LoopyCuts: Practical Feature-Preserving Block Decomposition for Strongly Hex-Dominant Meshing

MARCO LIVESU*, CNR IMATI, Italy
NICO PIETRONI*, University of Technology Sydney, Australia
ENRICO PUPPO, University of Genoa, Italy
ALLA SHEFFER, University of British Columbia, Canada
PAOLO CIGNONI, CNR ISTI, Italy

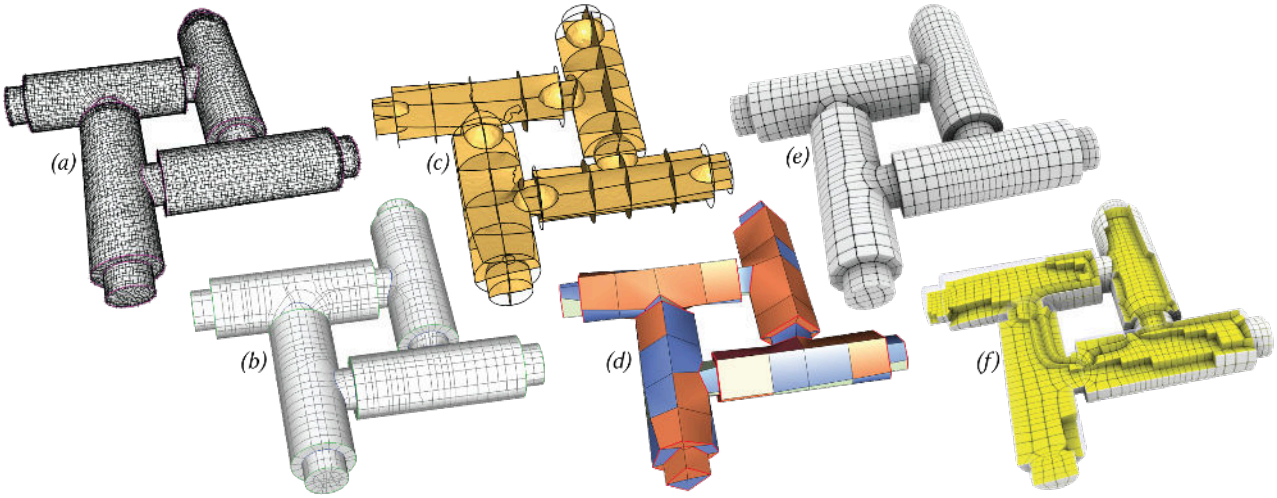


Fig. 1. Given a surface mesh and a curvature and feature aligned cross-field (a) *LoopyCuts* generates a sequence of field-aware cutting loops (b), and uses these loops to generate solid cuts through the object (c), decomposing the model into a metamesh consisting of hex (green), prism (blue) and other (orange) simple blocks (d). It converts the metamesh into a hex-mesh via midpoint refinement. The output hex-mesh (e,f) is well-shaped and well-aligned with the input field.

We present a new fully automatic block-decomposition algorithm for feature-preserving, *strongly* hex-dominant meshing, that yields results with a drastically larger percentage of hex elements than prior art. Our method is guided by a surface field that conforms to both surface curvature and feature lines, and exploits an ordered set of cutting loops that evenly cover the input surface, defining an arrangement of loops suitable for hex-element generation. We decompose the solid into coarse blocks by iteratively cutting it with surfaces bounded by these loops. The vast majority of the obtained blocks can be turned into hexahedral cells via simple midpoint subdivision. Our method produces pure hexahedral meshes in approximately 80% of the cases, and hex-dominant meshes with less than 2% non-hexahedral cells in the remaining cases. We demonstrate the robustness of our method on 70+ models, including CAD objects with features of various complexity, organic and synthetic shapes, and provide extensive comparisons to prior art, demonstrating its superiority.

CCS Concepts: • **Computing methodologies** → **Mesh models**; **Mesh geometry models**; **Volumetric models**; *Shape analysis*.

Additional Key Words and Phrases: mesh generation, volumetric meshing, hex-meshing, hex-dominant meshing, cross field

ACM Reference Format:
Marco Livesu, Nico Pietroni, Enrico Puppo, Alla Sheffer, and Paolo Cignoni. 2020. *LoopyCuts*: Practical Feature-Preserving Block Decomposition for Strongly Hex-Dominant Meshing. *ACM Trans. Graph.* 39, 4, Article 121 (July 2020), 17 pages. https://doi.org/10.1145/3386569.3392472

1 INTRODUCTION

Hexahedral and hex-dominant volumetric meshing of 3D shapes is a well investigated, yet still open, research topic. At their core, hexahedral meshing algorithms balance fidelity to the input surface geometry against element quality. They seek to generate meshes with well shaped, or box-like, elements whose outer surface closely aligns with that of the input model. To achieve high surface fidelity

*joint first authors

Authors' addresses: Marco Livesu, CNR IMATI, Genoa, Italy, marco.livesu@gmail.com; Nico Pietroni, University of Technology Sydney, Sydney, Australia; Enrico Puppo, University of Genoa, Genoa, Italy; Alla Sheffer, University of British Columbia, Vancouver, Canada; Paolo Cignoni, CNR ISTI, Pisa, Italy.

0730-0301/2020/7-ART121 $15.00
https://doi.org/10.1145/3386569.3392472

and to keep the element budget low, users prefer meshes whose external edges align with the surface curvature directions and follow geometric or semantic feature curves.

Despite multiple attempts, quality all-hex meshing remains elusive, and industrial models are still meshed using semi-manual block decomposition, a tedious and time consuming process [Lu et al. 2017]. Existing automatic methods for quality all-hex meshing are applicable to only a subset of inputs; while more general methods produce inferior quality meshes or fail to capture surface features. These challenges motivated the rise of hex-dominant approaches that can robustly handle a large spectrum of inputs. Unfortunately state-of-the-art hex dominant methods tend to produce excessive numbers of non-hex elements.

We propose *Loopy Cuts*, a practical and robust new meshing algorithm positioned in between these two extremes. LoopyCuts mimics manual block-decomposition, automatically generating all-hex feature-preserving meshes on a large spectrum of inputs and yields strongly hex-dominant meshes, with minuscule

non-hex element count, on the remaining inputs. Our method significantly outperforms prior approaches that attempted to automate block-decomposition (Section 2).

Our method is based on two simple observations. We first note that we can obtain a decomposition that produces well-shaped elements by choosing a set of *cutting surfaces* bounded by *cutting loops* distributed strategically across the input model's surface. We further note that by selecting a set of loops that are aligned with a surface cross-field and interpolate the input features, we can produce a decomposition that respects the geometric characteristics of the input model.

We generate the desired mesh as follows (Figure 1). Starting from a curvature and feature aligned cross-field (a), we extract a set of well-distributed loops on the object surface (b). We build cutting surfaces that interpolate these loops and adapt to the shape of the input model by using iso-surfaces of Hermite Radial Basis Functions (c). We use these surfaces to decompose the volume into a coarse complex of simple polyhedral blocks, the *meta-mesh*, terminating once the blocks satisfy our quality requirements (d). Finally, we refine these blocks via midpoint subdivision, producing a strongly hex-dominant mesh (e-f).

We validate our framework by testing it on more that 70 inputs of varying complexity, including both mechanical and organic models, which exhibit a range of geometric and user prescribed features (Section 7). Across all tested models, we obtained 76% pure hexahedral meshes; the remaining hex-dominant meshes contained less than 2% non-hexahedral elements. We highlight the advantages of our method by comparing our results to those produced using a range of existing hex and hex-dominant strategies, showing that we consistently produce quality meshes with significantly less non-hex elements than prior approaches.

Our overall contribution is a fully automatic method for block decomposition for strongly hex-dominant meshing. We generate comparable or better quality meshes than previous automatic and semi-manual approaches, while guaranteeing feature-preservation. This contribution is made possible by our novel technical ingredients: (i) the definition and computation of a field-aware loop network that evenly samples the surface and its features; (ii) the robust generation of valid smooth cutting surfaces that interpolate these loops; and (iii) a robust method to devise a cellular complex from the cut arrangement, which can then turned into a strongly hex-dominant mesh via simple midpoint refinement.

2 RELATED WORK

Hex-dominant meshes. In recent years hex-dominant methods have emerged as an alternative to pure hexahedral meshes. These methods offer superior robustness, and can produce quality meshes for objects with complex features at the cost of introducing some non-hexahedral elements. Early methods start from a tet mesh and use various grouping schemes to detect clusters of tetrahedra that, if merged together, form hexahedra. Pellerin et al. [2017] greedily explore the combinatorial space of all possible agglomerations of tets that produce hexahedra in a given simplicial mesh. Their approach produces hex-dominant meshes with a poor amount of cuboidal elements (under 60% across all examples shown). Yamakawa et al. [2002] similarly produce hybrid meshes with around 50% hexahedral cells. Sokolov et al. [2016a] obtain higher hex to tet ratios by first computing a guiding field that samples the volume at a regular grid, and then apply agglomeration to a mesh obtained by tetrahedralizing such grid. They obtain hex-dominant meshes with up to 95% hexahedral cells (although in the worst case they obtained less than 30%). Notably, the hybrid meshes these methods produce are non conforming, meaning that there are interfaces where e.g. a quadrilateral face of a hexahedron is touching two triangular faces of two tetrahedra. To grant mesh conformity, layers of zero-volume elements need to be positioned in between. Gao et al. [2017a] directly generate conforming hybrid meshes using polyhedral agglomeration. They use a similar tetrahedral mesh obtained by sampling a guiding field, but generate hexahedra via a set of local topological operators that modify mesh connectivity, thus granting mesh conformity at any time. Unfortunately, this approach cannot control the type of elements generated and can produce arbitrarily complex polyhedra (up to 40 facets in some examples, see Table 1 in [Gao et al. 2017a]). Levy and Liu [2010] obtain a hex-dominant mesh by applying the agglomeration scheme proposed in [Meshkat and Talmor 2000] to a CVT. None of these methods can generate a pure hexahedral mesh, even for simple shapes. Our method produces a pure hexahedral mesh in the majority of cases (76% of the models tested), and produces conforming meshes with less than 2% non-hexahedral cells in the remaining cases, much less than any prior method.

Hexahedral meshing. Generation of high-quality hexahedral meshes is a well researched, yet still open challenging problem; see [Blacker 2000; Owen 2009; Shepherd and Johnson 2008] for in depth reviews. Methods that overlay a regular [Lin et al. 2015; Schneiders 1996] or adaptive [Gao et al. 2019; Ito et al. 2009; Maréchal 2009] Cartesian grid onto the model to form hexahedra are unbeaten in terms of robustness, and can hexmesh virtually any shape. However meshes produced with these methods do not typically align to surface curvature, and cannot precisely incorporate surface features. Moreover, grid cells that intersect the surface are warped to approximate the input geometry, typically producing poorly shaped elements that

can hardly be optimized due to limits in the mesh structure. An extension of the grid idea consists in applying a uniform Cartesian grid to an orthogonal polyhedron (or *polycube* [Tarini et al. 2004]), and then use a volumetric map between the polycube and the target shape to position cubes in the interior of the object [Fang et al. 2016; Gregson et al. 2011; Huang et al. 2014; Livesu et al. 2013]. Although less robust than classical grid methods, polycube approaches have reached a fair level of maturity, and can reliably process datasets containing dozens of shapes [Fu et al. 2016]. None of these methods are able to preserve sharp creases that do not align with the grid, even for extremely simple shapes (Figure 2).

Frame-field-based methods rely on a map that is aligned to a given field, and define mesh edges as the integer iso-lines of such map [Huang et al. 2011; Jiang et al. 2014; Kowalski et al. 2016; Li et al. 2012; Nieser et al. 2011; Solomon et al. 2017]. Frame field methods can generate high quality meshes that align to both surface curvature and sharp features, which are seamlessly incorporated into the mesh connectivity. Sadly, not all frame fields are suitable to produce a hexaheral mesh. The generation of a hexable frame field remains an open problem that hinders the applicability and robustness of these techniques. In practice, frame field methods cannot handle even simple shapes like the ones showed in Figure 3 without requiring manual intervention. The most recent approaches can produce a field that is compliant with a given singular structure, and can therefore produce a valid hexahedral mesh if correctly initialized [Corman and Crane 2019; Liu et al. 2018]. However, it remains unclear how to compute a good singular graph for a given shape, and the process is still trial and error [Liu et al. 2018]. All in all, the most recent attempts to quality hexahedral meshing are either robust but lack the ability to align to surface curvature and sharp features, or produce high quality meshes but are extremely fragile. To this end, we believe *LoopyCuts* offers a good compromise between robustness and quality meshing. Similarly to frame field methods, the meshes we produce align to both curvature and features, and at the same time our approach exhibits a superior robustness, as demonstrated in Section 7.

Block Decomposition. Decomposition techniques aim to cut objects into parts, which can then be meshed conformingly using existing algorithms. Inside-out skeleton [Livesu et al. 2017, 2016] and medial-axis based decomposition approaches [Li et al. 1995; Quadros 2014; Sheffer et al. 1999] fail to generalize to complex shapes. Methods that start from a dense hexmesh and derive a coarse block decomposition from it [Cherchi et al. 2016; Gao et al. 2015, 2017b] may fail to align with features not present in the input mesh. Surface-driven block-decomposition techniques use . cuts to define either the primal [Blacker 1996; Liu and Gadh 1997; Miyoshi and Blacker 2000; Ruiz-Gironés et al. 2011] or the dual structure [Gao et al. 2018] of the mesh. Since the cuts used by dual methods do not correspond to mesh edges, these methods have challenges when attempting to capture input features. Both established [Shepherd and Johnson 2008] and recent [Kowalski et al. 2012; Wang et al. 2017] primal block-decomposition methods are limited in the set of geometries they can be applied to. In particular, they rely heavily on the sharp feature networks on the model surfaces, and cannot process free-form natural shapes, or shapes with smooth and rounded features.

