Robust Topological Construction of All-hexahedral Boundary Layer Meshes

MAXENCE REBEROL, KILIAN VERHETSEL, and FRANÇOIS HENROTTE,
Université catholique de Louvain, Belgique
DAVID BOMMES, University of Bern, Switzerland
JEAN-FRANÇOIS REMACLE, Université catholique de Louvain, Belgique

We present a robust technique to build a topologically optimal all-hexahedral layer on the boundary of a model with arbitrarily complex ridges and corners. The generated boundary layer mesh strictly respects the geometry of the input surface mesh, and it is optimal in the sense that the hexahedral valences of the boundary edges are as close as possible to their ideal values (local dihedral angle divided by 90°). Starting from a valid watertight surface mesh (all-quad in practice), we build a global optimization integer programming problem to minimize the mismatch between the hexahedral valences of the boundary edges and their ideal values. The formulation of the integer programming problem relies on the duality between boundary hexahedral configurations and triangulations of the disk, which we reframe in terms of integer constraints. The global problem is solved efficiently by performing combinatorial branch-and-bound searches on a series of sub-problems defined in the vicinity of complicated ridges/corners, where the local mesh topology is necessarily irregular because of the inherent constraints in hexahedral meshes. From the integer solution, we build the topology of the all-hexahedral layer, and the mesh geometry is computed by untangling/smoothing. Our approach is fully automated, topologically robust, and fast.

CCS Concepts: • **Mathematics of computing** → **Mesh generation**; • **Computing methodologies** → **Mesh geometry models**; **Volumetric models;**

Additional Key Words and Phrases: Hexahedral meshing, hex-dominant meshing, boundary layer, mesh generation

ACM Reference format:
Maxence Reberol, Kilian Verhetsel, François Henrotte, David Bommes, and Jean-François Remacle. 2023. Robust Topological Construction of All-hexahedral Boundary Layer Meshes. *ACM Trans. Math. Softw.* 49, 1, Article 2 (March 2023), 32 pages.
https://doi.org/10.1145/3577196

This research is supported by the European Research Council (project HEXTREME, ERC-2015-AdG-694020) and by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS). D. Bommes has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (AlgoHex, grant agreement No 853343).
Authors' addresses: M. Reberol, K. Verhetsel, F. Henrotte, and J.-F. Remacle, Université catholique de Louvain, Avenue Georges Lemaître 4-6, Louvain-la-Neuve, 1348, Belgique; emails: maxence.reberol@gmail.com, {kilian.verhetsel, francois.henrotte, jean-francois.remacle}@uclouvain.be; D. Bommes, University of Bern, Hochschulstrasse 6, Bern, 3012, Switzerland; email: david.bommes@unibe.ch.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
0098-3500/2023/03-ART2 $15.00
https://doi.org/10.1145/3577196

1 INTRODUCTION

Even though tetrahedral meshes are much easier to generate, hexahedral meshes are the preferred choice for advanced numerical simulations such as large deformation or **computational fluid dynamics (CFD)**.[1] For instance, hexahedral meshes surpass tetrahedral meshes by far for wall-bounded flows, because the hexahedral topology allows for orthogonal grid alignment in the wall-normal direction. Similar requirements also exist in other engineering disciplines, when interesting phenomena happen close to the model boundaries (e.g., skin effect).

Generating high quality conformal hexahedral meshes in arbitrary 3D domains is one of the most challenging open problems in mesh generation. This paper does not address the whole problem at once, but focuses on the boundary part. As mentioned before, many of the interesting physical phenomena to be captured by numerical simulations tend to happen near the object walls. Generating *boundary layer meshes* has thus always been an active research topic, although mostly for tetrahedral and polyhedral meshes. Different approaches have been proposed to address the large variety of ridge and corner topologies encountered in complex real-life geometrical models [9, 22, 24], and industrial solutions are available.[2,3]

Whereas it is rather easy to generate tetrahedral meshes for arbitrary ridge and corner topologies thanks to the flexibility of tetrahedra, the matter is somewhat harder with hexahedra due to the topological constraints in hexahedral meshes. The only straightforward way to build an all-hexahedral boundary layer mesh is to simply extrude a boundary quadrilateral mesh inwards. This approach mechanically generates a single hexahedron for each boundary quadrilateral face, and leads to low-quality meshes along feature curves. Some techniques have been developed to improve *a posteriori* the hexahedral boundary layer topology, but they are limited to hard-coded corner configurations and the general case remains unsolved.

This article essentially addresses the problem of building *good* hexahedra at ridges and corners. Starting from a watertight surface mesh (quad or quad-dominant in practice), we propose a novel robust method to build an all-hexahedral boundary layer, by first enumerating all possible configurations of hexahedra around boundary vertices, and then selecting the best combination, according to the model geometry and (optional) user-prescribed preferences. The advantage of our technique is its genericity and topological robustness. There are no manually-defined patterns, and it automatically finds valid (and optimal or close-to optimal) configurations at complicated feature curve junctions, where the optimal mesh is in general complex and non-intuitive (e.g., Figure 1). Our method generates a single layer of hexahedra, which can be subdivided to obtain a multi-layered all-hexahedral boundary layer. This opens the door to industrial-grade automatic hex-dominant meshing with high-quality all-hexahedral layers on the model boundary. On the theoretical side, this work allows to better understand the topological constraints in hexahedral meshing, and gives insights on how to satisfy them in a fully unstructured context.

Motivating example. Consider the volume below a pinched surface, as depicted in Figure 2. The dihedral angle varies along the central feature curve from 30° at the summit to 180° where the surface flattens off. The boundary layer obtained by simply extruding the boundary quads (which are all along regular) would yield pairs of hexahedra with very small angles (15°) near the summit, where it would be better to have a single row of hexahedra (with an edge of valence one on the feature curve). On the other hand, a single row of hexahedra with an edge of valence one all along the feature curve would not be appropriate either, as they would degenerate to flat hexahedra where

[1] Reduce CFD meshing time and improve accuracy with hexahedral boundary layer meshes, Cadence Pointwise.
[2] Meshing for CFD, Simmetrix.
[3] Simplify complex geometry CFD, Cadence Pointwise.

