Eurographics Symposium on Geometry Processing 2019
D. Bommes and H. Huang
(Guest Editors)

Volume 38 (2019), Number 5

Feature Preserving Octree-Based Hexahedral Meshing

Xifeng Gao[1,2], Hanxiao Shen[2], and Daniele Panozzo[2]

[1]Computer Science, Florida State University, USA
[2]Courant Institute of Mathematical Sciences, New York Unversity, USA

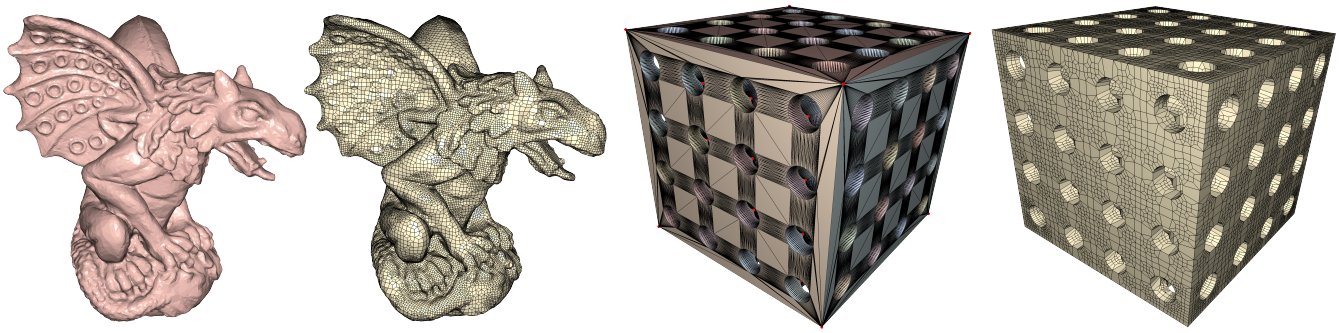


Figure 1: *We propose an automatic algorithm to convert a closed triangle mesh with complex features, e.g. high frequency (left) and sharp features (right), into a pure hexahedral mesh with a bounded Hausdorff distance from the input.*

Abstract

*We propose an octree-based algorithm to tessellate the interior of a closed surface with hexahedral cells. The generated hexahedral mesh (1) explicitly preserves sharp features of the original input, (2) has a maximal, user-controlled distance deviation from the input surface, (3) is composed of elements with only positive scaled jacobians (measured by the eight corners of a hex [SEK*07]), and (4) does not have self-intersections.*

We attempt to achieve these goals by proposing a novel pipeline to create an initial pure hexahedral mesh from an octree structure, taking advantage of recent developments in the generation of locally injective 3D parametrizations to warp the octree boundary to conform to the input surface. Sharp features in the input are bijectively mapped to poly-lines in the output and preserved by the deformation, which takes advantage of a scaffold mesh to prevent local and global intersections.

The robustness of our technique is experimentally validated by batch processing a large collection of organic and CAD models, without any manual cleanup or parameter tuning. All results including mesh data and statistics in the paper are provided in the additional material. The open-source implementation will be made available online to foster further research in this direction.

1. Introduction

Hexahedral meshes are a popular volumetric discretization widely used for solving partial differential equations in computer graphics and engineering applications. They are favored over unstructured meshes due to their support for higher-order spline constructions, which are popular in the isogeometric analysis community [HCB05] due to their slightly higher accuracy [SHG*19]. They are also one of the default mesh types used in many existing open-source and commercial computer-aided engineering analysis packages, e.g. PolyFEM [SDG*19], FEniCS [ABH*15], libMesh [KPSC06], MFEM [cod], Deal.II [AAB*18], ANSYS [ANS19], and ABAQUS [ABA19].

However, the automatic generation of such meshes is a challenging problem, for which robust solutions are still elusive. Most recent research efforts were spent in developing polycube or field-aligned methods, which have the potential to generate high-quality meshes with elements on the boundary nicely following sharp features. However, these algorithms cannot guarantee to produce an all-hex mesh, they cannot ensure feature preservation, and often contain inverted elements and globally intersected regions, requiring post-processing steps. The lack of robustness and guarantees hinders their use in automated analysis pipelines.

We introduce a new algorithm that attempts to produce a pure hex-mesh without self-intersections and with positive scaled Jaco-

bians (measured by the eight corners of a hex [SEK*07]), while capturing both the high frequency features on organic shapes and the sharp features on CAD models up to a user-specified geometric distance bound (Figure 1). Starting from a 3D, self-intersection free, closed surface, our method first creates a pure hex-mesh that has no inversions and intersections, and then deforms it to minimize the distance to the input surface by taking advantage of recent advancements in the generation of volumetric parametrizations. Local self-intersections are prevented using a flip-avoiding deformation [RPPSH17], while global intersections are prevented using a scaffold mesh [JSP17].

We generate an initial pure hex-mesh by revisiting classical results in octree mesh generation, one of the oldest hex-meshing techniques, that was originally introduced by [YS84], and is currently used in commercial hexahedral meshing tools [Mes18, Bol18]. While similarly relying on the adaptivity of an octree to use as few as possible elements to roughly approximate the volume of the input, we propose a simple yet effective algorithm to transform the octree into a pure hex mesh based on the observation that the dual-mesh of a non-conforming octree is conforming and made of a small set of polyhedral types, which can always be merged together into an adaptive and inversion-free pure hex mesh.

We then build a topologically bijective feature graph map between the boundary of the hex-mesh and the input, and employ locally injective volumetric deformations to align the features. During this process, we introduce a novel variational padding technique to add additional hexahedra to the mesh for better feature preservation.

For an input surface with annotated sharp features and a user-specified distance bound, a feature-aligned hex-mesh can be robustly generated by combining the two steps above. Since the edge length of the final mesh is not necessarily related to the given distance bound we aim to achieve, our algorithm iterates the two steps by varying the target edge length until it meets the stopping criteria.

We experimentally show the robustness and effectiveness of our method by batch-processing hundreds of models and producing meshes that have fewer elements and with a higher minimal element quality than a commercial, state of the art hexahedral meshing algorithm [Mar09, Mar16], included in the software package [Mes18].

We attach a reference implementation of our algorithm, with scripts to reproduce all results in the paper, in addition to the data (input/output) for all figures and for our dataset composed of 202 shapes. Both the implementation and the dataset will be released in the public domain to foster research in this area.

2. Related Work

We review the hex-meshing literature, with a focus on the space partitioning methods which are closely related to our contribution. We also review the literature in volumetric parametrization and deformation, since they are used extensively in our algorithm.

Paving and Sweeping. The first generation of hex-meshing algorithms was based on paving (i.e., inserting layers of cubes on a quad mesh boundary) and sweeping (i.e., extruding a quad mesh or a skeleton) [SJ08, GMD*16, LMPS16]. While ideal to produce high-quality elements in the vicinity of the boundary, the quality of the generated elements is poor in the interior where the multiple advancing fronts meet. They also require a high-quality quad mesh to start from and are extremely challenging to implement robustly due to the large number of special cases to handle [San16]. In contrast, our technique does not require a clean quad mesh or skeleton, it is robust and fully automatic, and can handle complex sharp features.

Spatial Partitioning. A hex-mesh can be trivially created by voxelizing the interior of a closed surface: the denser is the voxelization, the closer the geometric approximation will be. Starting from this regular lattice, many methods have been proposed [SB95, Sch96, SLSK04, ZB06a, ZZM07a, ZZM07b] to either project the vertices on the input surface or cut the elements intersecting it. Care needs to be taken to avoid inverted elements and to ensure that all the resulting elements will be hexahedra after the remeshing. The most popular approaches in this group use adaptive spatial partitioning to reduce the element count [SSW96, ZB06b, ISS09, Mar09, QZ10, EPOM11, ZLX13, EE14, OSE17, Mar16], further increasing the algorithmic complexity since adjacent octants will in general not conform to each other. These methods propose multiple templates to subdivide the octants for conforming transitions, but only a few [Mar09, Mar16] can robustly produce a hex-mesh for a clean input without introducing other kinds of elements. Another long lasting difficulty for these approaches is to accurately capture features on the input while maintaining the element quality satisfactory. Our algorithm uses an octree as a starting point, but ensures a pure hex output relying on a novel algorithm that uses its mesh dual (Section 3.2) and achieves feature preservation by warping the mesh using a locally injective deformation (Section 3.5). We provide an extensive comparison of our algorithm and the state-of-the-art, commercial implementation of [Mar09, Mar16] available in MeshGems [Mes18] in Section 4.

Polycube Deformation. A modern approach to hex-meshing uses polycube domains, i.e. the union of a set of cubes, to define the connectivity of the final mesh. In contrast with spatial partitioning, these methods strive to deform the surface, usually filled with tetrahedra, to generate an axis-aligned polycube [GSZ11, LVS*13, HJS*14, FXBH16, FBL16, LLWQ13, ZLL*18]. After the deformation, the polycube (which is a hex mesh by definition) is optionally subdivided and then mapped to the interior with the inverse of the deformation applied to the surface. The advantages of these methods is that they have superior singularity placement (assuming that the polycube is a good fit), but they have higher geometrical distortion, which can be ameliorated by an optional padding step [GSZ11, CAS*19], since they cannot introduce internal singularities. These methods are not robust due to two major drawbacks: (1) the volumetric deformation needs to exactly satisfy positional constraints and be bijective, a difficult geometrical problem for which no robust solutions currently exist [CSZ16] and (2) even if the map is bijective in the continuous sense, an extremely dense hex mesh might be required to avoid flips due to the piecewise-linear discretization of the hex mesh [CSZ16]. Our method takes inspiration from these approaches: we deform the dual of the octree mesh to fit the surface, using a method to prevent flipped elements. In case a solution could not be found by our parametrization algorithm, we locally refine until the map is sufficiently dense to admit a solution (Section 3.6). While we cannot prove that such a solution will al-

ways exist, we experimentally show that we are able to find one for hundreds of real-world models.

Field-Aligned Parametrization. Directional fields [VCD*16] have been very successful for quad meshing applications [BLP*12], and have been recently introduced in volumetric meshing algorithms [NRP11, HTWB11, LLX*12, JHW*14, SVB17, LZC*18]. They are used to guide a Poisson-based volumetric parametrization, which induces a hex-mesh if singularities are placed at integer locations and if the parametrization is locally injective [VCD*16]. An algorithm guaranteed to ensures these two properties is still elusive, and currently heuristics method are used to obtain these conditions in a weak sense [LBK16]. The robustness of these methods is not sufficiently high for practical applications, despite the high quality of the generated meshes when they succeed. Our approach produces meshes with inferior singularity placement, but it can do it robustly on hundred of models (Section 4).

Simplification and Quality Improvement. A large family of methods exist to simplify quad [DSSC08, DSC09a, DSC09b, TPC*10, BPP*10] and hexahedral meshes, using sheets collapse operations [BBS02, LS10] or altering the hexahedral duals [TK03, KLSO12]. The topological complexity can also be reduced using local operations [BLK11, TPP*11, MPKZ10, GDC15, GPW*17]. The geometrical quality of a hex-mesh can also be improved relocating its nodes without changing their connectivity [Knu00, Knu03, BDK*03, WSRRR*12, RGRS14, RGRS*15, LSVT15, XGC18]. All these methods are orthogonal to our contribution and could be used to post-processing our meshes for specific applications.