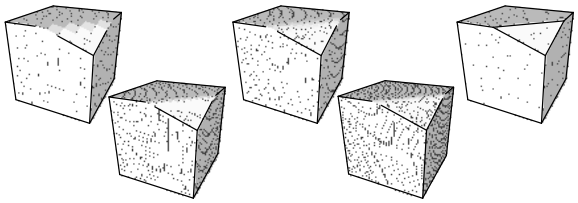


Fig. 2. Polycube- (left), and grid-based methods (middle), fail to conform to features not aligned to the major axes,resulting in loss of geometric fidelity and formation of elements with non planar facets. While geometric fidelity can be improved via refinement, the features cannot be matched (bottom). Our output meshes are by construction aligned with all surface features at a coarsest scale (right).

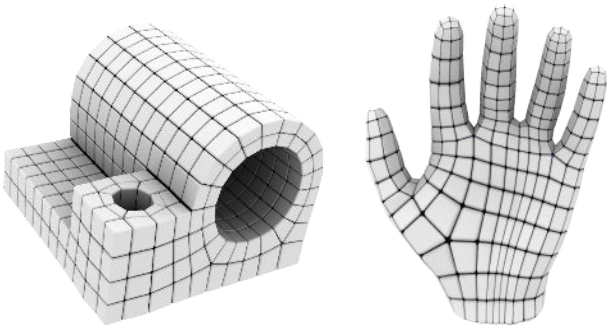
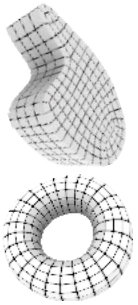


Fig. 3. State of the art tools based on volumetric fields [Liu et al. 2018] require manual intervention to mesh even simple objects like the joint and hand models. LoopyCuts automatically produces hexahedral meshes with comparable singular structure, without requiring a volumetric field.



Our method follows the block-decomposition approach popularized by these techniques, and inherits their core advantages. However, contrary to primal methods, it does not solely rely on the input feature curve networks; It can robustly handle generic free-form models with both smooth and sharp features (Figure 3) while preserving all desired features. The torus and the wave in the inset are examples which [Gao et al. 2018] and [Wang et al. 2017], respectively, list as failure cases. A recent system [Takayama 2019] lets users interactively design dual sheets that define a valid all-hex mesh topology. Our approach is inspired by this work and shares a similar use of Hermite RBF for the definition of internal surfaces, but generates high quality decomposition fully automatically.

Cross-fields and field-coherent loops. Generating and tracing direction fields on surfaces, or other spatial domains, is becoming a fundamental preprocessing step in numerous applications in computer graphics and geometry processing [de Goes et al. 2016; Vaxman et al. 2016]. Paths traced using most existing methods are not designed to be closed, and are typically terminated when approaching a singularity or another similarly directed path. A range of recent methods seek to connect cross-field singularities with short, field

aligned paths [Boier-Martin et al. 2004; Carr et al. 2006; Daniels II et al. 2009]. We trace closed field-coherent loops away from singularities following the approaches of [Campen et al. 2012; Pietroni et al. 2016], which both rely on the formalism introduced in [Kälberer et al. 2007]. We use the discrete graph based structure of [Pietroni et al. 2016] to efficiently trace such loops and compute field-aware geodesic distances.

3 OVERVIEW

We take in input a 3D model described by a closed triangle mesh $\mathcal{M}$, together with a set of line features demarcated as chains of edges on $\mathcal{M}$. We decompose the volume enclosed by $\mathcal{M}$ into blocks using a sequence of cuts, producing conforming blocks with shared surfaces. From such blocks, we extract a *meta-mesh* $\mathcal{MM}$ formed of polyhedral cells, which approximates the volume enclosed by $\mathcal{M}$. While choosing the cuts we exploit a strategy that tends to create polyhedral blocks with vertices of valence three, which can be easily split into hexahedra via midpoint subdivision.

Our process is mainly driven by two heuristics: a *loop sampling* strategy to generate a set of *loops* on $\mathcal{M}$, which will act as seeds to generate cutting surfaces; and the other to diffuse cutting surfaces, which interpolate such loops through the volume generating a *block decomposition*. In post-processing, we perform one step of midpoint subdivision to obtain a hexa-dominant decomposition and we smooth the resulting mesh with a state-of-the-art method in order to improve the shape of its cells. The whole process is fully automatic with a single parameter controlling the surface approximation error.

Loop sampling. Section 4. We define a space of *field-coherent* loops, similarly to [Pietroni et al. 2016], and a notion of distance in such space. We start by generating a set of loops that incorporate the line features in input, then we extend such set by a furthest point sampling technique in the space of loops. This strategy tends to generate an arrangement of loops that form a nearly uniform quad-dominant grid on the surface of $\mathcal{M}$. Loops are sampled to provide a set of cuts sufficient to: incorporate the input line features as edges of $\mathcal{MM}$; obtain a good approximation of the outer surface of $\mathcal{M}$; obtain polyhedral cells when cutting the volume through the loops. The set of extracted loops is usually larger than needed in the subsequent block decomposition process; it is sorted so that the loops containing line features are used first, while the following loops refine the subdivision of $\mathcal{M}$ into progressively smaller patches.

Block decomposition. Section 5. We pick loops from the sorted sequence in order; for each loop, we generate a Hermite surface that interpolates the loop and cuts through the volume. Each cutting surface intersects the surface of $\mathcal{M}$ and possibly other cutting surfaces, thus generating cells of the meta-mesh $\mathcal{MM}$. Both the arrangement of loops and the surface diffusion process are aimed at obtaining intersections just between nearly orthogonal lines and surfaces. In particular, each edge in the meta-mesh is obtained by intersecting exactly two surfaces (either two cutting surfaces, or a cut and the surface of $\mathcal{M}$); likewise, each vertex in the meta-mesh is obtained by intersecting exactly three surfaces. The meta-mesh is updated and analyzed after each cut and the decomposition stops as soon as all cells are valid polyhedra and the outer surface is approximated within the given threshold error.

Hexa-dominant meshing. Section 6. Most cells in the final meta-mesh are already either hexahedra or prisms. This is due to the fact that, being cuts guided by loops that intersect orthogonally, they tend to split the volume by intersecting in groups of three orthogonal surfaces at each vertex of $\mathcal{MM}$. However, some cells of $\mathcal{MM}$ may be more general polyhedra, possibly including vertices of valence two or valence four. This can be due either to vertices formed by the line features in input, or to special configurations that arise in the intersection of surfaces. We analyze the topology of $\mathcal{MM}$ and we attempt further cuts to eliminate such problematic configurations, whenever possible. After that, we apply one step of midpoint subdivision: hence, cells having all vertices with valence three are split into hexahedra. This means that if we manage to eliminate all cells with vertices with a valence different from three in $\mathcal{MM}$, we eventually obtain a full-hexa mesh; otherwise, a few non-hexa elements will remain. In our experiments, the majority of models can be decomposed into full-hexa meshes, while the number of non-hexa elements in the remaining models is very low. The final step of mesh smoothing is standard and allows us to obtain elements with balanced size and overall good shape.

4 COMPUTING CUTTING LOOPS

The input of this phase consists of mesh $\mathcal{M}$ and a set of line features on $\mathcal{M}$. Its output consists mainly of a sorted sequence (queue) of loops $Q = (\ell_1, \ldots, \ell_k)$.

We build the loops of Q in such a way that they induce a quad-dominant layout on $\mathcal{M}$, subdividing the surface into patches. Such layout embeds the line features in input and each patch is well approximated with a polygon; furthermore, Q is sorted so that the layout is uniformly refined while adding loops from Q in order.

In order to build the queue of loops, we impose a cross-field on $\mathcal{M}$ that we use to define a space of *field-coherent* loops, and a notion of distance in such space. The queue of loops is thus obtained with a customized furthest loop sampling process in such space.

Subsection 4.1 provides a few preliminary notions about line feature classification and cross field definition. Subsection 4.2 formally defines the space of loops and the distance on it. Subsection 4.3 describes how we initialize the queue of loops starting from the line features in input, while Subsection 4.4 describes how we replenish the queue with further loops. Subsection 4.5 describes how this method is implemented in the discrete setting. Finally, Subsection 4.6 describes the output from the discrete process, which is passed on to the block decomposition phase.

4.1 Preliminaries

Classification of line features. For each line feature in input ℓ on mesh $\mathcal{M}$, let p be a point on ℓ and let $(\mathbf{t}, \mathbf{b}, \mathbf{n})$ be a local frame at p, where $\mathbf{t}$ is the tangent of ℓ and $\mathbf{n}$ is the normal of $\mathcal{M}$ at p, thus $\mathbf{b}$ is the bi-normal on the tangent plane (see Figure 10). We say that p is a *corner* if the absolute value of the discrete curvature of ℓ at p is larger than a threshold $\hat{\theta}$, or p lies at the intersection of different line features, or p is a dangling endpoint of ℓ. We break the feature curves into segments bounded by corner vertices and treat each segment as a separate curve in the following.

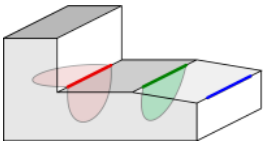


Fig. 4. Axamples of concave (red), flat (green) and convex (blue) features. In order to obtain a good volumetric decomposition, we generate one and two cutting surfaces through flat and concave features, respectively. Convex features generate no cuts, while they are embedded as edges of the meta-mesh.

Next we consider the discrete normal curvature of $\mathcal{M}$ in direction $\mathbf{b}$ at each point $p \in \ell$ and we integrate such value along ℓ. We say that ℓ is *flat* if the absolute value of the integrated curvature is smaller than a threshold $\bar{\theta}$; while it is either *convex* or *concave* if the integrated curvature is larger than $\bar{\theta}$, or smaller than $-\bar{\theta}$, respectively. For all our experiments we have used values $\hat{\theta} = 45°$ and $\bar{\theta} = 10°$. See Figure 4 for examples of concave, flat and convex features on a simple CAD model. The set of all *convex* feature lines $\mathcal{CF}$ is part of the output of this phase.

For the sake of simplicity, we classify features by simply thresholding the angle formed by consecutive edges along a feature for corner detection; and the dihedral angle between faces incident at each feature edge for feature classification. More sophisticated methods for detection of smooth features and curvature estimation can be used, though, which remain independent of our method [DeCarlo et al. 2003].

Cross field. We compute a cross-field $\mathbf{X}$ on the surface $\mathcal{M}$ using state-of-the-art methodology [Bommes et al. 2009; Diamanti et al. 2014]. We constrain the field to follow all line features in input and the main curvature directions at points with high curvature anisotropy, according to [Bommes et al. 2009], elsewhere. Note that the cross field is well defined also if no line feature is provided in input, e.g., for smooth organic objects.

4.2 Field-Coherent Loops

A *loop* is a simple closed line on the surface of $\mathcal{M}$. We trace field-coherent geodesic paths w.r.t. cross field $\mathbf{X}$, as in [Pietroni et al. 2016]. A formal definition of field-coherent paths and loops rests upon the stratification $\mathcal{M}_4$ of $\mathcal{M}$ as defined in [Kälberer et al. 2007] and briefly summarized in Appendix A.1. Informally we define a *field-coherent geodesic loop* through a point $p \in \mathcal{M}$ to be a closed curve that goes through p, is loosely following one of the directions of $\mathbf{X}$ and is as short as possible according to the anisotropic distance defined in Equation 3 (Appendix A.1). Roughly speaking, field-coherency forces a loop to approximately follow the underlying direction field on $\mathcal{M}_4$ until it gets back to its origin. Note that strictly aligning to $\mathbf{X}$ would often produce complex loops that spiral and self-intersect. Our strategic choice to balance field coherence with loop length consistently produces high quality cutting loops (Figure 5). We empirically verified that on average our tracing produces only about 5% of loops that self-intersect, which we discard. Such situations are typically triggered by atypical global structures of the field and arrangement of singularities.

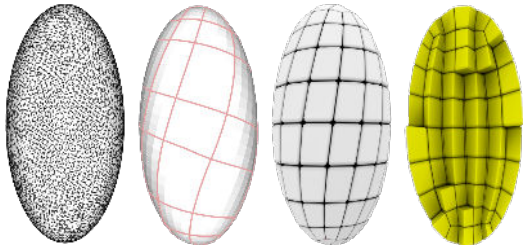


Fig. 5. Strictly aligning to a cross field (left) may result in complex loops that spiral and self-intersect (middle left). Our tracing system allows drifting to balance field coherency with loop length, yielding simple loops (middle right) that result in high quality topological structures (right).

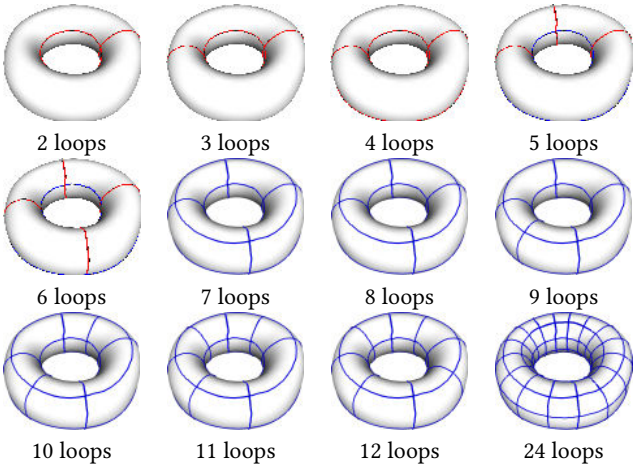


Fig. 6. A sequence of farthest loop sampling on a torus endowed with a curvature-aligned cross field. The sequence is initialized with the first vertical loop and the inner loop in the doughnut hole is identified as its farthest loop. In the following steps, loops are sampled alternately in the two homological classes, so that they tend to uniformly cover the surface of the torus. Blue loops satisfy the topological integrity constraint defined in Section 4.4, red loops do not. If a constraint that fosters the fulfillment of such condition is added, steps 6 and 7 would be swapped.

