when its discretization is sampled on volume elements (Figure 5). In this case, the position field values encode points on 3D grids permeating the interior of mesh elements, whose axes are determined by the previously computed orientation field. Since the grid does not change when translated by integer multiples of its cell width, only the fractional part of the position is relevant. In practice, our implementation samples the position field on vertices along with the orientation field, which is helpful in the extraction stage later on.

Smoothness of the position field is defined as a sum over the squared distances of suitably translated positions field values

$$E_p(\mathbf{p}, \boldsymbol{\tau}) := \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}(i)} d_p(\mathbf{p}_i + \lambda \mathbf{Q}(\mathbf{q}_{ij}) \boldsymbol{\tau}_{ij}, \mathbf{p}_j)^2, \tag{9}$$

where $\boldsymbol{\tau}_{ij} \in \mathbb{Z}^3$ $(i, j = 1, \ldots, n)$ is a set of 3D integer translations, $d_p(\mathbf{a}, \mathbf{b}) := \|\mathbf{a} - \mathbf{b}\|_2$, and λ is the position field's grid size, which is identical to the desired target edge length. The integer variables serve the same purpose as the rotation indices κ_{ij} from before: to remove excess integer jumps so that only fractional differences are

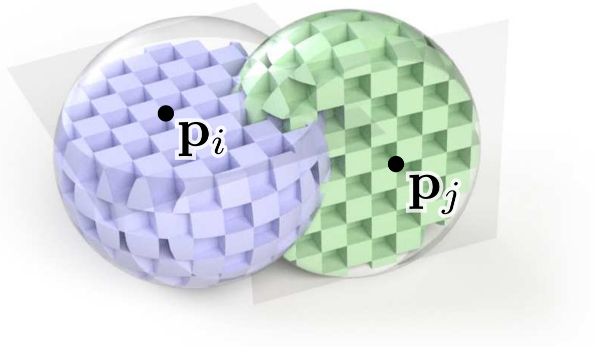


Fig. 5. Conceptual cutaway view of a position field discretized on a pair of overlapping volume elements. $\mathbf{p}_i$ and $\mathbf{p}_j$ encode three-dimensional positions located on integer positions of a 3D grid permeating their interior. Translating them by integer amounts does not change the underlying grid, hence only the fractional position is relevant. The second stage of our pipeline optimizes the smoothness of this field subject to a symmetry group of integer translations.

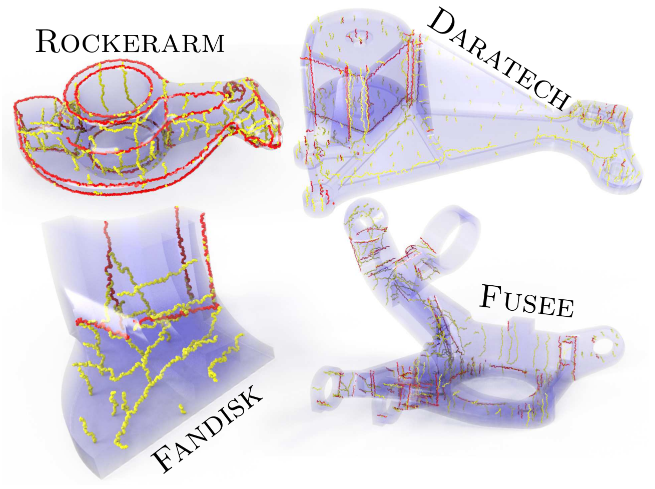


Fig. 6. Examples of singularity graphs obtained with our method. Red and yellow curves denote singularities of the orientation and position field, respectively.

measured. Optimizing the smoothness, again, entails finding both the field and the set of matchings minimizing the energy, i.e.

$$(\mathbf{p}^*, \boldsymbol{\tau}^*) = \operatorname{argmin} E_p(\mathbf{p}, \boldsymbol{\tau}), \tag{10}$$

The position field optimization relies on the same overall approach of performing non-linear Gauss-Seidel sweeps over the mesh in conjunction with a coarse-to-fine traversal of the multi-resolution hierarchy. One missing piece to realize this step is a way of obtaining the best position field matching for an adjacent pair of vertices, which is defined as

$$\boldsymbol{\tau}_{ij}^* := \underset{\boldsymbol{\tau}_{ij}}{\operatorname{argmin}}\, d_p(\mathbf{p}_i + \lambda \mathbf{Q}(\mathbf{q}_{ij})\boldsymbol{\tau}_{ij}, \mathbf{p}_j)^2. \tag{11}$$

where $\lambda \mathbf{Q}(\mathbf{q}_{ij})\boldsymbol{\tau}_{ij}$ the offset due to an integer number of steps on a grid with cell spacing λ and orientation $\mathbf{Q}(\mathbf{q}_{ij})$. Since the axes of this grid are orthogonal, the minimization has a simple explicit solution in terms of a component-wise rounding operation:

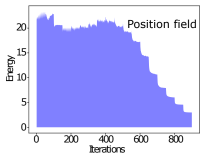
$$\boldsymbol{\tau}_{ij}^* = \operatorname{round}\left[\left(\lambda \mathbf{Q}(\mathbf{q}_{ij})\right)^{-1}\left(\mathbf{p}_j - \mathbf{p}_i\right)\right]. \tag{12}$$

With this last piece at hand, we can now define the position smoothing iteration applied to each level of the hierarchy:

```
function OPTIMIZE-POSITIONS($\mathbf{p}$)
    for $i = 1, \ldots, n$ do
        $\mathbf{p}_i' \leftarrow \mathbf{0}, k \leftarrow 0$
        for each $j \in \mathcal{N}(i)$ do
            $\mathbf{p}_i' \leftarrow \mathbf{p}_i' + \mathbf{p}_j + \lambda \mathbf{Q}(\mathbf{q}_{ij})\boldsymbol{\tau}^*(\mathbf{p}_i, \mathbf{p}_j)$
            $k \leftarrow k + 1$
            $\mathbf{p}_i \leftarrow \mathbf{p}_i'/k$
        $\mathbf{p}_i \leftarrow \mathbf{p}_i + \lambda Q^{-1}(\mathbf{q}_{ij}) \operatorname{round}\left(\lambda^{-1} Q(\mathbf{q}_{ij})(\mathbf{x}_i - \mathbf{p}_i)\right)$
```

The last step in line 8 rounds each position field value $\mathbf{p}_i$ to the integer position closest to the associated vertex position $\mathbf{x}_i$.Constraints can be incorporated by projecting the position field entry onto the tangent plane of boundary vertices following this rounding step.

