https://doi.org/10.1111/cgf.14393
This document is a preprint version.

COMPUTER GRAPHICS *forum*

At-Most-Hexa Meshes

Dennis R. Bukenberger[1], Marco Tarini[2], Hendrik P. A. Lensch[1]

[1]Eberhard Karls University, Tübingen, Germany [2]Università degli Studi di Milano, Italy

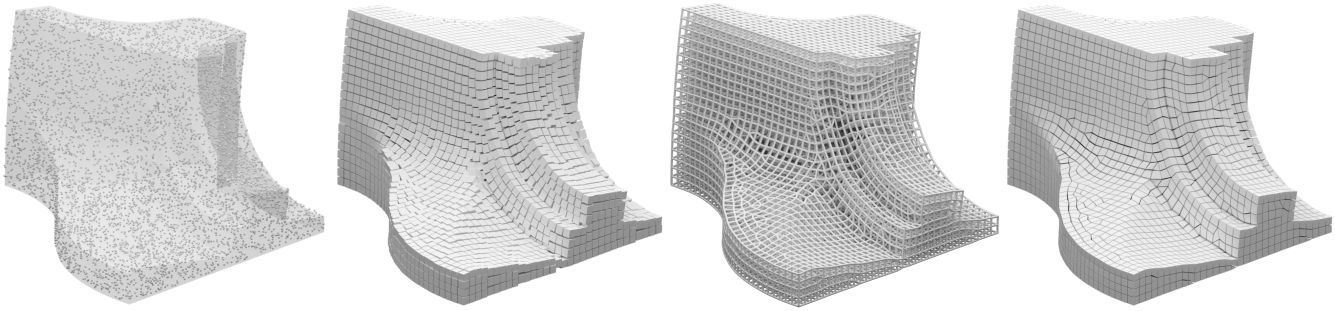


Figure 1: *Left to right: Stages of our pipeline from input samples to the final volume mesh with 9.5k primitives (*89.5% *hex).*

Abstract

Volumetric polyhedral meshes are required in many applications, especially for solving partial differential equations on finite element simulations. Still, their construction bears several additional challenges compared to boundary-based representations. Tetrahedral meshes and (pure) hex-meshes are two popular formats in scenarios like CAD applications, offering opposite advantages and disadvantages. Hex-meshes are more intricate to construct due to the global structure of the meshing, but feature much better regularity, alignment, are more expressive, and offer the same simulation accuracy with fewer elements. Hex-dominant meshes, where most but not all cell elements have a hexahedral structure, constitute an attractive compromise, potentially unlocking benefits from both structures, but their generality makes their employment in downstream applications difficult. In this work, we introduce a strict subset of general hex-dominant meshes, which we term "at-most-hexa meshes", in which most cells are still hexahedral, but no cell has more than six boundary faces, and no face has more than four sides. We exemplify the ease of construction of at-most-hexa meshes by proposing a frugal and straightforward method to generate high-quality meshes of this kind, starting directly from hulls or point clouds, e.g., from a 3D scan. In contrast to existing methods for (pure) hexahedral meshing, ours does not require an intermediate parameterization other costly precomputations and can start directly from surfaces or samples. We leverage a Lloyd relaxation process to exploit the synergistic effects of aligning an orientation field in a modified 3D Voronoi diagram using the L_∞ norm for cubical cells. The extracted geometry incorporates regularity as well as feature alignment, following sharp edges and curved boundary surfaces. We introduce specialized operations on the three-dimensional graph structure to enforce consistency during the relaxation. The resulting algorithm allows for an efficient evaluation with parallel algorithms on GPU hardware and completes even large reconstructions within minutes.

CCS Concepts

• ***Computing methodologies*** → ***Mesh geometry models;*** *Mesh models; Volumetric models; Point-based models;*

1. Introduction

Due to reduced element count and more harmonic structures, quad-meshes are often preferable over triangular-meshes for specific tasks like solving partial differential equations or CAD applications [BLP*13]. The same rivalry arises for volumetric meshes, where hexahedral meshes are often preferred over tetrahedral meshes [SHG*19]. Hex-meshes feature considerably fewer cells than tet-meshes for the same simulation accuracy, which is a desirable criterion for specific numerical solvers or finite element simulations. Their semi-regularity is a better fit for parallelized computation and makes their construction considerably more intricate due to the constraints implied by the global semi-regular lattice structure. On the other hand, tet-meshes are easier to construct and provide more direct control over the tessellation density, i.e., allowing for adaptive meshing resolution while still being conforming.

Hex-dominant meshes, where the majority of cells are still hexahedra, are a category of polyhedral meshes that are of interest because previous work [SRUL16, GJTP17] suggests that their construction can be achieved more reliably than in the more complicated case of *pure* hexahedral meshes (where all elements are hexahedra).

We propose to adopt a strict subset of hex-dominant meshes, called **"at-most-hexa meshes"**, where no cell exceeds six bounding faces, which are at maximum quadrangular, otherwise triangular. More specifically, all cells are either a hexahedron or are any valid polyhedron that can be obtained by starting from a hexahedron and collapsing a few of its edges (see Figure 2); at-most-hexa meshes are still conforming (free from T-junction).

Conceptually, the motivation for this choice is to ease the construction of the mesh as much as possible by relaxing the definition of hex-meshes while at the same time not sacrificing their usability by the downstream application. A linked motivation is that, because our cells can be considered special cases of hexas, many techniques applicable to pure-hexa meshes easily extend to deal with this new type of mesh. For example, internal representations, adjacency structures, and file formats designed for hex-meshes, can be readily adapted (Section 2).

The present work exemplifies how the construction of at-most-hexa meshes can be at least as reliable and fast as the construction of hex-dominant meshes. We are motivated by the intuition that at-most-hexa can be easier to process than the more general, hex-dominant meshes, which can feature arbitrary complex polyhedra while still inheriting most of the advantages from pure-hexa meshes.

To this end, we propose a pipeline based on a 3D Lloyd relaxation under the L_∞ norm for a harmonious hexahedral cell layout. Extracted geometry serves as the basis for a graph matching algorithm to identify all possible at-most-hexa primitives. The final mesh is then assembled with an iterative construction algorithm, prioritizing regular primitives over smaller polyhedra.

Many recently proposed approaches excel in solving a specific subproblem that contributes to the overall challenge of hexahedral meshing, but that has to be set into the perspective of what specifically tailored input is required. Our proposed concept is not explicitly designed to supersede all state-of-the-art techniques in every domain but extends this collection of possibilities with a novel start-to-finish procedure. A CAD-like specification, surface mesh, hex-dominant volume mesh, or solely a point cloud, as 3D scans acquire it, is already sufficient for our fully autonomous pipeline to produce at-most-hexa meshes. Nevertheless, we demonstrate how our results can compete or improve some established approaches in terms of hex-quantity and quality.

Key novelties and contributions of our proposed meshing approach can be briefly summarized with the following points:

• **At-Most-Hexa:** Primitives in our result meshes never exceed the base case of a hexahedral cell, which comes with many benefits over general hex-dominant meshes, as elaborated in the upcoming Section 2. We provide explicit graph matching routines with minimal branching for efficient primitive extraction. • **Simple, versatile input:** We do not assume input volume data, surface or volumetric parameterizations, frame-fields, or a consistent meshing of the surface. Hull meshes or sparse surface samples of arbitrary size and resolution are sufficient. • **Alignment:** Our method *can* be guided by an input orientation field if one is available, but this is not required. Initial orientations extrapolate from the input and then align to form an orthogonal vector field, further optimized during the relaxation process. • **Regularity and isometry:** The method strives to obtain regular meshing where most cells are hexahedral, most edges are regular, and cells are equally sized and well-shaped. • **Object hull:** Volumetric cells of the employed L_∞ Voronoi diagram materialize as final mesh primitives, thus there is no need to compensate for hull shrinkage as it is common with Delaunay graph meshing, e.g., with L_p-CVTs [LL10]. While this is trivial when the input is a closed mesh, we can also guarantee closed and feature-aligned result meshes starting from point cloud input because the kNN-graph provides robust and straightforward in/out labels, even for complex objects of higher genera. • **Implicit parallelism:** Our method leverages several sub-steps, where the construction and maintenance of the kNN-graph and the Lloyd relaxation rely on massively parallelized GPU code and the graph matching algorithms on multithreaded CPU implementations.

1.1. Related Work

Construction of Pure Hexa-Meshes There is a variety of hex-meshing approaches striving to automatically produce hex-meshes, which achieve high-shape quality, low-singularity, feature-aligned, or all-hex meshes [LBK16, SRUL16, GJTP17, LZC*18, CAS*19, Tak19, GSP19, LPP*20]. The required input for these approaches is, however, far from trivial to generate and primarily dictates the achieved result quality. Tetrahedral input meshes, parameterizations, mappings, frame fields, or singularity graphs often have to be determined beforehand, sometimes with heavy computation or manual effort to guide automation in the right direction. The relationship between source-mesh and input frame field also often poses a non-trivial causality dilemma of which to compute, i.e., derive from the other, first.

Approaches like QEx [EBCK13] for quad-mesh extraction from triangulations can be transferred to the hex-mesh extraction scenario as shown by Lyon et al. [LBK16] using parameterization [NRP11]. The all-hex meshing procedure proposed by Gregson et al. [GSZ11] relies on a given tet-mesh that is first transformed to PolyCubes [THCM04]. The all-hex method of Li et al. [LLX*12] focused on high-quality primitives using a singularity-restricted field. The octree-based method of Gao et al. [GSP19] does well in preserving features in the final hex-mesh by adapting the resolution where necessary but entirely omits all curvature and flow alignment. Takayama's method [Tak19] for all-hex meshes relies on user-defined planes (dual-sheets) bisecting the object. In a recent publication, Livesu et al. [LPP*20] developed this idea further and extracted loop cuts automatically from a mesh with a set of (sometimes manually) marked sharp feature edges and a frame field, resulting in high-quality hex-dominant meshes. On volumetric data, Zhang et al. [ZB06] published methods for high-quality quad- and hex-mesh extraction. As hex-meshes consist of hexahedral primitives with six quadrilateral faces each, one can easily comprehend that the surface of a pure hex-mesh is a quad-mesh. Calvo et al. [CI00], and Kremer et al. [KBLK14] proposed techniques to go the other way round and come up with a hex-mesh with only the quad-faced hull given. Early work by Shep-

herd et al. [SJ08] formulated general constraints for hexahedral meshing. More recent publications focus on the extraction of hex-meshes from tet-meshes using frame-fields or parameterizations [SVB17, GJTP17, RSR*18, CAS*19] or structural modification to make an object suitable for hex-remeshing [GPW*17]. Others describe constraints on the octahedral fields of existing hex-meshes to minimize singularities [LZC*18] which then qualify as "meshable" input for further elaborate orientation field computations [CC19]. Meshkat and Talmor [MT00] proposed a graph matching algorithm to extract hexahedra from a given tet-mesh. The publication by Sokolov et al. [SRUL16] extends this concept extensively with formal proofs and improves upon the results of Lévy and Liu [LL10] and Baudouin et al. [BRM*14]. Recently Pellerin et al. revisited the idea from tet- to hex-dominant mesh with a vertex-based approach [PJVR18]. A detailed review on the distinctions between Lévy and Liu's L_p-CVT based hex-dominant meshing approach and ours is featured in Section 6.2.

Analogy with quad-remeshing Many of the concepts above, and our construction technique, can be considered extensions of ideas originated in quad meshing. Our approach is inspired by the image stylization technique of Hausner [Hau01] to simulate mosaic images using Lloyd relaxations with an adapted metric and faces similar challenges as the quad-meshing approach by Pellenard et al. [PAM11] but with three dimensions. Furthermore, a smooth object-aligned octahedral flow field [SVB17, GPW*17] is created alongside the relaxation. We extended the method for fast computation of generalized Voronoi diagrams using graphics hardware [HIKL*99] to three-dimensional space. Our procedure utilizes a specially tailored graph matching algorithm to extract at-most-hexa primitives for the final mesh structure. As our approach relies on faces instead of tetrahedra [SRUL16], the graph matching search space significantly reduces to only one valid traversal path per primitive type.

Constructing Meshes from Point Clouds Our construction method, which can start from a point cloud, has direct conceptual predecessors in surface remeshing or point cloud reconstruction. Spatially harmonious triangulations of points in 2D or even on a 3D surface are easily determined with a Delaunay triangulation, the equivalent for points in a 3D volume results in a tetrahedral mesh structure. However, neither concept extends directly to quad or hexa meshes. The popular *Poisson surface reconstruction* [KBH06] utilizes the orientation of the samples to come up with a closed surface representation for a given point cloud. Although simple triangulations of these surfaces are trivially possible by extracting isosurfaces, quad meshes with feature-aligned topology are often the more favorable option. The concepts proposed by Jakob et al. [JTPSH15] and Schertler et al. [STJ*17] use flow-fields to extract such quad meshes from oriented point clouds. Other works [ZLGH10, BL18] proposed techniques that do not require scanned or precomputed normals for the point cloud to come up with a surface or, in the latter case, with a quad-mesh.

Hex-dominant Meshes The construction of hex-dominant meshes is studied as an interesting and convenient relaxation of pure hexa meshes. In recent works [SRUL16, GJTP17], hex-dominant meshes are obtained by fusing or conglomerating tetrahedral cells into more complex polyhedra, which are often but not always hexahedra. The hex-dominant approaches are categorized according to the polyhedra that are allowed for non-hexahedral cells. In the proposal by Sokolov et al. [SRUL16], these polyhedra are limited to quad-based pyramids, prisms, and tetrahedra (plus "slivers"). Our at-most-hexa meshes are a further relaxation, allowing for these shapes but also others, making the construction considerably less intricate than the one proposed by Sokolov et al. [SRUL16]. In the proposal of Gao et al. [GJTP17], there is no assumption made on the shape of the resulting non-hexahedral cells. While this maximally simplifies construction, the results are less usable for the reasons discussed below.

2. At-Most-Hexa Meshes

In this section, we define at-most-hexa meshes, which are the output of our construction technique, and outline a few of their favorable characteristics. In our meshes, each cell is definable as the locus of a tri-linear combination of the hexahedron's (possibly coinciding) vertices. In other words, each cell is a polyhedron that can be obtained by starting from a hexahedral cell and collapsing zero or more edges.