Each point p on $\mathcal{M}$, which is away from a field singularity, can be crossed by at most two field-coherent geodesic loops, which are orthogonal to one another at p, disregarding the traced curve's orientation. The *space of loops* $\mathbf{L}$ consists of the (infinite) collection of all non self-intersecting loops on $\mathcal{M}$.

We introduce a notion of (non-symmetric) distance between loops of $\mathbf{L}$, as the average over one loop of the shortest distance from each of its points to the other loop. Given the loops ℓ_i and ℓ_j we define:

$$d(\ell_i, \ell_j) = \frac{1}{|\ell_j|} \int_{\ell_j} \text{dist}(\ell_i, p_\theta) dp_\theta \tag{1}$$

where $\text{dist}(\ell_i, p_\theta)$ is the length of the shortest field-coherent geodesic path joining a point of ℓ_i to p_θ. To get an intuition for this distance, consider that two parallel loops can be close to one another or not depending on their geodesic distance on $\mathcal{M}$, while loops that

are either intersecting orthogonally, or wind about different handles of an object (of non-null genus) are always far apart.

Now considering a given set of loops $\mathcal{L} = \{\ell_1, \ldots, \ell_k\}$, and a loop $\ell \in \mathbf{L}$ not belonging to $\mathcal{L}$, we can generalize:

$$d(\mathcal{L}, \ell) = \frac{1}{|\ell|} \int_{\ell} \min_{\ell_i \in \mathcal{L}} \text{dist}(\ell_i, p_\theta) dp_\theta, \tag{2}$$

hence the notion of farthest loop $\bar{\ell}$ from $\mathcal{L}$ is well-defined as

$$\bar{\ell} = \text{argmax}_{\ell \in \mathbf{L}} d(\mathcal{L}, \ell).$$

On the basis of this distance, we will define a farthest loop sampling process. Figure 6 shows an example of farthest loop sampling sequence on a simple torus model.

4.3 Adding Feature Loops

We first consider the set of concave and flat features in input. Note that a line feature may already be a loop, as in Figure 7.a. We initialize our queue of loops Q by adding such closed features and extending the other features to loops. The order of this initial set is not relevant.

Note that open line features may end at cross-field singularities $\mathbf{X}$, where loops are undefined (see Figure 7.b). We extend each such feature into a loop by constraining the loop to run infinitesimally close to it on one of its sides, while leaving it free elsewhere. Depending on the side we choose for tracing, the loop may take different routes (see Figure 7.c-d). We extend concave features by tracing a loop on each side, which will generate two different cutting surfaces (see red feature in Figure 4). The relevant side of a loop is implicitly encoded by its orientation. Note that some loops may encompass more that one line feature, as in Figure 7.e. We proceed as follows:

- For each closed feature f: if f is *flat* we add it to Q with arbitrary orientation; if f is *concave*, we add two different copies of the loop coinciding with f with opposite orientations;
- We extend each open concave feature to two complete loops, defined as above, with opposite orientations, and add them to Q;
- Similarly, we extend each open flat feature to a single loop, by tracing it in arbitrary orientation, and add it to Q.

4.4 Sampling Further Loops

The arrangement of loops obtained from line features is usually not sufficient to induce a layout fine enough for our purposes. We extend this set further with a farthest loop sampling process, which starts at the given set Q and iteratively appends to the queue the loop that lies farthest from its content, according to the distance (2) defined in Subsection 4.2. We stop adding loops to Q as soon as the following two conditions are satisfied:

(1) **Topological integrity:** each loop ℓ_i in Q has at least three intersections with some other loop(s); in case loops ℓ_i and ℓ_j intersect in more than a place, such intersections cannot be consecutive along either loop.
(2) **Geometric fit:** each patch intercepted by the network of loops of Q on the surface $\mathcal{M}$ is approximated well enough with a corresponding polygon. In order to obtain a rough and quick estimation of accuracy, we just compare the areas of the patch and the polygon. The area of the patch is measured directly on $\mathcal{M}$. The polygon is obtained by joining the corners of the patch with straight line segments, while its area is estimated by triangulating it. The two areas must differ for no more than a threshold $\varepsilon/2$; the same threshold ε will be used in the subsequent block decomposition for a similar purpose; here we use half of the threshold to have a greater accuracy and ensure some redundancy in the loops we generate.

Note that, unlike the other thresholds used by our method, which are fixed once and for all, ε is the unique parameter that can be set by the user to control the geometric accuracy of our block decomposition with respect to the boundary of $\mathcal{M}$.

In Figure 6, the red loops violate topological integrity, which is satisfied once the 7th loop is added. At that point, the torus can be approximated by four triangular prisms. Depending on the value of ε further loops may be necessary to satisfy geometric fit.

4.5 Loop tracing and sampling in a discrete setting

We trace field-coherent loops by following the approach of [Pietroni et al. 2016]. A brief summary of the method is provided in Appendix A.2. Discretization impacts on our method in two aspects. First of all, we need to substitute the infinite space of loops $\mathbf{L}$ with a dense enough finite *pool* of loops $\mathcal{P}$ from which we sample. Second, while field-coherent loops in the continuum may intersect only orthogonally, discrete loop tracing may produce loops that overlap or intersect tangentially: we must avoid tangential intersections, because they would corrupt our layout. We provide a formal definition of tangential and orthogonal intersections in Appendix A.1.

We generate the pool $\mathcal{P}$ dynamically. Right after extending the line features to loops, we define set $\overline{Q} = Q \cup C\mathcal{F}$ as the set that contains all loops already in Q together with the set of all convex feature curves. We form the initial pool as follows.

(1) We sample all curves of $\overline{Q}$ at fixed intervals and, for each sample, we trace orthogonal loops that we add to $\mathcal{P}$. Traced loops are constrained to avoid tangential intersections with the elements of $\overline{Q}$;
(2) We perform a Poisson-Disk point sampling on $\mathcal{M}$ [Corsini et al. 2012] to obtain a set of well distributed seed points $\mathbf{P}$ and, for each $p \in \mathbf{P}$, we trace the two orthogonal loops, each constrained by the elements of $\overline{Q}$ as above, and we add such loops to $\mathcal{P}$.

Each time we add a new loop ℓ to Q, we replenish the pool with new loops obtained by sampling ℓ as described in item (1).

Rather than running a plain farthest point sampling, we use an additional criterion to promote the fulfilment of topological integrity. Let $\hat{Q}$ be the subset of Q made of loops that do not fulfill topological integrity, as defined in the previous subsection. We give higher priority to those loops in $\mathcal{P}$ that intersect at least one loop in $\hat{Q}$; among them, we select the loop ℓ that maximizes its distance from all loops in $\overline{Q}$ and we add ℓ to Q. Next, we remove from $\mathcal{P}$ all loops that intersect ℓ tangentially, and retrace them from their sources in $\mathbf{P}$, constrained to the updated $\overline{Q}$.

This step can be repeated at will and fosters the formation of a queue Q that satisfies topological integrity and consists of loops uniformly distributed over $\mathcal{M}$. Figure 8 shows a few steps of discrete loop generation on an object containing sharp features. Note how

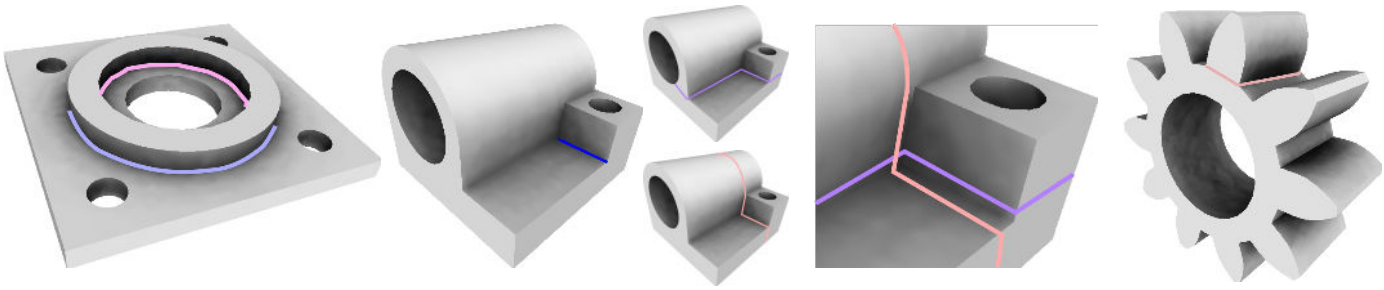


Fig. 7. (a) Closed concave curve features form two coincident loops each, with opposite orientations; (b) an open curve feature (c) is extended to form two distinct loops; (d) a closeup on field topology of traced loops (displaced from the feature in the rendering); (e) a loop connecting two open concave features (one hidden behind the gear tooth).

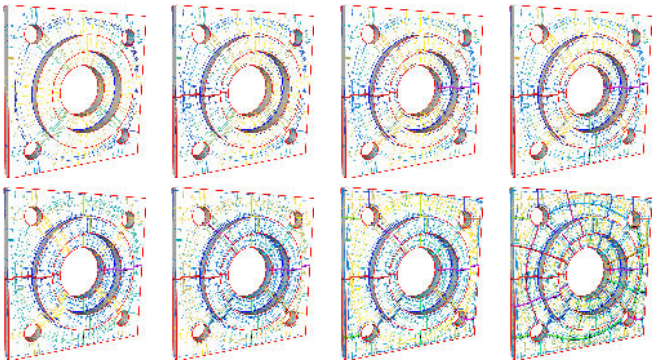


Fig. 8. Construction of the loop queue $\mathcal{Q}$ (thick lines). Loops corresponding to line features added in the top left image. Farthest loop sampling with priority on topological integrity shown in the other images (left to right, top to bottom). The thin lines are loops in the pool $\mathcal{P}$, where color denotes distance from the features already present in $\mathcal{Q}$: blue close, yellow far.

the pool $\mathcal{P}$ becomes progressively more dense as the process goes on.

4.6 Output from the discrete process

The output consists of the queue of loops $\mathcal{Q}$ and the set of convex features $\mathcal{CF}$. All elements in both sets are polylines defined by sequences of points on $\mathcal{M}$. For each point p on such a polyline, we also estimate the surface normal $\mathbf{n}$ at p and encode each polyline as a sequence of pairs $((p_1, \mathbf{n_1}), \ldots, (p_h, \mathbf{n_h}))$. Note that the surface normal is computed by using all triangles incident at p for flat and additional loops; while for concave loops we consider only the faces to the side of the feature that generated the loop. The number n_f of elements of $\mathcal{Q}$ enclosing the line features in input is reported, too.

Unfortunately, there is no guarantee that every open feature can be completed to a loop: indeed, loop tracing may produce a self-intersecting line, which is discarded (in the continuum setting, too). In case we cannot extend a line feature to a loop, we report the polyline corresponding to such feature as an *incomplete loop* in $\mathcal{Q}$. This just means that $\mathcal{Q}$ may contain some open line. The next decomposition phase is designed to deal with incomplete loops in a transparent way.

5 BLOCK DECOMPOSITION

The block decomposition phase receives in input: the mesh $\mathcal{M}$; the queue $\mathcal{Q}$ of loops, which starts with the (possibly incomplete) n_f loops covering all the non-convex features of the model; and the collection of its convex features $\mathcal{CF}$. The queue of loops is scanned in order and, for each loop, a cutting surface is generated, thus decomposing the volume Ω_M enclosed by $\mathcal{M}$ into progressively finer *blocks*, until some convergence condition is met. We build a *metamesh* $\mathcal{MM}$ consisting of polyhedral *cells*, each corresponding to a block in the decomposition; mesh $\mathcal{MM}$ approximates volume Ω_M – hence its outer surface approximates $\mathcal{M}$ – and it embeds all line features in input as chains of edges. In the following, we discuss in details the strategy used to form the cutting surfaces and to generate the metamesh: Subsection 5.1 provides the necessary tools; while Subsection 5.2 describes our method.

5.1 The metamesh: tetrahedra, blocks and cells

In order to support our block decomposition algorithm, we first build a *tetrahedral mesh* $\mathcal{MV}$ bounded by mesh $\mathcal{M}$ and filling volume Ω_M. Each time we cut the volume, the cutting surface is embedded by splitting the tetrahedra it intersects, so that the cutting surface can be represented as a collection of faces of $\mathcal{MV}$.

A *block* is a maximal set of tets of $\mathcal{MV}$ that are connected by adjacency without crossing any cutting surface (Figure 9 upper left). After each cut, we assign labels to the tets of $\mathcal{MV}$ with a simple flooding process, so that all tets forming a block are assigned the same label. Facets that are incident to tets with different labels are grouped to form the block faces; edges of the tet mesh that are incident at two such faces are grouped to form the block edges; and vertices that are incident at three such faces are the block vertices. Boundary faces and convex features in input are also incorporated to form boundary faces and edges of the outer blocks.

Note that faces, edges and vertices of each block, which result from this simple labeling process, are oblivious of the underlying microstructure of $\mathcal{MV}$ and define the combinatorial structure of the boundary of the block. In order to assess whether a block is a

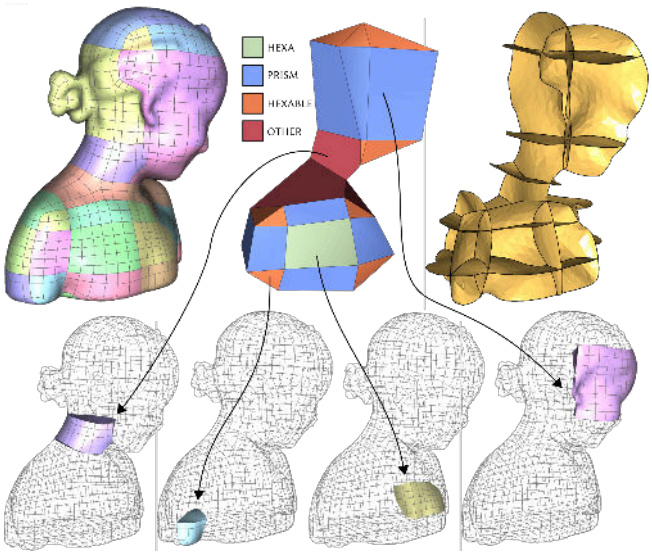


Fig. 9. We define blocks as clusters of a labeled tetrahedral mesh (top left) bounded by cutting surfaces (top right). Each block defines an element of the metamesh (top, middle), and can be of four types: cuboids (green), prisms (violet), polyhedra with all valence three vertices that yield hexahedra under midpoint refinement (orange), and other polyhedra which do not admit a pure hexahedral decomposition (red).