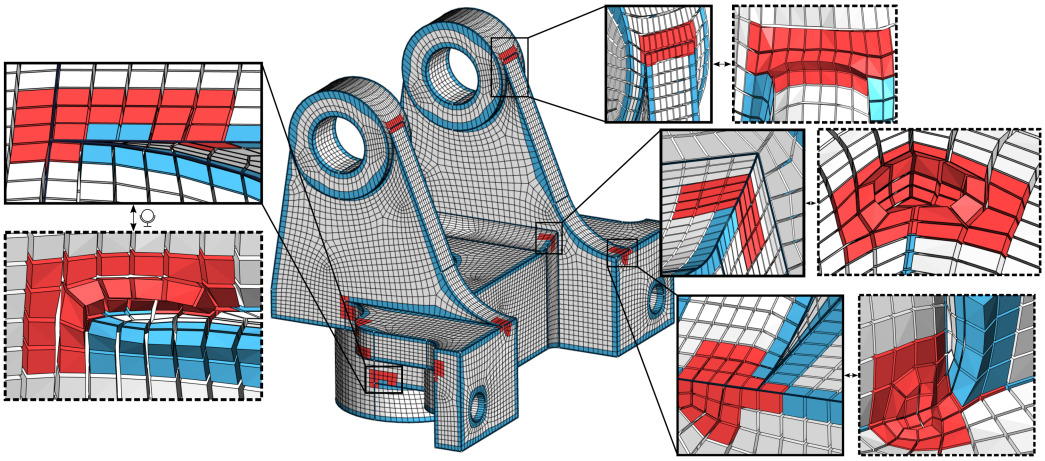


Fig. 1. All-hexahedral boundary layer on a model with complex hexahedral configurations required at non grid-like CAD corners. Hexahedron colors are: grey for extrusion of boundary quads, blue where the ideal hexahedral valence is respected on a feature curve, and red for non-intuitive configurations found by solving a combinatorial integer programming problem. Zooms in left and bottom-right contain *ski-jump* corners, top-right contains two pinched corners resolved together, and middle-right contains a dihedral angle transition from convex to flat along a feature curve. Boxes surrounded by dashed lines are showing the hexahedra viewed from the interior of the model.

the dihedral angle becomes 180°. Somehow, this feature curve calls for an appropriate transition between valence one and valence two, and it is the purpose of this paper to offer a robust and automatic procedure to do so. The optimal choice is to have valence one near the summit (elements in blue in Figure 2), and valence two when the surface is sufficiently leveled off. However, due to the inherent topology of hexahedral meshes, it is not possible to transition directly from valence one to valence two, and one has to use some transition configuration involving valence three edges (Figure 4). Our integer programming solver is able to automatically deal with these kinds of issues, and to find the optimal solution with a complex transition region, colored in red in Figure 2. The minimum element quality (SICN, explained in Section 5.1) is 0.4 with our approach, to be compared with 0.2 in case of direct extrusion. Our point is that finding a *good* all-hexahedral layer is not a simple task, even in situations, such as Figure 2 or the pyramid apex (Figure 10), where the boundary quad mesh is perfectly regular. In real-life CAD models (e.g., Figure 1), the problem is even more intricate due to the variability of the local geometrical features, the specificities of the prescribed boundary quad meshes, and the possible coupling with other nearby CAD features. Such complexity calls for an automatic and generic approach, which is the purpose of this paper.

Overview. Our boundary layer meshing process starts from a watertight surface mesh Q and produces an all-hexahedral layer H, which matches the midpoint subdivision of Q. The main steps are:

- Compute the ideal hexahedral valences at the boundary edges and at the boundary vertices of the input mesh Q, using the dihedral angles and solid angles (Section 3).
- Find the hexahedral valences at the boundary edges of Q by solving a global integer programming optimization problem (Section 4.2), which is the assembly of local boundary hexahedral configuration problems (Section 3.4) coupled by compatibility constraints (Section 4.1). The global problem is solved with a branch-and-bound search (Section 4.3) applied to subproblems built by domain decomposition (Section 4.4).

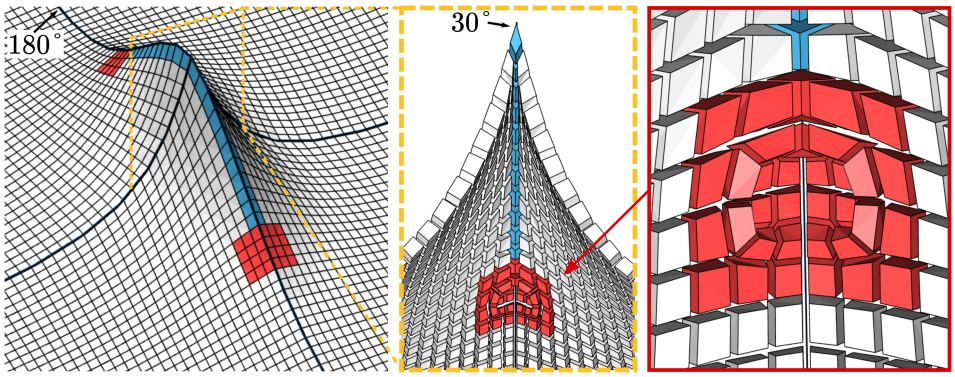


Fig. 2. (Left) All-hexahedral boundary layer meshing in case of a surface with a feature curve whose dihedral angle transitions from 30° (pinched region) to 180° (flat region). Our global integer solver automatically generates one row of hexahedra with valence one edges on the feature curve (blue color), and a transition region (red color) to connect with regular hexahedras with valence two edges on the feature curve. (Right) View from below, i.e., from inside the volume, of the generated all-hexahedral boundary layer.

- Build the boundary hexahedral configurations at the vertices of Q (Algorithm 1) and glue them (Section 4, Figure 7).
- Untangle and optimize the mesh geometry (Section 4.5), which is fixed on the boundary and free inside the volume.