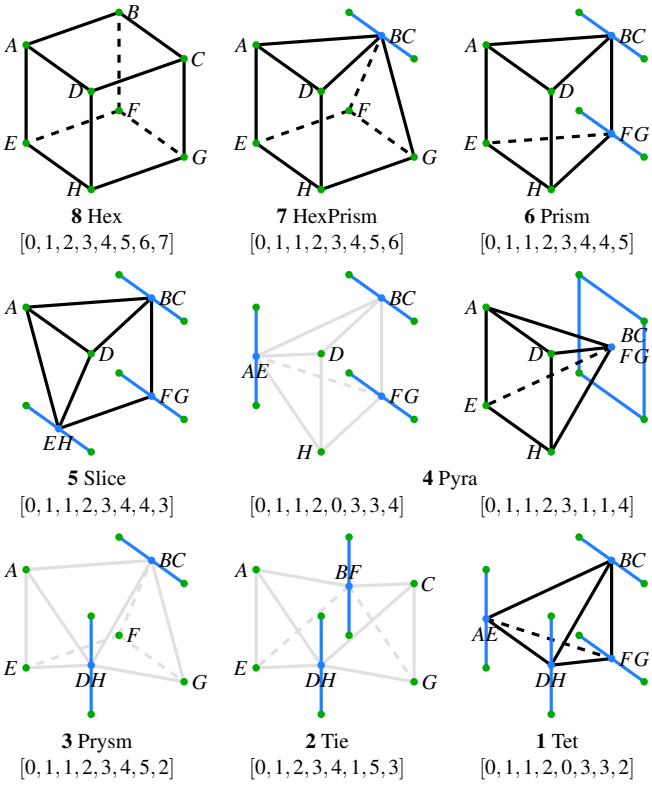


Figure 2: *The collection of at-most-hexa primitives featured in our meshes (except the grayed-out ones). The list is exhaustive up to rotational and reflection symmetries. Each configuration can be obtained by collapsing up to four non-adjacent edges of a hexahedron. This is modeled by duplicating the indices in a list of 8 vertices from the hexahedron.*

In Figure 2 we exhaustively list all the cell topologies that can be obtained in this way (up to symmetries). Not all possibilities are

useful for our purposes. Specifically we can exclude the following cases from further consideration:

- The "*Prysm*" (case 3) is topologically equivalent to the triangular *Prism* (case 6), with an additional diagonal on one quad face. Following the rules for extracting prisms (Figure 22), prysm constellations would be trivially included, thus result as a subset of all prisms. As the use of prysms would actively promote more triangular constellations and *sliver* elements, the regular prism is always the more favorable option. Therefore, prysm constellations are not considered in the extraction.
- The "*Tie*" (case 2) can be seen as the combination of two *Tetrahedra* (case 1) sharing one edge.

Two distinct configurations result in a pyramidal shape (case 4).

Volumetric Definition of Solid Objects Both pure tetra-meshes and pure hex-meshes, but not generic hex-dominant meshes, allow for a straightforward definition of the represented object's interior (and the boundary) as the union of their cells. A tetrahedral cell trivially defines as the set of all linear interpolations of the mesh vertices at its four corners; in hexahedral meshes, *and also in our meshes*, all cells are polyhedra which can be defined as the set of the trilinear interpolations of the eight vertices at their corners. Conveniently, the interiors of any two adjacent cells (cells sharing a face) are disjoint sets, and their union is a *simply connected* locus of points (i.e., no gap is left between them, despite cell faces not being necessarily planar). This construction does not extend trivially to generic hex-dominant meshes because it is unclear how to combine the vertices at the corner of an irregular polyhedron (with non-flat faces). Conversely, our at-most-hexa meshes generalize this situation. The cell's interior defines as the trilinear interpolation of the corners, but, in the occasional non-hexahedral cells, a few of its corners are instances of the same (x,y,z) mesh vertex. Additionally, when a cell reduces to a tetrahedron, the above definition is equivalent to a linear interpolation of the 4 surviving distinct vertices, meaning that the proposed structure generalizes both tetrahedral- and hexahedral-meshes.

Signal Interpolation Any scalar or vectorial signal sampled at the tet- or hex-mesh vertices can trivially interpolate for any point inside its interior (or boundary). Therefore, the same set of weights used to define an interior point p as a linear combination of the corners of its cell is employed to combine the signal defined at the vertices. This results in the definition of a scalar or vectorial field inside the mesh, which is C_0 everywhere (and C_∞ in the interior of the cells). Once again, this functional principle is inherited directly by at-most-hexa meshes, preserving all the properties, but not by general hex-dominant meshes (as a generalization of barycentric coordinates is not trivial even in 2D [HS17]).

Compact Representations Both pure tet- and hex-meshes can be internally represented as *indexed meshes* [BKP*10], a succinct data structure consisting of a set of vertices and a set of cells; tetra cells and hex cells are stored as a sequence of 4 (respectively, 8) indices of vertices at their corners, in some prescribed order. This can be useful for storing the mesh on drives, for example, in interexchange formats. A general hex-dominant mesh does not allow for such representations because non-hex elements have no structure that is known *a priori*. Conversely, cells of an at-most-hexa mesh can be represented the same way as hexahedra, where a few of the vertex indices at the corners (in non-hex cells) repeat as listed in Figure 2. This representation is not only convenient for storing meshes, allowing, for example, to reuse the same file formats of hexa meshes but also allows for easy application of common hex-meshing operations like subdivision [WSK06].

A note on cell convexity Our meshes are analogous to pure hex-meshes in that their cells are not necessarily convex unless special care is taken to ensure that every quadrangular face is exactly planar. While face planarity is implicitly pursued as a soft objective by our construction strategy, we did not experiment with its strict enforcement. The same issue arises with quad surfaces, where strictly enforced solutions have been proposed (PQ-meshes, [LPW*06]).

3. Overview of the Meshing Algorithm

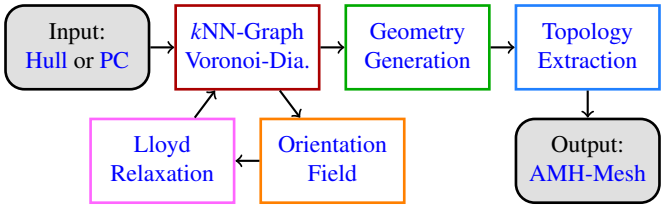


Figure 3: *The schematic pipeline of our method.*

3.1. Steps Breakdown

The diagram in Figure 3 outlines the basic steps of our pipeline:

▶ **Input Preparation:** The input object's volume (plus margins) is populated with points (sites) on a regular 3D grid, serving as seeds of a Voronoi diagram. Each site creates a cell in the diagram, and will generate one at-most-hexa primitive in the final output mesh.
▶ ***k*-Nearest-Neighbor Graph:** This graph stores the nearest-neighbor relationships between sites, and is crucial for an efficient evaluation of all subsequent steps.
▶ **Orientation Field:** Orientations from the input hull or point cloud are extrapolated once and further on jointly aligned during the relaxation following parallel and orthogonal flow constraints.
▶ **Lloyd Relaxation on Hex-like Cells:** By exchanging the Euclidean norm (L_2) in the Lloyd relaxation with the Chebyshev norm (L_∞), the cell structure becomes more cubical, which is suitable for further interpretation as a hex-like mesh structure.
▶ **Geometry Generation:** Geometry is established by materializing the hex-like cells as proportionally scaled unit cubes. Vertices and edges are created with a simple match-and-merge operation.
▶ **Topology Extraction:** With geometry established, tri- and quad-faces are collected to form the base for all at-most-hexa primitives. The primitives are then collected via graph matching, using small dedicated state-machine algorithms, effectively minimizing the required search space.
▶ **Output:** The final assembly routine allows for prioritizing regular hexahedra of high quality. Therefore, result meshes solely feature the at-most-hexa primitives listed in Figure 2 where the absolute majority are hexahedra.

3.2. Terminology

To counteract misunderstandings, we first want to establish a uniform terminology for involved entities: We will refer to the points in a Voronoi diagram as *sites* while the same point is a *node* in the associated graph structure. A *cell* refers to the associated space around each site. Points on the input surface, or point cloud points themselves, are referred to as *samples*. For simplicity, we call these entities s_i in all instances and the unique index i may also be used to identify corresponding properties like depth d_i, in/out-label l_i or a normal n_i. In the upcoming section for geometry extraction, cells will materialize as scaled unit cubes, each defined with eight *virtual* vertices. As the geometry extraction progresses, *virtual* vertices are merged to *real* ones. The geometric entities emerging from the topology extraction process feature hexa-, tetra-, and other polyhedra, collectively called *primitives*.

4. Relaxation

The relaxation is essential to capture characteristic features of the input. Therefore, the space inside and outside of the input object is populated with volumetric cells, which eventually align with the model's shape and curvature. Rather than relying on a predefined frame-field providing the orientation for volumetric cells, our relaxation optimizes the orientation and position of all sites in a collaborative process. Besides alignment to surface features, the goal of this stage is to obtain equally sized and cube-shaped cells.

Therefore, the necessary optimization is performed similarly to a Lloyd relaxation with specific constraints to favor the generation of hex-like cell structures. Eventually, the relaxation process outputs sites with optimized location and orientation, defining the input to the mesh extraction stage of Section 5.

4.1. Voronoi Diagram

The basis for the Lloyd relaxation is the underlying Voronoi diagram, which computes on a face-centered cubic (fcc) lattice [CS98, HAB*17] with at least 12^3 times more lattice points than sites in the diagram. This has proven to be a sufficiently high resolution that is still practically feasibly with limited GPU memory. Further, the fcc lattice is preferable over a regular cubical grid to avoid axis-aligned bias but is in contrast to lattice-guided approaches [YS03, NZH*18] merely a convenient way to label space. The diagram is computed on the GPU using a z-buffer [HIKL*99] extended for three dimensions. In this context, the publication Meshless Voronoi on the GPU [RSLL18] comes to mind. But, as elaborated in the following, our distance metric is not orientation-invariant as in a standard Voronoi diagram, which drastically complicates the integration of a cell. Therefore, this concept is not trivially suitable for our objective.

To propagate information between all sites $\mathbf{S}$, the relaxation relies on two different mechanisms: Lloyd iteration to optimize site positions and cell extents and a kNN-graph to align orientations and eventually promote hex-favorable grid structures.

Site Population In the first step, the object's bounding box plus margin is populated with sites $\mathbf{S}_v$ initialized on a regular or jittered grid, filling the entire volume. The number of sites in the diagram directly gives the final mesh's resolution. A partitioning is specified for one dimension of the bounding box and scaled accordingly for the others. It can be either used-defined to approach a certain target resolution or heuristically derived from the input, i.e., based on minimum widths in the input geometry. The whole set of sites in the diagram consists of two disjunct subsets $\mathbf{S} = \mathbf{S}_{pc} \cup \mathbf{S}_v$. Sites of $\mathbf{S}_v$, which are close to the hull, spawn a second set of surface samples $\mathbf{S}_{pc}$ positioned directly on the input hull. In Appendix B we propose to replace the $\mathbf{S}_{pc}$ set with an actual point cloud as an alternative to meshed input. However, for now, $\mathbf{S}_{pc}$ solely serves as query points for the orientation extrapolation.

Input Hull As the relaxation treats all cells equally, there is no distinction between cells in or outside the object. However, in the end, only the inside cells are relevant for further use. Therefore, inside-outside labels for all cells are determined using fast winding numbers [BDS*18] of the input surface. During the relaxation, the mesh also acts as a natural boundary, limiting the individual cell's extents and protects them from crossing the hull.

Distance Metric Sites thrive to increase the distance between each other during the relaxation, which eventually creates primitives of homogeneous size and maximum mesh isotropy. For a 2D example, if one would keep the Euclidean distance (L_2) as a metric, the majority of relaxed cells would resemble hexagons (like a honeycomb) because this is the densest 2D packing of circles [CW10]. Therefore, we employ the Chebyshev metric (L_∞) for our relaxation: 2D cells would now approximate squares [Hau01, MB12] and respectively, 3D cells actually become cubical [LL10, BRM*14]. In each iteration of the Lloyd relaxation, sites update with the geometric center of their cell, computed as the averaged position of their labeled lattice points.

4.2. kNN-Graph

To allow for fast information propagation during the relaxation, we incorporate a decentralized network between the sites, namely kNN-graphs, where each site links to its k nearest neighbors. Particularly, N_{26} and N_6 neighborhoods are used. N_{26} is purely based on geometric distances. $k = 26$ corresponds to $3\times3\times3-1$ cubes stacked in a 3D grid. In irregular arrangements, N_k might also contain sites that are not direct neighbors, but this hardly impacts the optimization. In Section 4.4, another N_6 neighborhood promotes hexahedral grid alignment. Whereas N_k includes neighbors based on their geometric distance alone, N_6 also incorporates a site's orientation: It features only the most suitable six neighbors from each direction (left, right, up, down, front, back) at an edge-length's distance e. As formulated in Equation 1, the N_6 can be derived as a subset of the N_k where M_i is a site's orientation and $\vec{r}$ corresponds to the coordinate axes.

$$N_6(i) = \left\{ \min_{j \in N_k(i)} || (s_i + (M_i \cdot \vec{r})e) - s_j ||_2 \right\}_{\vec{r} \in [\pm x, \pm y, \pm z]} \tag{1}$$

Furthermore, each node also maintains an n-hop-distance d_i (Equation. 2), which counts the number of steps required to reach the closest sample nodes on the hull.

$$d_i = \begin{cases} 0 & \text{if } s_i \in \mathbf{S}_{pc} \\ \min\limits_{j \in N_k(i)} \left[d_j\right] + 1 & \text{else} \end{cases} \tag{2}$$

Construction & Maintenance The kNN graph initializes by setting the neighbors of each node randomly. With a simple parallel update routine on all nodes $\mathbf{S}$, the randomly initialized graph becomes an actual nearest neighbor graph:

- For the node s_i collect the neighbors of all neighbors. $N_k^2(i) = \bigcup_{j \in N_k(i)} N_k(j)$.
- Sort by geometric distance $||s_i - s_j||_2$ where $j \in N_k^2(i)$. $\mathbf{N}_k^2(i) := \text{sort}(N_k^2(i))$
- Update $N_k(i)$ with the first k elements in $\mathbf{N}_k^2(i)$.

It can be shown [DML11] that only 7 update steps are required to get an almost perfect kNN approximation from random input connections. As the relaxation progresses, sites in the Voronoi diagram change their position with every step, and therefore, the graph also has to be updated with every iteration. However, once the graph is established, neighborhood fluctuation is marginal and usually only one, or to be sure two, update cycles have to be performed.