Hex-Dominant Meshing. Given the challenges in constructing hexahedral meshes with a high quality, low-element count, and good topological structure, a new family of methods has been introduced to generate hybrid meshes composed of a majority of hexahedral elements, but allowing a few general polyhedra [SRUL16, GJTP17]. These methods can robustly process complex CAD models, reliably creating high-quality meshes. Unfortunately, the use of these meshes requires the design of ad-hoc FEM methods [MKB*08, Bis14, RL16, SDG*18], which are not yet widely used by the graphics and engineering community. Our algorithm strives to fill the temporal gap needed for these hybrid approaches to become mainstream, providing a similar robustness but generating meshes that are directly usable in existing FEM pipelines.

Locally Injective Parametrization. The creation of discrete locally injective maps, i.e. parametrization without inverted elements, received a lot of attention in the last few years due to their wide applicability in geometry processing problems. Two main approaches exist: methods based on minimizing flip-preventing energies [HG00, SSGH01, SCOGL02, DMK03, SKPSH13, APL14, PL14, SS15, FLG15, KGL16, RPPSH17, SPSH*17, ZBK17] or methods to remove flips from existing discrete maps [Lip12, AL13, KABL15, FL16]. Our method relies on a variant of [RPPSH17] to fit the boundary of the hex mesh to the input surface and to add the padding layers. We selected this method due to its favorable scalability properties and availability of an open-source implementation, but other approaches could be used instead with minimal changes to our algorithm.

3. Method

Given (a) a manifold, watertight triangle mesh with annotated sharp features, (b) a small, user-specified tolerance ε, and (c) a desired edge length l, our algorithm attempts to produce an all-hexahedral mesh with the following properties: (1) the mesh has no non-manifold configurations, (2) all elements have positive scaled Jacobian [SEK*07], i.e. the Jacobian measured at the eight corners of a hexahedron are all positive, (3) its boundary is within ε distance from the input mesh, and (4) the boundary of the mesh has no self-intersections. We call a hex mesh satisfying these four properties *valid*.

Our fundamental principle of tackling this problem is to achieve the validity requirements one by one, and once a property is satisfied, it will never leave the valid space. For example, when we generate a manifold hex-mesh with only elements with positive scaled Jacobian (Sections 3.1-3.4), then its manifoldness and positive scaled Jacobian will be maintained throughout the deformation process for feature conforming. During the generation of the initial mesh, we also construct a scaffold [JSP17] surrounding that mesh to prevent any intersections from happening to our mesh throughout the rest of the pipeline. The feature alignment and geometric fidelity to the input are enforced by first constructing a topologically correct feature map, and then deforming the mesh to match the input using a locally injective volumetric parametrization (Section 3.5). Adaptive mesh refinement and padding is performed until the ε bound is satisfied (Section 3.6).

Pipeline. Figure 2 illustrates our pipeline in 2D. Starting from a closed, manifold triangular mesh that defines the domain to be meshed, we create an adaptive octree, with higher density in regions containing small geometric features (Section 3.1). The octree is then converted into a pure hexahedral mesh by computing its geometric dual and splitting/merging the non-hex cells (Section 3.2). At this step, we segment the mesh into inside and outside and treat the outside mesh as a scaffold mesh. The sharp features on the input mesh are then topologically mapped to the boundary of the hex mesh (Section 3.3), and both the mesh and the scaffold are padded both on the surface and around sharp features (Section 3.4). Finally, the hexahedral mesh is deformed using a locally injective map to match the geometry of the input surface (Section 3.5). Additional refinement might be necessary to ensure the topological bijective feature mapping (Section 3.3) or to satisfy the ε distance bound (Section 3.6).

3.1. Octree Generation

We construct an adaptive octree covering the input surface domain, with the maximum edge length not exceeding the target edge length l. Starting from the bounding box of the input, we recursively split its cells if their boundary intersects the input surface, or if they contain a vertex of the input mesh. To ensure a conforming pure hex-mesh with a "smooth" size transition, we also force additional splits to ensure (1) balancing, i.e. the difference between any two neighboring octants is less than or equal to one, and (2) pairing, i.e. if an octant's child has been split, then its siblings having the same octant as their parent should be split as well [Mar09].

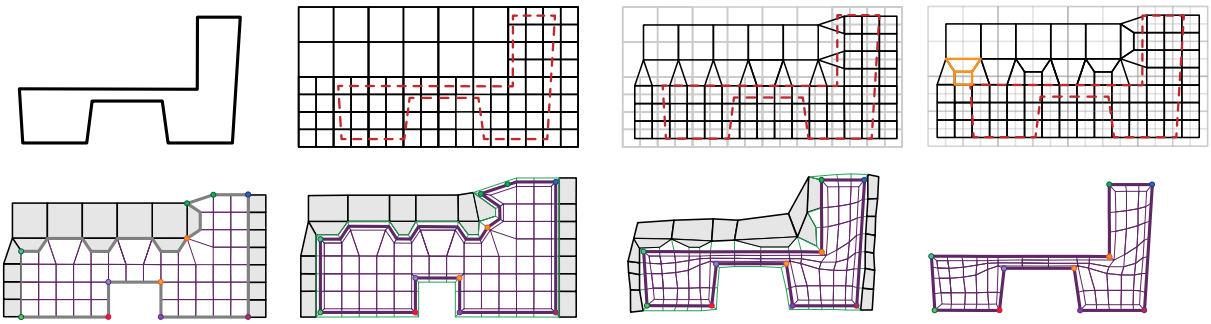


Figure 2: *2D illustration of our algorithm. Top row: adaptive quadtree constructed from the input, dual of the quadtree, and quadrilateral (quad-) mesh and scaffold mesh after splitting all cells. Bottom row: feature topological matching, padding of both the target mesh and the scaffold, mesh deformation to fit the input, and the output pure quad-mesh.*

3.2. Conversion to a Pure Hexahedral Mesh

Differences from [Mar09, Mar16]. The constructed octree can be viewed either as a non-conforming hex-mesh, with hanging nodes between adjacent octants with different sizes, or as a conforming hybrid mesh, composed of general polyhedral elements. The first point of view led to a wide array of techniques to explicitly split the cells containing hanging nodes into an all-hex mesh [SB95, ZB06a, Mar09, ISS09, Mar16]. Our construction takes advantage of the second point of view, i.e. it considers the octree structure as a hybrid mesh, and exploits the fact that its dual is a different hybrid mesh composed predominantly of hexahedral cells, based on which we introduce an easy all-hex hex-mesh conversion scheme as described below.

Our Approach. We first discuss the 2D case, i.e. the conversion of a quadtree into an all-quad mesh $\mathcal{M}$, and then show how to extend this construction to an octree.

2D. Thanks to the balancing rule, triangles in the dual quadtree are hanging nodes in the quadtree (Figure 3, left). Due to the pairing rule we used in the quadtree construction (Section 3.1) hanging nodes on the same axis *always* appear in pairs, and each pair of triangles form a *trapezoid* (Figure 3, left). Trapezoids are always isolated from each other, i.e. all the elements around them are quads. We can transform each trapezoid into a set of quads by splitting the interior quad with two additional vertices (Figure 3, right). This operation is purely local: since it does not modify the boundary and there is no propagation to other regions of the mesh. By executing this local operation on all trapezoids in the dual quadtree mesh, we obtain a pure quad mesh (Figure 2, top-right).

3D. Similarly, hanging nodes in an octree create pyramids in the dual octree (Figure 4, left). Thanks to the pairing rule we used in the octree construction (Section 3.1), four hanging nodes on the same plane *always* appear at the same time, forming a special configuration containing 4 pyramids, 1 cube, and 4 triangular prisms which we call a *3D frustum* (Figure 4, middle). The 3D frustum can be turned into a pure hex mesh composed of 13 hexahedra by adding 12 vertices, and reconnecting them into the configuration depicted in Figure 4, right. Differently from the 2D case, the nodes have to be inserted in the boundary, creating polyhedral cells around the splitted frustum. Surprisingly, there are only three types of polyhedral

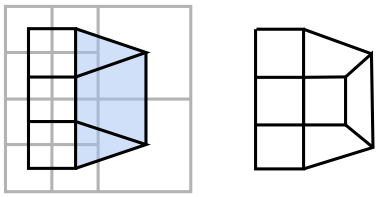


Figure 3: *The dual of a hanging node is a triangle. Triangles always appear in pairs with a quad in-between: we call this configuration a trapezoid (left, light blue). A trapezoid can be decomposed into a set of pure quads with a local splitting rule (right).*

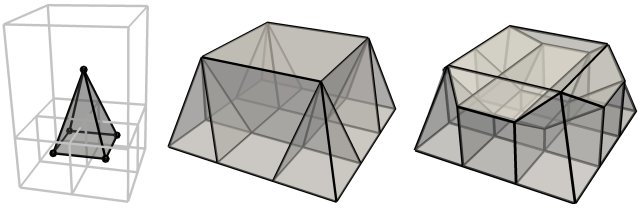


Figure 4: *The dual of a hanging node is a pyramid (left), and pyramids always appears in group of four (middle) due to the octree balancing rules. We call this configuration a* 3D frustum. *We convert each frustum into a collection of hexahedral elements by applying the splitting rule depicted on the right.*

elements that are generated after splitting all 3D frustums with the previous pattern: the polyhedra sandwitched between two frustums at the same octree level (Figure 5, top row), and those touching frustums of different levels (Figure 5, middle and bottom rows). By applying the corresponding splitting rule to each polyhedra as illustrated in Figure 5, we obtain a pure hexahedral mesh.

3.3. Feature Matching

We assume that the input surface is equipped with sharp features, each of which is defined as a collection of edges, when this information is missing, the input is regarded as an organic model with no sharp features. For the meshes in our CAD dataset, we extracted

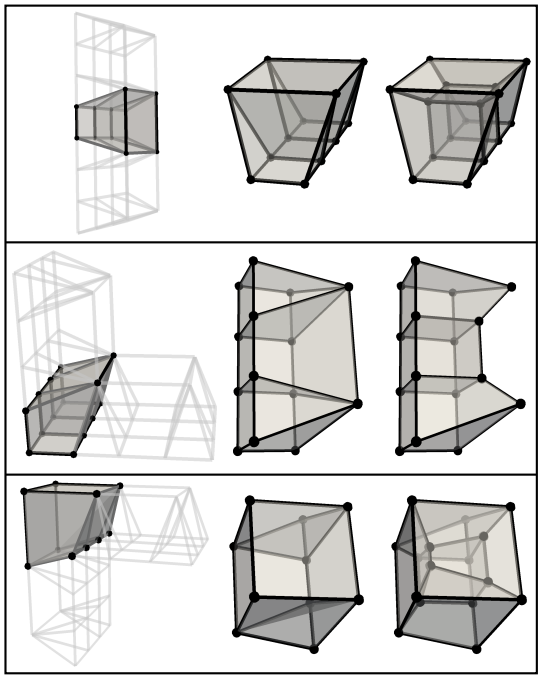


Figure 5: *Splitting rule for polyhedral elements touching two frustums at the same level (top). The remaining polyhedral elements are split with the two rules in the middle and bottom rows.*

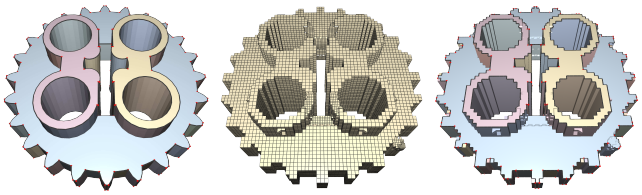


Figure 6: *Features are bijectively transferred from the input (left) onto the hex-mesh after all-hex conversion (middle and right).*

the sharp features automatically using a dihedral angle threshold of 140 degrees, and then manually fix them if some features are missed or incomplete. The sharp feature annotations are provided in the additional material.