The key idea behind the formulation of the global integer programming problem is that hexahedral configurations at boundary vertices are dual to the triangulations of the disk (Section 3), and we propose a novel integer formulation for the existence of disk triangulations constrained to boundary valences (Section 3.3).

Our approach scales well with model complexity and size, allowing us to deal robustly with real-life CAD models and their complex ridges/corners arrangements, which we believe is a significant achievement in the context of hexahedral meshing. Our technique possesses theoretical guarantees and has been applied successfully to the 114 CAD models of the MAMBO dataset (Section 5.1), with detailed figures and statistics available in the supplemental material. The implementation of the integer programming solver (based on the library *Gecode* [33]) is available in the supplementary material, and the complete implementation of our work is open-source and available[4] in the *Gmsh* [10] git repository.

2 RELATED WORK

Volume meshing approaches can be classified in a continuous spectrum that ranges from *surface-to-volume* (Section 2.1), where cells have faces that match elements on an input surface mesh, to *volume-to-surface* (Section 2.2), where the volume mesh is generated first and the surface geometry is recovered a posteriori by snapping, projection, and deformation.

2.1 Surface-to-volume Hex Meshing

Surface-to-volume meshing is the standard approach for industrial tetrahedral meshing (constrained Delaunay), especially for the generation of boundary layers. This approach is however less successful for hex meshing, because of the inherent and inflexible topological constraints in hexahedral topology.

[4]Branch *hexbl* in the *Gmsh* git repository https://gitlab.onelab.info/gmsh/gmsh/-/tree/hexbl.

Constrained hexahedral meshing consists in building a mesh matching a given quadrilateral boundary mesh. Theory shows that this is possible if the boundary has an even number of quads and no odd edge cycles [6]. Verhetsel et al. [38] developed a systematic approach to find *small* topological hexahedral meshes of the sphere, but is limited to quadrangulations with no more than 20 boundary quads. Furthermore, the smallest hexahedral meshes for simple boundaries may be surprisingly complex, e.g., one needs 36 irregularly-connected hexahedra to mesh the Schneiders' pyramid whose boundary is made of 16 quads.

For larger problems, attempts have been made to exploit the dual of the hexahedral mesh, which consist of surface sheets intersecting at chords. Given a hex mesh, the boundaries of the dual sheet are chords in the boundary quad mesh. The *Whisker Weaving* [18, 36] and the *dual cycle elimination* [15, 27] start from the chords of a boundary quad mesh and build internal sheets. *Whisker Weaving* is mainly topological and produces hexahedral meshes that are often tangled or of very low quality, and *dual cycle elimination* is limited to boundary quad meshes with no self-intersection loops, which is a serious limitation in practice. Despite sustained research efforts, the induced limitations are intractable, and constrained topological approaches have not led to *practical* all-hexahedral meshing, essentially due to the utter reliance on the suitability of the boundary quadrilateral mesh.

The all-hexahedral boundary layer method we propose falls in this category, but it supports *imperfect* quad meshes that may contain self-intersecting loops. The ability to work with a more general class of surface meshes comes from the fact that we are not trying to fill in the entire volume, but only to generate a single layer of elements.

The subdivision of each tetrahedron of a tetrahedral mesh into four hexahedra is another surface-to-volume method worth mentioning. It is as robust as tetrahedral meshing, and the model geometry is perfectly represented, but the resulting hexahedral mesh is very irregular and of rather poor quality for numerical simulations.

2.2 Volume-to-surface Hex Meshing

Another approach to hex meshing is to start by meshing the interior of the model with a grid [32] or an octree [7, 20, 25, 37] with size transition templates to ensure the internal mesh conformity. The difficulty is then to recover the geometry of the model boundary. Both References [25] and [7] use an externally extruded layer, and a boundary snapping and deformation procedure.

Octree-based hex meshing methods are the only robust fully automatic approaches which are able to deal with complicated CAD models (see the commercial mesher Hexotic[5]), but the resulting hexahedral meshes are far from ideal. In terms of topology, the meshes are irregular inside the volume (at octree transitions) and have a poor structure on the boundary, with many honeycomb-like patterns. Due to the padding layer, the hexahedral valence of the edges located on feature curves is always two, and thus not ideal at convex or concave ridges.

Accuracy on the model geometry is also limited. It is generally not possible to exactly match the model features because of the imposed structure coming from the interior hexahedral mesh. The boundary matching of the CAD features is only approximate, both in terms of geometry and of topology. For instance, octree-based methods change the apex of a pyramid (Section 5.6).

2.3 Simultaneous Surface and Volume Hex Meshing

Although not directly in relation with this work, it is worth mentioning that the recent trend in all-hexahedral meshing research is to determine first the *directions* of the hexahedra and then to generate the mesh.

[5]Hexotic: automated all-hex mesh generator based on the octree method.

With the PolyCube approach [21], there are three global directions (corresponding to axis x, y, z), and the mesh is obtained as the deformation of a grid, with no internal irregular edges. Because of the absence of irregular edges, PolyCube meshes may have difficulty to match CAD-like geometry, and selective padding [3] is generally required, with limitations similar to the ones of the grid/octree-based approach (inaccurate boundary recovery).

A more generic (but less robust) approach is to first build a frame-field [11], which allows the directions to permute around singularity lines, and then to build a hexahedral mesh matching the singular frame-field [19, 23, 28]. Frame-field based approaches are more flexible, but they rely on the correctness of the frame-field singularity graph, which is often lacking even for simple models [19, 30].

More fundamentally, both approaches use six orthogonal directions to represent hexahedra, and they are therefore unable to capture non grid-like corners such as the apex of a pyramid. They are promising lines of research with impressive results on smooth models, potentially with basic feature curves, but they are not mature enough to deal accurately with realistic CAD models.

2.4 A Posteriori Boundary Layer Insertion

Robust hexahedral meshing methods generally deliver meshes with low-quality hexahedra on the boundary, and techniques have been developed to specifically improve the boundary layer.