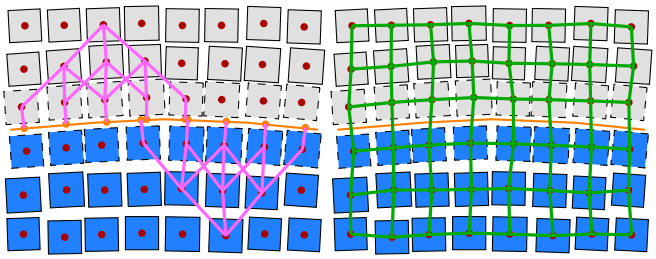


Figure 4: *Nodes with $d_i = 1$ (dashed) determine their initial orientation from samples on the hull. Nodes of $d_i > 1$ derive theirs from neighboring nodes closer to the surface using portions of the N_{26} graph. The right side shows the N_6 graph, transcending the outer hull so that adjacent cells on the in and outside can align.*

4.3. Constrained Relaxation

The Lloyd relaxation process is an iteration alternating two steps: 1. compute a Voronoi diagram based on the given site positions, 2. reposition each site to the geometric center of its cell. But as our employed distance metric is no longer orientation invariant, we also have to maintain individual orientations for all sites.

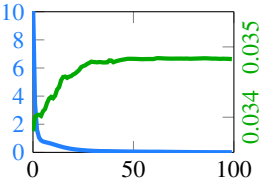
Orientation Initialization Each node in our graph carries its own orientation, defined by the three orthogonal base vectors: normal, tangent and bitangent, represented for interpolation by quaternions.

For samples $\mathbf{S}_{pc}$ on the input hull, orientation is determined as the surface normal plus principal curvature vectors [PdC76, Rus04]. As initialization, the orientations of $\mathbf{S}_{pc}$ are extrapolated once through the volume for all sites in $\mathbf{S}_v$. This is done in a wave-front propagation manner [OBB*13] over the discrete node positions of the graph. Portions of the N_k graph are shown in Figure 4 (left): Discrete n-hop-distances d_i as well as real geometric distances are employed to weight individual orientations during propagation.

Maintaining Orientations During relaxation, the orientation *and* the spatial arrangement of neighboring cells jointly align, resulting in the best fitting constellation concerning the geometrical constraints imposed by the input hull. In contrast to a predefined volumetric frame-field, orientations emerge from the alignment itself and are bound to the individual cells and their discrete site positions. Orientations of adjacent neighboring sites are optimized to be consistent, herein defined with invariance to axis-permutating rotations. This constraint is beneficial for the mesh extraction step in Section 5, where neighboring cells shall become adjacent primitives, forming hex-like structures.

Therefore, in each iteration, the orientation of a site is aligned to a distance-weighted combination of all orientations from its neighbors in N_k. This is realized analogously to the extrinsic smoothness energy minimization formulated by Jakob et al. [JTPSH15], which, summed up briefly, means: The base vectors for each site should point in close-to-parallel or orthogonal directions compared to their neighbor sites, regardless of their signs.

Convergence Whereas Lloyd relaxations are known to converge to Centroidal Voronoi Tesselations using the L_2 norm [DEJ06], it has yet to be shown that the same holds for higher dimensions or other norms. However, in practice, we could not provoke scenarios that showed tendencies of non-convergence or one that resulted in a bi-stable state.



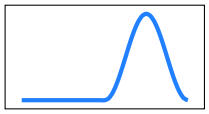
This graph plots the accumulated movement of all sites and the average volume of their cells over 100 relaxations. While the movement drops below numerical accuracy within the first 50 iterations, the volume also approaches a steady fix-point.

4.4. Regularization

So far, sites $\mathbf{S}_v$ freely move around during the relaxation, maximizing the distance to each other and orient themselves accordingly. However, for the upcoming geometry extraction step, sites should be positioned to form a grid if possible. The constellation on the left in Figure 5 resembling a brick wall is not unlikely to emerge with aligned orientations alone and without positional constraints. To counteract this brick wall alignment, we introduce a positioning scheme using the N_6 neighborhood. Equation 3 formulates the updated center c as the weighted sum of the Voronoi cell's geometric center c_g and the center of its six neighbors c_{N_6}. For constant $w(p) = 0$, the process is equivalent to Lloyd's algorithm.

$$c = (1 - w(p)) \cdot c_g + w(p) \cdot c_{N_6} \tag{3}$$

Here it is crucial to note that N_6 neighborhoods strictly exclude hull samples ($N_6 \cap \mathbf{S}_{pc} = \emptyset$). Therefore, the effect shown on the right in Figure 5 can benefit from outside cells, too, as illustrated in Figure 4. Improved results can be achieved using a variable weighting function that changes throughout the relaxation.



$$w(p) = \frac{1}{2} - \frac{\cos\left(4\pi\left(\max(\frac{1}{2}, p) - \frac{1}{2}\right)\right)}{2} \tag{4}$$

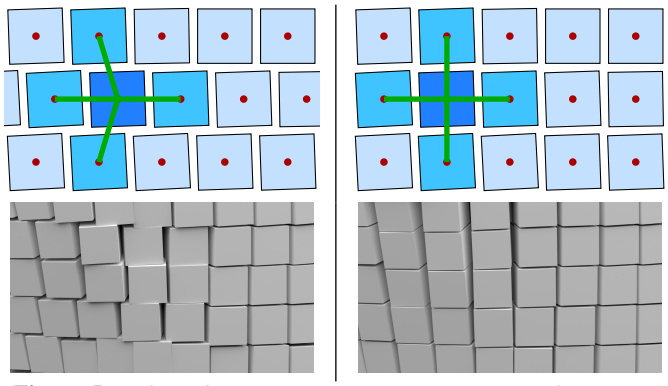


Figure 5: *Left: Coherent orientation is no guarantee for proper alignment of adjacent cells. Right: The N_6 graph reintroduces regularity to the relaxed system.*

Heuristic experiments suggest letting the first half of the relaxation run based on c_g centers alone, then increase the contribution of c_{N_6} centers (forcing the sites to from a hex grid) with a cosine curve peaking at 75% of the procedure and have them converge to 0 again towards the end of the relaxation. Other strategies for $w(p)$ like a linear, squared, quadratic, or sinusoidal decrease, increase, or both (as a peak) are possible but were outperformed by the curve, formally expressed in Equation 4 with progress $p \in [0,1)$.

Split Cells Some geometry might cause unfavorable constellations in the relaxed graph, like neighborhood clusters. A cluster occurs if a node is considered as a direct neighbor by more than 6 other nodes. As the N_6 graph constantly updates during the relaxation, one can quickly determine and resolve such clusters by splitting the affected node. However, if a node is split too early, e.g., with a cluster size of 7, the two resulting split nodes will have an underpopulated neighborhood, which is why we chose a split-limit of 10. The to-be-split node is replaced by two new nodes, inheriting its neighbors and linking to each other. Their geometric position is based on the split-node's site position, shifted in the positive or negative direction of the cells principal direction vector, respectively.

5. Mesh Extraction

The focus of this section is the post-relaxation domain, schematically outlined in Figure 6, which is to extract the geometry and topology of the mesh from the relaxed sites. Since every site position represents the center of a primitive, the relaxed Voronoi diagram, hence the kNN graph, only gives the dual of the anticipated hex-mesh, which its vertices should define. Geometry and topology are gathered for the at-most-hexa mesh by utilizing all the information that was accumulated for each site during the relaxation: 3D position and orientation, cell extent and volume, N_6 direct neighbors, in/out state, and n-hop distance. Simply put, our approach is to place actual hexahedra in all cells and fuse them wherever trivially possible. A closed mesh can still be guaranteed by introducing non-hex primitives where necessary.

The strict mechanisms for extracting the at-most-hexa mesh are designed to make inverted primitives impossible. Therefore, the resulting mesh does not contain negative Jacobians.

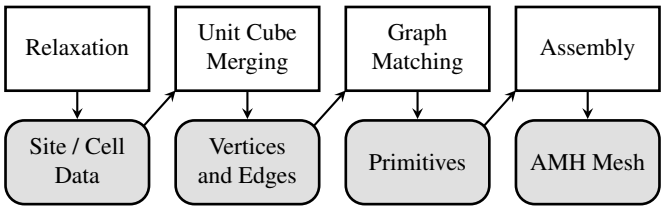


Figure 6: *The post-relaxation domain: Extracting vertices and edges from relaxed sites (5.1) followed by the collection and assembly (5.2) of at-most-hexa primitives for the final mesh.*

5.1. Stage One: Geometry

The geometric basis for the further steps is based on the materialization of all Voronoi cells with small cubes centered on their sites' position. This contrasts previous L_p-CVT based hex-dominant meshing concepts [LL10, BRM*14, SRUL16], which solely operate on the bi-graph of the relaxed diagram, by assembling hexahedral primitives from the Delaunay tetrahedralization. In our approach, each cube is scaled uniformly to approximate the extent of its corresponding cell and is rotated to the site's orientation. Boundary cells are ensured to grow such that the boundary faces align with the input hull or point cloud. This stage is illustrated in Figure 7 and as example in Figure 1 (center,left).

One can easily comprehend how two neighboring cubes should be connected: Take the quad of each cube that is facing the other cube and merge them into one. The eight *virtual* vertices of these two *virtual* quads shall become four *real* vertices of one *real* quad.

This task may sound fairly simple, but it is rather complex to determine robust connections geometrically. The left of Figure 7 shows relaxed cells; the right illustrates how uniform cubes are placed in this scenario. A simple snap-merge approach could succeed on very regular structures, but as soon as cells approximate curved surfaces, the distances between potential merge partners vary heavily due to keystone deformations of the cells. A merge criterion only based on geometric distance is therefore not very robust.

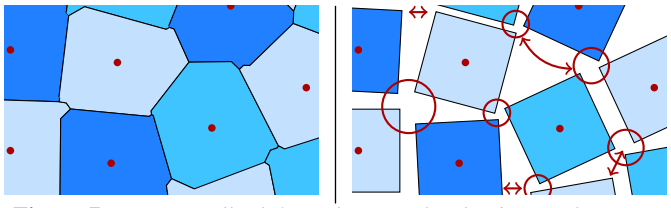


Figure 7: *Voronoi cells (left) and materialized cubes (right). Due to the cell's keystoning, the distances between virtual vertices may vary significantly and is therefore not a robust criterion for a merge.*

Our algorithm incorporates topology information provided by the six nearest neighbors N_6 of each node to approach this issue in the merging step. If there is a mutually unique link established between two cubes, they can be trivially interconnected. However, while the regularization in Section 4.4 vastly improves the number of mutually unique relationships, they cannot be established everywhere. Some node might be considered as a neighbor to fewer or more than six other nodes. Those complicated cases will be considered after the trivial cases.

Matching Scores To find the best suitable matches of the cube's quads to be merged, we first determine a score for each pair. Therefore, the merging order determines by a global score-sorted priority queue over all possible faces that could fuse two cubes. For each face of a cube, the algorithm queries suitable faces of all six neighbors and computes a geometric score φ as formulated in Equation 5, expressing how well the two cubes match.

$$\varphi(U,V) = \left(2 - \frac{\arccos(\vec{n}_u \cdot \vec{n}_v)}{\pi}\right) \sum_{i \in (0,1,2,3)} ||u_i - v_i||_2 \quad (5)$$

where u_i are vertices of face U, v_i vertices of face V and $\vec{n}_u, \vec{n}_v$ are the face normals respectively. Assuming the best suitable permutation for the vertices on the other face has been determined, the scoring function computes the accumulated pairwise distance between face vertices scaled by an angular component based on the face normals. For normals perfectly facing each other, the angular factor is 1, and it can grow up to 2 for face normals pointing in the same direction. Face pairs and corresponding neighbors are collected in the priority queue, sorted by ascending score values. Faces for which there is no inside neighbor are labeled as part of the outer hull of the final hex-mesh.

Vertex Merging With the face-merging order in place, we have to find a suitable way to merge vertices. In an entirely regular scenario, the corners of eight cubes would make up one vertex for the final mesh. However, in an irregular arrangement, sometimes more or fewer than eight *virtual* vertices make up one *real* vertex. Positions of *real* vertices in the final hex-mesh are set to be the geometric centers of the associated sets of *virtual* vertices. By working off the priority queue, the information which *virtual* vertices are to be merged comes in serial form and has to be assembled entirely before the vertices can actually be combined. Therefore, the merging process itself is split up into two steps: In the first run, all merge-relevant information is collected and the *virtual* vertices accumulate in merge-sets. In the second run, all merge-sets compute the *real* vertex position for the final hex-mesh.

Processing the priority queue As mentioned before, the majority of all cubes can be connected straightforwardly, but some may require extra care. Our implemented algorithm to process the priority queue follows a simple defensive strategy promoting only high-quality mesh output. Therefore, a bit-field is maintained, which keeps track of already merged and unmerged faces. If a face pair is up-next in the priority queue and one of the faces is already flagged as merged, the pair is skipped and left open.

Some of the skipped faces, e.g., as in the ambiguous assignment problem in Figure 8, are resolved implicitly by the vertex merging step. A cube's edge can collapse if two *virtual* vertices are in the same merge-set. Suppose this occurs on two opposing edges of a quad-face, then one side of a cube collapses to an edge.

Closing Cuts So-called *cuts* occur when there is a gap in the neighborhood topology at the time when matching scores are computed. These topological gaps are totally valid and relatively easy to fix by running another matching-score iteration before merging the vertices. If two cubes were not direct but indirect neighbors, it might happen that two of their *virtual* vertices will be together

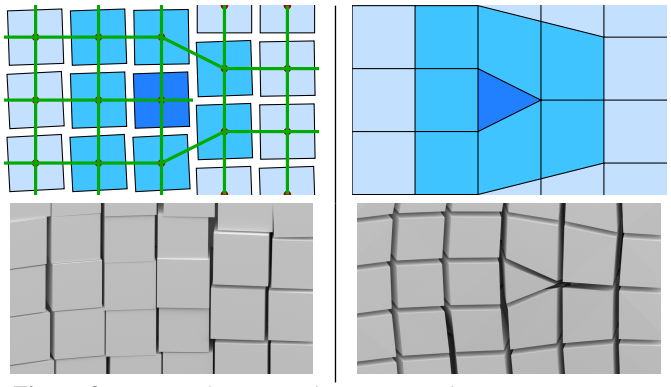


Figure 8: *Despite the N_6 regularization, ambiguous scenarios as on the left may still occur. The result on the right emerged from the automated matching in which a face is collapsed to an edge.*

in the same two merge-sets. If those two merge-sets form an edge that connects open faces from these cubes, they will be considered neighbors now, and the faces can be merged as well. Figure 9 illustrates this process in theory. A very prominent result can be found in Figure 15 on the long rounded vertical corner of our *Fandisk* result in the rightmost image.