The inset shows energy for the fertility model in Figure 9.



Field Singularities. The orientation and position fields are characterized by the presence of degeneracies named *singularities.* They can be detected by concatenating the computed integer rotations $\mathbf{r}_{ij}$ or translations $\boldsymbol{\tau}_{ij}$ around simple loops (i.e. connected chains of edges that do not self-intersect) and testing whether the concatenation becomes an identity upon returning to the starting point [Huang et al. 2011; Li et al. 2012; Ray et al. 2016].

Singularities effectively absorb excess rotations and translations, allowing our method to achieve a high level of isotropy. They are automatically introduced as part of optimizing the smoothness of the field. Figure 6 shows several visualizations of singularity graphs obtained using our method.

4 MESH EXTRACTION

Following field optimization, the second phase of our method draws on the information contained in the orientation and position fields to convert the input tetrahedral mesh into a manifold hex-dominant output mesh.

Recall that the position field records the offset of a local parameterization at every vertex, which is encoded as the 3D position of the nearest integer grid point. If we simply replace the positions of all input mesh vertices with their optimized position field counterparts, we obtain a new mesh, whose shape generally resembles the hex-dominant mesh we wish to extract. Figure 7 shows several interpolated versions of this transformation to illustrate its behavior. Note that the connectivity of the transformed mesh matches that of the input mesh—in other words, it has the structure of a manifold tetrahedral mesh, which will naturally require further modifications before it can be used as hex-dominant output. From a geometric viewpoint, we observe that the transformation creates many undesirable collapsed and inverted elements.

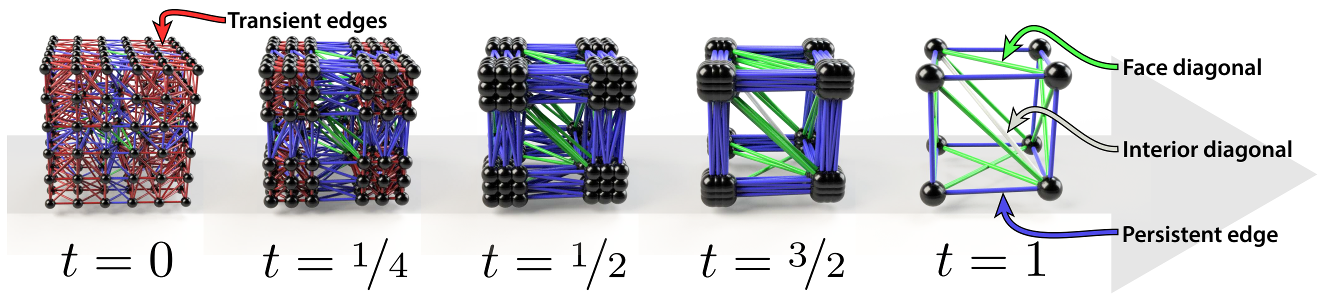


Fig. 7. Position field visualization for a cube-shaped tetrahedral object meshed into a single hexahedron. The sequence shows intermediate steps of a linear interpolation $\mathbf{v}_i(t) := (1-t)\mathbf{x}_i + t\mathbf{p}_i$ from the original vertex positions $\mathbf{x}_i$ to their associated position field values $\mathbf{p}_i$. The interpolation reveals how the position field maps most tetrahedral elements onto regions with zero volume (vertices as well as quad- and hex-diagonal edges). Colors in the above visualization are chosen according to the position field integer variables τ_{ij}, which classify the intended role of edges in the final mesh. Please see the supplemental material for an animated version of this figure.

The mesh extraction phase of our method then applies a sequence of local operators to fix problematic areas in the transformed mesh, removing collapsed regions and agglomerating tetrahedra into hexahedra or other types of polyhedra. The precise sequence of operations is controlled by the position field integer variables $\boldsymbol{\tau}_{ij}$, which specify the intended final role of every edge in the transformed mesh (Figure 7).

Similar to other field-aligned meshing techniques, the optimized fields are not guaranteed to be free of degeneracies and should therefore only be used as guides, hence, an important aspect of this phase is that each operation is only performed if it preserves the manifold structure and genus as an invariant. Apart from the difference in dimension, this approach constitutes a significant deviation from the IM mesh extraction algorithm, which unconditionally applied a single operation capable of producing non-manifold output.

4.1 Edge Classification

It will be helpful to classify edges into several groups analogous to the color scheme in Figure 7; the classification of an edge (i, j) is based on the associated position field integer value $\boldsymbol{\tau}_{ij}$.

(1) **Transient edges** satisfy $\boldsymbol{\tau}_{ij} = \mathbf{0}$. They are the most common and occur in larger clusters of edges that all map to a single vertex of the output mesh. Transient edges are thus not needed, and our extraction pipeline attempts to remove them.
(2) **Persistent edges** have an integer variable $\boldsymbol{\tau}_{ij}$ with exactly one nonzero coefficient ± 1. Persistent edges are aligned with the orientation field and connect clusters of transient edges. They correspond to the edges of the final mesh that should persist through the extraction pipeline, hence their name. Note that there might be multiple representatives of a persistent edge that will be merged during extraction.
(3) **Face and interior diagonals** have an integer variable $\boldsymbol{\tau}_{ij}$ with exactly two or three nonzero coefficients ± 1, respectively. These edges are not aligned with the orientation field and pass diagonally through faces or polyhedra of the output mesh. In both cases, the associated edges should be dissolved during extraction, merging the tetrahedral elements around them into hexahedra or other kinds of polyhedral elements.
(4) **Other edge** types must by definition involve an integer variable with at least one coefficient having magnitude > 1, which means that the target edge length λ is at least a factor of 1.5 smaller than the distance of the edge's vertices in the input mesh. For simplicity, our extraction pipeline does not attempt to deal with such edges—if they arise, it is necessary to either increase the target edge length λ or create a finer tetrahedral input mesh using TetGen [Si 2015] or a similar tool.

We also introduce a version of the position field energy that is defined per edge, i.e.

$$\varepsilon_{ij} = \|\mathbf{p}_i - \mathbf{p}_j + \lambda \mathbf{Q}(\mathbf{q}_{ij})\boldsymbol{\tau}_{ij}\|^2 \tag{13}$$

When our extraction pipeline applies local operators during sweeps over the mesh, it uses the values of ε_{ij} to prioritize regions where the position field is smoothest. Large values of ε_{ij} imply an ambiguous and potentially low-quality classification into the categories of the above list, hence the associated elements are considered last (at which point they are usually ignored since applying the corresponding operator would break manifoldness).