The straightforward way to add a boundary layer is to extrude all the vertices of the boundary [12], such that the newly inserted layer has boundary edges which are all adjacent to two cells. This may be better than the initial hex mesh, but it still contains low-quality hexahedra at convex and concave feature curves, and at corners. Reference [26] proposes a cross-imprinting method to build better elements at the intersection of two surfaces making a convex angle, but this is only applicable in a limited number of situations, with manual selection of the surfaces of interest.

In the more general case, it is not easy to recover good valences along feature curves because the geometry suggests choices that are incompatible with hex mesh topology. By looking at intersection of fundamental sheets, [14, 17] describe some configurations which are possible in hexahedral meshes and build a system to satisfy them with a greedy valence curve assignment. Reference [39] extends the approach by translating those configurations into linear constraints, so that the system can be solved by optimizing a linear integer programming problem. While there are similarities in spirit with our work, their approach is limited to 11 hard-coded corner configurations, and it is applied to already existing all-hexahedral meshes. Because a single valence is used per feature curve, their approach is global and the irregular ridge configurations are not resolved locally, but typically by falling back to quad extrusion on large portions of the boundary.

3 LOCAL BOUNDARY HEXAHEDRAL CONFIGURATION

A *hexahedral configuration* is the set of hexahedra connected to a vertex $v \in \Omega$. If the vertex v is on the boundary $\partial\Omega$ of the mesh, the set is called *boundary hexahedral configuration*, and is noted $\mathcal{H}_v$. Examples of boundary hexahedral configurations are presented in Figure 3. In this section, we define a local integer programming minimization problem (Equation (2)) to find the *optimal* boundary hexahedral configuration at a boundary vertex. However, the reader should be aware that this local problem is never solved alone in practice, but is meant to be assembled in the global problem (Equation (3)) introduced in the following section (Section 4).

Optimality of a boundary hexahedral configuration $\mathcal{H}_v$ is understood in the sense that the valences $(n_1, \ldots, n_m)$ of the m *boundary* edges connected to v, ordered consecutively around v, should be as close as possible to a set of prescribed *ideal* values $(x_1, \ldots, x_m)$. Only boundary edges are involved in this minimization process because they are the ones that determine the compatibility of neighboring boundary hexahedral configurations in the gluing process that occurs at a

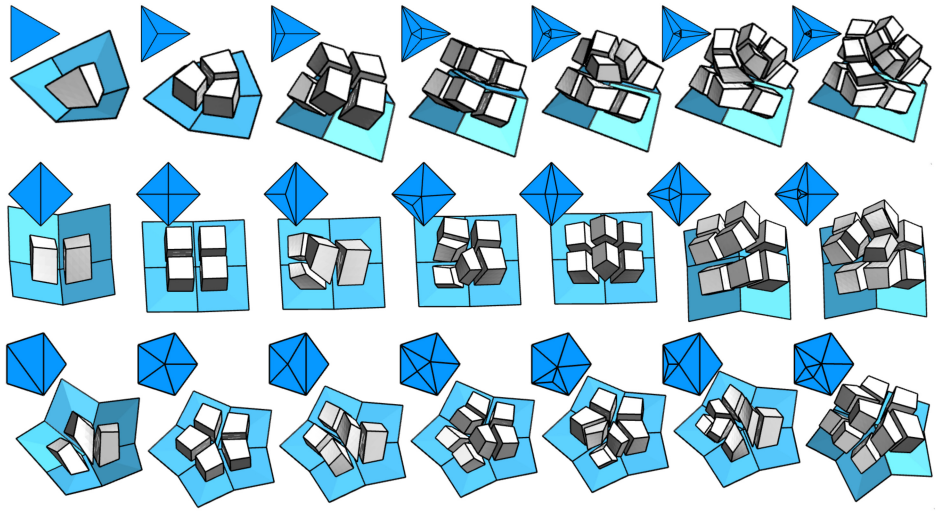


Fig. 3. Examples of boundary hexahedral configurations for boundary vertices of valence three (top row), four (middle row), and five (bottom row). For each configuration, the associated dual disk triangulation is shown in the upper left.

later stage (Section 4). To resolve the possible multiplicity of solutions with the same valences $(n_1, \ldots, n_m)$, an *ideal* number w_v of hexahedra adjacent to v is also prescribed.

The ideal valence real values $(x_1, \ldots, x_m)$ and w_v are determined on the basis of the dihedral angles α_i formed by $\partial\Omega$ at boundary edges, and the solid angle ω_v at the boundary vertex v. For right-angled hexahedra, the ideal valences are $x_i = \frac{\alpha_i}{\pi/2}$ at boundary edges, and $w_v = \frac{8\omega_v}{4\pi}$ at the vertex v.

The optimal local boundary hexahedral configuration is thus the solution of the minimization problem

$$\underset{\substack{\mathcal{H}_v;\\ 1 \le n_i \le 4}}{\operatorname{argmin}} \sum_{i=1..m} (n_i - x_i)^2 + \epsilon |\mathrm{val}(v) - w_v|, \tag{1}$$

with ϵ a coefficient small enough, e.g., $\epsilon = 10^{-3}$, so that the energy is always dominated by the first term. The second term is only used to discriminate configurations having identical boundary edge hexahedral valences $(n_1, \ldots, n_m)$, which only happens at concave corners, e.g., the top right configurations on Figure 3.

To solve the problem Equation (1), the technicality lies in the accurate characterization of the set of possible boundary hexahedral configurations $\mathcal{H}_v$ (with boundary edge valences n_i in the range $[1..4]$). Building on the duality between boundary hexahedral configurations and the triangulations of a disk (cf. next section), the keystone of our approach is to introduce an integer formulation (Proposition 1) to characterize the existence of disk triangulations matching prescribed valences at their boundary vertices, which are the duals of the boundary edges of $\mathcal{H}_v$. This approach allows us to translate boundary hexahedral configurations $\mathcal{H}_v$ into integer sets $\mathbf{n}$ which verify specific non-linear constraints (Proposition 1).