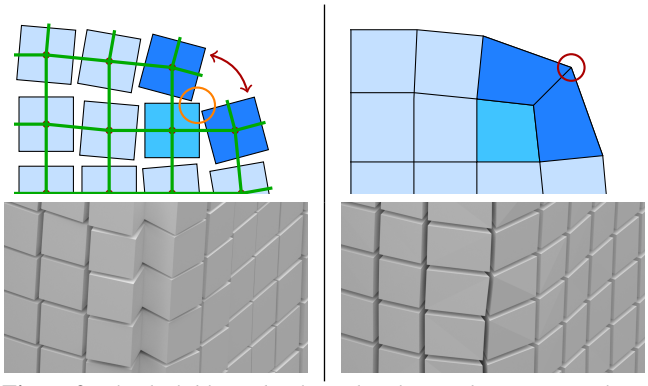


Figure 9: *The dark-blue cubes have dangling nodes (green) in their N_6 neighborhood. Indirect merges (red) are possible over common edges (orange) at small enclosing angles.*

After these steps, the positions of all *real* vertices are computed by averaging over all *virtual* vertices contained in their respective merge-sets. In some cases, the vertices generated by a merge can be a bit off from the anticipated surface. For example, mainly the narrowing geometry of the *Jumpramp* in Figure 18 provokes such merges, which in combination with the concave 90° edge, can cause displaced vertices. However, a simple optimization step, pulling vertices onto the input surface (or a $\mathbf{S}_{pc}$ surface patch in case of point cloud input) would allow for an easy fix in such scenarios, e.g., using the energy-term formulated for feature-aligned vertex placement in established surface-meshing methods [JTPSH15].

5.2. Stage Two: Topology

This is the point where actual mesh topology comes together. So far, the simple match-and-merge algorithm has only generated the final mesh vertices and associated edges derived from the

merged cube structures. This edge network already features trivial triangular and quadrangular faces, suitable for at-most-hexa primitives. However, this complex structure is not yet entirely suitable for our objective as it also includes constellations that are unresolvable with at-most-hexa primitives. We can identify such regions as spirals, shown in Figure 10, and resolve them by inserting additional edges. This can be done simultaneously to the identification of trivial tri- and quad-faces (**I.**). On this basis, small graph matching state-machine programs, as shown in Figure 22, extract all possible at-most-hexa primitives (**II.**) from the edge-network. The final mesh is then assembled (**III.**) from a quality-sorted priority queue.

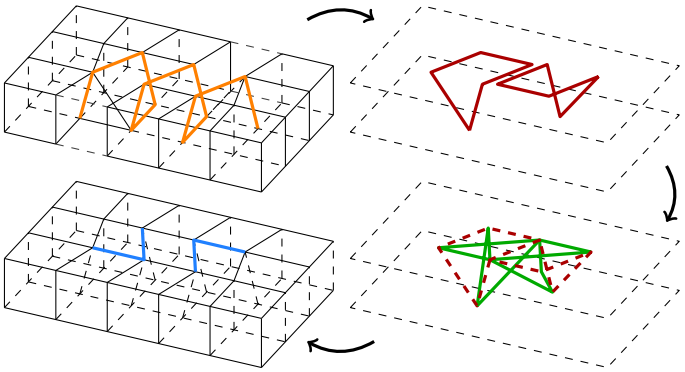


Figure 10: *Spiral structures can not be represented with at-most-hexa primitives. Identified penta-loops are supplemented with additional edges, which are only used where needed in the final mesh.*

I. Collect Faces Specified at-most-hexa primitives are based on triangular and quadrangular faces found in the given edge complex. Triangular faces are identified as loops of three adjacent edges or four edges for quadrangular faces, respectively. However, there are also cases where adjacent edge paths do not form closed loops of length three or four. As shown in Figure 10, topological missconfigurations can be caused during the relaxation in narrowing geometry and can not be resolved using at-most-hexa primitives, thus would cause holes and missing primitives in the final mesh. Spiraling edge structures are identified and broken down, therefore, becoming tri- and quadrangulatable. In an iteration over all edges, a set of penta-loops is gathered, defined as five adjacent edges, not interconnected by any other existing edge. Spirals can always be broken down into one or more (overlapping) penta-loops, which can be easily split up by creating five new interior edges per loop. Regardless of being a prior existing or newly added edge, they are only featured in the final mesh as part of a fitting primitive. Furthermore, it is suitable to split very skewed or non-planar quadrangular faces into two triangles by inserting a diagonal edge in some cases. For trapezoidal quads, we chose $\frac{\pi}{4}$ as the lower limit for corner angles and $\frac{\pi}{3}$ as the upper limit between opposing edges on trapezoidal quads. The shortest diagonal of such a quad is then added and treated equally to prior existing edges.

II. Graph Matching Algorithm Based on the collected tri- and quad-faces, our algorithm now generates the basic building blocks for the final mesh assembly, namely the at-most-hexa primitives. This poses a straightforward graph matching task, but the greedy search's complexity escalates quickly if not approached with care. With insight into possible outcomes, one can specify a strict set of rules to prune the search tree drastically and avoid incredible amounts of redundancy early on.

The algorithm in Listing 1 iterates over the available faces, generating the individual primitive types. The outer loop can be parallelized, such that all primitive types are processed at the same time. Pruning is achieved as the algorithm obeys the following rules:

- Every face f_i is considered as a possible starting point to assemble a primitive.
- Once a face f_i was a starting point, all possible primitives featuring f_i have been explored.
- Search paths from other starting points f_j including f_i would result in redundant results.
- Therefore, used starting points f_i are marked and allow for an early termination of redundant search paths f_j.

The state-machine programs featured in Figure 22 are designed for as little branching as possible. Therefore, a triangular face is favorable over a quad as a primary face due to only three open edges, hence branching directions; except for the hexahedron obviously and the pyramid due to symmetry. This preselection of suitable tri/quad faces is also implemented in the following algorithm. Our face-based state-machines can be formulated with as little as only *one*, or at maximum *two*, possible assembly sequences. This contrasts common tet-based hex-assembly [MT00,LL10,BRM*14,SRUL16], where the hexahedron alone can be formulated in 10 different constellations, featuring 5, 6 or 7 tetrahedra plus sliver elements.

```
1  primitives = {}
2  facesUsed = {}
3  for primType in [8,7,6,5,4,1]:
4      primitives[primType] = []
5      facesUsed[primType] = zeros(numFaces)
6      for fi, face in enumerate(faces):
7          if face is quad and primType in [8,4] \
8          or face is tri and primType in [7,6,5,1]:
9              newPrims = findPrims(fi, primType)
10             primitives[primType] += newPrims
11             facesUsed[primType][fi] = 1
```

Listing 1: *Loop over all faces with specialized state-machine algorithms, as shown in Figure 22. The outer loop may be parallelized for the individual at-most-hexa primitive types.*

Listing 2 describes the general algorithm to generate at-most-hexa primitives from a given starting face f_i, as illustrated in Figure 22. While there are unfinished primitives in the `openPrims` list, the algorithm queries for adjacent, unused and type-suitable candidate faces f_j (line 7). New candidate faces explicitly qualify by sharing one of the open edges in the unfinished primitive. A new primitive struct is generated by adding face f_j to the open primitive (line 8). If this leads to a complete primitive, it is added to the result set (line 10). If the new primitive is not yet complete but a valid state, it is added to the set of open primitives for the next round (line 13). If neither case is satisfied, the new primitive is an invalid state and rejected. For the same starting face f_i, the algorithm may return multiple primitives, i.e., two hexahedra sharing a common quad face, but never any duplicates.

```
def findPrims(fi, primType):
    donePrims = []
    openPrims = [[fi]]
    while len(openPrims):
        newOpenPrims = []
        for openPrim in openPrims:
            for fj in cFaces(openPrim, primType):
                newPrim = openPrim + [fj]
                if done(newPrim, primType):
                    donePrims.append(newPrim)
                    continue
                if valid(newPrim, primType):
                    newOpenPrims.append(newPrim)
        openPrims = newOpenPrims
    return donePrims
```

Listing 2: *The search for all possible primitives of a given type including a specified starting face.*

For the upcoming assembly routine, the at-most-hexa primitives are sorted by quality within their class. Minimum Scaled Jacobians (MSJ) are employed as an intuitive quality property. Not all primitives support this property as trivially as the hexahedron with 8 suitable vertices. However, as the primitives derive from a hexahedron by collapsing edges, each primitive trivially maps to a unit cube. Collapsed edges result in a Jacobian of 0 on affected vertices. Therefore, Jacobians are computed on non-hex primitives only where possible, namely on vertices connected to three edges.

III. Assembly With all available primitives at hand, we will now focus on the concept of assembling a full mesh. The upper histogram in Figure 12 shows the proportional amount of the different types from *all* collected primitives. As illustrated, each collection of primitives is internally sorted by quality (MSJ). With the following algorithm, the final mesh is assembled by incrementally adding the best suitable primitives to the existing set as exemplified in Figure 11. The relaxation produces a hex-dominant mesh, where the majority of hexahedra can be adopted on the fly to initialize the assembly's starting point in Listing 3: All available hexahedral primitives (type **8**) with a certain quality (MSJ > 0.5) are directly added to the `amhMesh` set. If there are conflicting hexahedra (i.e., partial overlaps) within this set by initialization, the lower quality primitives of conflict-pairs are removed until `amhMesh` is conflict-free. Further, faces used by two adjacent primitives are considered closed. Faces used only once are collected in the `openFaces` set.

The algorithm's design ensures it prioritizes larger (favorably hexahedral) elements of high quality. Smaller primitives serve as a fallback solution, especially the tetrahedron, as a last resort to fill up the smallest gaps in the volume. Therefore, the criteria for primitives to be considered candidates start high and will be automatically lowered if there are no elements to be added with the current settings, and reset if there was progress again.

Each iteration of the assembly algorithm in Listing 3 consists of three phases: • **First** (line 4-11) a preselection, where the main criteria for being considered in the next step are the `primType`, the minimum number of how many open faces a primitive should close `cLim`, and a minimum quality threshold `minQ` which we set at 0.25. Suitable primitives are collected in the `newPrims` set and sorted (lexicographically) by their qualification criteria; first by their two integer keys (primitive type and the number of closeable faces), then the float quality measure: `primType > c > prim.Q`. Therefore, the first `newPrim` element of this sorted list has the maximum primitive type, closes the most open faces, and is of the highest quality. • **Second** (line 13-19), primitives are added but only if they are not in conflict (`primInConflict()`) with the existing mesh. Our conflict definition follows the relaxed interface-conformity constraints for hex-dominant meshes [YS03], prohibiting partial overlaps or inclusions, i.e., two prisms in a hexahedron. If it is safe, `newPrim` is added to the `amhMesh` set and the `openFaces` set is updated with a *symmetric difference* set operation (△ in the pseudo-code). Possible conflicts may arise for `newPrims` with each newly added primitive, while they still await their turn, queued in `newPrims`. Therefore, this check has to be performed individually and not for all elements in the preselection. • **Lastly** (line 21-26), the qualification constraints for the next addable primitives are either lowered or reset.

```
primType = 8
cLim = 4
while primType > 0:
    newPrims = []
    for prim in primitives:
        prim.c = |openFaces ∩ prim|
        if prim.t >= primType \
        and prim.c >= cLim \
        and prim.Q > minQ:
            newPrims.append(prim)
    newPrims = lexSort(newPrims, keys = [t,c,Q])

    primsAdded = 0
    for newPrim in newPrims:
        if not primInConflict(newPrim):
            amhMesh.append(newPrim)
            openFaces = openFaces △ newPrim
            primsAdded += 1
        primitives.remove(newPrim)

    if not primsAdded:
        primType -= cLim < 4
        cLim = max(2, cLim-1)
    else:
        primType = 8
        cLim = 4
```

Listing 3: *Assembly algorithm to construct the at-most-hexa mesh.* `amhMesh` *is initialized with non-conflicting hexas (≥* `minQ`*).*

The lower histogram in Figure 12 gives the composition of the resulting at-most-hexa mesh after the assembly algorithm in Listing 3, most dominantly featuring hexahedral elements. However, the assembly routine is not necessarily as straightforward as Figure 11 suggests; the enclosing `while` loop is allowed to lower and reset the adding criteria multiple times. Further, the loop only may terminate when there was not a single tet (type **1**) left that would close up any more faces. Consequently, as the tetrahedron is the smallest possible primitive, the final mesh is completely filled up.

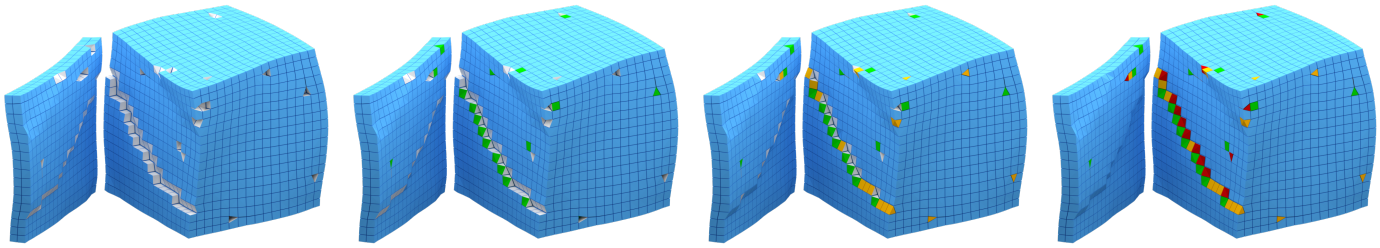


Figure 11: *Progress (left to right) of the assembly routine on a cut open example of the Twistcube. This visualizes iterations of the* `while` *loop in Listing 3, lowering the* `primType` *each round (8 hex > 7 hexPrism > 6 prism > 5 slice) to close open faces (white). The start* `amhMesh` *on the left is initialized with hex-only elements of high quality.*

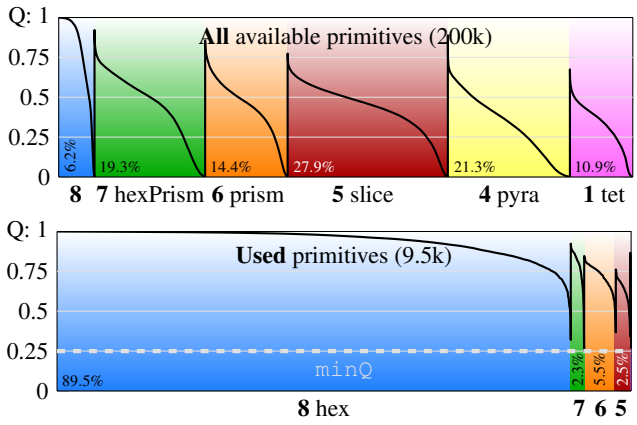
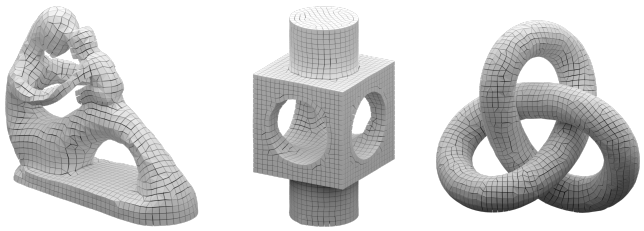


Figure 12: *Quality histograms of primitives for the Fandisk model, lex-sorted by* `hexType` *then quality (MSJ). Stats are shown before (top) and after the assembly (bottom). The final mesh only features high-quality (>*`minQ`*) primitives, primarily hexahedra.*

6. Experiments and Discussion

We have tested our construction strategy on a number of examples.