Partitioning. As shown in Figure 6 (left), the annotations induce a partitioning of the input surface into feature *corners* (where sharp turn happens on feature lines), *lines* (collection of segments), and *patches* (collection of connected triangles enclosed by feature lines). Our goal is to transfer all these entities onto the surface of the all-hex mesh generated in Section 3.2 (Figure 6 right).

The partitioning might contain *invalid* corner configurations that either are impossible to represent with an octree mesh (valence higher than 6) or will lead to very high geometric distortion (angles smaller than 30 degrees). We detect these configurations in the input annotations and tag them as invalid: our algorithm will still ensure validity (positive scaled Jacobian, within an ε distance from

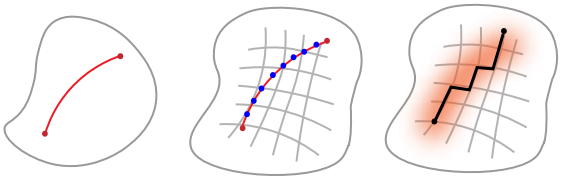


Figure 7: *To transfer a feature we first map its two endpoints (left), sample and map the feature itself to build a distance field (middle), and finally trace the line on the target domain using weighted Dijkstra (right).*

the input, and free of self-intersections) of the mesh around these regions, but it will not enforce exact representation of these features.

Transfer. The transfer of the features onto the surface of the hex-mesh could be solved by computing a bijective map between the source triangle mesh and the target quad-mesh, i.e. the boundary of the hex-mesh. Despite the recent advancements in cross parametrization techniques [SAPH04, KS04, APL14, APL15], a solution that supports arbitrary constraints, minimizes distortion, works robustly on hundreds of (possibly low quality) surface meshes, and ensures that the bilinear map of every mapped quad has positive Jacobian is still elusive. The last point is particularly important and specific for our setting, since it is required to ensure that the corresponding hex elements have positive scaled Jacobian. We propose a robust technique to achieve a bijective feature map by first topologically transferring the feature corners, lines, and regions (this section), and then computing their geometry with a bijective spatial deformation (Section 3.5).

We initially map every feature corner to the closest (spatially) vertex of the quad-mesh. We then map the feature lines sequentially, using the following procedure: (1) sample uniformly each feature lines on the input mesh (using $l/2$ as sampling distance) and map all the samples to the closest point on the edges of the quad mesh (Figure 7, middle), (2) define a distance field on the quad mesh, assigning to every vertex the distance to the closest mapped sample, and (3) we trace the feature line using Dijkstra's algorithm, using the distance field as weight (Figure 7, right). Every time a feature line is added, we consider it as a cut for subsequent feature tracing, ensuring that no intersections can be introduced.

With the corners and curves being bijectively mapped, we extract the patches segmented by the feature graph for both the input and the quad mesh. For each patch, we search for its correspondence by comparing the composed nodes and curves. Furthermore, for the found patch in the quad-mesh, we perform again the search to make sure the corresponding patch in the input is the same one, ensuring the bijectivity of patches: every patch of the input has a correspondence in the quad mesh and vice versa.

Bijective Map. A bijective map will not exist if the topology of the source and target domain is different or if the resolution of the target domain is insufficient. We address both cases with an adaptive refinement strategy. If the topology of the two domains is different, we split all the octree cells that intersect the patch in the input and repeat the procedure. If Dijkstra's algorithm fails to find a path, we

split all the octree cells that are intersecting the corresponding input feature line. While this procedure will eventually lead to a valid solution (since due to our balancing rules, the splits will propagate, increasing the resolution of the entire octree), it might introduce a lot of unnecessary elements. In practice, we found it to be effective and always succeed in only 1 or 2 iterations.

Remark on Feature Annotation. For many meshes converted from the CAD format, our simple, angle-based thresholding is sufficient to identify features. However, there are occasional issues mostly due to problems during the meshing fo the CAD surfaces which may require manual edit to correctly capture the feature lines. The most common cases are (1) features that are incomplete (for example, a ring which misses only one edge), (2) missing small features, and (3) spurious features that are geometrically close to real ones. In these cases, we manually fix them with an annotation tool that allows to add/remove triangle edges to the feature list.

3.4. Variational Padding and Untangling

The singularity distribution in the hexahedral mesh depends only on the octree adaptivity: it might be impossible to represent sharp features well without inserting additional singularities to compensate the geometric distortions to align the features and the input surface (Section 3.5). We propose to insert additional hex-mesh layers, introducing more singularities, around the surface and sharp features to alleviate such problem.

The challenge of adding additional hex elements remains in topologically maintaining the conformity between adjacent elements and geometrically preventing the newly inserted elements from inversion. This has been a notoriously difficult problem remaining unsolved [Mar16], since the insertion of new hex layers could be adjacent to an arbitrarily complicated surface. This essentially is equivalent to the unsolved untangling problem in 3D [Knu00].

Existing approaches tackle the geometric positioning of the new vertices by either simply extruding the surface along normal direction [SB95] or classifying the procedure into different configurations where each is handled by a mixture of complicated strategies [Mar16]. We propose a simple, general, and empirically robust solution by first, connectivity-wise, correctly inserting the new hex elements around the feature regions, and then computing a valid embedding using a variant of the algorithm proposed in [FL16, PTH*17], to untangle inversions presented in the hex-mesh.

Topological Padding. We distinguish features in a 3D model into two types: soft features that can be approximated by increasing the discretization resolution, e.g. the light red smooth curve in Figure 8, and hard features that have to be explicitly captured where refinement cannot help, e.g. the red straight line in Figure 9.

Soft features correspond to high frequency features that introduce concave regions. They are difficult for an underlining mesh resolution to capture, since the mesh has to be inversion-free limiting the space for positioning vertices. Figure 8 illustrates our solution in 2D. Given a mesh and a feature curve, an element (grey, Figure 8 left) with a concave corner is introduced in order to capture the concave part of the curve. By offsetting the polyline corresponding

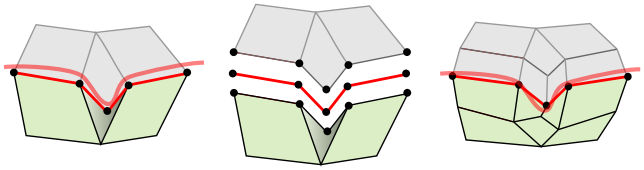


Figure 8: *A smooth curve (light red) is captured by a polyline (i.e. a set of red edges) of the mesh where a flipped element is introduced (left). Since the polyline partition the mesh into our target mesh (green) and the scaffold mesh (grey), by creating two padding layers around the polyline through offsetting it inward and outward, more degrees of freedom are given and the inverted element can be easily resolved.*

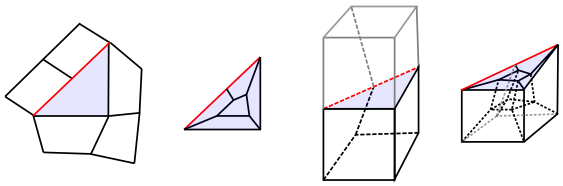
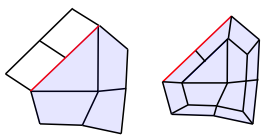


Figure 9: *Left: a degenerate element (blue) in 2D on a straight line feature (red) can be easily cured by inserting a ring of quads along the boundary of the degenerate element. Right: similarly, through the insertion of a layer of hexahedra along the boundary of those elements, the straight line feature can be precisely captured without introducing degenerates.*

to the feature (red, Figure 8 middle) to both sides (gray and green, Figure 8 middle) and connecting the corresponding vertices, the vertices near the concave region have more freedom to recover from the inversions (Figure 8 right). In 3D, this operation corresponds to offsetting the closed surface both inside and outside. We apply this strategy to all the models in our test, greatly reducing the need for refinement to satisfy the distance bound (Section 4).

Sharp features are points, curves, and surfaces that the user want to preserve precisely. An element on a sharp feature may degenerate when two or more of its edges are topologically labeled as features (blue, Figure 9 left). We resolve this issue by localizing the insertion to a small region. As illustrated in 2D in Figure 9 left, we can identify the degenerate quad element (blue) and insert a ring of elements by offsetting the boundary polyline of this quad element inward and connecting the corresponding vertices (Figure 9 second-left). This is equivalent to identifying the degenerate hex elements in 3D (Figure 9 second-right) and offsetting the boundary surface of the local mesh composing of those hex elements inwards (Figure 9 right shows the bottom part of the mesh after offsetting). To reduce the distortion, the degenerate region is expanded by adding one ring of elements. The inset shows such an example in 2D.



Untangling. After topologically inserting new elements, the hex-mesh may contain inversions. We isolate the inverted elements as local regions, expand the local regions by two rings of hex elements

for efficiency, and optimize the hex-mesh composed by these regions only. The challenge in our setting is that the mesh we wish to optimize is composed of hexahedra instead of tetrahedra. We propose a novel formulation combining the stitching idea [FL16] with the distortion energy introduced in [GPW*17]. The resulting energy can be efficiently minimized using the solver proposed in [RPPSH17], to obtain a general and effective mechanism to pad hex meshes.

Distortion Energy Term. To measure the distortion of the hex elements, we use the energy proposed in [GPW*17], which we summarize here to make the paper self-contained. We measure the distortion $\mathcal{D}_h(\mathbf{V})$ ($\mathbf{V}$ is a matrix storing the vertex positions of the entire mesh) of each hexahedron h as the sum of the distortions of its decomposition into eight tetrahedra t, one for each corner:

$$E_D(\mathbf{V}) = \sum_{h\in\mathcal{H}} \mathcal{D}_h(\mathbf{V}) = \sum_{h\in\mathcal{H}} \sum_{t\in h} \mathcal{D}_t(\mathbf{V}) \qquad (1)$$
$$\mathcal{D}_t(\mathbf{V}) = ||\mathbf{J}_t(\mathbf{V})||_F^2 + ||\mathbf{J}_t^{-1}(\mathbf{V})||_F^2$$

where $\mathcal{H}$ is the set of all hexahedra, and $\mathbf{J}_t$ is the Jacobian of the transformation of the tetrahedron t from its rest shape which is an *ideal* tetrahedron of the same volume as t. The difference between the current and ideal tetrahedron $\mathcal{D}_t(\mathbf{V})$ is measured with the symmetric Dirichlet energy [SS15]. We require the ideal shape to be a right-angled tetrahedron: we compute the target shape as the tetrahedron of the corresponding corner of a cube, which has the same volume as h.