Note that the local optimal boundary hexahedral configuration is defined by a L^2-norm minimization problem (Equation (1)) as it is the straightforward choice, but any objective function can be used in our framework as long as it is bounded and computable from the local configuration, thanks to our branch and bound solver (Section 4.3).

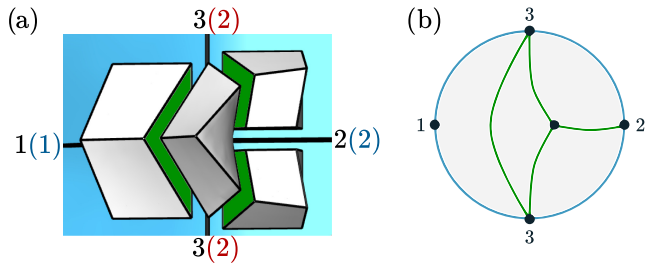


Fig. 4. Example of boundary hexahedral configuration for the transition from valence one on a convex feature curve (a, left) to valence two on a flat region (a, right). While the ideal valences are $\mathbf{n}^* = (1, 2, 2, 2)$, such configuration is not possible (no disk triangulation). The closest solution that allows the left-to-right $1 - 2$ transition is $\mathbf{n} = (1, 3, 2, 3)$. This boundary hexahedral configuration is shown on the left (a), with its associated dual disk triangulation on the right (b).

3.1 Duality between Hexahedral Configuration and Triangulation

Hexahedral configurations at interior vertices are dual to the triangulations of the sphere [28]. Similarly, boundary hexahedral configurations around boundary vertices are dual to the triangulations of the disk [19]. This means that in a boundary hexahedral configuration, there exists a one-to-one correspondence between the edges/faces/cells adjacent to the boundary vertex and the vertices/edges/faces of the dual triangulation, respectively. This duality has a simple geometric interpretation [28]: the dual triangulation is the intersection of the boundary hexahedral configuration $\mathcal{H}_v$ with an infinitesimal sphere centered at v. Besides its theoretical value, this duality is also very useful in practice because it is much easier and intuitive to think in terms of disk/sphere triangulations than in terms of topological constraints in hexahedral meshes. The duality is illustrated in Figure 3, where we show multiple examples of boundary hexahedral configurations and their associated disk triangulations.

As mentioned above, we are interested in finding boundary hexahedral configurations verifying boundary valences desiderata which come from the model geometry. Although there is in general an infinite number of possible configurations around a vertex, not so many of them subsist when studying problems of smaller size with explicit boundary valence constraints.

For instance, Figure 4 depicts the transition from a convex to a flat region, i.e., from a dihedral angle of 90° to a dihedral angle of 180°. In a boundary hexahedral configuration, it is not possible to transition directly from valence one to valence two because there is no disk triangulation for $\mathbf{n} = (1, 2, 2, 2)$. The local solution is to use valence three on the sides ($\mathbf{n} = (1, 3, 2, 3)$), but this means that we are no longer respecting the ideal valences. To better understand how it is possible to arrange hexahedra around boundary vertices, we have to study the problem of existence of boundary hexahedral configurations in a more systematic way.

3.2 Existence of a Boundary Hexahedral Configuration

The existence problem can be formulated as follows:

Given an ordered set of m strictly positive integer numbers $\mathbf{n} = (n_1, \ldots, n_m)$, does a boundary hexahedral configuration exist with m boundary edges, the valences of which are $\mathbf{n}$?

By exploiting the previous duality (Section 3.1), we can reformulate the problem in terms of disk triangulations:

Problem 1. Constrained disk triangulation problem.

Given a set of integer valences $\mathbf{n} = (n_1, \ldots, n_m)$, does a triangulation of the disk exist that has m boundary vertices, the valences of which are $\mathbf{n}$?

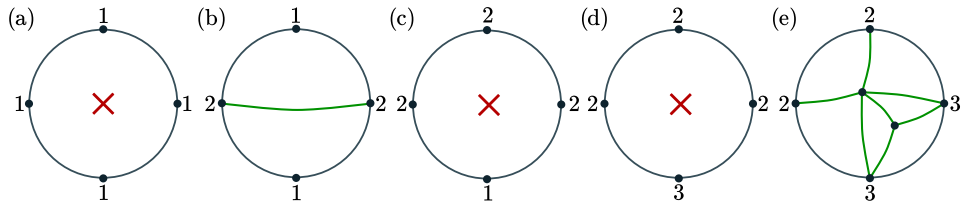


Fig. 5. Existence of triangulations of the disk for different prescribed sets of boundary vertex valences. In some simple situations (a,c,d), there is no matching triangulation. Absence of disk triangulation for (a) means that there is no ideal hexahedral configuration for the pyramid apex (junction of four convex ridges).

Examples of Problem 1 are shown in Figure 5. It is interesting to note that apparently simple configurations may have no solution. The first example (Figure 5(a)) is the pyramid apex, with four edges of ideal valence one meeting at a corner. No hexahedral configuration can thus match these ideal valences, and other valences must be chosen, e.g., $\mathbf{n} = (1, 2, 1, 2)$ as illustrated on Figure 5(b). Such *non grid-like* corners, where ideal valences cannot be matched with hexahedral elements, appear frequently in CAD models. They can however be identified theoretically by checking whether or not the existence problem (Problem 1) has a solution. The motivation of our work is to resolve such situations, and to determine which boundary hexahedral configurations should be used whenever the *ideal* ones are not possible.

To answer Problem 1, one can generate a tabulation of disk triangulations, by using the algorithm in Appendix A.2 of [19] or the planar graph library *plantri* [2]. As the list of disk triangulations is infinite, we restrict ourselves to useful ones, i.e., the ones with a limited number of boundary vertices (e.g., eight), and with limited vertex valences (e.g., six). Once the exhaustive list has been generated (available in the supplemental material), answering Problem 1 reduces to a simple query in a database. But this existence problem assumes the valences $\mathbf{n}$ are known in the first place. Their determination implies ensuring compatibility of local problems at all vertices of $\partial\Omega$, as discussed later in Section 4.