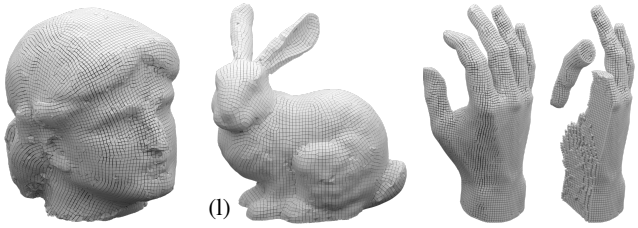
#: 75.8% | V: 86.9% #: 87.8% | V: 93.9% #: 65.0% | V: 81.1%

Figure 13: *The percentage of hexahedral cells (#) and volume (V) obtained by our construction algorithm, on models with complex geometries and high genera.*

6.1. Requirements on input

One strength of our construction strategy is the versatility in terms of input. The input shape can be given as a triangular boundary mesh, a generic hex-dominant volume mesh, or even, as described in the dedicated Appendix B, an unstructured point cloud.

Normal and orientation initialization is sampled from the input hull or point cloud. Surface orientation can be robustly extracted using winding numbers [BDS*18] for meshes, or various strategies for point clouds [JBG19]. A frame-field is not required, and cell orientations get implicitly aligned to boundaries during the relaxation. The method is robust with varying sampling density in the input, and the input boundary is not required to be closed (as exemplified by the *Bunny* dataset and the *Minerva* dataset in Figure 14, which are open at the bottom). Manifolds of higher genera and complex geometries can also be meshed correctly, as shown in Figure 13.



#: 77.6% | V: 88.9% #: 76.7% | V: 88.2% #: 81.5% | V: 90.8%

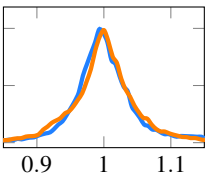
Figure 14: *Minerva [Bol09] features over 100k primitives, reconstructed from a point cloud of only 8k points. The Bunny [Sta14] and the Hand were given as meshes and feature around 48k each. Stats show the percentage of hexahedra as: #: number | V: volume.*

6.2. Experimental Results

We tested our construction method on several 3D objects, both with a mechanical and an organic shape. For comparison purposes against competing approaches, we included popular 3D test objects. We used the real 3D scan data of *Minerva* in Figure 14 and synthetic point cloud examples for our results of the *Cylinder* in Figure 19 and *Igea* in Figure 25. Table 1 lists numerical evaluations of our results compared to other approaches. Shown results and measurements reflect the native outcome of our procedure, without any further optimization, which would be possible [LSVT15].

Comparisons & Quality Table 1 offers comparisons, in terms of mesh quality, of the results obtained with our method with the ones resulting from existing hex-meshing approaches. As confirmed by direct visual comparison, shown in Figure 19, our results are on par with the state-of-the-art in terms of quality. This is in spite of our method working with much fewer assumptions on the input, which is a main motivation in our work. Many competing method methods require a starting tet-mesh, and additional inputs such as additional frame-fields [LLX*12, SRUL16, SVB17, LPP*20] (which can be computed as part of the method, such as in [GJTP17]), volumetric parameterizations [GSZ11, LBK16, RSR*18, CAS*19], and hand-crafted singularity graphs [LZC*18, CC19], dual-sheets [Tak19] or feature-edge selections [LPP*20].

These required inputs can be constructed in separate, non-trivial and preliminary steps, each subject to own lines of research: E.g., the construction of a tet-mesh from a boundary tri-mesh [SJ08, Si15, HZG*18], of a tri-mesh from a point cloud [BTS*17], of a volumetric directions-field from boundary geometry [BRM*14, SVB17], or sharp-feature lines from a boundary mesh [MAR*20]. Despite the recent advancements in each of these fronts, we consider it advantageous to bypass the need for these tasks.



As the relaxation strives to maximize uniformity, equidistantly distributed sites generate primitives of approximately the same size. The inset histogram shows the distribution of edge lengths of the *Fandisk* model of Gao et al. compared to our result. The plot is zoomed in on the median peak, which was scaled to 1.

Figure 12 reports quality measure histograms for the *Fandisk* model of all gathered primitives from the relaxation (top) and the used ones in the final mesh (bottom). The hexahedral cells in our output feature very regular shapes and a high Average Scaled Jacobians [PTS*08], matching or even superior to the ones obtained with state-of-the-art procedures. As expected, the Minimum Scaled Jacobians never fall below our threshold `minQ` of 0.25. In terms of the number of hexahedral elements, the proportions of hexahedral primitives in our at-most-hexa meshes ($> 80\%$) exceed recent hex-dominant procedures. Feature alignment and homogeneous edge-flow is similar to the one obtained with existing meshing algorithms focusing on this feature [GSZ11].

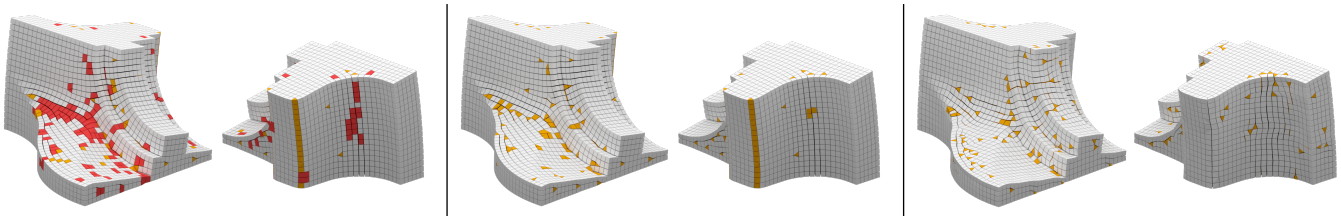
Figure 16 shows a comparison that highlights the limitation of existing all-hex methods [Tak19, LPP*20], in terms of reliance on complex-to-produce input. In these cases, irregular cell shapes, widely varying element sizes, and inverted elements are produced with the competing strategies. Authors of LoopyCuts [LPP*20] attribute this failure case to a limitation of their procedure, which only performs well on suitable input frame-fields with evenly distributed singularities.

Figure 15 illustrates another direct comparison with the hexa-dominant result of Gao et al. [GJTP17], highlighting all non-hexahedral elements. The crucial difference is that their results feature generic polyhedra with more than eight vertices and faces with more than four vertices. In contrast, our method produces *at-most-hexa* meshes, with the consequences discussed in Section 1. In this experiment, we modify our construction algorithm to refine an existing generic hexa-dominant mesh into one valid at-most-hexa mesh: we simply skip the relaxation part to generate the geometry and directly break down the input structure, and then reassemble the final elements as usual.

As an experiment, we also tested the same procedure starting from existing irregular tet-meshes, such as the ones produced by a *TetWild* [HZG*18] (see results in Figure 20). In this case, the input fails to provide any flow or direction information to guide the construction, so it is expected that the results will be less regular and less hexa-dominant. Nevertheless, by lowering the quality threshold `minQ` to 0.1, the assembly routine managed to recover 21.4% hexahedral elements.

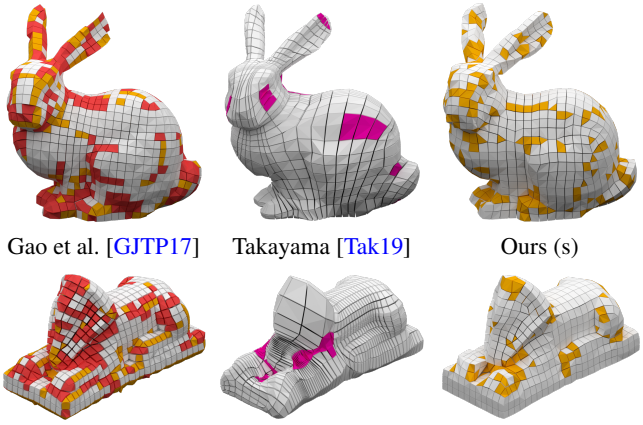
		# prims	hex (%)	vol (%)	MSJ	ASJ	t
anc101	Ours	84 551	**93.3**	96.9	**0.261**	**0.983**	1590
	[LL10]	105 000	77.1	-	-	-	720
	[GSP19]*	188 886	-	-	0.094	0.865	46.3k
aries	Ours	14 004	94.8	97.6	**0.255**	**0.987**	1255
	[GSP19]*	22 547	-	-	0.092	0.813	4568
bunny	Ours (l)	49 243	**76.7**	88.2	0.252	0.939	1639
	Ours (s)	3201	**71.4**	84.9	0.258	0.935	153
	[SRUL16]	-	60.6	88.6	-	**0.950**	469
	[RSR*18]	-	-	**92.2**	-	-	-
	[GJTP17]	2135	66.2	65.2	0.285	0.953	-
	[Tak19]*	2832	-	-	–0.771	0.749	-
	[GSP19]*	29 698	-	-	0.292	0.790	-
	[LPP*20]*	2172	-	-	**0.451**	0.911	112
cylinder	Ours (g)	1665	**93.4**	97.0	**0.373**	**0.973**	22
	Ours (n)	1671	85.8	93.1	0.298	0.964	16
	[SRUL16]	-	64.7	90.9	-	0.960	327
	[RSR*18]	-	-	**99.3**	-	-	-
fandisk	Ours	9523	**89.5**	**95.3**	0.303	0.973	780
	[SRUL16]	-	51.4	77.8	-	0.969	10
	[GJTP17]	7069	88.4	87.7	**0.668**	**0.986**	-
	[Tak19]*	1774	-	-	0.217	0.905	-
fertility	Ours	4997	**75.8**	**86.9**	**0.261**	0.946	645
	[SRUL16]	-	33.6	78.4	-	0.930	1121
	[GJTP17]	4769	72.0	72.6	–0.330	**0.960**	1429
hanger	Ours	12 706	91.1	95.8	0.256	**0.976**	1905
	[GSP19]*	26 918	-	-	0.155	0.828	1536
	[Tak19]*	1382	-	-	**0.333**	0.944	-
igea	Ours	27 015	76.8	**88.2**	**0.251**	0.941	1228
	[GJTP17]	12 936	**81.0**	81.0	–0.230	**0.970**	-
rod	Ours	7409	87.2	94.1	**0.355**	**0.968**	541
	[GSP19]*	26 918	-	-	0.155	0.828	1536
	[Tak19]*	600	-	-	0.221	0.763	-
sculpt	Ours	5247	77.1	89.1	0.251	**0.934**	667
	[GSP19]*	15 202	-	-	0.104	0.759	826
	[LPP*20]*	168	-	-	**0.806**	0.918	18
sphinx	Ours	2100	**81.5**	**90.9**	**0.312**	0.963	200
	[GJTP17]	2170	76.7	77.9	0.158	**0.971**	-
	[GSP19]*	45 348	-	-	0.182	0.814	-
	[LPP*20]*	3944	-	-	–0.803	0.808	672
Ours	hand	47 132	81.5	90.8	0.253	0.950	521
	holeblock	7491	87.8	93.9	0.250	0.970	175
	jumpramp	1460	97.3	98.7	0.776	0.994	9
	minerva	100 567	77.6	88.9	0.251	0.940	1235
	trefoil	8526	65.0	81.1	0.252	0.917	362
	twistcube (s)	1301	88.9	95.6	0.488	0.978	7
	twistcube (m)	5695	96.2	98.5	0.571	0.992	16
	twistcube (l)	15 841	96.2	98.4	0.316	0.990	76
	twistcube (t)	2836	21.4	41.3	0.125	0.801	-

Table 1: *Quality measures for our results and results obtained with competing approaches. We list the total number of primitives, as well as the proportions of hexahedra and their volume. The quality metric of Scaled Jacobians (best is 1) is given with **M**inimum and **A**verage. The Cylinder (Figure 19) is included as the native (n) and guided (g) version, the Twistcube (Figure 20) with three resolutions (s,m,l) and the tet-based example (t). The total construction time (in seconds) is also reported.*



Gao et al. [GJTP17] | #: 88.36% | V: 87.74% [GJTP17] + Ours | #: 90.6% | V: 96.1% Ours | #: 89.5% | V: 95.3%

Figure 15: *A direct comparison of results for the Fandisk model. Cells with more than six faces (larger than hexahedra), which are avoided in our solutions, are colored in red. Cells that are not hexahedral, but are at-most-hexahedral, are colored in yellow. The center column shows the results we obtain using, as input, the hex-dominant from the left. Stats show the percentage of hexahedral cells (#) and their volume (V).*



Gao et al. [GJTP17] Takayama [Tak19] Ours (s)

Gao et al. [GJTP17] Livesu et al. [LPP*20] Ours

Figure 16: *Results of the Bunny and Sphinx with about the same resolution each. General (larger than hexa) polyhedra are shown in red, smaller ones in yellow and inverted primitives in magenta.*

Comparison to L_p-based Meshing Lévy and Liu [LL10] extensively studied Centroidal Voronoi Tessellations (CVT) under the L_p norm. While their focus lies on the formulation and derivation of L_p-CVTs, they also propose their use for quad and hex-dominant meshing. There are clear distinctions between their approach and ours: Whereas our approach aims to materialize the hexahedral cells of the diagram itself, Lévy and Liu utilize the diagram bi-graph for the geometry extraction. Similar to Sokolov et al. [SRUL16], the graph matching approach of Meshkat and Talmor [MT00] is employed to assemble hex-like cells from the diagram's Delaunay tetrahedralization. As their mesh vertices are actually former cell centroids, they have to compensate for the resulting shrinkage on the most outer hull layer. This is not required in our approach where a cell itself corresponds to a mesh primitive, thus there is no gap between generated geometry and the targeted hull. The concept of Lévy and Liu also does not allow for control over the mesh regularity, as their cell's positional alignment is solely based on the relaxation. In contrast to that, our internal graph structures can enforce specific favorable alignments during the relaxation, as introduced in Section 4.4 with the N_6 graph. Furthermore, our N_k graph provides the possibility to propagate and interpolate orientations throughout the diagram, thus it allows for cells aligned to the input hull but also to each other. In the diagram of Lévy and Liu, individual cell orientations are queried from anisotropy matrices associated with outer hull faces, based on single nearest-neighbor connections. Therefore, adjacent cells are not necessarily similarly oriented but solely depend on their closest connection to the outer hull. This effect is prominently visible in Figure 17 on the cut-open *Anc101* model: In Lévy and Liu's L_p-CVT result, the rounded cap of the pin-cavity dominates large portions of the interior cells. In our result, the small cavity only has a minor influence as the cells primarily align with the dominant outer shape of the model.