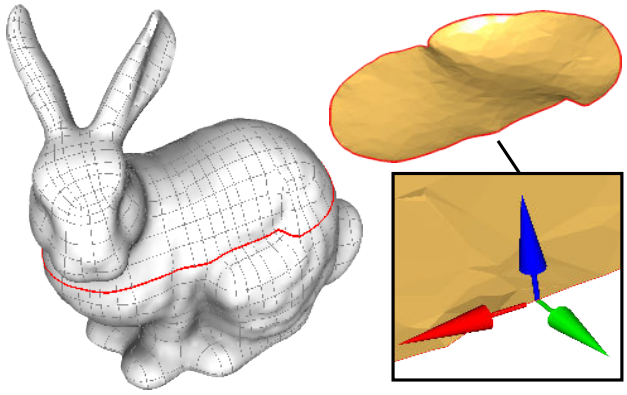


Fig. 10. A cutting loop and its associated cut. Cuts are defined as cubic HRBF that interpolate loop points and the binormal (blue) to the surface (green) and loop tangent (red).

polyhedron we require that: its volume has genus zero; each face of the block is homeomorphic to a disc and bounded by at least three edges; there are no dangling edges inside faces. All such conditions can be easily checked in a combinatorial way.

If a block is a polyhedron, we add a corresponding *cell* to the current metamesh $\mathcal{MM}$ (Figure 9 upper middle). The metamesh is initially empty, and it is regenerated after each cut by adding all valid polyhedral cells. Note that the geometry of each cell of $\mathcal{MM}$ is defined as follows: its vertices are fixed from the corresponding intersections of surfaces (Figure 9 upper right); its edges are straight line segments joining pairs of vertices; and its faces are polygons bounded by such edges. Since such polygons are not necessarily planar, the surface of each polygon is approximated by triangulating it. Block decomposition ends when the metamesh $\mathcal{MM}$ fills the whole Ω_M, while approximating its outer surface $\mathcal{M}$ sufficiently well. See below for details.

5.2 Cutting the volume

The decomposition initially consists of a unique block and its outer surface is possibly subdivided just from the convex features in $\mathcal{CF}$. Unless we are in the trivial case – i.e., there are just convex features and they alone define a valid polyhedron approximating the input sufficiently well – this initial block needs to be subdivided on order to obtain a metamesh.

Stop condition. In all cases, we use the first n_f loops in $\mathcal{Q}$, which contain all the non-convex features, to generate cutting surfaces; next we scan the rest of $\mathcal{Q}$ and we decompose the volume with further cutting surfaces, until the following two conditions are met:

(1) The metamesh $\mathcal{MM}$ has the same topological type of the domain Ω_M;
(2) The outer surface of $\mathcal{MM}$ approximates the input $\mathcal{M}$ sufficiently well. Similarly to the phase of loop generation, the quality of approximation is evaluated by measuring the difference of area between each outer face of $\mathcal{MM}$ and its corresponding patch of surface on $\mathcal{M}$.

Quality of approximation is controlled with the same threshold ε used during loop generation.

Generating cuts. For each loop $\ell \in \mathcal{Q}$, we generate a cutting surface as follows. The loop comes in the form $((p_1, \mathbf{n_1}), \ldots, (p_h, \mathbf{n_h}))$ where each p_i is a point on $\mathcal{M}$ and $\mathbf{n_i}$ is the surface normal at p_i; we set a local frame at p_i as already described in Section 4.1 (Figure 10). Now we seek a surface that interpolates ℓ and is tangent to $\mathbf{n_i}$ at all p_i's, i.e., all $\mathbf{b_i}$'s are normal to the cutting surface. Similarly to [Takayama 2019], we follow [Macêdo et al. 2011] to generate a cubic Hermite radial basis function Φ_ℓ defined over the whole volume and constrained to the following conditions at all $i = 1, \ldots, h$:

$$\Phi_\ell(p_i) = 0$$

$$\nabla\Phi_\ell(p_i) \parallel b_i.$$

Computing a cut is very efficient, as it amounts to solving a (dense) $4h \times 4h$ linear system which depends on the size of the loop, and not on the complexity of the underlying volume mesh. Function Φ_ℓ is evaluated at all vertices of the tetrahedral mesh $\mathcal{MV}$ and its zero isosurface satisfies our requirements. We use a marching tetrahedra algorithm [Doi and Koide 1991] to extract the connected component S_ℓ of such zero isosurface, which has loop ℓ on its boundary. Note that we discard other connected components that possibly exist and intersect the volume elsewhere (Figure 11).

Loop pairing. Note that surface S_ℓ may cut the surface $\mathcal{M}$ in places other than ℓ. This will certainly happen in case ℓ is an incomplete loop, and it may happen also if S_ℓ pierces the shell $\mathcal{M}$ along other loops. See Figure 12 for examples.

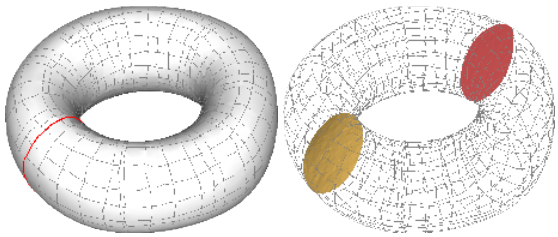


Fig. 11. Cutting along the red loop requires a HRBF whose zero level set intersects the torus at two disjoint connected components. We take just the component incident at the starting loop, discarding the other (in red).

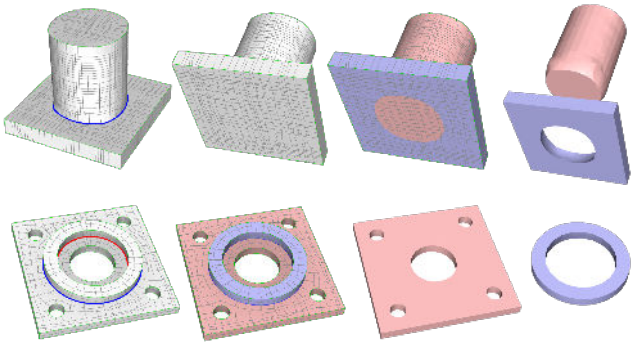


Fig. 12. Top: vertically cutting along the blue loop at the base of the cylinder to extrude it generates a new loop at the bottom of the model which does not conform with the cross field. Bottom: horizzontally cutting along the blue loop generates a membrane that also interpolates the red lop at the inner side of the same feature. In this case a new cutting membrane that interpolates both loops is computed, and used to halve the model.

Let $\bar{\ell}$ be one such loop. In some cases, especially with CAD objects, there might exist some loop $\hat{\ell}$ from $\mathcal{Q}$ that approximates $\bar{\ell}$ closely. The existence of $\hat{\ell}$ is easy to check right after tet classification in the marching tetrahedra algorithm, before splitting the mesh. For each loop in $\mathcal{Q}$, we simply check whether the value of function Φ_ℓ at its vertices, as well as the angle between its bi-normal and $\nabla\Phi_\ell$ are close enough to zero. A loop $\hat{\ell}$ is taken as a good representative for $\bar{\ell}$ if such conditions are met at all its vertices. In all our experiments, the two thresholds for the function value and the angle are set at 0.1 and 10°, respectively.

In case we find one such loop $\hat{\ell}$, we discard the current cutting surface and we generate another one by constraining it at both ℓ and $\hat{\ell}$. The same mechanism applies even if we find more than one such loop, in case the cutting surface has multiple boundary loops.

If we do not find any pairing loop for $\bar{\ell}$, we keep the current cutting surface as is. Note that $\bar{\ell}$ usually will not be field-coherent in the sense defined in the previous section, therefore it might interact with other loops from $\mathcal{Q}$ in an uncontrolled way. The presence of such loops is necessary to comply with situations like the one depicted in Figure 12 (top), which cannot be captured by the surface cross field. On the other hand, their interaction with the other loops may produce artifacts that may need special care during post-processing, as explained in Section 6.

Sanity checks. The cutting surface S_ℓ may intersect other cutting surfaces inserted previously inside the volume. Intersections are found trivially on the basis of the labeling of cells of any dimension in $\mathcal{MV}$, which is updated after any new cut with simple flooding. Before accepting S_ℓ as a valid cut, we make the following sanity checks:

- No boundary tangency: S_ℓ does not contain any triangle, edge or vertex lying on the surface of $\mathcal{M}$, except for vertices and edges belonging to its boundary loops;
- No cut tangency: S_ℓ does not contain any triangle already belonging to another cutting surface;
- No multiple intersections at edges: each edge of S_ℓ may belong to at most another cutting surface;
- No multiple intersection at vertices: each vertex of S_ℓ may belong to at most other two cutting surfaces;
- No bubbles: the boundary of the intersection between S_ℓ and any other cutting surface consists of an open line.

In case surface S_ℓ violates any of the conditions above, we discard the cut and revert to the previous configuration. In this way, we are certain that cuts intersect two by two at edges and three by three at vertices. Consequently, all inner vertices of the metamesh will have valence three. On the outer surface of $\mathcal{MM}$, however, few vertices of valence different from three may be generated, as we will discuss in the next section.

6 HEX-DOMINANT MESHING

Once enough cuts have been incorporated in the volume and the metamesh satisfies both topological and geometric convergence criteria expressed in Section 5.2, $\mathcal{MM}$ is formed of valid polyhedral cells. If all vertices of each such cell have valence three, then cells can be turned into hexahedra with a single step of midpoint refinement [Li et al. 1995]. Due to our cutting strategy and sanity checks reported in the previous section, we guarantee that the valence three condition is satisfied at all internal vertices of the metamesh. On the surface, however, awkward configurations may lead to the generation of vertices with valence 2. In case any vertex with such valence is present, we run a simple heuristic that tries to remove them by either adding or removing cuts. This process is not guaranteed to converge to a pure hexahedral mesh, and may also oversimplify $\mathcal{MM}$, affecting geometric convergence. In the latter case, we just revert to the initial version of the metamesh, and use the procedure described at the end of this section to produce a hex-dominant mesh.

Balancing vertex valence. Vertices with valence 2 arise whenever two loops, or a loop and a convex feature, have two consecutive intersections. A visual example is given in Figure 13. Note that, while we try to avoid this circumstance during loop sampling, by imposing the topological integrity constraint (see Section 4.4), this can happen when: some loop cannot be used to produce a cut because it violates sanity checks; some new loop originated from cuts tangentially intersects twice some other loop or feature; convex features in input were already in such configuration. While we provide no formal

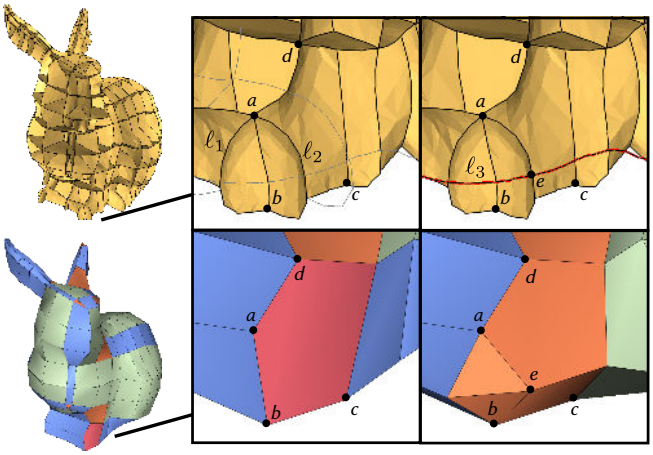


Fig. 13. Top left: loops ℓ_1, ℓ_2 intersect at two distinct points, a, b, and are connected by three chains of edges. Both vertices have valence two with respect to the cells of the metamesh that stay at their left and right (w.r.t. the right cell a is adjacent to b, d, and b is adjacent to a, c). Consequently, the resulting polyhedron (bottom left, in red) does not produce hexahedra under midpoint refinement. Right: adding loop ℓ_3 in the metamesh produces a new point, e, which splits the previous chains connecting a, b, also balancing vertex valence. The resulting polyhedron (in orange) is now hexable with midpoint refinement.

proof, we empirically observed that these are the only practical circumstance originating valence two vertices.

As shown in Figure 13 a possible remedy to balance vertex valences and secure a local full hexahedral decomposition consists in selecting a third loop and either cut through it, or just use it to form surface edges in $\mathcal{MM}$. Since we strive for a minimal decomposition, among the two solutions we opt for the latter, bacause it does not change the number of blocks. If no such loop is found within the non used loops of $\mathcal{Q}$, we try to recover by reverting one of the two offending cuts (the last that was inserted). Removing one loop may open analogous issues in the metamesh elsewhere. We keep removing loops (and related cuts) until either all issues are resolved, or the stop condition defined in Section 5.2 is violated. In the latter case, we revert to the original $\mathcal{MM}$. We observed that this strategy succeeds in approximately 50% of the cases. Finally, if the models contains sharp features, surface vertices may also have valences higher than 3 (imagine the tip of a pyramid with squared base). Since these vertices directly depend from the input creases, we do not attempt any remedy, and keep them in the output mesh, which will therefore be hex-dominant.