Stitching Energy Term. Initially, all the hex elements are dissembled into independent cubes and therefore, all the vertices except those on the boundary have multiple copies. This term encourages the multiple copies of each vertex to move into the same position, allowing us to merge them into a single vertex. We define a cluster c for each set of vertices that needs to be merged, and add a quadratic penalty term that encourages each cluster to contract:

$$E_S(\mathbf{V}) = \sum_{c\in\mathcal{C}} \sum_{p\in c} (\| p(\mathbf{V}) - \frac{1}{|c|} \sum_{p\in c} p(\mathbf{V}) \|_2^2) \qquad (2)$$

where c is a vertex clusters, and p is the position of one of the vertices in the cluster. Ideally, we would like E_S to be exactly zero, in practice we try to collapse each cluster after every optimization iteration, and we accept the merge if it does not introduce elements with negative Jacobian.

Solver. The geometry of the local regions (the vertices of the subset of the mesh are denoted as $\mathbf{V}'$) is optimized by minimizing the following energy:

$$E_F(\mathbf{V}') = E_D(\mathbf{V}') + \lambda_S E_S(\mathbf{V}'), \qquad (3)$$

where the weighting strategy is adapted from [FL16] to set as $\lambda_S^{update} = min(\lambda_{min} * max(\lambda_S \frac{E_D}{E_S}, 1), \lambda_{max})$ with $\lambda_S = 1$ as the initialization, $\lambda_{min} = 10^4$ and $\lambda_{max} = 10^{16}$.

Since our input is dissembled into a hex mesh consisting of valid hex elements only (i.e., all scaled Jacobians are positive), we use the SLIM solver [RPPSH17] to minimize the energies, which employs the flip-avoiding line search [SS15] to avoid flips on the 8 corner tetrahedra. Figure 10 demonstrates a 3D example by applying our algorithm on a pyramid.

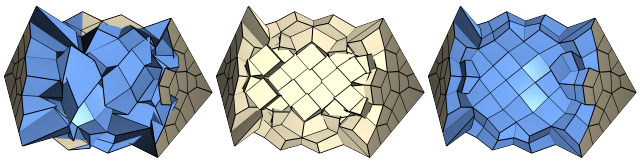


Figure 10: *Given a mesh containing inverted hexahedra (left), our untangling technique initializes an independent cube to each hexhedron (middle), and stitches them back while maintaining the inversion-free property of the mesh (right).*

3.5. Geometric Fitting

The hexahedral mesh is finally deformed to minimize the distance between its boundary and the input triangular mesh. This is achieved by computing a locally injective map, similarly to how we insert the padding, but with an additional set of energy terms encouraging the feature corners, lines, and patches on both meshes to align.

Feature Preservation. The feature is aligned by applying the energy term $E_B(V)$ proposed in [GPW*17]:

$$E_B(V) = \sum_{p\in corner} ||p(V) - \tilde{p}||^2 + \sum_{p\in line} ||p(\mathbf{V}) - \tilde{p} - a_l \vec{d}_l||^2 + \sum_{p\in\mathcal{M}} ||\vec{n}\cdot(p(\mathbf{V}) - \tilde{p})||^2 \qquad (4)$$

where $\tilde{p}$ is the closest surface position for each boundary vertex, $\vec{d}_l$ is the feature line tangent at $\tilde{p}$, a_l is an auxiliary variable added to the system for feature line constraints, and $\vec{n}$ is the normal at vertex $\tilde{p}$.

Note that, while all the features are topologically captured for all of our results, we do not enforce the preservation of the corner formed by two connecting feature curves if its angle is <30 degrees. This avoids close to degenerate hexahedra to be introduced close to these regions. If exact preservation is required, this additional filtering can be disabled, but this will consistently introduce badly shaped elements close to these corners.

Fitting Energy. The final mesh is computed by minimizing the following energy:

$$E(\mathbf{V}) = E_D(\mathbf{V}) + \lambda_B E_B(\mathbf{V}), \qquad (5)$$

where the same weighting strategy is used by setting as $\lambda_B^{update} = min(\lambda_{min} * max(\lambda_B \frac{E_D}{E_B}, 1), \lambda_{max})$ with $\lambda_B = 50$ at the beginning.

Remark on Scaffolding. As done in [JSP17], the outer boundary of the scaffold mesh should be fixed to guarantee that both the scaffold and the target meshes contain no self-intersections during deformation, however, we experimentally found that letting the scaffold freely deform in 3D leads to a much easier and faster convergence of the geometric fitting to satisfy the feature alignment and the Hausdorff distance bound. Therefore, we freeze the scaffold boundary only when we detect self-intersections which never happens for all of our experiments.

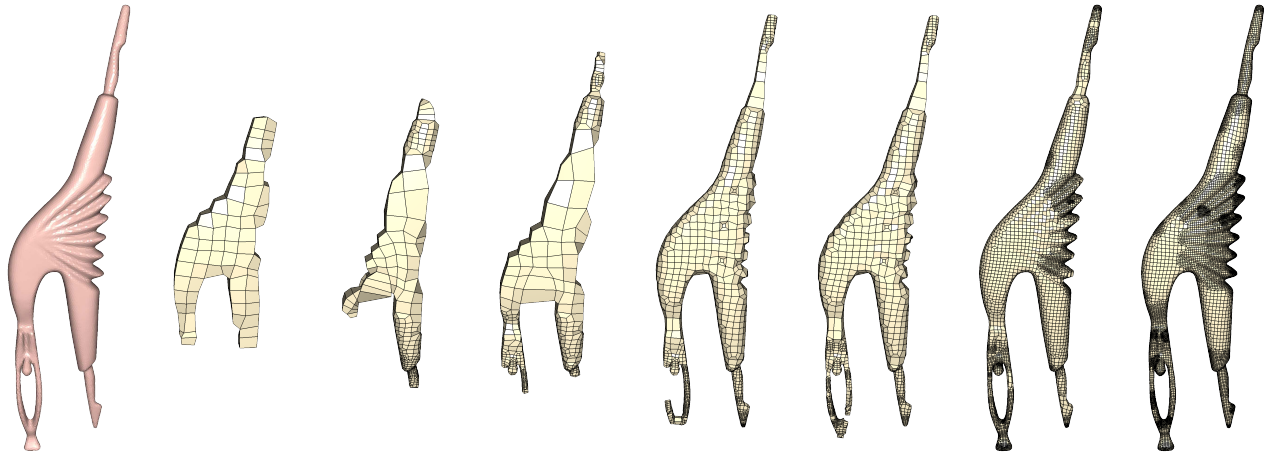


Figure 11: *The same model is meshed using* $\varepsilon = 0.5$, 0.2,0.15, 0.1, 0.05, 0.005, *and* 0.001 *respectively. The corresponding ratios of the number of singular edges to all edges for the meshes are 140/1037, 944/4738, 1093/5891, 2616/15449, 2657/14989, 18454/99983, and 37380/207646, respectively.*

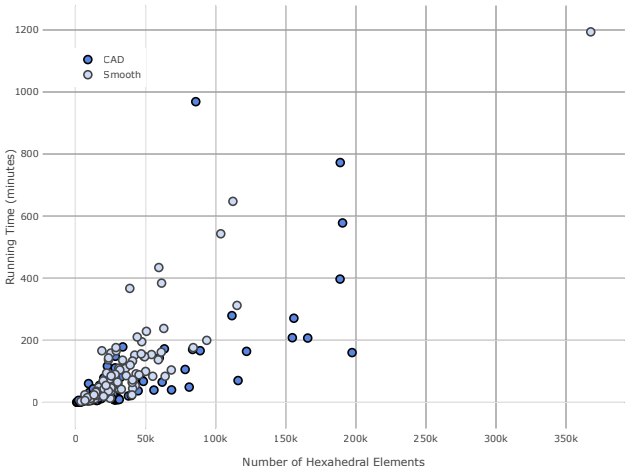


Figure 12: *The running time (vertical axis) of our algorithm is mostly dependent on the size of the* output *mesh (horizontal axis).*

3.6. Stopping Criteria

We minimize the energy for at most 30 minimization iterations (Eq. 5), and we stop earlier if two conditions are satisfied: (1) the mesh is within the user-specified ε maximum Hausdorff distance bound, and (2) the change of the average surface deviation from the input, measured using Metro [CRS03], between two consecutive iterations is small enough (default is 1%). While the first condition provides the geometric fidelity, the second ensures the geometric smoothness of the mesh on the boundary. Typically, the geometry of the mesh will become stable and smooth when the energy in Eq. 5 reaches a local minimum. While our optimization greatly reduces the energy in the first few iterations and the geometry is already smooth enough, it may require many iterations for the energy to reach the local minimum, we use the direct measure of the change of the surface geometry instead, i.e. condition (2), to stop the optimization earlier. If both conditions are satisfied (checked using [CRS03]), we output the resulting mesh. Otherwise, we force a split on the octree cells that intersects hexes outside the ε bound and repeat the procedure from Section 3.2.

The number of required splits is highly correlated with the user parameters l and ε. A smaller l reduces the chances of breaking the bound, but increases the overall mesh density, and similarly a larger ε reduces the chances of breaking the bound, but leads to meshes with a higher geometric difference from the input. Figure 11 shows