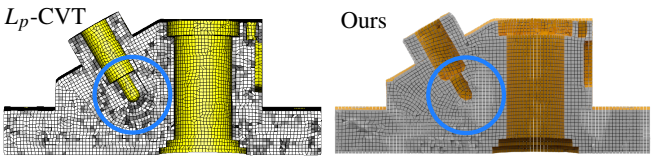


Figure 17: *In the L_p-CVT mesh [LL10] the rounded cavity cap dominates the interior alignment; our cells are more aligned to the outer shape. Here we adopted the color scheme of Lévy and Liu with yellow hull faces and white for the exposed cut-open interior.*

Regularity and Alignment The results presented in Figure 18 are prime examples of the synergy of hull samples and cell orientations during the relaxation with no frame-field given. On planar surfaces, as on the *Jumpramp*, only surface normals are determined robustly, primary curvature directions are just random guesses. Since there was no consistent curvature field, the cell alignment of the flanks dominates during the relaxation, and the orthogonally aligned orientations follow along, so the resulting arrangement is as regular as possible. Nevertheless, hex-like cells also intuitively align to more distinctively shaped geometry like the *Fandisk* in Figure 15, featuring creases, flat, angled, curved, and narrowing regions.

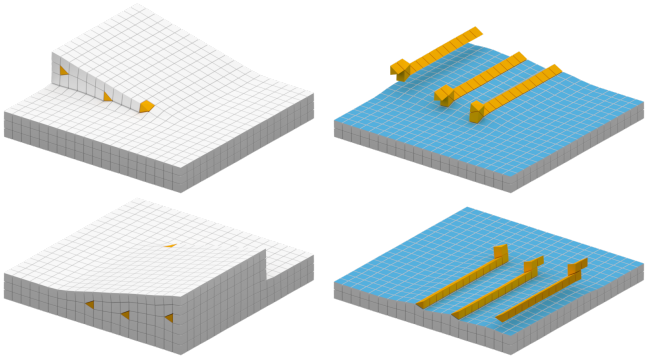
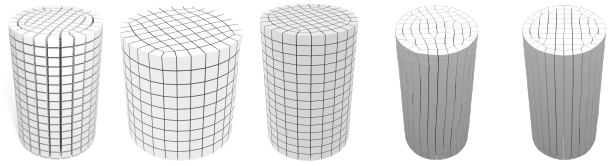


Figure 18: *The Jumpramp is challenging for hex-meshing approaches based on regular parameterizations or integer mappings. Nevertheless, our relaxation is flexible enough to cope with the angled and narrowing geometry. Inner faces are shaded blue, non-hex primitives (all triangular prisms) are highlighted in yellow.*

The relaxation process also promotes homogeneity throughout the whole object. For example, the *Cylinder* in Figure 19 features equally sized primitives in its center as well as on the outer hull. Some hex-meshing algorithms tend to mimic proportions of the hull in deeper layers which leads to an unnecessarily high resolution towards the core of an object [LBK16, SRUL16, SVB17]. Cells are not explicitly bound to stay close to their initial position or neighborhood. Although objects like the *Cylinder* or the *Holeblock* (Figure 13) can't provide much vertical support, cells do not twist out of control and maintain a quite regular vertical alignment.



[LZC*18] [CAS*19] [CC19] Ours (n) Ours (g)

Figure 19: *Results of other recent work using tet-meshes, polycube mappings, and singularity graphs as input compared to ours based on hull points only. This also compares a native (n) outcome of our procedure with an experimentally guided (g) version.*

The choice for initial site positions on an axis-aligned regular grid is well suited for objects that also feature axis-aligned parts like the *Jumpramp*, the *Holeblock*, or the CAD models in Figure 21. Nevertheless, the relaxation is very flexible and able to approximate organic shapes without explicitly axis-aligned parts faithfully. But to be fair, even organic shapes like *Minerva* or *Fertility* are usually not given with an arbitrary rotation but are also often oriented for at least axis-aligned symmetry.

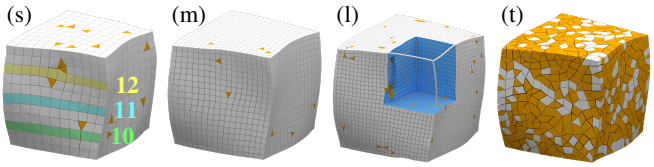


Figure 20: *The Twistcube [JTPSH15] in three different resolutions (s,m,l) and with a cutout. The rightmost cube (t) is based on a Delaunay tetrahedralization [HZG*18] instead of our relaxed input.*

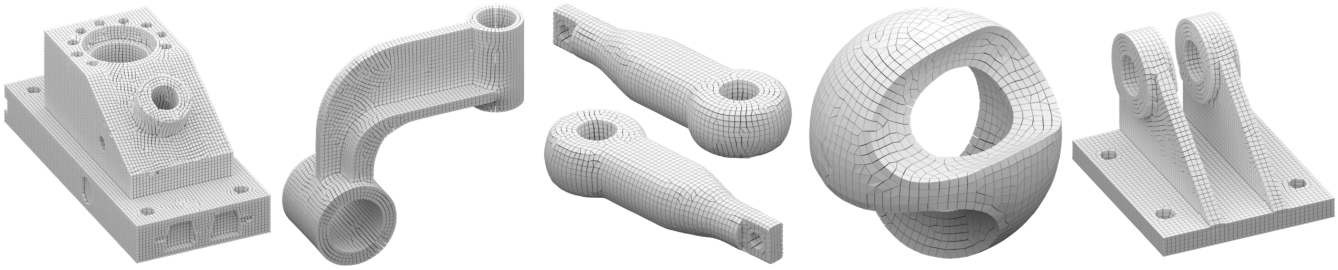
The *Twistcube* in Figure 20 illustrates how the relaxation also maximizes isotropy on objects with non-axis-aligned surfaces: The curved point cloud hull acts as inside/outside separation on the initially regular grid cells. This rasterized initialization is resolved in the relaxation with a varying number of homogeneously shaped primitives instead of squashed and stretched ones.

Guidance Due to the relaxation, the mesh's flow naturally aligns to object curvature, based on surface features alone. Nevertheless, it is quite easy to supplement this process with guiding structures inside the object to control the internal flow-field and orientations. For the *Cylinder* in Figure 19, we extended $\mathbf{S}_{pc}$ with additional samples on two orthogonal planes, intersecting on the rotational axis of the object, similar to dual-sheet meshing [Tak19]. This pushes the anticipated regularity for simple geometric shapes even further by guiding the relaxation to obey symmetry or similar characteristics.

Performance Due to the decentralized graph structure (Section 4.2), the relaxation part of our pipeline is easily parallelized. Timings were measured with an implementation in *CUDA*, run on a *GeForce GTX 1080Ti* graphics card. The performance of the relaxation heavily depends on the selected parameters, e.g., grid resolution and the dimensions of the input structure. Included results were created using 150 relaxation iterations. The *Minerva* object, one of the larger reconstructions listed in Table 1, features about 225k Voronoi cells (including the space outside of the object). One complete relaxation iteration includes: Computing Voronoi cells and re-centering their sites, updating the N_6 and N_{26} neighborhood graphs and n-hop distances, updating and interpolating separate individual positions and orientations, and combining them with the progress-dependent weighing function. With a neighborhood size of $k = 26$, one iteration for these 225k elements is done in 2.43s. For reconstructions of coarser resolution, e.g., the *Jumpramp* model, our implementation reaches about 18 iterations per second. The vertex merging steps for geometry generation as well as the graph matching for topology extraction are also easily parallelizable and finish within a few seconds on a multithreaded Python CPU implementation: As described in Section 5.2, the outer loop in Listing 1 can be parallelized for the individual primitive types to be acquired and sorted simultaneously. How much time is spent in each step varies and depends on the chosen parameters as well as on the input object: The GPU relaxation time correlates with the chosen mesh resolution, thus the overall number of cells in the diagram. An object's shape directly influences the time that is consumed by the primitive-collection and mesh-assembly routines. For blocky objects like the *Twistcube* in Figure 11, the assembly routine starts with a strongly hex-dominant initialization. In organic shapes or contorted CAD models like the *Fandisk* in Figure 15, the initialization contains larger quantities of non-hex cavities to be fill-up, hence also a larger pool of at-most-hexa primitives to choose from. This assembly routine (Listing 3) is the only serial CPU operation in our pipeline. Nevertheless, the straight ordering of the priority queue (Figure 12) allows for an efficient execution of the assembly, which also terminates within minutes.

An extensive comparison of *number-of-primitives vs. time* stats with a recent publication might be rather misleading due to the following reasons: Our relaxation currently (see Outlook) operates on all cells within the objects bounding box volume. And as mentioned above, filigree objects (Figure 13) would do rather poorly compared to massive blocky objects (Figure 20). On the other hand, timings of other approaches with similar primitive count usually also heavily depend on the task to be solved and the given input conditions. Our heterogeneous mixture of Cuda GPU-, single-, and multithreaded Python CPU code introduces further bias on the comparison with single-core C++ implementations of competing methods [LL10,GJTP17,Tak19,LPP*20]. Nevertheless, Table 1 lists our measured timings for the included result.

Figure 12 lists about 200k gathered primitives of which only 9.5k were used in the final mesh. This brute-force approach seems wasteful but still outperformed more advanced alternatives. An interleaved routine, alternating assembly steps with primitive collection queries *only where needed* caused overall too much overhead. The simplest method proved to be the fastest: Gather primitives in parallel and run the assembly on sorted data.



#: 93.3% | V: 96.9% #: 91.1% | V: 95.8% #: 87.2% | V: 94.1% #: 77.1% | V: 89.1% #: 94.8% | V: 97.6%

Figure 21: *More results on CAD models. Stats show the percentage of hexahedra as: #: number | V: volume.*

6.3. Outlook

Additional Guidance As experimentally introduced in Figure 19, the relaxation can be supplied with guidance from additional geometry. Possible scenarios in future research could be to explore the utilization of common and more explicit guiding structures. Dual-sheets, frame-fields, or singularity graphs are currently not required in our method but could improve result quality.

Non-uniform Cells Cells of non-uniform size and/or shape bear geometric challenges which have yet to be explored. However, solving them may pay out in the form of improved hex-mesh quality or adaptive meshing options. Formulating non-hex cells in the relaxation or cells scaled anti-proportional to the n-hop distance would allow for a more resourceful and detail-focused evaluation.

Exclude far away Cells To further improve the relaxation performance, it could be beneficial to investigate mechanisms to exclude far-away outside cells at an early stage of the relaxation. A combined criterion with an n-hop distance > 1 would already exclude many cells from further computation, probably without too much impact on the closest relevant cells on the hull's inside.

7. Conclusion

In this work, we present a new and innovative way to construct at-most-hexa mesh structures. Further, our proposed procedure succeeded in bridging the gap between surface point clouds and volume hex-mesh generation. The quantity and quality of hexahedral elements in our meshed results are improved compared to established state-of-the-art algorithms. We introduce at-most-hexa meshes, a novel class of hex-dominant meshes, where non-hex elements are linear combinations of hexahedra. This allows for trivial interpolation of scalar or vector signals within the volume and greatly simplifies the internal representation due to the suitability for indexed meshes. Contrasting common requirements for such tasks, arbitrary meshes or point clouds of various sizes and densities are sufficient as input. The resolution for the resulting mesh can be selected independently. The proposed approach consists of two parts, starting with a Lloyd relaxation that eventually reintroduces constrained regular structures. In contrast to previous L_p relaxation methods using Delaunay tetrahedralizations, our geometry extraction is based on the actual materialization of the relaxed Voronoi cells. At-most-hexa primitives are extracted with specialized state-machine programs, and the final mesh assembles from a quality-sorted priority queue. The interaction between fixed hull or point cloud samples and relaxed cells generates a homogeneous orthogonal vector field for feature-aligned mesh structures. Therefore, the approach is suitable to reconstruct organic and curved objects as well as flat surfaces, sharp edges, or angled geometry with well-aligned hexahedral primitives.

Acknowledgments

This work has been partially funded by the *Deutsche Forschungsgemeinschaft* (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC number 2064/1 – Project number 390727645.