Mesh generation. From a topological perspective, the generation of the output mesh starting $\mathcal{MM}$ consists in a simple midpoint refinement. In order to obtain a good geometric approximation of the input mesh, we carefully reposition newly inserted surface vertices. Specifically, each surface face of $\mathcal{MM}$ corresponds to a disk like patch of surface triangles of the tetmesh. Assuming the $\mathcal{MM}$ face contains n sides, we first parameterize each patch on a discrete $n-$gon centered at the origin of the uv space with geometric Tutte, and position the metamesh midpoint at the point on the tetmesh that maps to the origin in uv space. This approach guarantees that the new point is projected to the input surface, regardless of the geometric distance between the $\mathcal{MM}$ face and its corresponding patch in the tet mesh. Edge midpoints are sampled in the same way, using the parameterization associated to one of the two patches adjacent to its sides, and finding the coordinates of the corresponding edge midpoint in parametric space. We eventually improve surface smoothness by applying plain laplacian smoothing for interior vertices, and laplacian smoothing in tangent space for surface vertices. Similarly to [Livesu et al. 2015], we smooth vertices inside features along their corresponding feature lines, and keep vertices at the intersection of multiple features fixed at their original position.

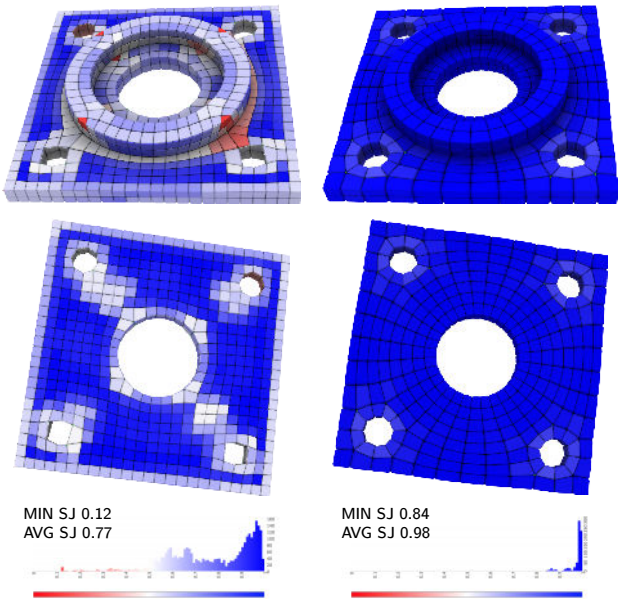


Fig. 14. Even though the polycube mesh (left) is almost four times larger than ours (2.7K vs 0.7K elements), it is not as effective at capturing the sharp features on the object (see the square-like little holes at the bottom). Meshes derived from our block-decomposition naturally align with features, and yield a quality mesh with much higher average and minimum scaled Jacobians. Polycube mesh was computed with [Livesu et al. 2013]; both hexmeshes were optimized for maximal minimum Jacobian with Edge-Cone Rectification [Livesu et al. 2015].

7 RESULTS

We validate our method on a vast range of models, mostly sourced from Hexalab [Bracci et al. 2019] and Thingi10K [Zhou and Jacobson 2016] databases. We tested CAD-like models containing sharp creases of different complexity (Figures 16, 17), organic shapes such as animals or statues, synthetic shapes, and tubular objects (Figures 18, 19). *LoopyCuts* does not require parameter tuning: we used $\varepsilon = 0.1$ to produce all our results. We report statistics in Table 1. Overall, *LoopyCuts* produced full hexahedral meshes in 76% of the cases, and hex-dominant ones in the remaining 24%. As discussed in the previous section, all non-hexahedral elements lie on the outer

Table 1. Output statistics: input mesh size (num. vertices); number of cuts; number of hexahedra (H), prisms (P), andother hexable (O) and non hexable (N) polyhedra in the metamesh; number of hexahedra and other polyhedra after midpoint refinement as well as ratio between them; computation time.

			metamesh				Output Mesh			
Model	**Verts**	**Cuts**	**H**	**P**	**O**	**N**	**H**	**O**	**%**	**Time**
Plate 1	21K	1	2	0	0	0	16	0	100%	4s
Plate 2	4K	1	2	0	0	0	16	0	100%	4s
Plate 3	8K	2	2	1	0	0	28	0	100%	3s
Plate 4	45K	9	29	9	4	0	346	0	100%	42s
Bearing Plate	12K	16	43	12	16	0	712	0	100%	78s
Beveled Shoulder	13K	10	14	0	0	0	112	0	100%	22s
Bimba	29K	24	97	82	40	0	1560	0	100%	81s
Blade	30K	48	214	217	104	16	4102	23	99.4%	203s
BladeFEM	13K	70	490	107	77	12	5338	23	99.6%	200s
Bolt	14K	19	32	56	24	0	784	0	100%	47s
Bone	7K	24	54	80	18	0	1016	0	100%	27s
BracketInches	26K	90	732	438	574	64	14573	89	99.3%	979s
Bunny	25K	32	162	88	46	0	2172	0	100%	112s
Busto Bimba	8K	28	109	97	42	0	1768	0	100%	73s
Cactus	3K	39	86	136	20	0	1616	0	100%	41s
Cat	12K	20	111	71	72	0	1888	0	100%	65s
Clef	5K	61	0	200	24	0	1328	0	100%	42s
Cube w circle	2K	11	56	16	8	0	812	0	100%	43s
Cube Carved	14K	2	14	6	4	0	212	0	100%	26s
Cube Minus Sphere	15K	5	3	3	1	0	64	0	100%	3s
Cup	38K	7	29	4	0	0	256	0	100%	88s
Cylinder Plate	11K	4	0	12	0	0	88	0	100%	10s
Dancer	27K	61	87	208	96	9	2776	18	99.4%	249s
Des6	14K	90	1321	310	312	23	15598	34	99.8%	639s
Deckel	20K	25	329	92	131	36	4784	72	98.5%	128s
Dog	8K	38	54	123	43	0	1816	0	100%	116s
Ellipse	12K	11	24	28	8	0	392	0	100%	29s
Eraser Ball	12K	39	144	211	178	0	3900	0	100%	129s
Femur	13K	47	270	154	18	2	3524	4	99.8%	102s
Gear	14K	23	39	6	6	0	406	0	100%	50s
Gyroidpuzzle	23K	73	181	68	104	24	2788	39	98.4%	450s
Halved Oblique Scarf	15K	75	622	54	40	13	5854	23	99.6%	281s
Hand	7K	61	77	100	15	0	1300	0	100%	137s
Hanger	12K	21	64	10	0	10	688	14	98%	40s
Hinge	20K	24	6	19	24	0	428	0	100%	68s
Holes2	8K	61	45	39	12	0	858	0	100%	394s
Impeller	30K	30	16	96	0	0	1024	0	100%	205s
Indorelax	20K	61	40	120	124	19	2146	41	98.1%	296s
Inlay Dovetail	27K	11	18	0	0	0	144	0	100%	44s
Joint	4K	12	56	12	0	0	650	0	100%	51s
Kiss	25K	56	145	278	397	41	6662	88	98.6%	413s
Kitten	12K	22	87	66	34	0	1728	0	100%	71s
Knob	18K	22	53	24	22	0	744	0	100%	53s
Kong	39K	21	55	100	55	0	1520	0	100%	93s
Lever Arm	10K	31	50	37	24	0	818	0	100%	46s
Mech10	2K	7	16	15	3	0	230	0	100%	8s
Mech Piece	33K	5	3	4	0	0	XX	0	100%	21s
Mechanical02	12K	33	34	102	63	0	1372	0	100%	65s
Mechanical05	12K	87	502	298	269	8	8398	13	99.8%	488s
Mechanical06 (teaser)	17K	93	287	265	101	0	4744	0	100%	533s
Mechanical08	5K	73	652	121	24		11348	42	99.6%	453s
Metatron	9K	71	47	14	0	0	568	0	100%	121s
Mid2FEM	17K	33	72	20	0	0	908	0	100%	57s
Motor Tail	22K	54	54	44	2	0	916	0	100%	163s
Nugear	36K	86	370	98	30	8	3964	15	99.6%	727s
Pig	16K	16	124	54	38	0	1592	0	100%	51s
Pinion	24K	30	0	0	0	0	240	0	100%	61s
Prism	7K	0	0	1	0	0	12	0	100%	1s
Rabbit	21K	60	200	176	50	0	3040	0	100%	460s
Rod	12K	21	11	28	14	0	704	0	100%	47s
Sculpt	12K	14	12	8	0	0	168	0	100%	18s
Sphinx	28K	60	178	267	70	0	3944	0	100%	672s
Teapot	17K	34	97	65	65	13	2380	24	99%	145s
Torque	10K	16	57	58	11	0	880	0	100%	38s
Torque1	15K	57	155	97	72	0	7320	0	100%	797s
Torus	8K	35	41	82	0	0	1968	0	100%	350s
Trebol	21K	14	14	45	25	0	552	0	100%	32s
Tris Open	8K	9	15	0	0	0	120	0	100%	13s
Tris Closed	13K	3	3	1	0	0	30	0	100%	4s
U-joint	15K	9	8	0	4	0	48	0	100%	9s
Vessel	7K	61	10	134	22	0	1000	0	100%	80s
Vertebrae	20K	59	711	343	298	32	7682	42	99.5%	357s
Wave	8K	7	9	5	0	0	136	0	100%	7s
Wedge	12K	1	1	1	0	0	14	0	100%	2s
Wheel	38K	83	419	302	115	21	7316	33	99.6%	872s
Wrench	7K	11	7	6	0	0	100	0	100%	5s

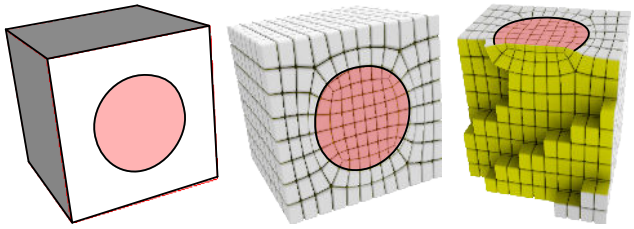


Fig. 15. Left: an input cube with a user-demarcated circular feature on one of its faces. Middle: the hexahedral mesh produced by LoopyCuts. Right: cut through view showing the inner mesh connectivity (the cube is rotated to highlight the singular structure beneath the circular feature).

surface. Following midpoint subdivision all cell faces are quadrilateral, and all internal or non-feature vertices have valence three w.r.t. each of their incident cells. The valence of surface vertices at the corners of the feature network depends on the network valence (e.g. on an 8 sided pyramid the valence of the tip will be 8). After subdivision most cells have 6 faces, but we have no theoretical bound on face count. In our experiments we encountered cells with up to 10 faces. In all cases, the ratio between the number of hexahedra and the total number of elements in our hex-dominant meshes was above 98%. As Figure 20 shows, we generate full hexahedral meshes across many inputs for which state-of-the-art hex-dominant meshing techniques introduce multiple non-hex elements. In general, our method produces meshes with much higher percentage of hex elements than previous hex-dominant methods such as polyhedral agglomeration [Gao et al. 2017a] and PGP3D [Ray et al. 2018; Sokolov et al. 2016b]. Meshes produced with the former have only 60% to 90% hex elements (80% on average) and contain complex hybrid elements that cannot be turned into hexahedra with midpoint refinement, and may have up to 40 facets (see Table 1 in [Gao et al. 2017a]). Meshes produced with PGP3D span from 33% to 95%.

Sharp crease alignment. One of the key advantages of our tool is its ability to directly incorporate sharp creases into the block decomposition. Methods that work in the dual space typically use snapping to place edges on the creases after dualization. Similarly, methods based on a parameterization snap function values to make sure that each crease is interpolated by an integer isoline. Both procedures are inherently fragile, and may introduce poor (or even flipped) elements in the mesh. Using LoopyCuts features can be directly incorporated into the metamesh as surface edges, and therefore reproduced exactly into the resulting mesh (Figure 14). As Figure 15 shows it fully supports non-geometric features; any curve defined on the surface can be tagged as a feature and either used for cutting, or just included in the metamesh as a chain of surface edges. We currently support feature curves made of chains of consecutive mesh edges; isolated (point-like) features could also be addressed by tracing two orthogonal loops starting from each such point.

Mesh structure. As discussed in [Cherchi et al. 2016; Gao et al. 2017b] hexmeshes with a coarse block structure are preferable in a variety of engineering applications, e.g. because they can be encoded more efficiently, and their domains can be used to fit tensor products

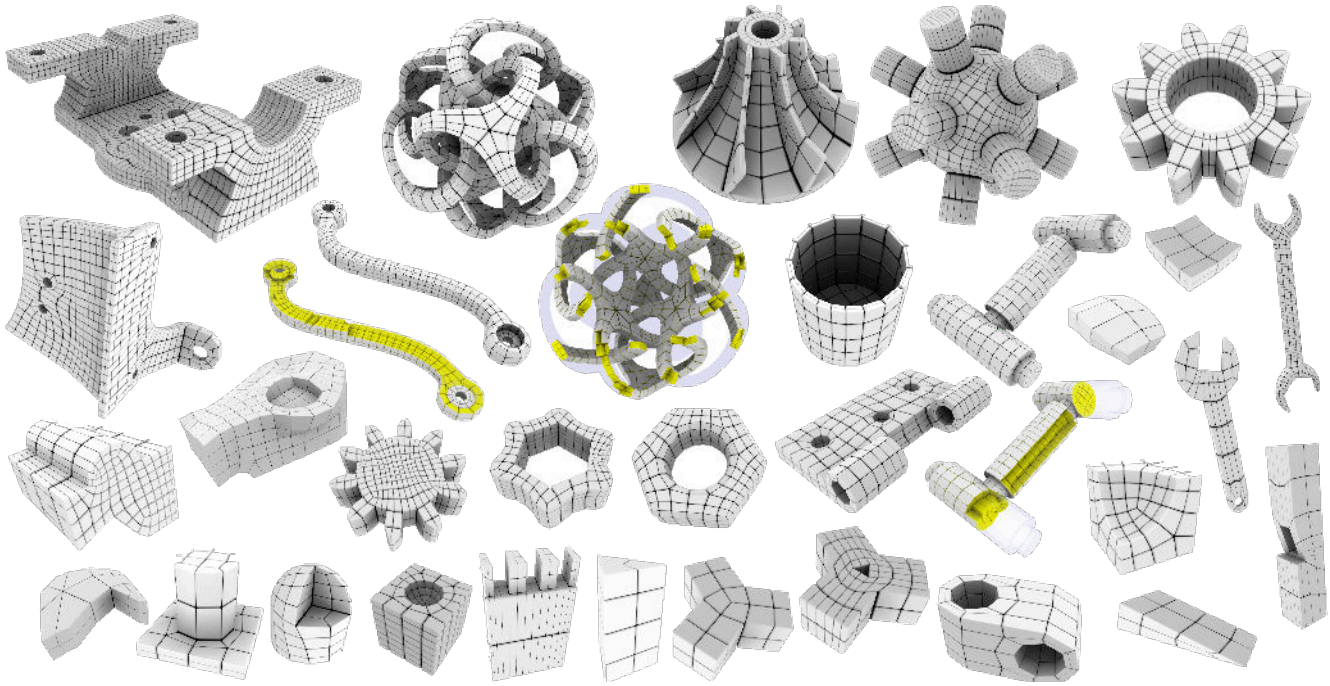


Fig. 16. A gallery of full hexahedral meshes of CAD models produced with our method.