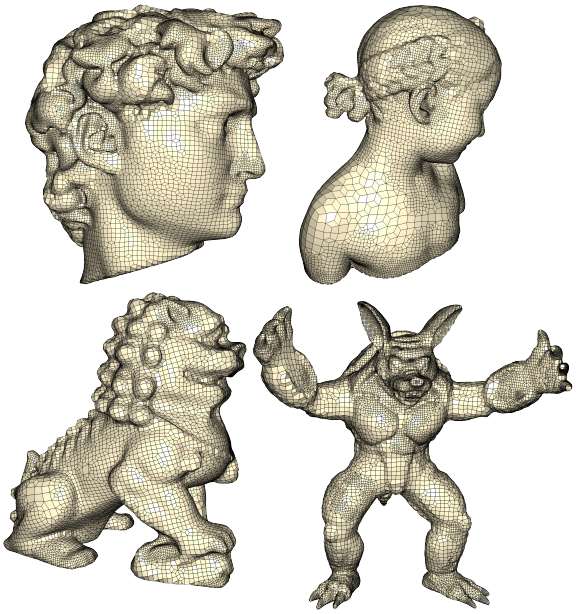


Figure 13: *Models with high frequency features.*

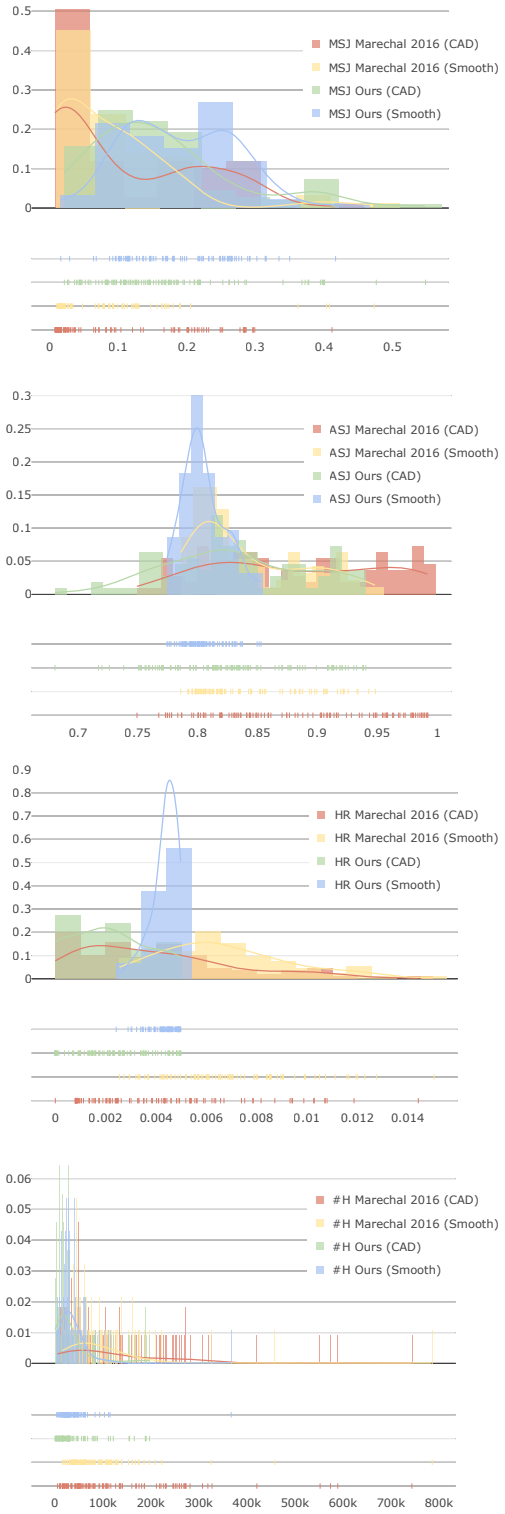


Figure 14: *Histograms for both our methods and [Mes18] over the entire datasets of: Minimum Scaled Jacobian (MSJ, top), Average Scaled Jacobian (ASJ, second top), Hausdorff distance ratio wrt the bounding box diagonal (HR, second bottom), and Hex count (#H, bottom).*

the effect of ε. As shown in the figure, the ratio of the number of singular edges to the number of all the edges in the mesh is not obviously correlated. We found this parameter much simpler to use than competing alternatives available in commercial software, where the maximal tree refinement or the desired number of elements are used.

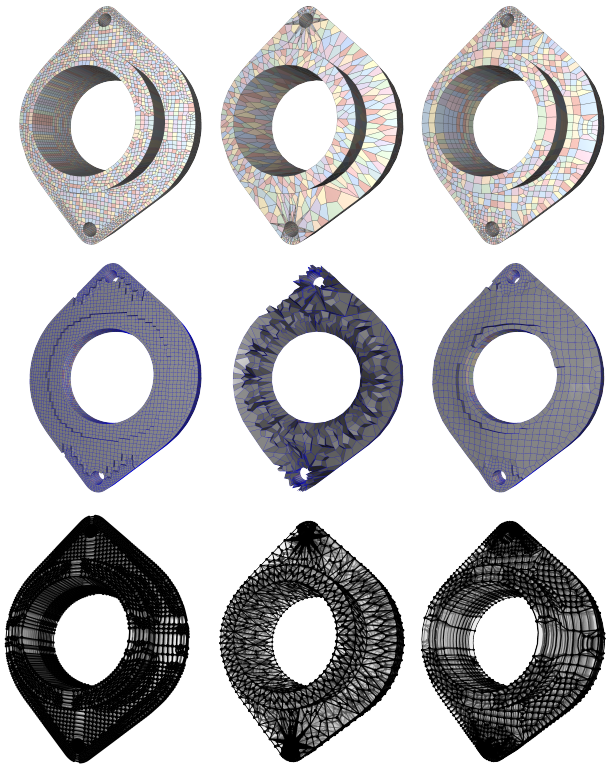


Figure 15: *Hex-meshes are generated from MeshGem [Mes18] (left column), by subdividing one tetrahedron into four hexes from a tet-mesh (middle column), and our approach (right column). Different visualizations of the hex-meshes are shown: the global structure (top row), cutting through of the interior (middle row), and the entire global structure with transparency.*

4. Results

We tested our single thread implementation on cluster nodes with 2 Xeon E5-2690v4 2.6GHz CPUs and 250GB memory, running 4 processes in parallel on each node, each with 60Gb of reserved memory. All experiments used $l = \frac{b}{2^6}$ (b is the longest side of the bounding box) and the same Hausdorff distance bound of $\varepsilon = 0.005d$ (d is the diagonal of the bounding box). The reference implementation used in the experiments and scripts to reproduce all results are included in the additional material.

Robustness Testing. We tested our algorithm on two benchmarks. The first is a dataset containing all the 93 organic models introduced in [FBL16]. The second is a new dataset of CAD models with manually annotated sharp features: the dataset was created by selecting and manually annotating 109 CAD models from [FBL16]

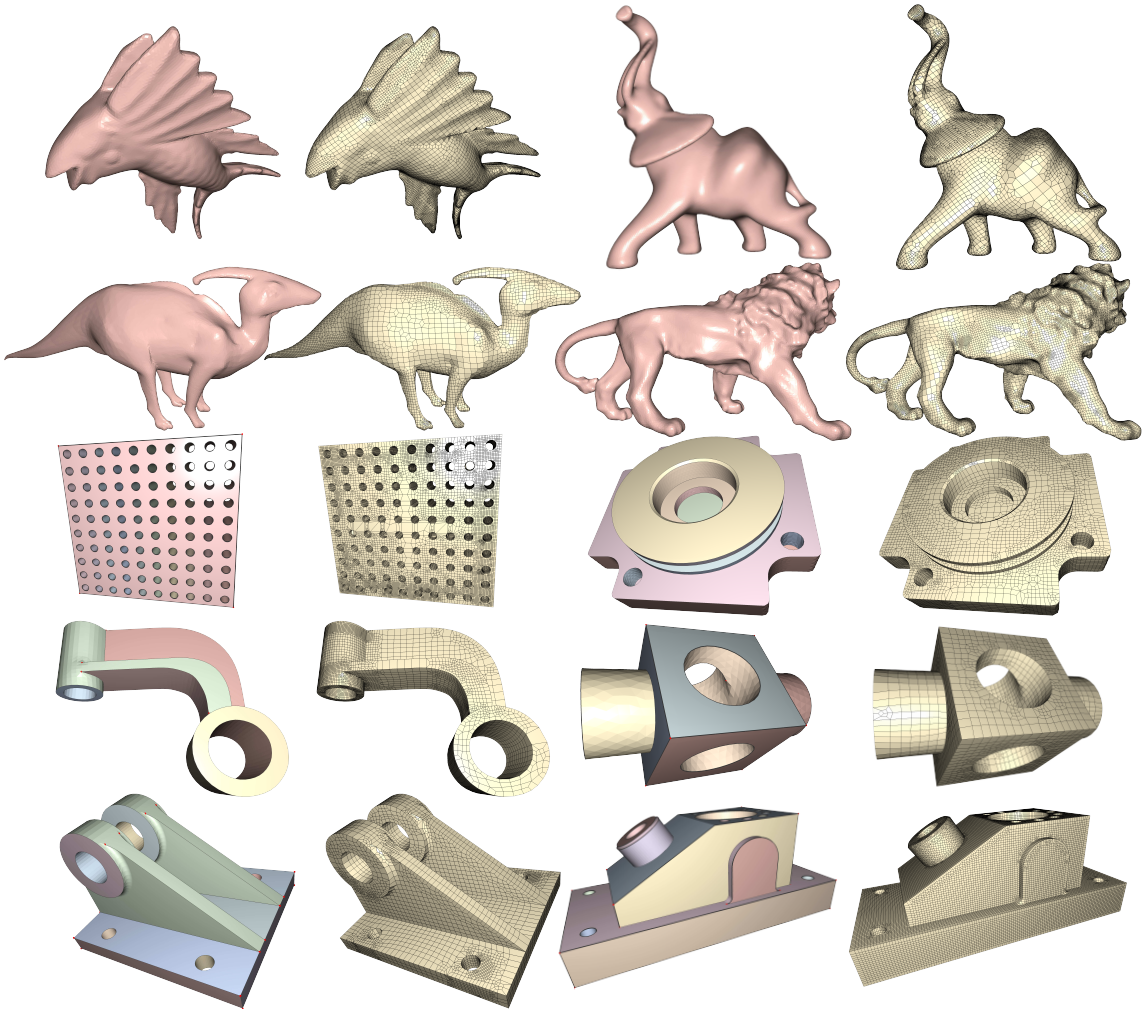


Figure 16: *Sample models in our dataset.*

(13 models) and from [Dre18] (96 models). The manual annotations are created using a combination of manual editing and automatic detection via dihedral angle thresholding. Our algorithm successfully meshed both datasets. A sampler of our results is shown in Figure 16. In addition, Figure 13 demonstrates that our algorithm works well for models with high-frequency features. The input data (obj format), together with the annotations (fgraph format) and the outputs (vtk and mesh format) are available in the additional material for further inspection. The timings for our method are reported in Figure 12 which demonstrates that the running time is growing approximately linearly with the number of hexahedral elements in the output mesh. To statistically measure the feature preservation of our method, for each model, we compute the error as the ratio between the deviation of corners, curves, and patches independently from the resulting mesh to the original input and the diagonal length of the model. Over the entire dataset, we achieved feature preservation with errors (Min/Average/Max): $0/7.06\times10^{-5}/9.56\times10^{-4}$ for corners, $3.98\times10^{-10}/2.11\times10^{-4}/1.13\times10^{-3}$ for curves, and $5.35\times10^{-5}/6.11\times10^{-4}/3.5\times10^{-3}$ for patches.

Comparison with Octree Methods. We compared our results against the implementation of [Mar09, Mar16] available in MeshGem-Hexa [Mes18], a state-of-the art, commercial hex-mesher. We use it to process the entire dataset and compare it with ours (using default parameters for both methods). We report aggregated statistics for both methods in Figure 14. We also show a representative visual comparison in Figure 17, and attach all the results in both the vtk and mesh formats for further inspection. Our method produces results with a *minimal scaled Jacobian* considerably higher than MeshGem for both smooth and CAD models (Figure 14, top). The *averaged scaled Jacobian* is similar for both methods (Figure 14, middle): it is interesting to note that while our method distributes the distortion more evenly, MeshGem favors perfect elements, producing an average quality close to 1 (ideal) for a few models. MeshGem does not explicitly control the geometric approximation error, obtaining meshes with a high variance in Hausdorff distance to the input (Figure 14, bottom), while our method keeps it bounded, producing meshes with a distance that is always smaller than ε (0.5% of the bounding box in our experiments). On the other hand, MeshGem