References

[BDS*18] BARILL G., DICKSON N. G., SCHMIDT R., LEVIN D. I., JACOBSON A.: Fast winding numbers for soups and clouds. *ACM Transactions on Graphics (TOG) 37*, 4 (2018), 43. 5, 11, 20

[BKP*10] BOTSCH M., KOBBELT L., PAULY M., ALLIEZ P., LÉVY B.: *Polygon mesh processing*. AK Peters/CRC Press, 2010. 4

[BL18] BUKENBERGER D. R., LENSCH H. P. A.: Hierarchical Quad Meshing of 3D Scanned Surfaces. *Computer Graphics Forum 37*, 5 (2018), 131–141. doi:10.1111/cgf.13497. 3

[BLP*13] BOMMES D., LÉVY B., PIETRONI N., PUPPO E., SILVA C., TARINI M., ZORIN D.: Quad-mesh generation and processing: A survey. *Comput. Graph. Forum 32*, 6 (Sept. 2013), 51–76. doi:10.1111/cgf.12014. 1

[Bol09] Dept. of Math. Bologna University Scan Repository, 2009. Online; accessed Oktober-2017, http://www.dm.unibo.it/~morigi/homepage_file/research_file/scan_db/res_scan.html. 11

[BRM*14] BAUDOUIN T. C., REMACLE J.-F., MARCHANDISE E., HENROTTE F., GEUZAINE C.: A frontal approach to hex-dominant mesh generation. *Advanced Modeling and Simulation in Engineering Sciences 1*, 1 (2014), 8. 3, 5, 7, 9, 12

[BTP*19] BRACCI M., TARINI M., PIETRONI N., LIVESU M., CIGNONI P.: Hexalab.net: An online viewer for hexahedral meshes. *Computer-Aided Design 110* (2019), 24 – 36. URL: https://www.hexalab.net/, doi:10.1016/j.cad.2018.12.003. 17

[BTS*17] BERGER M., TAGLIASACCHI A., SEVERSKY L. M., ALLIEZ P., GUENNEBAUD G., LEVINE J. A., SHARF A., SILVA C. T.: A survey of surface reconstruction from point clouds. In *Computer Graphics Forum* (2017), vol. 36, Wiley Online Library, pp. 301–329. 12

[CAS*19] CHERCHI G., ALLIEZ P., SCATENI R., LYON M., BOMMES D.: Selective padding for polycube-based hexahedral meshing. In *Computer Graphics Forum* (2019), vol. 38, Wiley Online Library, pp. 580–591. 2, 3, 11, 14

[CC19] CORMAN E., CRANE K.: Symmetric moving frames. *ACM Trans. Graph. 38*, 4 (2019). 3, 11, 14

[CI00] CALVO N. A., IDELSOHN S. R.: All-hexahedral element meshing: Generation of the dual mesh by recurrent subdivision. *Computer Methods in Applied Mechanics and Engineering 182*, 3-4 (2000), 371–378. 2

[CS98] CONWAY J. H., SLOANE N. J. A.: *Sphere packings, lattices and groups*. Springer, 1998. 5

[CW10] CHANG H.-C., WANG L.-C.: A simple proof of thue's theorem on circle packing. *arXiv preprint arXiv:1009.4322* (2010). 5

[DEJ06] DU Q., EMELIANENKO M., JU L.: Convergence of the lloyd algorithm for computing centroidal voronoi tessellations. *SIAM journal on numerical analysis 44*, 1 (2006), 102–119. 6

[DML11] DONG W., MOSES C., LI K.: Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web* (2011), ACM, pp. 577–586. 6

[EBCK13] EBKE H.-C., BOMMES D., CAMPEN M., KOBBELT L.: Qex: robust quad mesh extraction. *ACM Transactions on Graphics (TOG) 32*, 6 (2013), 168. 2

[GJTP17] GAO X., JAKOB W., TARINI M., PANOZZO D.: Robust hex-dominant mesh generation using field-guided polyhedral agglomeration. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 114. 2, 3, 11, 12, 13, 14

[GPW*17] GAO X., PANOZZO D., WANG W., DENG Z., CHEN G.: Robust structure simplification for hex re-meshing. *ACM Transactions on Graphics 36*, 6 (2017). 3

[GSP19] GAO X., SHEN H., PANOZZO D.: Feature preserving octree-based hexahedral meshing. In *Computer Graphics Forum* (2019), vol. 38, Wiley Online Library, pp. 135–149. 2, 12

[GSZ11] GREGSON J., SHEFFER A., ZHANG E.: All-hex mesh generation via volumetric polycube deformation. In *Computer graphics forum* (2011), vol. 30, Wiley Online Library, pp. 1407–1416. 2, 11, 12

[HAB*17] HALES T., ADAMS M., BAUER G., DANG T. D., HARRISON J., LE TRUONG H., KALISZYK C., MAGRON V., MCLAUGHLIN S., NGUYEN T. T., ET AL.: A formal proof of the kepler conjecture. In *Forum of mathematics, Pi* (2017), vol. 5, Cambridge University Press. 5

[Hau01] HAUSNER A.: Simulating decorative mosaics. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, pp. 573–580. 3, 5, 19

[HIKL*99] HOFF III K. E., KEYSER J., LIN M., MANOCHA D., CULVER T.: Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 277–286. 3, 5

[HS17] HORMANN K., SUKUMAR N.: *Generalized barycentric coordinates in computer graphics and computational mechanics*. CRC Press, 2017. 4

[HZG*18] HU Y., ZHOU Q., GAO X., JACOBSON A., ZORIN D., PANOZZO D.: Tetrahedral meshing in the wild. *ACM Transactions on Graphics (TOG) 37*, 4 (2018), 60. 12, 14

[JBG19] JAKOB J., BUCHENAU C., GUTHE M.: Parallel globally consistent normal orientation of raw unorganized point clouds. *Computer Graphics Forum 38* (08 2019), 163–173. doi:10.1111/cgf.13797. 11

[JTPSH15] JAKOB W., TARINI M., PANOZZO D., SORKINE-HORNUNG O.: Instant field-aligned meshes. *ACM Trans. Graph. 34*, 6 (2015), 189–1. 3, 6, 8, 14

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2006), SGP '06, Eurographics Association, pp. 61–70. 3

[KBLK14] KREMER M., BOMMES D., LIM I., KOBBELT L.: Advanced automatic hexahedral mesh generation from surface quad meshes. In *Proceedings of the 22nd International Meshing Roundtable*. Springer, 2014, pp. 147–164. 2

[LA] LEVY B., ALONSO L.: Graphite. URL: http://alice.loria.fr/index.php/software/3-platform/22-graphite.html. 17

[LBK16] LYON M., BOMMES D., KOBBELT L.: Hexex: robust hexahedral mesh extraction. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 123. 2, 11, 14

[LL10] LÉVY B., LIU Y.: L p centroidal voronoi tessellation and its applications. In *ACM Transactions on Graphics (TOG)* (2010), vol. 29, ACM, p. 119. 2, 3, 5, 7, 9, 12, 13, 14

[LLX*12] LI Y., LIU Y., XU W., WANG W., GUO B.: All-hex meshing using singularity-restricted field. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 177. 2, 11

[LPP*20] LIVESU M., PIETRONI N., PUPPO E., SHEFFER A., CIGNONI P.: Loopycuts: Practical feature-preserving block decomposition for strongly hex-dominant meshing. *ACM Transactions on Graphics 39*, 4 (2020). doi:10.1145/3386569.3392472. 2, 11, 12, 13, 14

[LPW*06] LIU Y., POTTMANN H., WALLNER J., YANG Y.-L., WANG W.: Geometric modeling with conical meshes and developable surfaces. In *ACM SIGGRAPH 2006 Papers*. 2006, pp. 681–689. 4

[LSVT15] LIVESU M., SHEFFER A., VINING N., TARINI M.: Practical hex-mesh optimization via edge-cone rectification. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 141. 11

[LZC*18] LIU H., ZHANG P., CHIEN E., SOLOMON J., BOMMES D.: Singularity-constrained octahedral fields for hexahedral meshing. *ACM Transactions on Graphics (TOG) 37*, 4 (2018), 93. 2, 3, 11, 14

[MAR*20] MATVEEV A., ARTEMOV A., RAKHIMOV R., BOBROVSKIKH G., PANOZZO D., ZORIN D., BURNAEV E.: Def: Deep estimation of sharp geometric features in 3d shapes, 2020. arXiv:2011.15081. 12

[MB12] MOUTON T., BÉCHET E.: Lloyd relaxation using analytical voronoi diagram in the l_infinite norm and its application to quad optimization. *Proceedings of the 21st International Meshing Roundtable* (2012). 5

[MT00] MESHKAT S., TALMOR D.: Generating a mixed mesh of hexahedra, pentahedra and tetrahedra from an underlying tetrahedral mesh. *International Journal for Numerical Methods in Engineering 49*, 1-2 (2000), 17–30. 3, 9, 13

[NRP11] NIESER M., REITEBUCH U., POLTHIER K.: Cubecover–parameterization of 3d volumes. In *Computer graphics forum* (2011), vol. 30, Wiley Online Library, pp. 1397–1406. 2

[NZH*18] NI S., ZHONG Z., HUANG J., WANG W., GUO X.: Field-aligned and lattice-guided tetrahedral meshing. In *Computer Graphics Forum* (2018), vol. 37, Wiley Online Library, pp. 161–172. 5

[OBB*13] ORZAN A., BOUSSEAU A., BARLA P., WINNEMÖLLER H., THOLLOT J., SALESIN D.: Diffusion curves: a vector representation for smooth-shaded images. *Communications of the ACM 56*, 7 (2013), 101–108. 6

[PAM11] PELLENARD B., ALLIEZ P., MORVAN J.-M.: Isotropic 2d quadrangle meshing with size and orientation control. In *Proceedings of the 20th International Meshing Roundtable*. Springer, 2011, pp. 81–98. 3

[PdC76] PERDIGÃO DO CARMO M.: Differential geometry of curves and surfaces. 6

[PJVR18] PELLERIN J., JOHNEN A., VERHETSEL K., REMACLE J.-F.: Identifying combinations of tetrahedra into hexahedra: A vertex based strategy. *Computer-Aided Design 105* (2018), 1–10. 3

[PTS*08] PÉBAY P. P., THOMPSON D., SHEPHERD J., KNUPP P., LISLE C., MAGNOTTA V. A., GROSLAND N. M.: New applications of the verdict library for standardized mesh verification pre, post, and end-to-end processing. In *Proceedings of the 16th International Meshing Roundtable* (2008), Springer, pp. 535–552. 12

[RSLL18] RAY N., SOKOLOV D., LEFEBVRE S., LÉVY B.: Meshless Voronoi on the GPU. *ACM Transactions on Graphics 37*

(2018). URL: https://hal.inria.fr/hal-01927559, doi:10.1145/3272127.3275092. 5

[RSR*18] RAY N., SOKOLOV D., REBEROL M., LEDOUX F., LÉVY B.: Hex-dominant meshing: mind the gap! In *SPM 2018 - International Conference on Solid and Physical Modeling* (Bilbao, Spain, June 2018). URL: https://hal.inria.fr/hal-01927557. 3, 11, 12

[Rus04] RUSINKIEWICZ S.: Estimating curvatures and their derivatives on triangle meshes. In *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on* (2004), IEEE, pp. 486–493. 6

[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, 2019. arXiv:1903.09332. 1

[Si15] SI H.: Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software (TOMS) 41*, 2 (2015), 1–36. 12

[SJ08] SHEPHERD J. F., JOHNSON C. R.: Hexahedral mesh generation constraints. *Engineering with Computers 24*, 3 (2008), 195–213. 3, 12

[SRUL16] SOKOLOV D., RAY N., UNTEREINER L., LÉVY B.: Hexahedral-dominant meshing. *ACM Transactions on Graphics (TOG) 35*, 5 (2016), 157. 2, 3, 7, 9, 11, 12, 13, 14

[Sta14] The Stanford 3D Scanning Repository, 2014. Online; accessed Oktober-2017, http://graphics.stanford.edu/data/3Dscanrep/. 11

[STJ*17] SCHERTLER N., TARINI M., JAKOB W., KAZHDAN M., GUMHOLD S., PANOZZO D.: Field-aligned online surface reconstruction. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 77. 3

[SVB17] SOLOMON J., VAXMAN A., BOMMES D.: Boundary element octahedral fields in volumes. *ACM Trans. Graph. 36*, 3 (May 2017), 28:1–28:16. doi:10.1145/3065254. 3, 11, 12, 14

[Tak19] TAKAYAMA K.: Dual sheet meshing: An interactive approach to robust hexahedralization. *Computer Graphics Forum (proceedings of Eurographics) 38*, 2 (2019), 37–48. doi:10.1111/cgf.13617. 2, 11, 12, 13, 14

[THCM04] TARINI M., HORMANN K., CIGNONI P., MONTANI C.: Polycube-maps. In *ACM transactions on graphics (TOG)* (2004), vol. 23, ACM, pp. 853–860. 2

[WSK06] WADA Y., SHINBORI J., KIKUCHI M.: Adaptive fem analysis technique using multigrid method for unstructured hexahedral meshes. In *Key Engineering Materials* (2006), vol. 306, Trans Tech Publ, pp. 565–570. 4

[YS03] YAMAKAWA S., SHIMADA K.: Fully-automated hex-dominant mesh generation with directionality control via packing rectangular solid cells. *International journal for numerical methods in engineering 57*, 15 (2003), 2099–2129. 5, 10

[ZB06] ZHANG Y., BAJAJ C.: Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. *Computer methods in applied mechanics and engineering 195*, 9-12 (2006), 942–960. 2

[ZLGH10] ZHANG L., LIU L., GOTSMAN C., HUANG H.: Mesh reconstruction by meshless denoising and parameterization. *Computers & Graphics 34*, 3 (2010), 198 – 208. Shape Modelling International (SMI) Conference 2010. doi:10.1016/j.cag.2010.03.006. 3

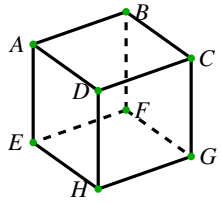
Appendix A: At-Most-Hexa Primitives

The base case of our at-most-hexa mesh is a simple hexahedron with 8 vertices and 6 quad faces. All other allowed primitives are listed in Figure 22 and derive from this base case by collapsing up to four edges.

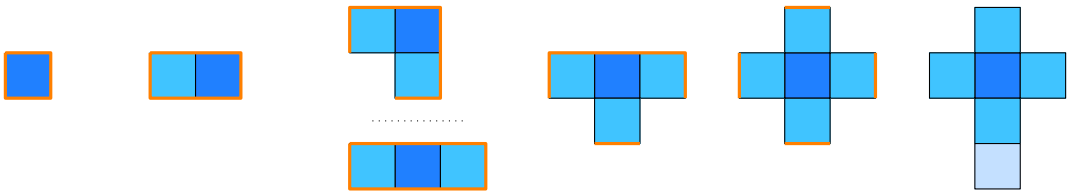
State-Machine Programs The search for primitives in the set of quadrangular and triangular faces can be performed very efficiently, using specifically tailored state-machine algorithms for each primitive type. Starting from the initialization state with only one face, adding a face is the only supported operation to transition from one state to the next. If a transition generates a valid state, the search path continues or is discarded otherwise. The illustration in Figure 22 shows the progress of the individual state-machines with color-coded faces and edges. The *primary face* is the starting position, providing the first set of *open edges*. *Secondary faces* may only be added on *open edges* of the *primary face*. *Tertiary faces* are determined indirectly by a specific set of open edges, e.g., in the penultimate state of the hexahedron, the last four open edges unambiguously specify the last quad face. Since only faces in a direct adjacent neighborhood (fix size) are considered for a state transition, the search's complexity is in $\mathcal{O}(n)$.

Interexchange Format At-most-hexa primitives can be formulated as linear combinations of the 8 vertices of a hexahedron. We use this property to store our results as simple indexed *mesh* files, commonly used for hex-meshes. Therefore, corner vertex indices are duplicated as listed in Figure 2 with 8-vertex-index sets for the different primitives, respectively. Out in the wild, this format could easily be read by dedicated hex-meshing applications. However, some internal routines may struggle with collapsed edges or faces: *Graphite* [LA] can load our meshes but may not be able to differentiate between interior and hull faces robustly. Nevertheless, the *HexaLab* [BTP*19], an online visualizer/analyzer that is designed for pure-hexa meshes, can be used to examine our results, which are provided in the supplemental material.