4.2 Extraction via Polyhedral Agglomeration

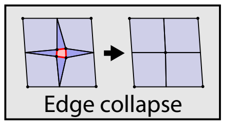
Seen from a high level, our extraction algorithm alternates between consecutive *coarsening* and *splitting* phases. The coarsening phase reduce the complexity of the input mesh by removing edges, vertices, and faces from it. However, this phase alone does not suffice: an algorithm that is only based on local coarsening operators can run into complex interlocking configurations of elements where any further operation would break manifoldness, thereby inhibiting forward progress; an instance of this problem was studied by Dougherty et al. [2004].

The second phase of our algorithm thus performs additional splitting operations where needed. These steps release interlocked configurations, and they reduce the complexity of elements by splitting high-degree faces and polyhedra into simpler configurations. This alternating sequence of iterations repeats until no more edges can be collapsed or split, or after a maximum of 10 iterations has passed. In our experiments, we find that most models converge after an average number of 3 iterations. A few models contain elements that are alternately coarsened and split, eventually triggering the 10 iteration limit. These are ambiguous elements which have a satisfactory quality in both coarsen and split configurations and hence do not cause problems.

We now discuss each operator in turn. Note that the color scheme in the small inset figures differs from that of Figure 7: red refers to components to be removed, and blue and green are used to highlight distinct components.

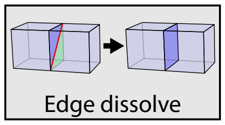
4.2.1 *Coarsening Operators.*

Edge Collapse. The first coarsening operator traverses all transient mesh edges using a priority queue that is ordered according to the edge energy ε_{ij}, visiting smoother regions of the mesh first. It then attempts to collapse each transient edge into a single vertex using a suitable update to the surrounding mesh connectivity. Collapse operations that would result in a non-manifold or genus-changing configuration are rejected but can be retried in a later phase. The underlying topological test is identical for all operators and is described in Section 4.2.3. If successful, the collapsed vertex is assigned an averaged orientation and position field value computed from its constituent vertices.

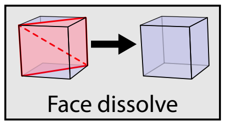


Note that position field adjustments will generally change the smoothness energy ε_{ij} of connected vertices and could potentially even trigger a change in the high-level classification of adjacent edges. We associate a discrete timestamp with every edge that is also pushed on the priority queue. When an edge obtained from the queue does not match its global timestamp, the information is stale and can safely be ignored.

Edge Dissolve. The second coarsening operator traverses all edges classified as face diagonals according to ε_{ij} and attempts to remove them to create a larger face. In the inset shown on the right, this operation entails converting two polyhedra with seven faces into two hexahedra with six faces each. Invalid operations are skipped as before.

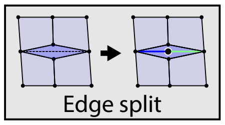


Face Dissolve. The third coarsening operator traverses all edges classified as interior diagonals (dashed), attempting to remove all adjacent faces and thereby agglomerating tetrahedra into hexahedra, prisms, or other polyhedra. The algorithm also attempts to remove the leftover face diagonal edges (solid red) as part of a separately triggered set of atomic operations.



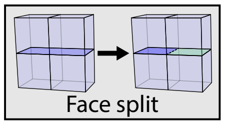
4.2.2 *Splitting Operators.* Following coarsening, the second of the two alternating extraction phase performs splits according to a set of criteria to be discussed, which increase the number of vertices, faces, and edges. Splits are intended to simplify high-degree polyhedral elements by breaking them into smaller parts as well as trigger sequences of coarsening steps that were previously inaccessible due to interlocking configurations. Note that some of the splitting operations occur multiple times —once as a sweep over the mesh (indicated using a self-loop in Figure 2), and once as a local step triggered by another operation (an edge split e.g. triggers a face split, which in turn triggers a polyhedral split). Each step occurs in an atomic unit; recursively triggered steps are not required to succeed for the parent operation to be committed to the mesh.

Edge Split. Consider the example configuration shown in the right inset: a pair of elongated faces (dark blue) containing an overly long edge (dashed) interrupts the otherwise regular structure of the mesh. Ideally, the two faces should be collapsed and removed, but there are no transient edges that would allow such an operation to occur.



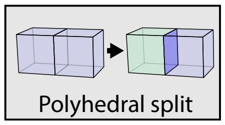
We handle this and a variety of other similar configurations by detecting overly long edges in faces and inserting a vertex at their barycenter. A subsequent face split operation (discussed next) then inserts the requisite transient edge, enabling an edge collapse in the next extraction phase. An edge is defined as being overly long if the ratio of its horizontal or vertical length (determined according to the orientation field) compared to the corresponding length of the smallest adjacent edge rounds to a value ≥ 2. In this case, we split the edge in the middle by inserting a vertex with an interpolated rotation and position field value.

Face Split. The face split operation traverses every face of the mesh and attempts to insert an additional persistent or transient edge that would cause the face to separate into two parts. The operator simply loops over every possible pair of edges in the face, classifying the resulting hypothetical edges using their orientation and position field values. The operator then evaluates the smoothness energy ε_{ij} for each candidate from the set of transient and persistent edges and finally attempts to carry out the lowest-energy face split.



When a localized face split operation is triggered by a preceding edge split, the search over edges constrains one of the endpoints to lie on the newly created vertex. A face split operation in turn triggers two recursive polyhedral split operations associated with each of the newly created faces.

Polyhedral Split. The preceding splitting operators increase the number of faces of the surrounding mesh elements, which may occasionally create polyhedra with a large number of faces. These elements are not representative of the mesh encoded by the position field—they are simply artifacts of the lack of a sufficient number of edges in the input mesh and therefore undesirable. The last splitting operator processes these large polyhedra, breaking them into smaller components.



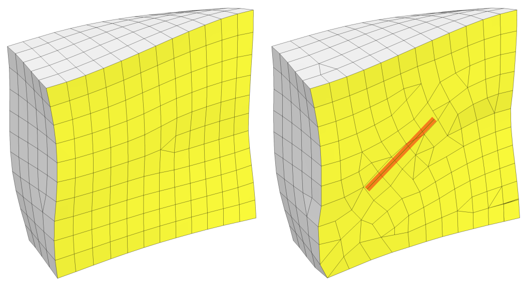


Fig. 8. Our field optimization supports alignment constraints on the boundary (left) and in the interior (right).