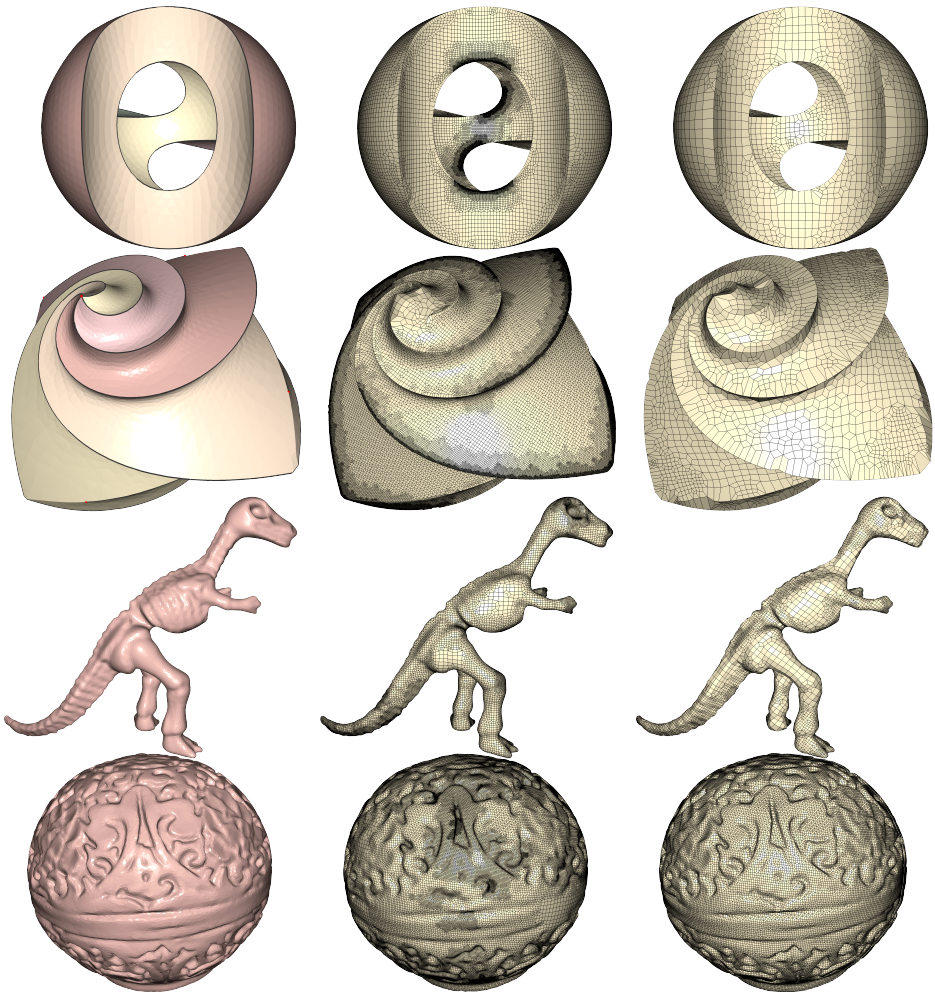


Figure 17: *Comparison between meshes generated by MeshGem Hexa (middle) and our algorithm (right). The input triangle mesh is shown on the left.*

produces meshes in seconds, while our approach can cost as many as more than 20 hours for a mesh with complex features (Figure 12, the dot at up-right corner).

Structure Irregularity. One of the big differences of a hex-mesh from a tet-mesh is its tensor-product nature which can be used to construct high-order splines. From irregular edges, a hex-mesh can be decomposed into cuboid blocks, the number of which is largely determined by the number of irregular edges and the connections between them. Figure 15 (top) shows the decomposed blocks for hex-meshes created by MeshGem, a direct one to four subdivision method of a tet-mesh, and our method, respectively. Their corresponding number of singular edges (number of all the edges) are 8270/143404, 3157/13637, and 4498/34528, where the ratio of the mesh by the direct tet decomposition is the largest. Their corresponding scaled Jacobians (MSJ/ASJ) are 0.180/0.844, 0.003/0.254, and 0.164/0.7667, where the quality of the mesh by the direct tet decomposition is the worst. As shown in Figure 15 (bottom), since both MeshGem and our methods are octree-based, it is not surprising that the resulting hex-meshes have similar patterns. However, since our method generates a coarser resolution for the same model than MeshGem, as a result, the structure of our mesh is much simpler. Compared to the hex-mesh from the direct subdivision of a tet-mesh, the octree structure is much simpler (Figure 15, middle row).

5. Conclusion

We introduced a fully automatic algorithm to generate hexahedral meshes conforming to the sharp features of the input, manifold, with positive scaled Jacobian, and without self-intersections on their boundary. The robustness of our approach has been experimentally verified by batch processing hundreds of models including both CAD and organic shapes.

Our technique has three main limitations: (1) it has a large memory footprint, (2) it has long running times, and (3) it only produce meshes with positive scaled Jacobians, i.e. the eight tetrahedra created at the corners of each hex have positive area.

Table 1: *Statistics of the input and output meshes of the models shown in the paper. #V/#H stands for the vertex and hex element numbers, ASJ and MSJ are the average and minimum scaled Jacobian. Std. is the standard deviation of the distribution of scaled Jacobians (in percentage) of all elements in a mesh. HR indicates the Hausdorff distance. Please refer to the supplemental document for the statistics of all models we have experimented with.*

Models	[Mar09, Mar16, Mes18]				Ours			
	#V/#H	MSJ/ASJ/Std.	HR(%)	Time (m)	#V/#H	MSJ/ASJ/Std.	HR(%)	Time (m)
3-octa-flower (Fig. 17)	854616/743861	0.008/0.786/0.032	1.07	0.48	20379/16616	0.045/0.738/0.041	0.48	35.90
anc101 (Fig. 16)	154675/135982	0.017/0.797/0.044	0.54	0.02	212847/188886	0.094/0.865/0.027	0.30	772.76
aries155_1 (Fig. 16)	85030/73878	0.030/0.892/0.029	0.82	0.07	27548/22547	0.092/0.813/0.031	0.36	76.13
armadillo (Fig. 13)	—	—	—	—	72728/60340	0.159/0.779/0.023	0.49	320.97
bimba (Fig. 13)	—	—	—	—	63679/55035	0.056/0.792/0.026	0.49	168.90
block (Fig. 16)	53985/47282	0.071/0.914/0.025	0.48	0.37	11517/9343	0.177/0.797/0.032	0.41	60.73
cheese 4 (Fig. 1)	644766/552706	0.009/0.778/0.046	0.22	0.56	103152/78245	0.142/0.717/0.033	0.45	106.26
chinese_lion (Fig. 13)	196616/174268	0.023/0.810/0.030	1.05	0.13	69278/59102	0.266/0.797/0.024	0.46	137.42
coverrear (Fig. 16)	104071/92084	0.015/0.846/0.039	0.40	0.11	38136/32116	0.134/0.781/0.037	0.25	80.95
dancer (Fig. 11)	49810/40947	0.019/0.816/0.026	0.52	0.05	37679/30958	0.193/0.807/0.022	0.45	110.37
david (Fig. 13)	506449/458518	0.015/0.804/0.033	0.75	0.28	127778/112314	0.126/0.795/0.026	0.49	647.77
dilo (Fig. 16)	100110/84977	0.0173/0.798/0.029	0.85	0.08	32637/26553	0.100/0.791/0.022	0.42	89.61
dino (Fig. 17)	115189/98515	0.020/0.796/0.028	0.59	0.08	35471/29065	0.248/0.777/0.022	0.43	166.39
elephant (Fig. 16)	130971/113341	0.048/0.799/0.029	0.70	0.10	38805/31642	0.232/0.781/0.021	0.48	104.84
gargoyle (Fig. 1)	—	—	—	—	157008/135737	0.070/0.785/0.025	0.48	634.22
hanger (Fig. 16)	54027/44863	0.025/0.851/0.031	0.50	0.10	33002/26918	0.155/0.828/0.028	0.23	25.60
holes10 (Fig. 16)	214268/186549	0.009/0.843/0.037	0.67	0.23	138138/115987	0.110/0.813/0.031	0.01	70.58
lion_recon (Fig. 16)	185901/161870	0.078/0.796/0.029	0.80	0.12	134140/115245	0.179/0.789/0.025	0.46	312.60
red_circular_box (Fig. 17)	864366/786129	0.017/0.805/0.032	0.67	0.50	409011/367583	0.216/0.817/0.027	0.29	1193.77
sculpture (Fig. 17)	353541/307789	0.011/0.787/0.037	0.23	0.23	19068/15202	0.104/0.759/0.036	0.16	13.76
wooden_fish (Fig. 16)	178530/153458	0.014/0.801/0.029	0.54	0.12	61284/50616	0.276/0.798/0.023	0.45	229.09

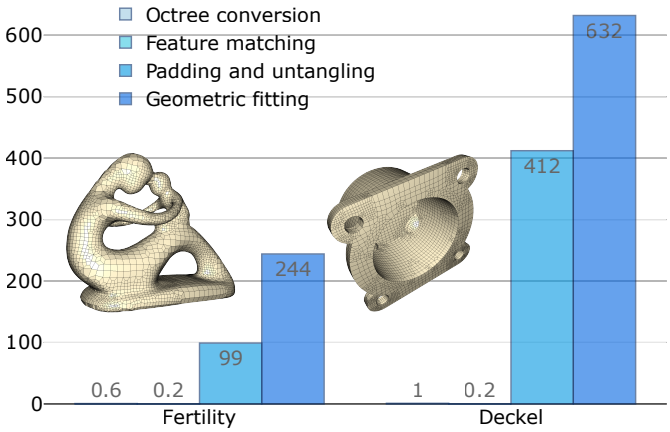


Figure 18: *Timings of one iteration of the four steps of our pipeline on an organic model and a CAD model. The vertical axis denotes the time in seconds.*

Figure 18 shows the breakdown of timings of one iteration of our pipeline on an organic model and a CAD model, respectively. For the four steps of our pipeline, both the padding and untangling, and geometric fitting steps dominant the timing since they both involve nonlinear global optimization. Similar breakdown statistics applies for the memory usage, where the peak memory for the first two steps and the last two steps are respectively the same, and it is 10 times larger of the last two steps than the first two steps.

The memory limitation could be ameliorated by computing the final deformation on subsets of the mesh, or by replacing SLIM with a localized optimization, which has a smaller memory requirement. The timing limitation is due to the use of locally injective volumetric parametrization on large meshes, and will likely become less relevant as more advanced parametrization algorithms are discovered.

The third limitation is more critical: having a positive scaled Jacobian is a necessary, but not sufficient, condition to ensure a bijective geometric map, e.g. the tri-linear map [JWR17], and indeed in 5 meshes in our results (out of 202) we have problematic elements. Luckily, for these five meshes, there is only one non-bijective element per model and is on the boundary which can be simply discarded to obtain a valid mesh. While these inverted elements are not a problem for solving Poisson problems, it prevents using these meshes for other PDEs like non-linear elasticity. This is a limitation shared with most recent hexahedral meshing and improvement methods [Knu00, Knu03, BDK*03, Mar09, GSZ11, NRP11, HTWB11, LLX*12, WSRRR*12, JHW*14, LVS*13, HJS*14, RGRS14, RGRS*15, LSVT15, GMD*16, GDC15, LMPS16, Mar16, FXBH16, FBL16, LBK16, GPW*17, CAS*19] and we believe is an important direction to find a way to integrate conservative inversion checks such as [Zha19] into a mesh optimization framework.

We believe that our technique will have a large impact in the research community, since it will allow to automatically generate hundreds of coarse, feature-preserving hexahedral meshes, providing a complete pipeline for generating IsoGeometric Aanalysis preferable hex-meshes by coupling it with [GPW*17], opening the door to black-box finite element analysis pipelines on hexa-

hedral meshes and allowing statistically significant comparisons between different types of meshes in graphics and engineering applications [SHG*19].