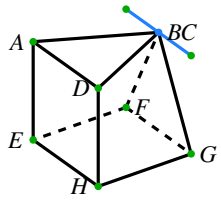
8 hex



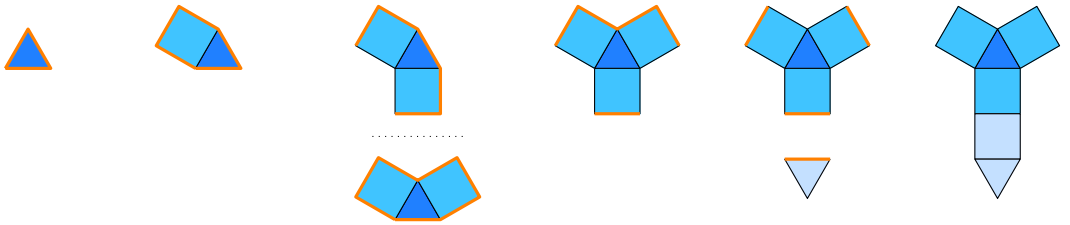
[Q:1, T:0, OE:4] [Q:2, T:0, OE:6] [Q:3, T:0, OE:6|8] [Q:4, T:0, OE:6] [Q:5, T:0, OE:4] [Q:6, T:0, OE:0]



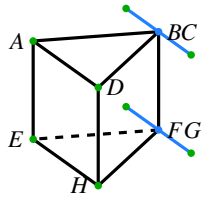
7 hexPrism



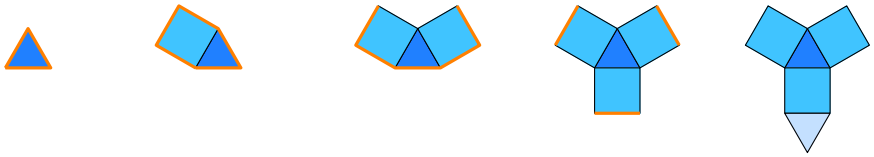
[Q:0, T:1, OE:3] [Q:1, T:1, OE:5] [Q:2, T:1, OE:5|7] [Q:3, T:1, OE:5] [Q:3, T:2, OE:4] [Q:4, T:2, OE:0]



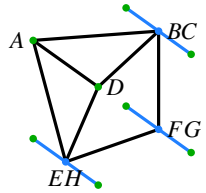
6 prism



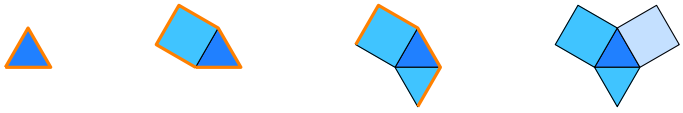
[Q:0, T:1, OE:3] [Q:1, T:1, OE:5] [Q:2, T:1, OE:5] [Q:3, T:1, OE:3] [Q:3, T:2, OE:0]



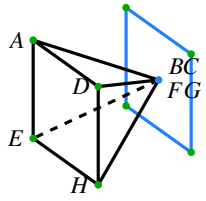
5 slice



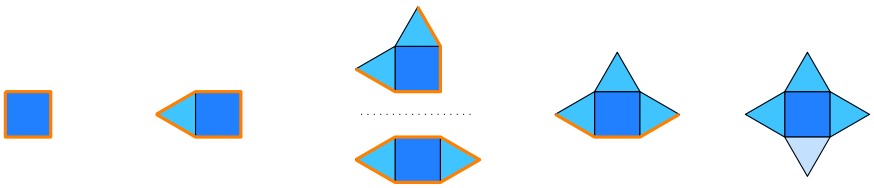
[Q:0, T:1, OE:3] [Q:1, T:1, OE:5] [Q:1, T:2, OE:4] [Q:2, T:2, OE:0]



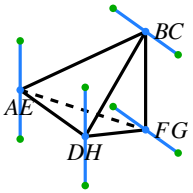
4 pyra



[Q:1, T:0, OE:4] [Q:1, T:1, OE:5] [Q:1, T:2, OE:4|6] [Q:1, T:3, OE:3] [Q:1, T:4, OE:0]



1 tet



[Q:0, T:1, OE:3] [Q:0, T:2, OE:4] [Q:0, T:3, OE:3] [Q:0, T:4, OE:0]

Figure 22: *State-machine programs for the different (used) at-most-hexa primitives, with an initialization state of only one face on the left and completed primitives on the right. Valid states are encoded in triples: Number of **Q**uads / **T**ris and the number of **O**pen **E**dges. Colors highlight open edges, primary, secondary and tertiary faces.*

Appendix B: Point Cloud Input

In addition to supporting closed manifold input, we also propose a simple extension to our method to require point clouds as input only. This feature is elaborated analogously to Section 4.

Motivation

While surface reconstruction techniques can easily start from point cloud samples directly, established hex-meshing algorithms require complex input like parameterizations, mappings, other volume or surface meshes. The goal of any signal processing pipeline should be to avoid the concatenation of lossy processing steps and the accumulation of errors. Even for well-researched procedures, it is always important to study alternative ways – if only to prove a point. So if we can go from point samples to a surface mesh and from a surface mesh to a volume mesh - why not skip this surface precomputation step and generate the volume mesh directly from point cloud samples? Nevertheless, a surface mesh is indirectly generated anyway as the outer hull of the volume mesh. Therefore, we propose an approach to bridge this gap and produce high-quality hex-dominant meshes and watertight surfaces solely from point cloud surface samples.

Site Population

The whole set of sites in the Voronoi diagram consists of two disjunct subsets $\mathbf{S} = \mathbf{S}_{pc} \cup \mathbf{S}_v$: $\mathbf{S}_v$ initializes as before. Additionally, to represent the surface, each 3D point cloud sample creates one additional site, forming the second set of sites $\mathbf{S}_{pc}$. Sites in $\mathbf{S}_{pc}$ have a fixed location - the one of their point cloud sample - and a fixed normal derived from the local neighborhood. Both properties do not change during the optimization. The sites $\mathbf{S}_{pc}$ serve two purposes: They separate inside from outside cells, ensuring that the created boundary precisely aligns to the implicit hull. Further, they enforce that the orientation of all other interior sites $\mathbf{S}_v$ will adapt to the surface features.

Point Cloud Hull

In the Voronoi diagram, both site-sets $\mathbf{S}_v$ and $\mathbf{S}_{pc}$ spawn cells. $\mathbf{S}_v$ cells are uniformly shaped with the Chebyshev metric following the sites' orientation and therefore will approximate cuboids; for $\mathbf{S}_{pc}$ cells, an anisotropic transformed norm is applied: These cells are compressed along the direction of their normal and stretched along the direction of their tangent and bitangent as illustrated in Figure 23 (with factors 0.5 and 1.5). Even sparsely sampled point clouds provide a sufficiently dense hull to separate sites floating around the in- and outside of the object [Hau01]. Throughout the relaxation, the height of the $\mathbf{S}_{pc}$ cells is slowly reduced to 0 to ensure that the hull is not covering any volume (Figure 24, right). Furthermore, sites $\mathbf{S}_{pc}$ are not allowed to move during the relaxation. This way sites $\mathbf{S}_v$ can move up to the outer hull from the in- and outside of the object but will not pass the hull. Sites that start somewhere on this hull will be pushed either in or out of the object within only a few iterations.

kNN-Graph

There are no modifications required for the kNN-graphs to support the point cloud input. The N_{26} is allowed to include neighbors from $\mathbf{S}$, therefore natively interconnecting $\mathbf{S}_{pc}$ and $\mathbf{S}_v$ sites. As before, N_6 only operates on $\mathbf{S}_v$.

Inter- and Extrapolation

Analogously to Section 4.3, all sites initialize with orientations propagated from the hull, in this case, point cloud samples. After the relaxation, an in/out label is propagated.

Orientation Initialization Instead of sampling the hull individually, nodes of level $d_i = 1$ query their orientation from the closest point cloud neighbors ($d_i = 0$), which are naturally included in their k nearest neighbor list. A $\mathbf{S}_v$ may query multiple $\mathbf{S}_{pc}$ nodes, and a $\mathbf{S}_{pc}$ node may be shared by multiple $\mathbf{S}_v$ nodes as illustrated on the left in Figure 23.

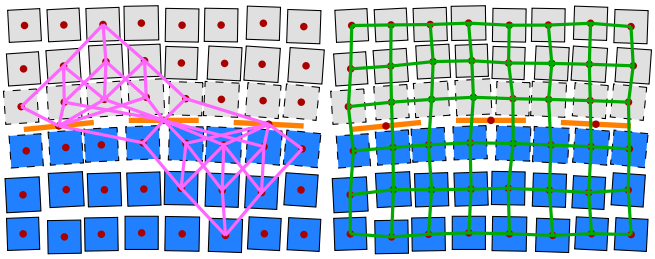


Figure 23: *Nodes with $d_i = 1$ (dashed) determine their in/out label from the dot product of direction $\vec{v}_i$ and normal $\vec{n}_i$. Nodes of $d_i > 1$ derive their label from neighboring nodes closer to the surface, as illustrated with the graph in magenta. The right side shows the N_6 graph (green), transcending the outer hull, so that adjacent cells on the in- and outside are able to align nevertheless.*

Inside/Outside Determining inside-outside labels is no longer as trivial as with a hull given. Although algorithms specialise in winding numbers for point clouds, we propose to query the information at hand using the neighborhood graph and simple geometry. First, this labeling operation simplifies by focusing only on nodes of level $d_i = 1$, close to the hull. All other nodes with $1 < d_i$ easily derive their state from already labeled nodes in their direct neighborhood. In/out labels for nodes with $d_i = 1$ depend on their normals pointing *towards* or *away* from hull nodes as shown in Figure 23. Vector $\vec{v}_i$ in Equation 6 is the normalized average direction from node s_i to its direct neighbors $s_j \in \mathbf{S}_{pc}$ with $j \in N_{26}(i)$. As formulated in Equation 7, the in/out label l_i derives from the sign of the dot product between direction vector $\vec{v}_i$ and normal $\vec{n}_i$.

$$\vec{v}_i = \frac{1}{|N_{26}(i) \cap \mathbf{S}_{pc}|} \sum_{j \in N_{26}(i) \cap \mathbf{S}_{pc}} \frac{s_j - s_i}{||s_j - s_i||_2} \tag{6}$$

$$l_i = \begin{cases} \text{in} & \text{if } \vec{v}_i \cdot \vec{n}_i > 0 \\ \text{out} & \text{else} \end{cases} \tag{7}$$

Close to sharp edges, this procedure may cause false-positives. The scenario illustrated in Figure 24 (left) shows a case where the majority of direct point cloud neighbors is *just around the corner*. Therefore, the derived normal $\vec{n}_i$ and averaged direction $\vec{v}_i$ point into the same direction. This is a perfectly valid behavior, but one can also easily fix such outliers by querying the labels of their N_6 neighborhood as illustrated in Figure 24 (center): If a node is labeled as *inside* and has less than three direct neighbors, which are also labeled as inside, its label is switched to *outside*.

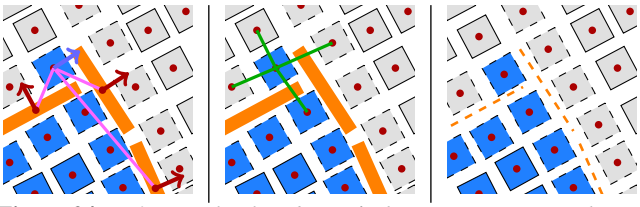


Figure 24: *Left: A node identifies as **in** because its primary direction is in accord with the majority of its hull neighbors. Center: A majority vote of the N_6 neighborhood easily fixes false-positives. Right: $\mathbf{S}_{pc}$ cells converge to 0 height at the end of the relaxation, so the hull no longer covers any volume.*

Compared to winding numbers [BDS*18], our approach trivially blends in with the rest of the graph operations and does not add any further computation steps. Since we limit our labeling to points that are closest to the oriented hull and propagate results for farther points, our results are practically error-free.

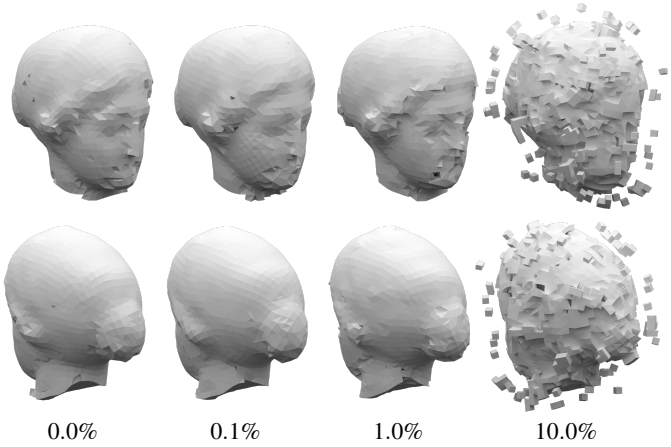


Figure 25: *Reconstructions of the Igea artifact from 5k points perturbed with random noise vectors. Noise magnitude levels are given in percent of the length of the bounding-box's diagonal. Raw output of the relaxation and merging pipeline, no at-most-hexa topology.*

Bring the Noise Due to our focus on simple and minimal input, we never used more than 10k randomly sampled points for included results. The quality of actual 3D scans is usually higher than what we aimed for: Modern mid-range 3D scanners can produce high-density results with millions of points arranged in nice regular patterns. However, robustness to bad input is always crucial for reconstruction tasks. To explore the limits of our procedure, we artificially perturbed input samples with noise vectors (sphere samples with limited random magnitude). Figure 25 illustrates results, using different magnitudes of noise which are given in percent of the length of the object's bounding-box diagonal. Normals and orientations were computed *from* the noisy point clouds to simulate unreliable data. Furthermore, we did not apply any topology reconstruction or remeshing steps to these results. Even with a certain amount of noise, our method can concisely reconstruct the object with only minor flaws on the hull. Enough noise can also provoke failures, as shown on the right in Figure 25. With noise magnitudes larger than the cell size, the hull becomes very vague: The in/out labels of the outermost layer can no longer be determined with certainty, which is why there are loose hexahedra floating around. However, already from the second layer on inwards, the mesh is fine and follows a smooth orientation field.

Relaxation

The Lloyd relaxation process with point cloud input is equivalent to the one introduced in Section 4.3 with one exception: As there is no hull separating interior and exterior cells, the sites $\mathbf{S}_{pc}$ are not altered during the relaxation. They remain in their initial position during the whole process and also maintain their normal direction. However, tangent and bitangent are free to be updated according to the frame field generated during the relaxation. By keeping their normals fixed, the hull barrier is guaranteed to stay intact, but the cells can align with dominant surface features.

Limitations Voronoi cells relax under the assumption to approximate a cubical shape with uniform edge lengths. Cells within geometry, thinner than the average relaxed edge length, are in a very unrelaxed state and might be squeezed outside of the object. However, with samples placed sufficiently dense, it becomes improbable for cells to pass the outer hull, and even thin geometry can be represented, e.g., with only one layer of hexahedra.