3.3 Integer Formulation of the Constrained Disk Triangulation Problem

Before turning to the global compatibility problem presented in Section 4, the constrained disk triangulation problem (Problem 1) must be formulated into integer constraints suitable for a global combinatorial optimization problem. Our reformulation is a combination of necessary theoretical conditions, and a recursive reduction to the simplest possible triangulation of the disk, the triangle. For the sake of conciseness, the notation $\exists\mathcal{DT}_m^a(\mathbf{n})$ is introduced to indicate that a triangulation of the disk with m boundary vertices of valence $\mathbf{n} = (n_1, \ldots, n_m)$ exists and is solution of Problem 1. The subscript m is the number of disk vertices, and the superscript a is the primal vertex in the input surface mesh to which the disk triangulation is dual. Both can be omitted when there is no ambiguity. We will also use $\#(n_i = k)$ as the number of boundary vertex valences that are equal to k, i.e., $\#(n_i = k) := |\{n_i,\ i \in [1..m] \mid n_i = k\}|$. We also recall that the term *valence* is used to specify to the number of triangles incident to a vertex (and not the number of incident edges), because it is the same number as the boundary edge hexahedral valence in the primal.

The first condition concerns boundary vertices of valence one. The only possible triangulation of the disk with two successive boundary vertices of valence one is the triangle (i.e., $\exists\mathcal{DT}_3((1, 1, 1))$). Except in that very case, two consecutive valences cannot be equal to one in any triangulation of the disk. Moreover, as a boundary vertex of valence one necessarily forms a triangle with its two neighbors, a reduced triangulation problem of size $m-1$ can be obtained by removing this triangle and decreasing by one unity the valence of the neighbor vertices, as depicted in Figure 6(a).

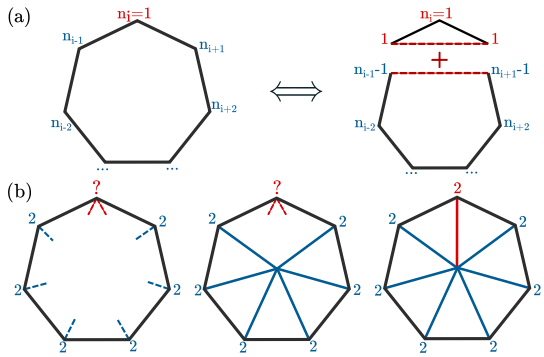


Fig. 6. (a) A valence one boundary vertex necessarily forms a triangle with its neighbors. By removing this triangle, a triangulation problem of size $m-1$ is obtained. (b) A sequence of valence two boundary vertices must connect to the same interior vertex. If all but one boundary vertices are of valence two, then the last vertex must also be of valence two.

Condition 1. *In the triangulation of a disk with $m > 3$ boundary vertices, two consecutive boundary vertex valences cannot be equal to one. If $m = 3$, either all or none of the boundary vertices are of valence one.*

$$\exists \mathcal{DT}_m(\mathbf{n}), m > 3, n_i = 1 \implies n_{i-1} \neq 1 \text{ and } n_{i+1} \neq 1$$
$$\exists \mathcal{DT}_3(\mathbf{n}) \implies \#(n_i = 1) = 0 \text{ or } 3$$

The second condition, illustrated in Figure 6(b), states that there is never $m-1$ boundary vertices of valence two in a triangulation of the disk with m boundary vertices. The reason is that a sequence of valence two boundary vertices necessarily connect all to the same interior vertex. If now the sequence of valence two vertices is of size $m-1$, then the polygon that remains when all triangles have been removed is a quadrangle whose opposite boundary vertices (the ends of the sequence) cannot accept additional incident edges, otherwise their valence would be three, instead of two. The only solution is thus that the last boundary vertex is also of valence two, which contradicts the initial assumption that the sequence was of size $m-1$. The cases (c) and (d) in Figure 5 are examples where Condition 2 is not verified.

Condition 2. *A necessary condition for the existence of a triangulation of the disk with m boundary vertices is that the number of boundary vertices with valence two is different from $m-1$:*

$$\exists \mathcal{DT}_m(\mathbf{n}) \implies \#(n_i = 2) \neq m-1$$

The two above conditions are necessary. They are, however, insufficient to resolve all cases. There exist sets of valences $\mathbf{n} = (n_1, \ldots, n_m)$ that fulfill both conditions, but for which no triangulation exists. Still, we conjecture that these conditions are sufficient if they are applied recursively to all reduced problems (of size $k \leq m$) obtained by applying the ablation of valence one vertices, as shown on Figure 6(a). The complete reduction algorithm is given in Appendix A.

Proposition 1. *A triangulation of the disk $\mathcal{DT}_m(\mathbf{n})$ whose boundary vertex valences match the set $\mathbf{n} = (n_1, \ldots, n_m)$ exists if and only if one has, for all reduced problem $\mathbf{n}^s = (n_1^s, \ldots, n_k^s)$ recursively obtained by ablation of valence one vertices*

$$\#(n_i^s = 2) \neq k-1$$
$$k > 3, n_i^s = 1 \implies n_{i-1}^s \neq 1 \text{ and } n_{i+1}^s \neq 1$$
$$k = 3 \implies \#(n_i^s = 1) = 0 \text{ or } 3$$

This proposition is a powerful reformulation of the constrained triangulation existence problem (Problem 1). As the conditions are a list of integer constraints, they can be integrated and propagated in constraint-based optimization techniques, such as the branch and bound search introduced later (Section 4.3). In practice, Proposition 1 contains non-linear *counting* constraints and conditional ones, so one must use a sufficiently flexible integer solver, such as *Gecode* [33]. For a concrete example, we provide the explicit list constraints for a regular boundary vertex (disk of size 4) in Appendix A.

It is important to note that $\forall i \in [1..m],\ n_i = 2$ is a trivial solution that always satisfies the constraints. The corresponding disk triangulation has a central interior vertex connected to all boundary vertices (e.g., right of Figure 6(b)). In the primal, this corresponds to the extrusion of a boundary quadrilateral mesh, where each boundary vertex is duplicated inside the volume.