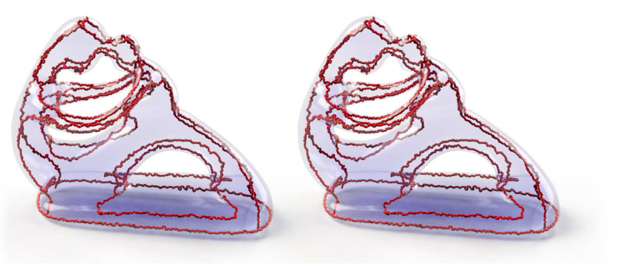


Fig. 9. Our orientation field optimization is robust to different initializations. The fields obtained starting from a constant (left) and random (right) initialization are indistinguishable. **Left**: constant initialization. **Right**: random initialization

The polyhedral split operation traverses every polyhedron in the mesh and attempts to insert an additional face to the interior that would cause the polyhedron to separate into two parts. A split candidate must be constructed from previously existing mesh edges arranged in a simple loop. Our implementation uses depth first search to enumerate all possible loops that start and end on the same edge, filtering non-manifold configurations. We then select the loop that deviates less, i.e. it is more planar, with respect to one of the main axes of the face. The polyhedra in our meshes are fortunately small enough so that the exponential number of edges found in this way remains unproblematic.

4.2.3 Topological Invariants. Our algorithm starts from a manifold tetrahedral mesh and preserves its manifoldness by preserving the following invariants throughout the coarsening and splitting phases.

(1) Each face of a polyhedron is topologically a circle.
(2) Each polyhedron is topologically a sphere.

Designing mesh operations that efficiently enforce the above conditions on a general polyhedral mesh can be tremendously challenging. To keep implementation complexity at bay while allowing for strong guarantees, we opted for the following simpler approach: before each operation, we extract a sufficiently large topological sphere encapsulating the affected element(s) using breadth first search. After applying the operations to the extracted mesh, our algorithm scrutinizes the resulting connectivity, only accepting the operation if the topological invariants are not violated. While this is far from optimal from an efficiency perspective, it allowed for a drastically simplified implementation.

5 RESULTS

We implemented our algorithm in C++, using *Eigen* for linear algebra routines. We run all our experiments on a workstation with a 6-cores Intel processor clocked at 3.5 Ghz and 64 Gb of memory, using only one thread.

Parameters. Our algorithm only requires one parameter, the target edge-length l of the hex-dominant mesh, which controls the output mesh density. Our pipeline requires a clean, sliver-free input triangle mesh. Its resolution is not important since we convert it into a dense tetrahedral mesh with an edge-length of 30% l using tetgen [Si 2015].

Optionally, our method can preserve sharp features prescribed by the user. Since our test datasets do not explicitly contain this information, we detect them using a simple heuristic: we tag all the edges with a dihedral angle lower than 150 degrees as sharp edges. Vertices on sharp edges are tagged as features, and are not enforced explicitly: we align both fields to the boundary *only* on non-feature vertices. The alignment to the sharp features is a consequence of the energy minimization and it happens automatically.

Internal Constraints. Internal constraints can be easily incorporated into our field optimization framework in the same way as we handle boundary alignment. Figure 8 shows an example where we added a line alignment constraint in the interior of the twisted cube.

Robustness. To test the robustness of our algorithm, we process the complete database proposed in [Fu et al. 2016], with a target edge length of 2.5 times the average edge length of the input mesh. Our algorithm successfully produced hex-dominant meshes for all 106 models. We attach the remeshed database as additional material: For each model we provide the mesh file in ascii format, and a short mp4 clip showing its interior during a plane sweep. The size of the meshes in the database ranges from 0.06 to 3.6 millions tetrahedra, our running time from 1 minute to 10 hours, the hexahedral ratio from 48% to 91%, and the average Jacobian from 0.93 to 0.99. We refer to Table 1 for more detailed timings and quality measurements for the results shown in the paper. In figures 10, 11, 13, 14, and 15, we show histograms of the scaled Jacobian of the hexahedral elements.

Comparison with [Ray et al. 2016]. Section 3.1 introduces a novel volumetric orientation field representation and a corresponding smoothing algorithm. We processed the same models used in figure 11, 13, and 15 of [Ray and Sokolov 2015] with our algorithm and obtained visually similar fields. Remarkably, also the singularity structure is similar (Figure 6), suggesting that the two measures of smoothness are deeply related. One major advantage of our representation is the explicit modeling of the matchings, which supports higher-order singularities and exact singularity prescription. In contrast to methods based on spherical harmonics [Huang et al. 2011; Ray et al. 2016], our smoothing algorithm is insensitive to the initialization, as demonstrated in Figure 9. For all the other results in the paper and in the additional material, we use a random initialization.

Comparison with [Sokolov et al. 2016]. Figures 10 and 11 compare our algorithm with [Sokolov et al. 2016] on two datasets provided by the authors of [Sokolov et al. 2016]. Our algorithm produces meshes

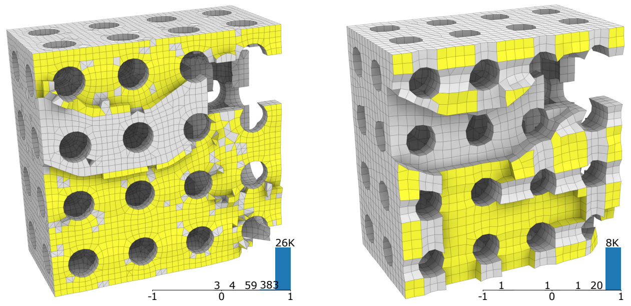


Fig. 10. A comparison between a remeshing of the *cheese* dataset created by [Sokolov et al. 2016] (left) and by our algorithm (right). The hex ratio is 60.43% (left) and 88.82% (right). The hexahedral element quality of the meshes are 0.96/-0.005/0.05 (left) and 0.95/-0.65/0.05 (right), measured by the scaled Jacobian in the format of average/minimum/standard deviation.