Acknowdgement

We thank Jérémie Dumas, Yixin Hu, Zhongshi Jiang, and Teseo Schneider for valuable discussion, suggestion, and helping with some initial experiments. This work was partially supported by the First Year Assistant Professor Funding 043525 from Florida State University. This work was supported in part through the NYU IT High Performance Computing resources, services, and staff expertise. This work was partially supported by the NSF CAREER award under Grant No. 1652515, the NSF grant OAC-1835712, the NSF grant DMS-1436591, a gift from Adobe Research, and a gift from nTopology.

References

[AAB*18] ALZETTA G., ARNDT D., BANGERTH W., BODDU V., BRANDS B., DAVYDOV D., GASSMOELLER R., HEISTER T., HELTAI L., KORMANN K., KRONBICHLER M., MAIER M., PELTERET J.-P., TURCKSIN B., WELLS D.: The `deal.II` library, version 9.0. *Journal of Numerical Mathematics 26*, 4 (2018), 173–183. 1

[ABA19] ABAQUS INC.: Abaqus FEA. http://www.simulia.com, 2019. 1

[ABH*15] ALNÆS M. S., BLECHTA J., HAKE J., JOHANSSON A., KEHLET B., LOGG A., RICHARDSON C., RING J., ROGNES M. E., WELLS G. N.: The FEniCS project version 1.5. *Archive of Numerical Software 3*, 100 (2015). 1

[AL13] AIGERMAN N., LIPMAN Y.: Injective and bounded distortion mappings in 3d. *ACM Trans. Graph. 32*, 4 (July 2013), 106:1–106:14. 3

[ANS19] ANSYS INC.: ANSYS®. https://www.ansys.com/, 2019. 1

[APL14] AIGERMAN N., PORANNE R., LIPMAN Y.: Lifted bijections for low distortion surface mappings. *ACM Trans. Graph. 33*, 4 (2014), 69:1–69:12. 3, 5

[APL15] AIGERMAN N., PORANNE R., LIPMAN Y.: Seamless surface mappings. *ACM Trans. Graph. 34*, 4 (July 2015), 72:1–72:13. 5

[BBS02] BORDEN M. J., BENZLEY S. E., SHEPHERD J. F.: Hexahedral sheet extraction. In *Proc. of the 11th International Meshing Roundtable* (2002), pp. 147–152. 3

[BDK*03] BREWER M., DIACHIN L., KNUPP P., LEURENT T., MELANDER D.: The mesquite mesh quality improvement toolkit. In *Proc. of the 12th International Meshing Roundtable* (2003), pp. 239–250. 3, 12

[Bis14] BISHOP J.: A displacement-based finite element formulation for general polyhedra using harmonic shape functions. *International Journal for Numerical Methods in Engineering 97*, 1 (2014), 1–31. 3

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

[BLP*12] BOMMES D., LÉVY B., PIETRONI N., PUPPO E., A C. S., TARINI M., ZORIN D.: State of the art in quad meshing. In *Eurographics STARS* (2012). 3

[Bol18] Bolt. http://www.csimsoft.com/boltoverview, 2018. Accessed: 2018-06-05. 2

[BPP*10] BOZZO A., PANOZZO D., PUPPO E., PIETRONI N., ROCCA L.: Adaptive quad mesh simplification. In *Eurographics Italian Chapter Conference 2010* (2010). 3

[CAS*19] CHERCHI G., ALLIEZ P., SCATENI R., LYON M., BOMMES D.: Selective padding for polycube-based hexahedral meshing. *Computer Graphics Forum 38*, 1 (2019), 580–591. 2, 12

[cod] MFEM: Modular finite element methods library. `mfem.org`. 1

[CRS03] CIGNONI P., ROCCHINI C., SCOPIGNO R.: Metro: Measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2 (2003), 167–174. 8

[CSZ16] CAMPEN M., SILVA C. T., ZORIN D.: Bijective maps from simplicial foliations. *ACM Trans. Graph. 35*, 4 (July 2016), 74:1–74:15. 2

[DMK03] DEGENER P., MESETH J., KLEIN R.: An adaptable surface parameterization method. In *Proceedings of the 12th International Meshing Roundtable* (2003), pp. 201–213. 3

[Dre18] DREXEL: Drexel cad repository, 2018. URL: http://edge.cs.drexel.edu/repository/. 10

[DSC09a] DANIELS J., SILVA C. T., COHEN E.: Semi-regular quadrilateral-only remeshing from simplified base domains. In *Computer Graphics Forum* (2009), vol. 28, Wiley Online Library, pp. 1427–1435. 3

[DSC09b] DANIELS II J., SILVA C. T., COHEN E.: Localized quadrilateral coarsening. In *Proceedings of the Symposium on Geometry Processing* (2009), SGP '09, pp. 1437–1444. 3

[DSSC08] DANIELS J., SILVA C. T., SHEPHERD J., COHEN E.: Quadrilateral mesh simplification. *ACM Trans. Graph. 27*, 5 (Dec. 2008), 148:1–148:9. 3

[EE14] ELSHEIKH A. H., ELSHEIKH M.: A consistent octree hanging node elimination algorithm for hexahedral mesh generation. *Advances in Engineering Software 75*, Supplement C (2014), 86 – 100. 2

[EPOM11] EBEIDA M. S., PATNEY A., OWENS J. D., MESTREAU E.: Isotropic conforming refinement of quadrilateral and hexahedral meshes using two-refinement templates. *International Journal for Numerical Methods in Engineering 88*, 10 (2011), 974–985. 2

[FBL16] FU X.-M., BAI C.-Y., LIU Y.: Efficient volumetric polycube-map construction. In *Computer Graphics Forum* (2016), vol. 35, Wiley Online Library, pp. 97–106. 2, 9, 12

[FL16] FU X.-M., LIU Y.: Computing inversion-free mappings by simplex assembly. *ACM Trans. Graph. 35*, 6 (Nov. 2016), 216:1–216:12. 3, 6, 7

[FLG15] FU X.-M., LIU Y., GUO B.: Computing locally injective mappings by advanced mips. *ACM Trans. Graph. 34*, 4 (July 2015), 71:1–71:12. 3

[FXBH16] FANG X., XU W., BAO H., HUANG J.: All-hex meshing using closed-form induced polycube. *Transactions on Graphics (Proc. SIGGRAPH 2016) 35*, 4 (2016). 2, 12

[GDC15] GAO X., DENG Z., CHEN G.: Hexahedral mesh re-parameterization from aligned base-complex. *ACM Trans. Graph. (SIGGRAPH '15) 34*, 4 (2015), 1–10. 3, 12

[GJTP17] GAO X., JAKOB W., TARINI M., PANOZZO D.: Robust hex-dominant mesh generation using field-guided polyhedral agglomeration. *ACM Trans. Graph. 36*, 4 (July 2017), 114:1–114:13. 3

[GMD*16] GAO X., MARTIN T., DENG S., COHEN E., DENG Z., CHEN G.: Structured volume decomposition via generalized sweeping. *IEEE TVCG 22*, 7 (2016), 1899–1911. 2, 12

[GPW*17] GAO X., PANOZZO D., WANG W., DENG Z., CHEN G.: Robust structure simplification for hex re-meshing. *ACM Trans. Graph. 36*, 6 (Nov. 2017), 185:1–185:13. 3, 7, 12

[GSZ11] GREGSON J., SHEFFER A., ZHANG E.: All-hex mesh generation via volumetric polycube deformation. *CGF 30*, 5 (2011), 1407–1416. 2, 12

[HCB05] HUGHES T., COTTRELL J., 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. 1

[HG00] HORMANN K., GREINER G.: MIPS: An efficient global parametrization method. In *Curve and Surface Design: Saint-Malo 1999*. 2000, pp. 153–162. 3

[HJS*14] HUANG J., JIANG T., SHI Z., TONG Y., BAO H., DESBRUN M.: L1-based construction of polycube maps from complex shapes. *ACM Trans. Graph. 33*, 3 (2014), 25:1–25:11. 2, 12

[HTWB11] HUANG J., TONG Y., WEI H., BAO H.: Boundary aligned smooth 3d cross-frame field. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 143:1–143:8. 3, 12

[ISS09] ITO Y., SHIH A. M., SONI B. K.: Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates. *Int. J. Numer. Meth. Engng*, 77 (2009), 1809–1833. 2, 4

[JHW*14] JIANG T., HUANG J., WANG Y., TONG Y., BAO H.: Frame field singularity correction for automatic hexahedralization. *IEEE TVCG 20*, 8 (Aug. 2014), 1189–1199. 3, 12

[JSP17] JIANG Z., SCHAEFER S., PANOZZO D.: Simplicial complex augmentation framework for bijective maps. *ACM Trans. Graph. 36*, 6 (Nov. 2017), 186:1–186:9. 2, 3, 7

[JWR17] JOHNEN A., WEILL J., REMACLE J.: Robust and efficient validation of the linear hexahedral element. *CoRR abs/1706.01613* (2017). URL: http://arxiv.org/abs/1706.01613, arXiv:1706.01613. 12

[KABL15] KOVALSKY S. Z., AIGERMAN N., BASRI R., LIPMAN Y.: Large-scale bounded distortion mappings. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH Asia) 34*, 6 (2015). 3

[KGL16] KOVALSKY S. Z., GALUN M., LIPMAN Y.: Accelerated quadratic proxy for geometric optimization. *ACM Trans. Graph. 35*, 4 (July 2016), 134:1–134:11. 3

[KLSO12] KOWALSKI N., LEDOUX F., STATEN M. L., OWEN S. J.: Fun sheet matching: towards automatic block decomposition for hexahedral meshes. *Engineering with Computers 28*, 3 (2012), 241–253. 3

[Knu00] KNUPP P. M.: Hexahedral mesh untangling and algebraic mesh quality metrics. In *Proceedings, 9th International Meshing Roundtable* (2000), pp. 173–183. 3, 6, 12

[Knu03] KNUPP P. M.: A method for hexahedral mesh shape optimization. *International journal for numerical methods in engineering 58*, 2 (2003), 319–332. 3, 12

[KPSC06] KIRK B. S., PETERSON J. W., STOGNER R. H., CAREY G. F.: libMesh: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. *Engineering with Computers 22*, 3–4 (2006), 237–254. 1

[KS04] KRAEVOY V., SHEFFER A.: Cross-parameterization and compatible remeshing of 3d models. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 861–869. 5

[LBK16] LYON M., BOMMES D., KOBBELT L.: Hexex: Robust hexahedral mesh extraction. *ACM Trans. Graph. 35*, 4 (July 2016), 123:1–123:11. 3, 12

[Lip12] LIPMAN Y.: Bounded distortion mapping spaces for triangular meshes. *ACM Trans. Graph. 31*, 4 (July 2012), 108:1–108:13. 3

[LLWQ13] LI B., LI X., WANG K., QIN H.: Surface mesh to volumetric spline conversion with generalized poly-cubes. *IEEE TVCG 19*, 9 (2013), 1539–1551. 2

[LLX*12] LI Y., LIU Y., XU W., WANG W., GUO B.: All-hex meshing using singularity-restricted field. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 177:1–177:11. 3, 12