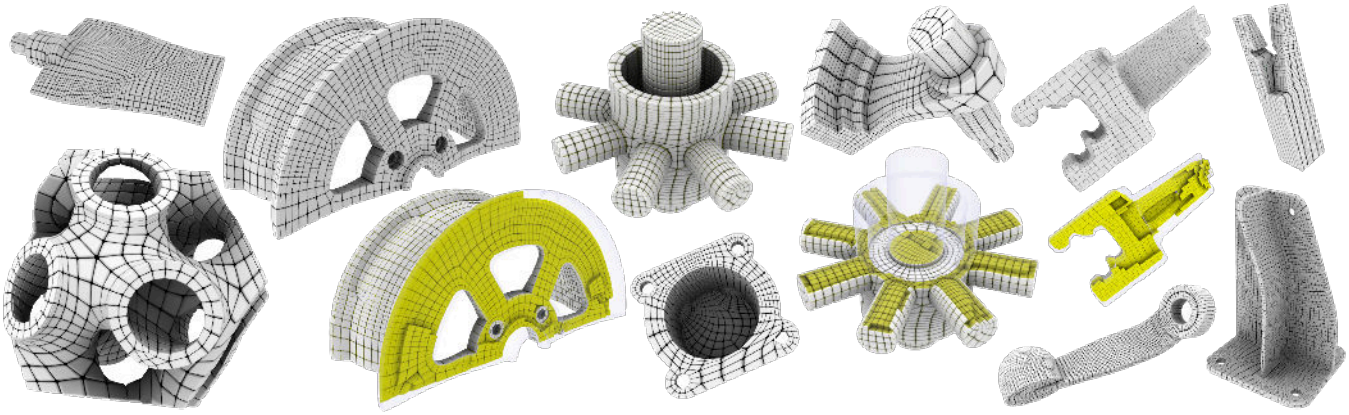


Fig. 17. A gallery of hex dominant meshes of CAD models produced with our method.

to solve PDEs with IGA. Our meshes are primarily driven by the input field, which is then propagated inwards by the cutting system. As demonstrated by the output cross-sections shown, the internal structure of our meshes is coarser and more regular compared to those produced by octree-based methods, on par with polycube and frame field methods (Figure 21).

Comparison with [Takayama 2019]. Takayama propose a system to create hexahedral meshes by interactively tracing dual sheets. While there are fundamental differences between the two methods (dual vs primal, interactive vs fully automatic) there are also a few analogies. We both use HRBF to define cuts, and both use a reference tetrahedral mesh to store information about blocks and to extract the final mesh. As Figure 22 shows the meshes produced by the two systems are of comparable quality. However, as acknowledged by the author, designing a mesh in the dual space is quite challenging, and the generation of complex models like the bunny may take hours of work even for experts. Additionally, due to implementation constraints, the system proposed in [Takayama 2019] cannot use cuts that do not globally split the object in multiple connected components, making meshing impossible on objects with non-zero genus, such as the torus. Our system supports such cuts.

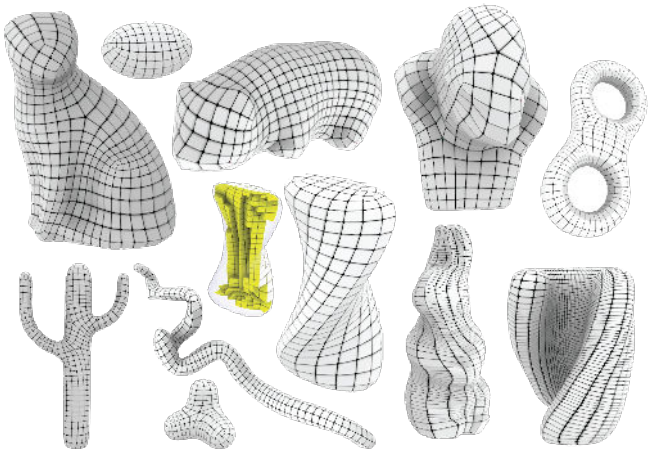


Fig. 18. A gallery of hexahedra meshes of organic and synthetic models produced with our method.

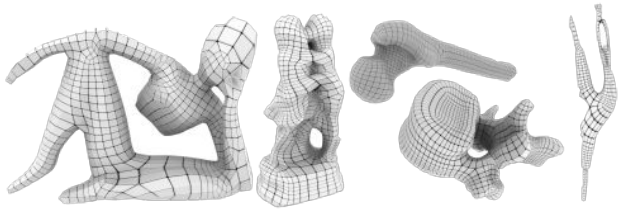


Fig. 19. A gallery of hex dominant meshes of organic models produced with our method.

Customization. Our LoopyCuts algorithm is fully automatic, but can support different controls for customizing the block decomposition. In particular, users can: (i) prescribe a customized cut sequence; (ii) manually pair loops to define cuts that interpolate multiple cycles; (iii) mark an open or closed curve as a feature, in order to incorporate it in the metamesh; (iv) customize convergence, stopping the cutting process at any moment. Figure 23 demonstrates the use of a subset of these features to produce a coarse full hexahedral mesh of a model with complex feature lines for which the automatic mode produces a dense hybrid mesh.

Implementation details. We implemented *LoopyCuts* as a single threaded C++ application, using VCG [Visual Computing Lab 2018] for field processing, CinoLib [Livesu 2017] for polyhedral meshes, Tetgen [Si 2015] for tetrahedralization, and Eigen [Guennebaud et al. 2010] for numerics. Cross fields aligned to line features and surface curvature were computed using MIQ [Bommes et al. 2009] and PolyVector Fields [Diamanti et al. 2014]. Sharp creases were automatically detected by thresholding dihedral angles, whereas other features were manually marked. Note that both field computation and feature detection are external to our algorithm, and alternative techniques may be used. *LoopyCuts* is agnostic to how this information is computed.

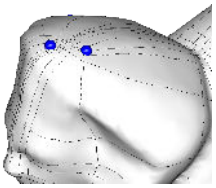
Table 2. Element quality comparisons against [Gao et al. 2019] and [Takayama 2019]. For each method we report minimum and average Scaled Jacobians. Note that [Gao et al. 2019] already optimize their meshes for maximum quality, while [Takayama 2019] and LoopyCuts do not. For these latter methods we therefore report quality before and after mesh optimization, performed with [Livesu et al. 2015].

	[Gao et al. 2019]	[Takayama 2019]		*Loopy Cuts*	
Bone	0.41/0.80	0.05/0.74	0.43/0.86	-0.43/0.85	0.47/0.89
Bunny	0.29/0.79	-0.77/0.74	-0.78/0.81	-0.37/0.88	0.45/0.91
Joint	0.19/0.88	0.25/0.93	0.69/0.96	0.39/0.87	0.72/0.95
Rod	0.04/0.81	0.22/0.76	0.55/0.87	0.10/0.84	0.32/0.93

8 CONCLUSIONS

We presented a new approach to block-decomposition for hex-dominant meshing. As demonstrated *LoopyCuts* outperforms prior hex-dominant meshing methods providing pure hexahedral meshes for 76% of the inputs, and hybrid meshes with less than 2% of non-hexahedral cells for the remaining ones. Our strategic choice to operate directly on the primal mesh allows us to preserve all input feature curves, contrary to many prior approaches. Our technical contributions consist of: a novel method to trace geodesic field aligned closed loops; the definition of a metric that induces a regular sampling of such loops on the surface; and a robust algorithm to transform a labeled tetrahedral mesh into a polyhedral mesh. We will release our data and a reference implementation of *LoopyCuts* in a public GitHub repository. In particular, pure hexahedral meshes will be shared on Hexalab [Bracci et al. 2019].

Limitations and Future work. While we obtained valid results on all the models we processed, there is no theoretical guarantee that the decomposition produced from a given sequence of loops will satisfy all topological and geometric constraints. We envision three potential failure cases. First, our loop formation strategy relies on the underlying cross-field. On surfaces where the cross field directions change multiple times, the resulting loops may be too complicated or the tracing may not be able to close loops properly, avoiding self-intersections. Second, tangential intersection between cuts may create inner pockets and force the algorithm to revert the majority of the cuts, leading to an incomplete decomposition where the topology of each block is not rich enough to define a proper polyhedron. Lastly, we cannot guarantee that the loop set defined in Section 4 is large enough to accommodate a valid decomposition of the input object. Our method can produce valid but poor quality meshes in two known scenarios: when the field has a poor singularity layout, exhibiting close by singular vertices that trigger an uneven distribution of sampled loops (this resulted in the uneven meshing of the Sphinx in Figure 20, see inset) or when long and highly non planar loops traverse surface regions with high normal variation. Cuts associated with such loops may be poorly shaped, producing decompositions that do not tightly conform to the input driving field. While rare, a few such configurations occurred during our experiments (Figure 24). Finally, a practical limitation of our current implementation stems from the fact that we use simple



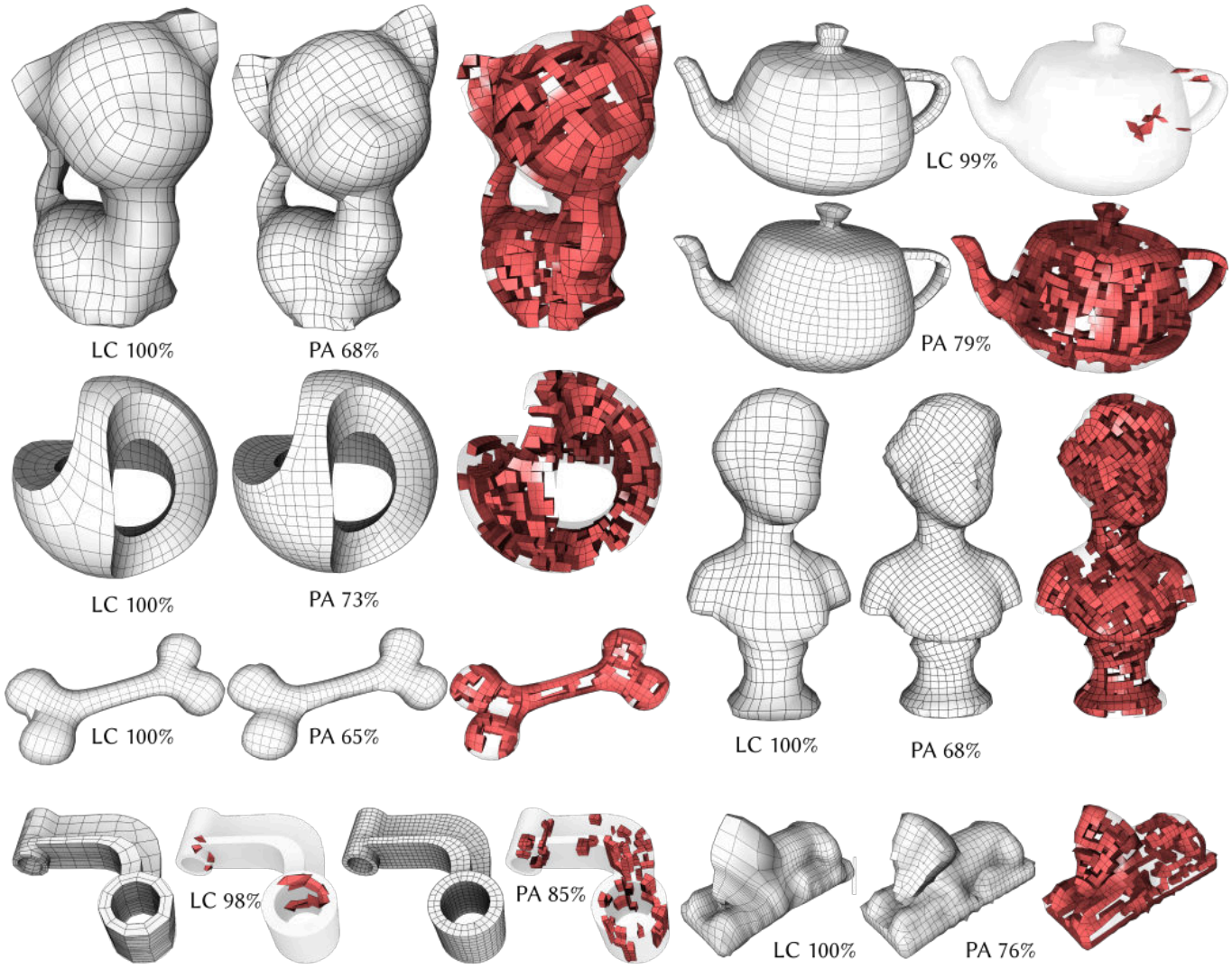


Fig. 20. Visual comparison between Loopy cuts (LC) and the polyhedral agglomeration (PA) method proposed in [Gao et al. 2017a]. LC produces hexahedral meshes for many of their results. In all the other cases, the amount of non hexahedral elements (in red) never exceeds 2%, significantly less than PA.