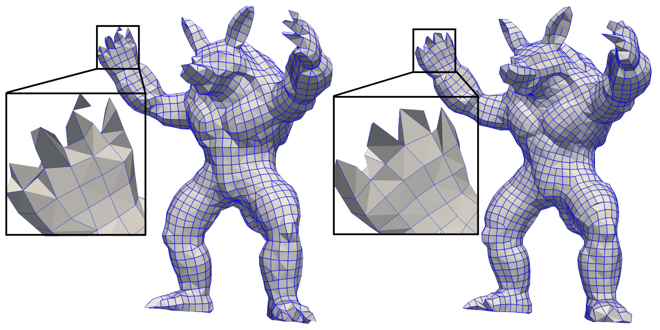


Fig. 12. Our extraction algorithm is guaranteed to produce manifold outputs (right), while the greedy method proposed by [Jakob et al. 2015] (left) often introduces non-manifold artifacts on features smaller than the target edge length.

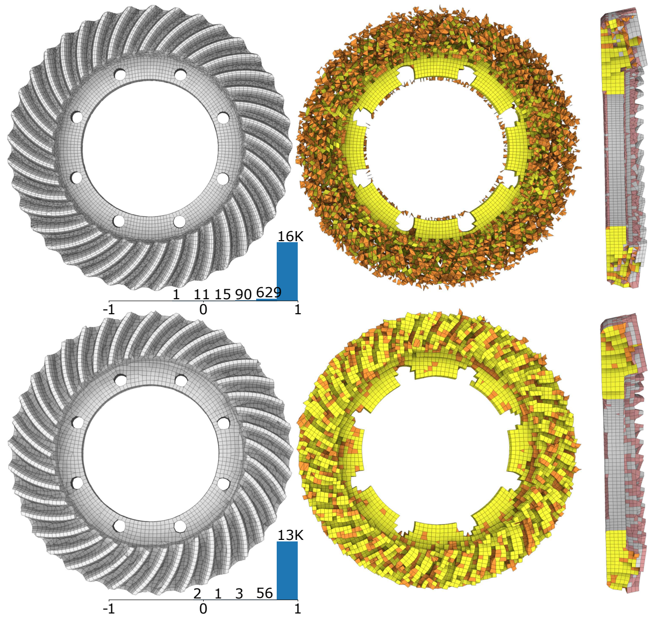


Fig. 11. A comparison between a remeshing of a challenging example with thin features, computed with [Sokolov et al. 2016] (top) and our algorithm (bottom). From left to right: an external view, an internal view obtained by peeling one layer of elements, a side view with a vertical cut. Our algorithm produces a more regular output (hex ratio 73.77% vs 35.27%) and the hex elements have a higher quality (0.97/0.04/0.04 vs 0.91/-0.23/0.08).

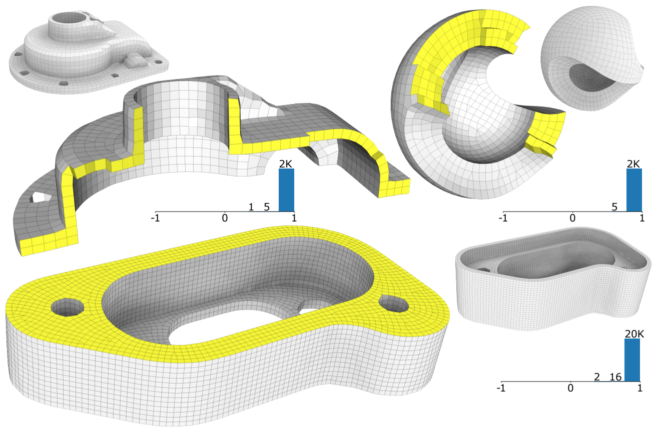


Fig. 13. A selection of CAD models meshed with our algorithm. Our meshes are highly isotropic and preserve sharp features.

with higher regularity and similar hexahedral element quality (as detailed in the figure captions). Note that the results from [Sokolov et al. 2016] have been optimized with [Brewer et al. 2003], while ours are not.

Comparison with [Jakob et al. 2015]. Our extraction algorithm (Section 4) can be easily adapted to process 2D orientation and position fields. In Figure 12, we compare our extraction algorithm with the greedy one proposed in [Jakob et al. 2015]: While the results between the two extraction methods are visually similar (Figure 12), the major advantage of our approach is that it is guaranteed to produce a manifold output, even for a large target edge length.

Gallery. CAD models with multiple holes and sharp features are challenging models for field-aligned parametrization methods. Our algorithm can process them robustly, while preserving sharp features (Figure 13). The orientation field also naturally aligns to smooth features of organic shapes, due to its extrinsic formulation (Figure 14). We show two additional challenging examples in Figure 15, and we refer to the additional material for more results.

6 LIMITATIONS AND CONCLUDING REMARKS

We introduced a fully automatic and robust algorithm to create manifold hex-dominant meshes, and validated it on a database with more than a hundred models. We expect that our algorithm will have a major impact in the graphics and CAD community and we will release a reference implementation to foster replicability of results and future research in this area.

Our current implementation is single-threaded, and not suited to process datasets with billions of tetrahedra. However, this is not a limitation of the algorithm, which can be fully parallelized

Dataset	#T_{in}	#H_{out}	H_{nbr}	H_{vol}	Quality	#Inv	#Int	#P	Hierarchy	RoSy	PoSy	Extraction
Cheese (Fig. 10)	765911	8439	0.89	0.89	0.95/-0.65/0.05	2/25/0	1	20	0.24	3.91	7.07	58.88
ASM_Limited (Fig. 11)	1457332	13010	0.74	0.72	0.97/0.04/0.04	0/56/0	8	41	0.34	5.16	12.24	139.36
Casting (Fig. 13)	218533	1916	0.83	0.83	0.98/0.53/0.04	0/4/1	0	17	0.07	1.1	1.86	6.21
Lock (Fig. 13)	1702872	20503	0.91	0.91	0.99/0.35/0.02	0/48/1	3	17	0.45	6.71	14.05	176.39
Sculpt (Fig. 13)	224353	1794	0.73	0.73	0.97/0.73/0.04	0/1/0	0	22	0.04	0.83	1.69	3.2
Elephant (Fig. 14)	1029536	9505	0.73	0.73	0.96/-0.35/0.06	3/26/3	2	31	0.21	3.61	7.48	52.32
Fertility (Fig. 14)	516140	4769	0.72	0.73	0.96/-0.33/0.07	4/23/0	2	24	0.12	1.9	4.02	17.77
Igea (Fig. 14)	1234724	12936	0.81	0.81	0.97/-0.23/0.05	1/33/0	2	22	0.24	4.08	9.84	69.35
Rocker (Fig. 14)	526133	5238	0.78	0.80	0.98/-0.78/0.05	1/11/1	2	18	0.11	1.9	4.12	18.3
Daratech (Fig. 15)	766733	8867	0.87	0.87	0.98/0.18/0.04	0/18/0	2	19	0.22	3.43	6.65	57.55
Mazewheel (Fig. 15)	637011	6938	0.81	0.81	0.97/0.28/0.04	0/36/0	3	18	0.17	2.72	5.02	45.33