Concerning Proposition 1, it is clear that the existence conditions are necessary when applied recursively to the reduced triangulations, but we have no formal theoretical proof that they are also sufficient in general. We have, however, an experimental proof limited to the values that are used in practice. It can thus be claimed that

Proposition 2. *Proposition 1 is true for $m \leq 6$ and $n_j \leq 4$.*

To verify Proposition 2 experimentally, boundary valences sets $\mathbf{n}$ have been extracted from all disk triangulations generated by the planar graph software *plantri* [2], and we have generated in parallel the valence sets $\mathbf{n}$ verifying the conditions in Proposition 1 with our integer solver. Both lists match exactly, and our C++ implementation is available in the supplemental material. Note that we have not pushed the experiment further to larger ranges, because higher valences (e.g., $m \geq 6, \forall j, n_j \geq 4$) yield millions of unique disk triangulations, and it becomes computationally expensive. We are nevertheless confident that Proposition 1 should be true in general.

3.4 Optimal Boundary Hexahedral Configuration

With the new integer formulation (Proposition 1), we can rewrite the boundary-part of the optimal boundary hexahedral configuration problem (Equation (1)) as an integer programming problem whose unknowns are the boundary edge hexahedral valences $\mathbf{n} = (n_1, \ldots, n_m)$:

$$\begin{aligned} \underset{\mathbf{n},\ 1 \leq n_i \leq 4}{\operatorname{argmin}} \quad & \sum_{i \in [1..m]} (n_i - x_i)^2 \\ \text{subject to} \quad & \exists \mathcal{DT}(\mathbf{n}) \quad \text{(see Proposition 1)} \end{aligned} \tag{2}$$

This optimization problem considers all possible boundary hexahedral configurations because there is a one-to-one correspondence with the disk triangulations (Section 3.1) and the integer constraints (Proposition 1) characterize all possible disk triangulations.

The interior-part of (Equation (1)), $\epsilon|\mathrm{val}(v) - w_v|$, allows discriminating triangulations of the disk with identical boundary valences. In practice, such triangulations occur by subdivision of internal triangles having no vertex on the boundary. This issue can be resolved independently, as it is not useful to integrate these internal degrees of freedom in (Equation (2)).

As the set $(\forall i, n_i = 2)$ is always a solution, the minimization problem (Equation (2)) is guaranteed to have a solution. Once the optimal set $\mathbf{n}$ is known, the corresponding triangulation of the disk $\mathcal{DT}(\mathbf{n})$ is obtained by searching a tabulated list containing all the disk triangulations up to a maximum valence. Given the triangulation $\mathcal{DT}(\mathbf{n})$, the associated boundary hexahedral configuration is then built with Algorithm 1. Finally, the geometry is determined by untangling and geometry optimization (Section 4.5). A few examples of boundary hexahedral configurations built this way are depicted in Figure 3. Note that in practice there is no point in solving a single instance

ALGORITHM 1: Build a boundary hexahedral configuration from a disk triangulation

Input: Boundary vertex v and disk triangulation $\mathcal{DT}$ made of N_{dv} vertices, N_{de} edges and N_{dt} triangles
Output: Hexahedra $\{H_{dt},\ dt \in [1, N_{dt}]\}$
for *each dual vertex* $dv \in [1, N_{dv}]$ **do**
  create a new vertex v_{dv};
  create a new edge $e_{dv} = (v, v_{dv})$;
end
for *each dual edge* $de = (dv_1, dv_2) \in [1, N_{de}]$ **do**
  create a new vertex v_{de};
  create a new edge $e_{de1} = (v_{dv1}, v_{de})$;
  create a new edge $e_{de2} = (v_{dv2}, v_{de})$;
  create a new quad face $f_{de} = (e_{dv1}, e_{dv2}, e_{de1}, e_{de2})$;
end
for *each dual triangle* $dt = (de_1, de_2, de_3) \in [1, N_{dt}]$ **do**
  collect the three quad faces $f_{de1}, f_{de2}, f_{de3}$ (contain 9 unique edges and 7 unique vertices);
  create the opposite vertex v_{dt};
  create the three missing edges (adjacent to v_{dt});
  create the three missing faces (adjacent to v_{dt});
  create the hexahedron H_{dt} from the six faces;
end

of (Equation (2)), but it becomes interesting when the problems at all the boundary vertices are considered together (Section 4).

The input from the primal mesh, required to setup (Equation (2)), consists of boundary vertices and their adjacent edges. Other nodes and edges needed to form the hexahedra are generated by Algorithm 1. Our method can therefore be applied to any vertex of a surface polygonal mesh (e.g., Figure 4(a)). In most cases, we have applied it to quadrilateral surface meshes, but the method would equally work with more general surface meshes (e.g., triangular mesh in Figure 12).

4 ALL-HEXAHEDRAL BOUNDARY LAYER PROBLEM

Being now able to build an optimal boundary hexahedral configuration at each vertex of the boundary mesh, we turn to the problem of making individual boundary hexahedral configurations compatible with each other to form an all-hexahedral boundary layer. There are two ways to *connect* configurations: either by gluing adjacent configurations (Figure 7), or by merging them with overlaps.

The *gluing* approach is light and easy to manage, and also works with any polygonal surface meshes (e.g., quad-dominant). It is the one followed in this paper. The main drawback is that the generated all-hexahedral mesh is not conformal with an initial quadrilateral surface mesh, but with its midpoint subdivision. So the edge lengths will be two times smaller in the hexahedral layer than in the surface mesh, but it can be somehow mitigated by building a coarser surface mesh in the first place.