Laplacian smoothing to relax mesh vertices after midpoint refinement. Therefore, currently produced meshes may contain elements with negative Jacobian, or have minor local projection flaws introduced by the smoother. These defects can be addressed with more advanced optimization schemes, see Figure 14 and Table 2, where a few meshes have been optimized using [Livesu et al. 2015] and compared with [Gao et al. 2019; Livesu et al. 2013; Takayama 2019].

Our method can be augmented with additional user controls. Further indirect control can be provided by editing the guiding surface field using interactive tools such as those described by [Jakob et al. 2015]. Such tools can be used to prescribe additional soft constraints on the meshing process promoting alignment to secondary features or symmetries. Addressing all the above aspects is an interesting avenue for future research.

ACKNOWLEDGMENTS

We are deeply grateful to Nicholas Vining for help with paper editing and proofing. This work is supported by NSERC and by the Italian Ministry of Education, University and Research under Program PRIN 2015, Project DSurf, Grant $N.2015B8TRFM002$.

REFERENCES

T. Blacker. 1996. The Cooper Tool. In *Proc. 5th Int. Meshing Roundtable*. 13–29.
T. Blacker. 2000. Meeting the challenge for automated conformal hexahedral meshing. In *Proc. 9th Int. Meshing Roundtable*. 11–20.
I. Boier-Martin, H. Rushmeier, and J. Jin. 2004. Parameterization of triangle meshes over quadrilateral domains. In *Proc. Eurographics Symp. on Geom. Proc.* 193–203.
D. Bommes, H. Zimmer, and L. Kobbelt. 2009. Mixed-integer quadrangulation. *ACM Trans. Graph.* 28, 3 (2009), 77.
M. Bracci, M. Tarini, N. Pietroni, M. Livesu, and P. Cignoni. 2019. HexaLab. net: an online viewer for hexahedral meshes. *Computer-Aided Design* 110 (2019), 24–36.
M. Campen, D. Bommes, and L. Kobbelt. 2012. Dual loops meshing: quality quad layouts on manifolds. *ACM Trans. Graph.* 31, 4 (2012), 110.

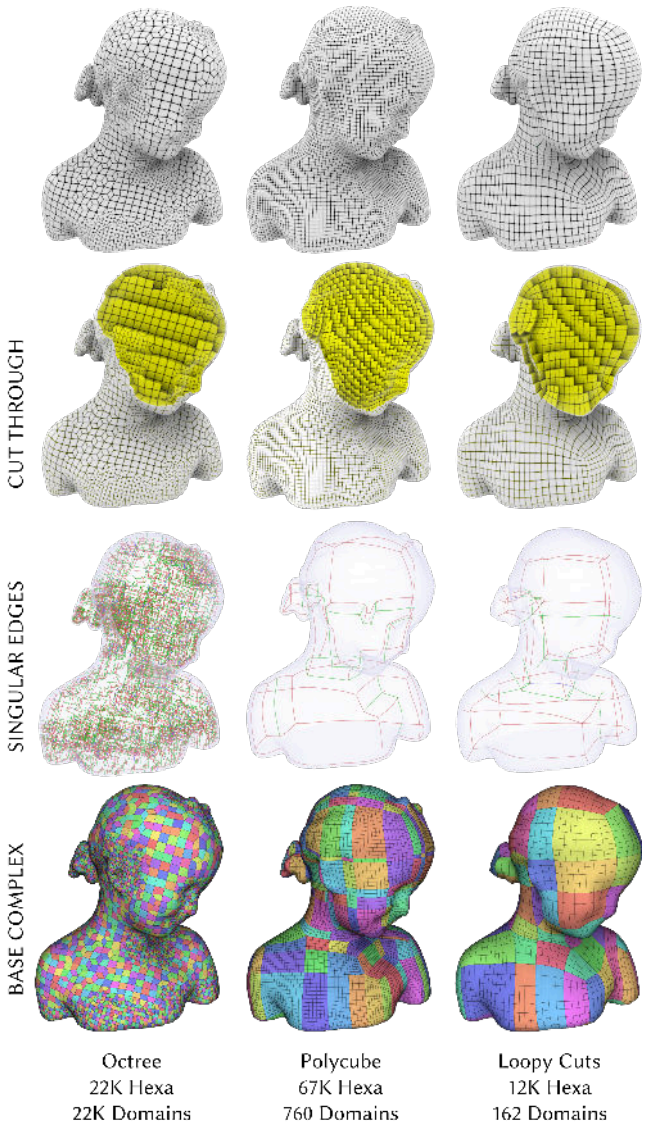


Fig. 21. Hexahedral meshes of the Bimba model, obtained with an octree method [Gao et al. 2019], a polycube method [Fu et al. 2016], and LoopyCuts. The octree method's output does not align with surface curvature, and has a dense singular structure where each hexahedron is a separate $1\times1\times1$ domain of the base complex. Our cutting method is guided by a surface field, and – similarly to polycubes – generates a sparse singular structure that aligns with the boundary, producing a coarse block layout.

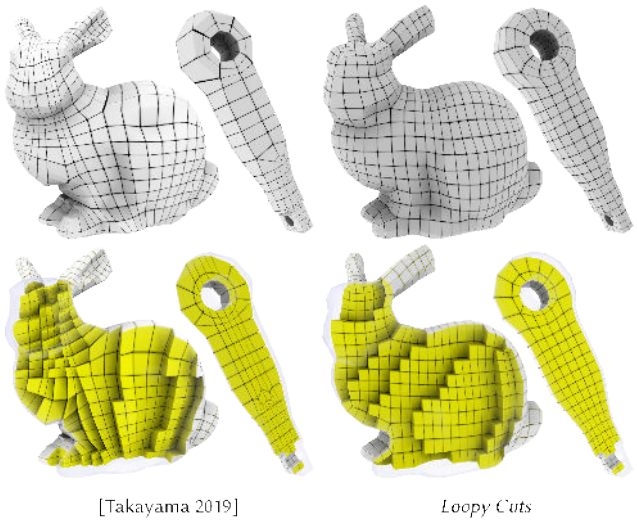


Fig. 22. Hexmeshes obtained semi-manually using dual sheet modeling [Takayama 2019] (left) and LoopyCuts (right). While the meshes have comparable quality, manually designing a hexmesh in the dual space is a complex process that may takes hours even for experts. Loopy cuts took 112 and 47 seconds to automatically hexmesh the Bunny and Rod, respectively.

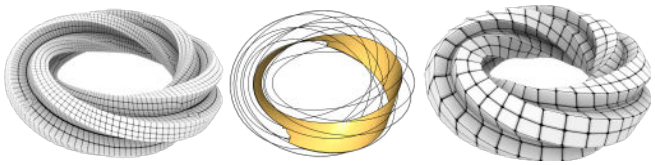


Fig. 23. Left: in automatic mode LoopyCuts produces a dense hybrid mesh. Non hexahedral elements are induced by cuts that deviate from the sharp creases (middle). Right: In user-assisted mode users can selectively apply only the wanted cuts, producing a much coarser full hexahedral mesh (right).

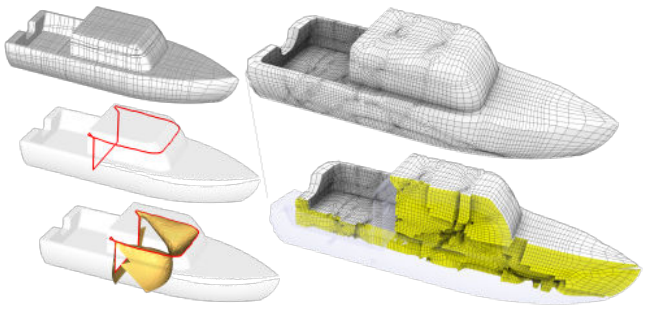


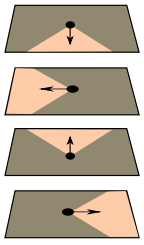
Fig. 24. Complex long loops that traverse surface areas with high normal variation may introduce poor cuts (left), that eventually produce a mesh where the surface edge flow is not compliant with the input cross field (right). In our experimentation we encountered only a few pathological cases of this kind, over more than a hundred models tested.

N. A. Carr, J. Hoberock, K. Crane, and J. C. Hart. 2006. Rectangular Multi-chart Geometry Images. In *Proc. Eurographics Symp. on Geom. Proc.* 181–190.
G. Cherchi, M. Livesu, and R. Scateni. 2016. Polycube Simplification for Coarse Layouts of Surfaces and Volumes. *Comput. Graph. Forum* 35, 5 (2016), 11–20.
E. Corman and K. Crane. 2019. Symmetric Moving Frames. *ACM Trans. Graph.* 38, 4 (2019), 1–16.
M. Corsini, P. Cignoni, and R. Scopigno. 2012. Efficient and Flexible Sampling with Blue Noise Properties of Triangular Meshes. *IEEE Trans. Vis. Comput. Graph.* 18, 6 (June 2012), 914–924.
J. Daniels II, C. T. Silva, and E. Cohen. 2009. Semi-regular Quadrilateral-only Remeshing from Simplified Base Domains. *Comput. Graph. Forum* 28 (July 2009), 1427–1435.
F. de Goes, M. Desbrun, and Y. Tong. 2016. Vector field processing on triangle meshes. In *ACM SIGGRAPH 2016 Courses*. ACM, 27.
D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella. 2003. Suggestive Contours for Conveying Shape. *ACM Trans. Graph.* 22, 3 (2003), 848–855.