Table 1. Timings and statistics for the models shown in the paper. From left to right: number of input tetrahedra, number of output polyhedra, ratio of the number of hexahedra to polyhedra, ratio of the volume of hexahedra to polyhedra, quality of hex elements (measured as average scaled Jacobian, minimum scaled Jacobian and standard deviation), number of degenerated elements (inverted hexahedra/inverted polyhedral/collapsed polyhedra), number of self-intersected elements, the maximal number of faces (for polyhedra), timings in minutes for hierarchy construction, RoSy optimization, PoSy optimization, and mesh extraction.

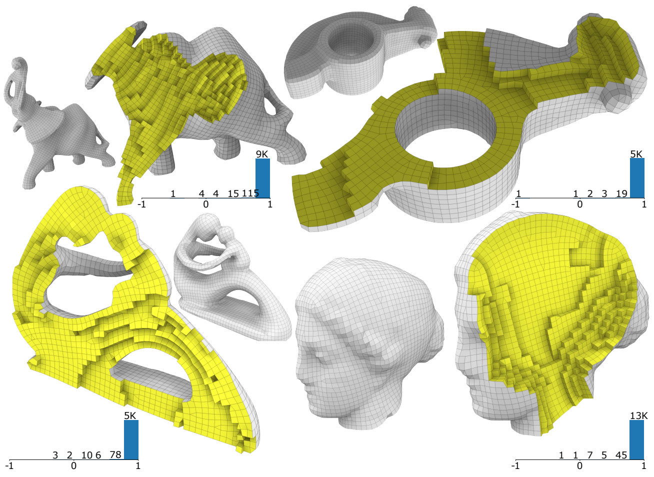


Fig. 14. A selection of models with smooth surface features. Note how our algorithm naturally aligns to them, even if they are not explicitly marked as features.

using the same strategy proposed in [Jakob et al. 2015] — Extending this algorithm to work in a distributed system is a challenging and exciting research direction.

Similarly to the original IM algorithm, the number of position singularities increases with the target density. The curl-correction step proposed in [Sokolov et al. 2016] could be applied to slightly decrease the number of singularities, but it is not as effective as its 2D counterparts [Ray et al. 2006; Diamanti et al. 2015]. Successfully tackling this problem would increase the regularity of our meshes, at the price of isotropy.

Our approach might introduce inverted, self-intersecting, or collapsed elements. However, they rarely appear in our experiments (Table 1). Our results are obtained without doing any further mesh improvement, which could likely resolve these rare cases. Differently from the 2D case, where it is straightforward to convert hybrid meshes to pure quadrilateral with a subdivision step, this is not the case for 3D.

We believe that automatic generation of hex-dominant meshes will play a key role in the nearby future: adaptive spline technology is quickly advancing [Sederberg et al. 2003; Giannelli et al. 2012; Dokken et al. 2013; Kang et al. 2015], and it is likely that a general solution to construct higher-order isogeometric basis will be soon discovered. Such a technology, when paired with our algorithm, would allow to do simulations and analysis sidestepping the expensive and time consuming manual meshing process that is currently used both in industry and in academia.

ACKNOWLEDGMENTS

We thank Bruno Levy for providing the mechanical models for our experiments and comparisons, Yixin Hu for preparing video clips, Zhongshi Jiang for helping with the statistics, Alexandra Trif for coding a preliminary prototype, and Olga Sorkine-Hornung for the insightful discussions. The elephant, fertility, and rocker-arm models are part of the AIM@SHAPE Shape Repository. This work was supported in part by the NSF CAREER award 1652515 and by the MIUR project DSurf.

REFERENCES

2016. AMPS. http://www.ampstech.com/ampstech/Asp/Solid.asp. (2016). Accessed: 2016-09-15.
2016. ANSYSTurbogrid. http://www.ansys.com/Products/Fluids/ANSYS-TurboGrid/ANSYS-TurboGrid-Features#1. (2016). Accessed: 2016-09-15.
2016. Apex. http://www.mscapex.com/apex-modeler/. (2016). Accessed: 2016-09-15.
2016. Autodesk Simulation-Mechanical. http://www.autodesk.com/products/simulation-mechanical/features/all/gallery-view. (2016). Accessed: 2016-09-15.
2016. BETA CAE. http://www.beta-cae.com/ansa.htm. (2016). Accessed: 2016-09-15.
2016. Bolt. http://www.csimsoft.com/boltoverview. (2016). Accessed: 2016-09-03.
2016. Cart3D. http://people.nas.nasa.gov/~aftosmis/cart3d/. (2016). Accessed: 2016-09-15.
2016. CUBIT. http://cubit.sandia.gov/. (2016). Accessed: 2016-09-15.
2016. Harpoon. http://www.sharc.co.uk/index.htm. (2016). Accessed: 2016-09-15.
2016. Hexotic. https://www.rocq.inria.fr/gamma/gamma/Membres/CIPD/Loic.Marechal/Research/Hexotic.html. (2016). Accessed: 2016-09-15.
2016. HEXPress. http://www.numeca.com/en/products/automeshtm/hexpresstm. (2016). Accessed: 2016-09-15.
2016. HyperMesh. http://www.altairhyperworks.com/product/HyperMesh. (2016). Accessed: 2016-09-15.