[LMPS16] LIVESU M., MUNTONI A., PUPPO E., SCATENI R.: Skeleton-driven adaptive hexahedral meshing of tubular shapes. In *Computer Graphics Forum* (2016), vol. 35, Wiley Online Library, pp. 237–246. 2, 12

[LS10] LEDOUX F., SHEPHERD J.: Topological modifications of hexahedral meshes via sheet operations: a theoretical study. *Engineering with Computers 26*, 4 (2010), 433–447. 3

[LSVT15] LIVESU M., SHEFFER A., VINING N., TARINI M.: Practical hex-mesh optimization via edge-cone rectification. *Transactions on Graphics (Proc. SIGGRAPH 2015) 34*, 4 (2015). 3, 12

[LVS*13] LIVESU M., VINING N., SHEFFER A., GREGSON J., SCATENI R.: Polycut: monotone graph-cuts for polycube base-complex construction. *ACM Trans. Graph. 32*, 6 (2013), 171. 2, 12

[LZC*18] LIU H., ZHANG P., CHIEN E., SOLOMON J., BOMMES D.: Singularity-constrained octahedral fields for hexahedral meshing. *ACM Trans. Graph.* (2018). 3

[Mar09] MARÉCHAL L.: *Advances in Octree-Based All-Hexahedral Mesh Generation: Handling Sharp Features*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 65–84. 2, 3, 4, 10, 12

[Mar16] MARÉCHAL L.: All hexahedral boundary layers generation. *Procedia Engineering 163* (2016), 5 – 19. 25th International Meshing Roundtable. 2, 4, 6, 10, 12

[Mes18] MeshGems. http://meshgems.com/volume-meshing-meshgems-hexa.html, 2018. Accessed: 2018-06-05. 2, 9, 10, 12

[MKB*08] MARTIN S., KAUFMANN P., BOTSCH M., WICKE M., GROSS M.: Polyhedral finite elements using harmonic basis functions. In *Proceedings of the Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2008), SGP '08, Eurographics Association, pp. 1521–1529. URL: http://dl.acm.org/citation.cfm?id=1731309.1731340. 3

[MPKZ10] MYLES A., PIETRONI N., KOVACS D., ZORIN D.: Feature-aligned t-meshes. *ACM Trans. Graph. 29* (July 2010), 117:1–117:11. 3

[NRP11] NIESER M., REITEBUCH U., POLTHIER K.: Cubecover- parameterization of 3d volumes. *CGF 30*, 5 (2011), 1397–1406. 3, 12

[OSE17] OWEN S. J., SHIH R. M., ERNST C. D.: A template-based approach for parallel hexahedral two-refinement. *Computer-Aided Design 85*, Supplement C (2017), 34 – 52. 24th International Meshing Roundtable Special Issue: Advances in Mesh Generation. 2

[PL14] PORANNE R., LIPMAN Y.: Provably good planar mappings. *ACM Trans. Graph. 33*, 4 (2014), 76:1–76:11. 3

[PTH*17] PORANNE R., TARINI M., HUBER S., PANOZZO D., SORKINE-HORNUNG O.: Autocuts: Simultaneous distortion and cut optimization for uv mapping. *ACM Trans. Graph. 36*, 6 (Nov. 2017), 215:1–215:11. 6

[QZ10] QIAN J., ZHANG Y.: *Sharp Feature Preservation in Octree-Based Hexahedral Mesh Generation for CAD Assembly Models*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 243–262. 2

[RGRS14] RUIZ-GIRONÉS E., ROCA X., SARRATE J.: Optimizing mesh distortion by hierarchical iteration relocation of the nodes on the cad entities. *Procedia Engineering 82* (2014), 101–113. 3, 12

[RGRS*15] RUIZ-GIRONÉS E., ROCA X., SARRATE J., MONTENEGRO R., ESCOBAR J. M.: Simultaneous untangling and smoothing of quadrilateral and hexahedral meshes using an object-oriented framework. *Advances in Engineering Software 80* (2015), 12–24. 3, 12

[RL16] REBEROL M., LÉVY B.: Low-order continuous finite element spaces on hybrid non-conforming hexahedral-tetrahedral meshes. *ArXiv e-prints* (May 2016). arXiv:1605.02626. 3

[RPPSH17] RABINOVICH M., PORANNE R., PANOZZO D., SORKINE-HORNUNG O.: Scalable locally injective mappings. *ACM Trans. Graph. 36*, 2 (2017), 16:1–16:16. 2, 3, 7

[San16] SANDIA N. L.: Cubit. https://cubit.sandia.gov/, 2016. 2

[SAPH04] SCHREINER J., ASIRVATHAM A., PRAUN E., HOPPE H.: Inter-surface mapping. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 870–877. 5

[SB95] SCHNEIDERS R., BUNTEN R.: Automatic generation of hexahedral finite element meshes. *Computer Aided Geometric Design 12*, 7 (1995), 693 – 707. Grid Generation, Finite Elements, and Geometric Design. 2, 4, 6

[Sch96] SCHNEIDERS R.: A grid-based algorithm for the generation of hexahedral element meshes. *Engineering with Computers 12*, 3 (Sep 1996), 168–177. 2

[SCOGL02] SORKINE O., COHEN-OR D., GOLDENTHAL R., LISCHINSKI D.: Bounded-distortion piecewise mesh parameterization. In *Proceedings of the Conference on Visualization* (2002), pp. 355–362. 3

[SDG*18] SCHNEIDER T., DUMAS J., GAO X., BOTSCH M., PANOZZO D., ZORIN D.: Poly-spline finite element method. *CoRR abs/1804.03245* (2018). URL: http://arxiv.org/abs/1804.03245, arXiv:1804.03245. 3

[SDG*19] SCHNEIDER T., DUMAS J., GAO X., ZORIN D., PANOZZO D.: PolyFEM. https://polyfem.github.io/, 2019. 1

[SEK*07] STIMPSON C. J., ERNST C. D., KNUPP P., PÉBAYAND P. P., THOMPSON D.: The verdict geometric quality library, 2007. 1, 2, 3

[SHG*19] SCHNEIDER T., HU Y., GAO X., DUMAS J., ZORIN D., PANOZZO D.: A large scale comparison of tetrahedral and hexahedral elements for finite element analysis. *CoRR abs/1903.09332* (2019). URL: http://arxiv.org/abs/1903.09332, arXiv:1903.09332. 1, 13

[SJ08] SHEPHERD J. F., JOHNSON C. R.: Hexahedral mesh generation constraints. *Eng. with Comput. 24*, 3 (June 2008), 195–213. 2

[SKPSH13] SCHÜLLER C., KAVAN L., PANOZZO D., SORKINE-HORNUNG O.: Locally injective mappings. In *Symposium on Geometry Processing* (2013), pp. 125–135. 3

[SLSK04] SU Y., LEE K., SENTHIL KUMAR A.: Automatic hexahedral mesh generation for multi-domain composite models using a hybrid projective grid-based method. *Computer-Aided Design 36*, 3 (2004), 203–215. 2

[SPSH*17] SHTENGEL A., PORANNE R., SORKINE-HORNUNG O., KOVALSKY S. Z., LIPMAN Y.: Geometric optimization via composite majorization. *ACM Trans. Graph. 36*, 4 (July 2017), 38:1–38:11. 3

[SRUL16] SOKOLOV D., RAY N., UNTEREINER L., LÉVY B.: Hexahedral-dominant meshing. *ACM Trans. Graph. 35*, 5 (June 2016). 3

[SS15] SMITH J., SCHAEFER S.: Bijective parameterization with free boundaries. *ACM Trans. Graph. 34*, 4 (July 2015), 70:1–70:9. 3, 7

[SSGH01] SANDER P. V., SNYDER J., GORTLER S. J., HOPPE H.: Texture mapping progressive meshes. In *ACM SIGGRAPH* (2001), pp. 409–416. 3

[SSW96] SCHNEIDERS R., SCHINDLER R., WEILER F.: Octree-based generation of hexahedral element meshes. In *IN PROCEEDINGS OF THE 5TH INTERNATIONAL MESHING ROUNDTABLE* (1996), pp. 205–215. 2

[SVB17] SOLOMON J., VAXMAN A., BOMMES D.: Boundary element octahedral fields in volumes. *ACM Trans. Graph. 36*, 3 (May 2017). 3

[TK03] TAUTGES T. J., KNOOP S. E.: Topology modification of hexahedral meshes using atomic dual-based operations. In *Proc. of the 12th International Meshing Roundtable* (2003), pp. 415–423. 3

[TPC*10] TARINI M., PIETRONI N., CIGNONI P., PANOZZO D., PUPPO E.: Practical quad mesh simplification. *CGF 29*, 2 (2010), 407–418. 3

[TPP*11] TARINI M., PUPPO E., PANOZZO D., PIETRONI N., CIGNONI P.: Simple quad domains for field aligned mesh parametrization. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 142:1–142:12. 3

[VCD*16] VAXMAN A., CAMPEN M., DIAMANTI O., PANOZZO D., BOMMES D., HILDEBRANDT K., BEN-CHEN M.: Directional field synthesis, design, and processing. In *Computer Graphics Forum* (2016), vol. 35, Wiley Online Library, pp. 545–572. 3

[WSRRR*12] WILSON T. J., SARRATE RAMOS J., ROCA RAMÓN X., MONTENEGRO ARMAS R., ESCOBAR SÁNCHEZ J. M.: Untangling and smoothing of quadrilateral and hexahedral meshes. 3, 12

[XGC18] XU K., GAO X., CHEN G.: Hexahedral mesh quality improvement via edge-angle optimization. *Computers & Graphics 70* (2018), 17 – 27. CADGraphics 2017. 3

[YS84] YERRY M. A., SHEPHARD M. S.: Automatic three-dimensional mesh generation by the modified octree technique. *International Journal for Numerical Methods in Engineering 20*, 11 (1984), 1965–1990. 2

[ZB06a] ZHANG Y., BAJAJ C.: Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. *Computer methods in applied mechanics and engineering 195*, 9 (2006), 942–960. 2, 4

[ZB06b] ZHANG Y., BAJAJ C.: Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. *Computer Methods in Applied Mechanics and Engineering 195* (Feb. 2006), 942–960. 2

[ZBK17] ZHU Y., BRIDSON R., KAUFMAN D. M.: Blended Cured Quasi-Newton for Geometric Optimization. *ArXiv e-prints* (Apr. 2017). arXiv:1705.00039. 3

[Zha19] ZHANG S.: Subtetrahedral test for the positive jacobian of hexahedral elements. http://www.math.udel.edu/~szhang/research/p/subtettest.pdf, 2019. Accessed: 2019-04-03. 12

[ZLL*18] ZHAO H., LEI N., LI X., ZENG P., XU K., GU X.: Robust edge-preserving surface mesh polycube deformation. *Computational Visual Media 4*, 1 (Mar 2018), 33–42. 2

[ZLX13] ZHANG Y. J., LIANG X., XU G.: 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. 2

[ZZM07a] ZHANG H., ZHAO G., MA X.: Adaptive generation of hexahedral element mesh using an improved grid-based method. *Computer-Aided Design 39*, 10 (2007), 914–928. 2

[ZZM07b] ZHANG H., ZHAO G., MA X.: Adaptive generation of hexahedral element mesh using an improved grid-based method. *Computer-Aided Design 39*, 10 (2007), 914–928. 2