O. Diamanti, A. Vaxman, D. Panozzo, and O. Sorkine-Hornung. 2014. Designing N-PolyVector Fields with Complex Polynomials. *Comput. Graph. Forum* 33, 5 (2014), 1–11.
A. Doi and A. Koide. 1991. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE Trans. Inf. and Sys.* E74, 1 (1991), 214–224.
X. Fang, W. Xu, H. Bao, and J. Huang. 2016. All-hex Meshing Using Closed-form Induced Polycube. *ACM Trans. Graph.* 35, 4 (2016), 124:1–124:9.
X.-M. Fu, C.-Y. Bai, and Y. Liu. 2016. Efficient Volumetric PolyCube-Map Construction. In *Comput. Graph. Forum*, Vol. 35. 97–106.
S. Gao, Z. Zheng, R. Wang, Y. Liao, and M. Ding. 2018. Dual Surface Based Approach to Block Decomposition of Solid Models. In *Proc. 27th Int. Meshing Roundtable.*
X. Gao, Z. Deng, and G. Chen. 2015. Hexahedral mesh re-parameterization from aligned base-complex. *ACM Trans. Graph.* 34, 4 (2015), 142.
X. Gao, W. Jakob, M. Tarini, and D. Panozzo. 2017a. Robust Hex-dominant Mesh Generation Using Field-guided Polyhedral Agglomeration. *ACM Trans. Graph.* 36, 4 (July 2017), 114:1–114:13.
X. Gao, D. Panozzo, W. Wang, Z. Deng, and G. Chen. 2017b. Robust structure simplification for hex re-meshing. *ACM Trans. Graph.* 36, 6 (2017), 185.
X. Gao, H. Shen, and D. Panozzo. 2019. Feature Preserving Octree-Based Hexahedral Meshing. In *Comput. Graph. Forum*, Vol. 38. 135–149.
J. Gregson, A. Sheffer, and E. Zhang. 2011. All-Hex Mesh Generation via Volumetric PolyCube Deformation. *Comput. Graph. Forum* 30, 5 (2011), 1407–1416.
G. Guennebaud, B. Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org. (2010).
J. Huang, T. Jiang, Z. Shi, Y. Tong, H. Bao, and M. Desbrun. 2014. L1-Based Construction of Polycube Maps from Complex Shapes. *ACM Trans. Graph.* 33, 3 (2014), 25.
J. Huang, Y. Tong, H. Wei, and H. Bao. 2011. Boundary aligned smooth 3D cross-frame field. In *ACM Trans. Graph.*, Vol. 30. ACM, 143.
Y. Ito, A. M. Shih, and B. K. Soni. 2009. Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates. *Int. Jou. Num. Meth. Eng.* 77, 13 (2009), 1809–1833.
W. Jakob, M. Tarini, D. Panozzo, and O. Sorkine-Hornung. 2015. Instant field-aligned meshes. *ACM Trans. Graph.* 34, 6 (2015), 189–1.
T. Jiang, J. Huang, Y. Wang, Y. Tong, and H. Bao. 2014. Frame field singularity correction for automatic hexahedralization. *IEEE Trans. Vis. Comput. Graph.* 20, 8 (2014), 1189–1199.
F. Kälberer, M. Nieser, and K. Polthier. 2007. QuadCover: Surface Parameterization using Branched Coverings. *Comput. Graph. Forum* 26, 3 (2007), 375–384.
N. Kowalski, F. Ledoux, and P. Frey. 2016. Smoothness driven frame field generation for hexahedral meshing. *Computer-Aided Design* 72 (2016), 65–77.
Nicolas Kowalski, Franck Ledoux, Matthew L Staten, and Steve J Owen. 2012. Fun sheet matching: towards automatic block decomposition for hexahedral meshes. *Engineering with Computers* 28, 3 (2012), 241–253.
M. Lanthier, A. Maheshwari, and J.-R. Sack. 2001. Approximating shortest paths on weighted polyhedral surfaces. *Algorithmica* 30, 4 (2001), 527–562.
B. Lévy and Y. Liu. 2010. Lp Centroidal Voronoi Tesselation and its Applications. *ACM Trans. Graph.* 29, 4 (2010), 119.1–119.11.
T. S. Li, R. M. McKeag, and C. G. Armstrong. 1995. Hexahedral meshing using midpoint subdivision and integer programming. *Comput. Methods App. Mech. Eng.* 124, 1-2 (1995), 171–193.
Y. Li, Y. Liu, W. Xu, W. Wang, and B. Guo. 2012. All-hex Meshing Using Singularity-restricted Field. *ACM Trans. Graph.* 31, 6 (2012), 177.1–177.11.
H. Lin, S. Jin, H. Liao, and Q. Jian. 2015. Quality Guaranteed All-hex Mesh Generation by a Constrained Volume Iterative Fitting Algorithm. *Computer-Aided Design* 67, C (2015), 107–117.
H. Liu, P. Zhang, E. Chien, J. Solomon, and D. Bommes. 2018. Singularity-constrained octahedral fields for hexahedral meshing. *ACM Trans. Graph.* 37, 4 (2018), 93.
S. Liu and R. Gadh. 1997. Automatic Hexahedral Mesh Generation by Recursive Convex and Swept Volume Decomposition. In *Proc. 6th Int. Meshing Roundtable.* 217–231.
M. Livesu. 2017. cinolib: a generic programming header only C++ library for processing polygonal and polyhedral meshes. https://github.com/mlivesu/cinolib/. (2017).
M. Livesu, M. Attene, G. Patané, and M. Spagnuolo. 2017. Explicit Cylindrical Maps for General Tubular Shapes. *Computer-Aided Design* 90 (2017), 27 – 36.
M. Livesu, A. Muntoni, E. Puppo, and R. Scateni. 2016. Skeleton-driven Adaptive Hexahedral Meshing of Tubular Shapes. *Comput. Graph. Forum* 35, 7 (2016), 237–246.
M. Livesu, A. Sheffer, N. Vining, and M. Tarini. 2015. Practical Hex-Mesh Optimization via Edge-Cone Rectification. *ACM Trans. Graph.* 34, 4 (2015).
M. Livesu, N. Vining, A. Sheffer, J. Gregson, and R. Scateni. 2013. PolyCut: Monotone Graph-Cuts for PolyCube Base-Complex Construction. *ACM Trans. Graph.* 32, 6 (2013).
J. H.-C. Lu, W. R. Quadros, and K. Shimada. 2017. Evaluation of user-guided semi-automatic decomposition tool for hexahedral mesh generation. *Jou. Comput. Design Eng.* 4, 4 (2017), 330–338.
I. Macêdo, J. P. Gois, and L. Velho. 2011. Hermite Radial Basis Functions Implicits. *Comput. Graph. Forum* 30, 1 (2011), 27–42.
L. Maréchal. 2009. Advances in Octree-Based All-Hexahedral Mesh Generation: Handling Sharp Features. In *Proc. 18th Int. Meshing Roundtable.* 65–84.
S. Meshkat and D. Talmor. 2000. Generating a mixed mesh of hexahedra, pentahedra and tetrahedra from an underlying tetrahedral mesh. *Int. Jou. Num. Meth. Eng.* 49, 1-2 (2000), 17–30.
K. Miyoshi and T. Blacker. 2000. Hexahedral Mesh Generation Using Multi-Axis Cooper Algorithm. In *Proc. 9th Int. Meshing Roundtable.* 89–97.
M. Nieser, U. Reitebuch, and K. Polthier. 2011. CubeCover - Parameterization of 3D Volumes. *Comput. Graph. Forum* 30, 5 (2011), 1397–1406.
Steven Owen. 2009. A Survey of Unstructured Mesh Generation Technology. http://www.andrew.cmu.edu/user/sowen/survey/hexsurv.html. (2009).
J. Pellerin, A. Johnen, and J.-F. Remacle. 2017. Identifying combinations of tetrahedra into hexahedra: a vertex based strategy. *Procedia Engineering* 203 (2017), 2–13. 26rd International Meshing Roundtable (IMR26).
N. Pietroni, E. Puppo, G. Marcias, R. Scopigno, and P. Cignoni. 2016. Tracing Field-Coherent Quad Layouts. *Comput. GraphForum* 35, 7 (2016), 485–496.
W. R. Quadros. 2014. LayTracks3D: A New Approach to Meshing General Solids using Medial Axis Transform. *Procedia Engineering* 82 (2014), 72 – 87. 23rd International Meshing Roundtable (IMR23).
N. Ray, D. Sokolov, M. Reberol, F. Ledoux, and B. Levy. 2018. Hex-dominant meshing: mind the gap! *Computer-Aided Design* 102 (2018), 94–103.
E. Ruiz-Gironés, X. Roca, and J. Sarrate. 2011. Using a computational domain and a three-stage node location procedure for multi-sweeping algorithms. *Advances in Eng. Software* 42, 9 (2011), 700–713.
R. Schneiders. 1996. A grid-based algorithm for the generation of hexahedral element meshes. *Eng. with Computers* 12, 3 (1996), 168–177.
A. Sheffer, M. Etzion, A. Rappoport, and M. Bercovier. 1999. Hexahedral Mesh Generation using the Embedded Voronoi Graph. *Eng. with Computers* 15, 3 (1999), 248–262.
J. F. Shepherd and C. R. Johnson. 2008. Hexahedral mesh generation constraints. *Eng. with Computers* 24, 3 (2008), 195–213.
H. Si. 2015. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Software* 41, 2 (2015), 11.
D. Sokolov, N. Ray, L. Untereiner, and B. Lévy. 2016a. Hexahedral-Dominant Meshing. *ACM Trans. Graph.* 35, 5 (2016), 157:1–157:23.
Dmitry Sokolov, Nicolas Ray, Lionel Untereiner, and Bruno Lévy. 2016b. Hexahedral-dominant meshing. *ACM Transactions on Graphics (TOG)* 35, 5 (2016), 1–23.
J. Solomon, A. Vaxman, and D. Bommes. 2017. Boundary element octahedral fields in volumes. *ACM Trans. Graph.* 36, 3 (2017), 28.
K. Takayama. 2019. Dual Sheet Meshing: An Interactive Approach to Robust Hexahedralization. *Comput. Graph. Forum* 38, 2 (2019), 37–48.
M. Tarini, K. Hormann, P. Cignoni, and C. Montani. 2004. Polycube-maps. In *ACM Trans. Graph.*, Vol. 23. ACM, 853–860.
A. Vaxman, M. Campen, O. Diamanti, D. Panozzo, D. Bommes, K. Hildebrandt, and M. Ben-Chen. 2016. Directional field synthesis, design, and processing. In *Comput. Graph. Forum*, Vol. 35. 545–572.
Visual Computing Lab. 2018. *The VCG Library.* Italian National Research Council - ISTI. http://vcg.isti.cnr.it/vcglib/
R. Wang, C. Shen, J. Chen, H. Wu, and S. Gao. 2017. Sheet operation based block decomposition of solid models for hex meshing. *Computer-Aided Design* 85 (2017), 123 – 137. 24th International Meshing Roundtable (IMR24).
S. Yamakawa and K. Shimada. 2002. Hex-dominant mesh generation with directionality control via packing rectangular solid cells. In *Proc. Geometric Modeling and Processing.* IEEE, 107–118.
Q. Zhou and A. Jacobson. 2016. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797* (2016).

A COMPUTING CUTTING LOOPS

A.1 Field-coherent loops



Following [Kälberer et al. 2007], the four components of a cross field $\mathbf{X}$ can be separated on a stratification $\mathcal{M}_4$ of manifold $\mathcal{M}$ into four sheets, defined as follows (inset). For every point p of $\mathcal{M}$, except the singularities of $\mathbf{X}$, consider four copies p_0, $p_{\frac{\pi}{2}}$, p_π and $p_{\frac{3}{2}\pi}$, each consisting of p together with one of the four directions of $\mathbf{X}$ at p, such that $p_0 = -p_\pi$ and $p_{\frac{\pi}{2}} = -p_{\frac{3}{2}\pi}$. We call each such copy p_θ of p a *point-arrow* meaning that it incorporates both a position on $\mathcal{M}$ and a direction on its tangent space. Space $\mathcal{M}_4$ consists of four sheets,

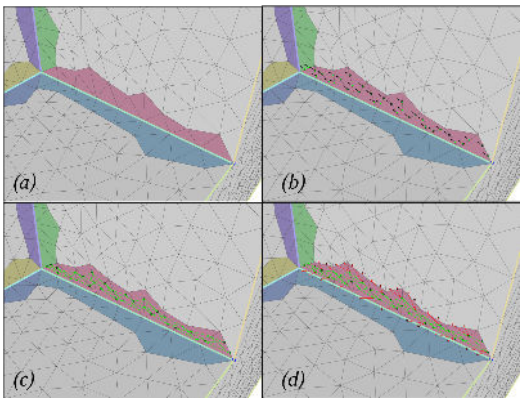


Fig. 25. Tracing loops along curve features requires modifying the graph $\mathcal{G}$ to avoid the endpoint singularities: we consider the triangles on one side of a curve feature (a), we choose the Steiner points of $\mathcal{G}$ of that side (b) and we weight to zero the red arcs inside the corridor (c), and inhibit the green arcs exiting from the corridor (d).

each corresponding to $\mathcal{M}$ less the singularities of $\mathbf{X}$, such that the point-arrows p_θ defined above are distributed among the layers to form a smooth direction field (see inset). Generally speaking, if $\mathbf{X}$ has singularities, the direction field on $\mathcal{M}_4$ turns about such singularities, thus sliding between different sheets, and $\mathcal{M}_4$ consists of a single connected component. Space $\mathcal{M}_2$ is the quotient space of $\mathcal{M}_4$ obtained by identifying pairs of point-arrows p_θ and $-p_\theta$, thus consisting of two sheets, each endowing a line field. Manifold $\mathcal{M}$ can be also seen as a quotient space of $\mathcal{M}_4$, by identifying the four point-arrows at each point p.

Following [Pietroni et al. 2016], a smooth (oriented) curve ℓ on $\mathcal{M}_4$ is said to be a *field-coherent* path if its tangent direction at all points does not differ for more than $\pi/4$ from the underlying direction field on $\mathcal{M}_4$ (pink wedges in the inset). With abuse of notation, we denote by ℓ also the corresponding curves on $\mathcal{M}_2$ and $\mathcal{M}$, regarded as quotient spaces of $\mathcal{M}_4$. Two paths ℓ_1 and ℓ_2 are said to intersect *tangentially* if they intersect in $\mathcal{M}_2$; while they intersect *orthogonally* if they intersect in $\mathcal{M}$ but they do not intersect in $\mathcal{M}_2$.

The drift of a field-coherent path w.r.t. $\mathbf{X}$ comes from the angle between the direction field and the tangent of ℓ at each point along it. We adopt an anisotropic metric on $\mathcal{M}_4$ that increases the length of a path proportionally to its amount of drift:

$$\|w\|_{\mathbf{X}} = |w|(1 + \alpha \frac{\measuredangle(p_\theta, w)}{\pi/4}) \tag{3}$$

where w is a tangent vector at p, $|w|$ is its Euclidean norm, p_θ is the reference direction on $\mathcal{M}_4$, $\measuredangle$ measures the unsigned angle between a pair of vectors, and α is a parameter that tunes the amount of penalty for the drift. A field-coherent *geodesic* path between to point-arrows on $\mathcal{M}_4$ is a field-coherent path joining them that is shortest according to the above metric. We define a *field-coherent geodesic loop* to be a non-null field-coherent geodesic path that starts and ends at the same point.

A.2 Tracing loops

We trace field-coherent geodesic loops with the discrete graph-based approach of [Pietroni et al. 2016], which brings to the stratified structure $\mathcal{M}_4$ the technique of [Lanthier et al. 2001] to evaluate geodesic paths and distances. In short, in [Lanthier et al. 2001] shortest paths and distances are found by a Dijkstra search on an extended graph $\mathcal{G}$, which is built over $\mathcal{M}$ edges and vertices, plus Steiner points sampled on edges and arcs connecting vertices of $\mathcal{M}$ and Steiner points across each triangle. In our case, four point-arrows are generated per vertex and per Steiner point, which are properly arranged on $\mathcal{M}_4$, and just field-coherent arcs connecting them are considered. The graph $\mathcal{G}$ is built once, and used in all subsequent processing. One important advantage of this method is that crossings and overlaps of paths can be handled in a robust, combinatorial way that does not involve numerics: it is possible to precisely identify whether two paths intersect orthogonally or tangentially by simply comparing arcs that belong to the same triangle of $\mathcal{M}$ and checking their underlying direction fields.

Measuring distances. Given a set of loops $\mathcal{L}$, each represented with a polyline, and a loop ℓ not belonging to $\mathcal{L}$, we evaluate distance $d(\mathcal{L}, \ell)$, as defined in Section 4.2 as follows. We set a source for each node of graph $\mathcal{G}$ traversed by each loop $\ell_i \in \mathcal{L}$ and we run a Dijkstra propagation. Then distance $d(\mathcal{L}, \ell)$ is computed, after propagation, by collecting distances at all nodes traversed by ℓ. Note that a single Dijkstra propagation is sufficient to set distances at all nodes of $\mathcal{G}$, thus it is sufficient to evaluate distances from all loops in a pool of candidate loops.

Constrained propagation. Given a set of constraints $\overline{\mathcal{L}}$, as in Section 4.5, we prevent any new loops from tangentially intersecting loops in $\overline{\mathcal{L}}$ by blocking Dijkstra propagation along arcs in $\mathcal{G}$ that are orthogonal to those arcs belonging to paths already in $\overline{\mathcal{L}}$.

A.3 Extending features to loops

In order to force loops to run along line features, we modify the graph $\mathcal{G}$ as follows:

- Given a line feature f, we create two *corridors*, each made of a strip of triangles of $\mathcal{M}$ incident at f, one for each side of f (see Figure 25.a);
- For each face in a corridor, we consider all Steiner nodes of $\mathcal{G}$ that are coherent with the direction of f and that lie on edges crossing the corridor (see Figure 25.b);
- We reduce the weight of arcs connecting pairs of such nodes in the corridor (green arcs in Figure 25.c);
- We inhibit all arcs that connect such nodes with nodes at the boundary of the corridor (red arcs in Figure 25.d).

For each line feature f, we create one seed node per side of f and we trace a set of candidate loops from all such nodes. Note that, in the modified graph, each path that enters a corridor is forced to traverse it totally, and paths traversing several corridors (i.e., joining or bridging different line features) are favoured because of their reduced cost (see Figure 7.d). Note that each feature may be traversed by multiple loops in the set of candidates. In the process of generating the loops that extends line features we select loops in a greedy manner, preferring the ones that span the largest length of open features.