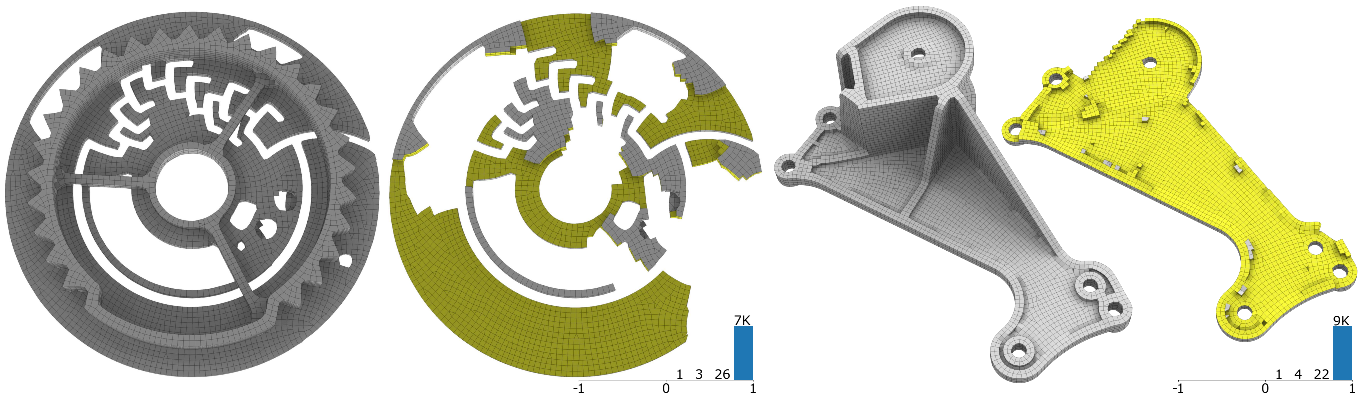


Fig. 15. Two extremely challenging mechanical objects, containing multiple holes and sharp features.

2016. Kubrix. http://www.itascacg.com/software/kubrix. (2016). Accessed: 2016-09-15.

2016. LBIE. http://www.cs.utexas.edu/~bajaj/cvc/software/LBIE.shtml. (2016). Accessed: 2016-09-15.

2016a. MeshGems. http://meshgems.com/volume-meshing-meshgems-hexa.html. (2016). Accessed: 2016-09-15.

2016b. MeshGemsHybrid. http://meshgems.com/volume-meshing-meshgems-hybrid.html. (2016). Accessed: 2016-09-15.

2016. PAMGEN. https://trilinos.org/packages/pamgen/. (2016). Accessed: 2016-09-15.

2016. SiemenPLM. https://www.plm.automation.siemens.com/en_us/products/lms/virtual-lab/structures/meshing.shtml. (2016). Accessed: 2016-09-15.

2016. TexMesher. http://texmesher.com/tex.html. (2016). Accessed: 2016-09-15.

2016. Trelis. http://www.csimsoft.com/trelis.jsp. (2016). Accessed: 2016-09-15.

2016. XBX. http://texmesher.com/kbx.html. (2016). Accessed: 2016-09-15.

Steven E. Benzley, Ernest Perry, Karl Merkley, Brett Clark, and Greg Sjaardema. 1995. A comparison of all hexagonal and all tetrahedral finite element meshes for elastic and elasto-plastic analysis. In *In Proceedings, 4th International Meshing Roundtable*. 179–191.

J.E. Bishop. 2014. A displacement-based finite element formulation for general polyhedra using harmonic shape functions. *Internat. J. Numer. Methods Engrg.* 97, 1 (2014), 1–31.

David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. 2013. Quad-Mesh Generation and Processing: A Survey. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 51–76.

David Bommes, Henrik Zimmer, and Leif Kobbelt. 2009. Mixed-integer Quadrangulation. *ACM Trans. Graph.* 28, 3 (July 2009), 77:1–77:10.

M. Brewer, L. Diachin, P. Knupp, T. Leurent, and D. Melander. 2003. The Mesquite Mesh Quality Improvement Toolkit. In *Proc. of the 12th International Meshing Roundtable*. 239–250.

A. O. Cifuentes and A. Kalbag. 1992. A performance study of tetrahedral and hexahedral elements in 3-D finite element structural analysis. *Finite Elements in Analysis and Design* 12, 3–4 (1992), 313–318.

Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Integrable PolyVector Fields. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)* 34, 4 (2015).

Tor Dokken, Tom Lyche, and Kjell Fredrik Pettersen. 2013. Polynomial Splines over Locally Refined Box-partitions. *Comput. Aided Geom. Des.* 30, 3 (March 2013), 331–356.

Randall Dougherty, Vance Faber, and Michael Murphy. 2004. Unflippable Tetrahedral Complexes. *Discrete & Computational Geometry* 32, 3 (2004), 309–315.

Xianzhong Fang, Weiwei Xu, Hujun Bao, and Jin Huang. 2016. All-hex meshing using closed-form induced polycube. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 124.

Xiaoming Fu, Chongyang Bai, and Yang Liu. 2016. Efficient Volumetric PolyCube-Map Construction. *Computer Graphics Forum (Pacific Graphics)* 35, 7 (2016).

Carlotta Giannelli, Bert JüTtler, and Hendrik Speleers. 2012. THB-splines: The Truncated Basis for Hierarchical Splines. *Comput. Aided Geom. Des.* 29, 7 (Oct. 2012), 485–498.

James Gregson, Alla Sheffer, and Eugene Zhang. 2011. All-Hex Mesh Generation via Volumetric PolyCube Deformation. *CGF* 30, 5 (2011), 1407–1416.

Jin Huang, Tengfei Jiang, Zeyun Shi, Yiying Tong, Hujun Bao, and Mathieu Desbrun. 2014. L1-based Construction of Polycube Maps from Complex Shapes. *ACM Trans. Graph.* 33, 3 (2014), 25:1–25:11.

Jin Huang, Yiying Tong, Hongyu Wei, and Hujun Bao. 2011. Boundary Aligned Smooth 3D Cross-frame Field. *ACM Trans. Graph.* 30, 6 (Dec. 2011), 143:1–143:8.

Yasushi Ito, Alan M. Shih, and Bharat K. Soni. 2009. Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates. *Int. J. Numer. Meth. Engng* 77 (2009), 1809–1833.

Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant Field-aligned Meshes. *ACM Trans. Graph.* 34, 6 (Oct. 2015), 189:1–189:15.

Tengfei Jiang, Jin Huang, Yiying Tong Yuanzhen Wang, and Hujun Bao. 2014. Frame Field Singularity Correction for Automatic Hexahedralization. *IEEE TVCG* 20, 8 (Aug. 2014), 1189–1199.

Hongmei Kang, Jinlan Xu, Falai Chen, and Jiansong Deng. 2015. A New Basis for PHT-splines. *Graph. Models* 82, C (Nov. 2015), 149–159.

Bo Li, Xin Li, Kexiang Wang, and Hong Qin. 2013. Surface Mesh to Volumetric Spline Conversion with Generalized Poly-cubes. *IEEE TVCG* 19, 9 (2013), 1539–1551.

Yufei Li, Yang Liu, Weiwei Xu, Wenping Wang, and Baining Guo. 2012. All-hex Meshing Using Singularity-restricted Field. *ACM Trans. Graph.* 31, 6 (Nov. 2012), 177:1–177:11.

Marco Livesu, Alessandro Muntoni, Enrico Puppo, and Riccardo Scateni. 2016. Skeleton-driven Adaptive Hexahedral Meshing of Tubular Shapes. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 237–246.

Marco Livesu, Nicholas Vining, Alla Sheffer, James Gregson, and Riccardo Scateni. 2013. PolyCut: monotone graph-cuts for PolyCube base-complex construction. *ACM Trans. Graph.* 32, 6 (2013), 171.

Max Lyon, David Bommes, and Leif Kobbelt. 2016. HexEx: Robust Hexahedral Mesh Extraction. *ACM Trans. Graph.* 35, 4, Article 123 (July 2016), 11 pages.

Loïc Maréchal. 2009. Advances in octree-based all-hexahedral mesh generation: handling sharp features. In *proceedings of the 18th International Meshing Roundtable*. Springer, 65–84.

Sebastian Martin, Peter Kaufmann, Mario Botsch, Martin Wicke, and Markus Gross. 2008. Polyhedral Finite Elements Using Harmonic Basis Functions. In *Proceedings of the Symposium on Geometry Processing (SGP '08)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 1521–1529.

Matthias Nieser, Ulrich Reitebuch, and Konrad Polthier. 2011. CubeCover- Parameterization of 3D Volumes. *CGF* 30, 5 (2011), 1397–1406.

Steven J. Owen and Sunil Saigal. 2000. H-Morph: an indirect approach to advancing front hex meshing. *Internat. J. Numer. Methods Engrg.* 49, 1-2 (2000), 289–312.

Nicolas Ray, Wan Chiu Li, Bruno Lévy, Alla Sheffer, and Pierre Alliez. 2006. Periodic Global Parameterization. *ACM Trans. Graph.* 25, 4 (Oct. 2006), 1460–1485.

Nicolas Ray and Dmitry Sokolov. 2015. On Smooth 3D Frame Field Design. *CoRR* abs/1507.03351 (2015).

Nicolas Ray, Dmitry Sokolov, and Bruno Lévy. 2016. Practical 3D Frame Field Generation. *ACM Trans. Graph.* 35, 6, Article 233 (Nov. 2016), 9 pages.

Maxence Reberol and Bruno Lévy. 2016. Low-order continuous finite element spaces on hybrid non-conforming hexahedral-tetrahedral meshes. *CoRR* abs/1605.02626 (2016). http://arxiv.org/abs/1605.02626

Thomas W. Sederberg, Jianmin Zheng, Almaz Bakenov, and Ahmad Nasri. 2003. T-splines and T-NURCCs. *ACM Trans. Graph.* 22, 3 (July 2003), 477–484.

Jason F. Shepherd and Chris R. Johnson. 2008. Hexahedral Mesh Generation Constraints. *Eng. with Comput.* 24, 3 (June 2008), 195–213.

Hang Si. 2015. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw.* 41, 2, Article 11 (Feb. 2015), 36 pages.

Dmitry Sokolov, Nicolas Ray, Lionel Untereiner, and Bruno Lévy. 2016. Hexahedral-Dominant Meshing. *ACM Trans. Graph.* 35, 5 (June 2016), 157:1–157:23.

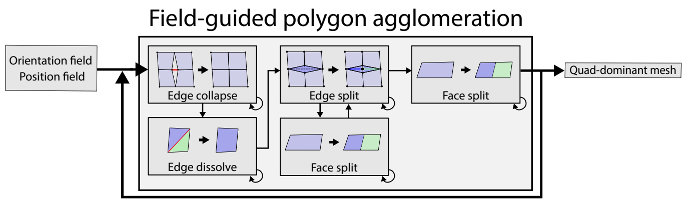


Fig. 16. The **field-guided agglomeration phase** converts the fields into a stream of local transformations applied to the input mesh, each preserving both manifold structure and genus. Self-loops indicate repeated application of an operation over the entire mesh until no more qualifying elements are left. Certain operations occur multiple times—once as a sweep over the mesh (with self-loop), and once as a local step triggered by another operation (an edge split e.g. triggers a face split).

Yi Su, KH Lee, and A Senthil Kumar. 2004. Automatic hexahedral mesh generation for multi-domain composite models using a hybrid projective grid-based method. *Computer-Aided Design* 36, 3 (2004), 203–215.

Srinivas C. Tadepalli, Ahmet Erdemir, and Peter R. Cavanagh. 2010. A Comparison of the Performance of Hexahedral and Tetrahedral Elements in Finite Element Models of the Foot. In *ASME 2010 Summer Bioengineering Conference, Parts A and B*. Naples, Florida, USA.

Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. 2016. Directional Field Synthesis, Design, and Processing. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 545–572.

Soji Yamakawa and Kenji Shimada. 2003. Fully-automated hex-dominant mesh generation with directionality control via packing rectangular solid cells. *Internat. J. Numer. Methods Engrg.* 57, 15 (2003), 2099–2129.

Hongmei Zhang, Guoqun Zhao, and Xinwu Ma. 2007. Adaptive generation of hexahedral element mesh using an improved grid-based method. *Computer-Aided Design* 39, 10 (2007), 914–928.

Yongjie Zhang and Chandrajit Bajaj. 2006. Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. *Computer methods in applied mechanics and engineering* 195, 9 (2006), 942–960.

Y. J. Zhang, X. Liang, and Guoliang Xu. 2013. A robust 2-refinement algorithm in octree or rhombic dodecahedral tree based all-hexahedral mesh generation. *Computer Methods in Applied Mechanics and Engineering* 256 (2013), 88–100.

APPENDIX - 2D EXTRACTION VIA POLYGON AGGLOMERATION

We can adapt our extraction algorithm to convert any manifold triangle mesh into a quad-dominant mesh while maintaining the genus, number of holes and manifoldness of the input model. Similarly to our 3D extraction, all edges can be classified into transient, persistent, diagonal, and long edges. To remove non-persistent edges, the extraction algorithm alternates between *coarsening* and *splitting* operators. As shown in Figure 16, the coarsening operators are edge collapses (to remove transient edges) and edge dissolves (to remove diagonal edges). The splitting operators are edge splits (to remove long edges) and face splits (to prevent large polygons). These operators are sequentially executed and iterated until no more elements can be collapsed or split.