Remark 1. The merging approach, on the other hand, could be directly conformal with a boundary quadrilateral mesh, which makes it appealing, but it is also much more complex, and hardly practical. The coupling between the adjacent configurations is indeed more intricate and the topological interdependencies may even become global, with a very strong sensitivity to the quad mesh topology and geometry. In convex regions, the mesh may fully connect on opposite sides of the

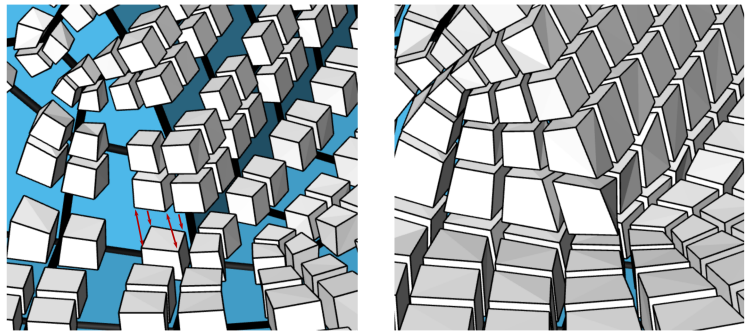


Fig. 7. The boundary hexahedral configurations defined at each vertex of the input boundary mesh (left) are glued together (red arrows) to form the all-hexhahedral boundary layer (right).

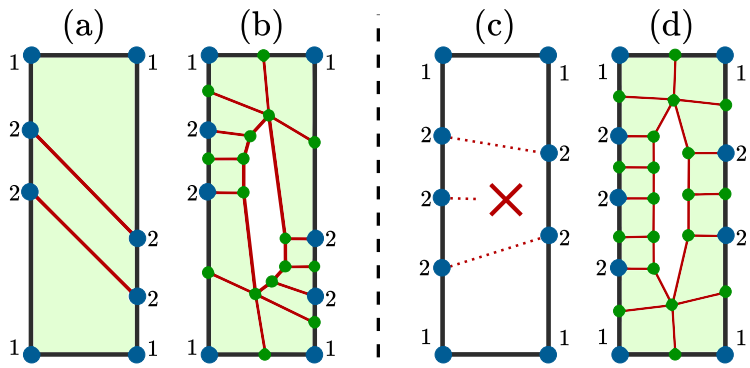


Fig. 8. 2D analog for the creation of boundary layers matching ideal valences. Blue dots represent boundary nodes with their ideal valences, and green dots in (b) and (d) represent the added mid-point subdivision dots. Ideal valences can match a boundary layer in case (a), but not in case (c). (a) is however sensitive to a global coupling between vertices on opposite sides of the domain, which results in a distorted quadrangle. In the cases (b) and (d), the boundary layer can be successfully generated, because the global coupling is released by the mid-point subdivision.

model and the problem tends to be more alike unstructured all-hexahedral meshing than the mere construction of a boundary layer. A 2D analog (much simpler than a real 3D case) is illustrated on Figure 8, where we can see that mesh (a) is global (and not even a boundary layer) and much more sensitive to the boundary than mesh (b) where the boundary is subdivided. The 3D overlap-merge option also requires advanced compatibility constraints, which we have not derived.

4.1 Compatibility Constraints

The main advantage of the *gluing approach* is that the compatibility constraints between adjacent configurations are rather simple. The two ends of an edge e_i of the initial boundary mesh belong to a pair of adjacent boundary hexahedral configurations to be glued together. As the hexahedral valence of an edge is unique, compatibility is automatically enforced by assigning a unique integer unknown n_i to each edge e_i of the input boundary mesh, which is quite natural. In terms of triangulation constraints, the valence of the boundary vertex associated to the edge e_i in both triangulations of the disk must be identical.

Condition 3. *If $e_i = (a, b)$ is an edge of the polygonal boundary mesh Q, with dv_j^a and dv_k^b its dual vertices in the disk triangulations $\mathcal{DT}^a(\mathbf{n}^a)$ and $\mathcal{DT}^b(\mathbf{n}^b)$, respectively, then the corresponding boundary vertex valences are identical:*

$$n_j^a = n_k^b$$

If Condition 3 is verified, then the building of the all-hexahedral boundary layer amounts to matching/gluing sets of quadrilateral faces adjacent to the edges on the boundary mesh, as illustrated with red arrows in Figure 7. In practice, the vertex identifications resulting from the gluing are enumerated, and the identified vertices are then merged in the mesh datastructure. This process is straightforward and unambiguous, because around each boundary edge, the two glued hexahedral configurations necessarily present a topologically matching set of quadrilateral faces, as a consequence of the fact that the common edge has a unique hexahedral valence.

Remark 2. Condition 3 is sufficient to successfully proceed with the *gluing* approach, but it falls short for the *overlap-merge* option (Remark 1). Although most of the local merges will be successful, complicated situations (combination of many irregular valences) will occur where the condition is verified but the merging would nevertheless break the hexahedral topology. More sophisticated (non-local) conditions are required, but we were not able to formalize them, despite significant effort. For instance, in the 2D analogy presented in Figure 8, no valid boundary layer can match the ideal valences in case (c) because of the well-known topological condition that a quadrilateral mesh requires an even number of edges on its boundary.

4.2 All-hexahedral Boundary Layer Integer Programming Problem

We build a global integer programming problem by combining the local problems (Section 3.4) and ensuring compatibility constraints (Condition 3). Consider a polygonal boundary mesh Q with N_v vertices and N_e edges. At each edge $e_i, i \in [1..N_e]$, an ideal hexahedral valence $x_i \in \mathbb{R}^+$ is determined on a geometrical basis, and we note $n_i \in [1..4]$ the unknown integer hexahedral valence. The goal of the global problem is to find a triangulation of the disk at each boundary vertex $v_j, j \in [1..N_v]$, fulfilling all compatibility constraints (Condition 3), and such that the valences $\mathbf{n} = (n_1, \ldots, n_{N_e})$ minimize the distance with the ideal ones $\mathbf{x} = (x_1, \ldots, x_{N_e})$, i.e.,

$$\begin{aligned} \underset{\mathbf{n},\ 1 \le n_i \le 4}{\text{minimize}} \quad & \sum_{i=1..N_e} (n_i - x_i)^2 \\ \text{subject to} \quad & \forall j \in [1..N_v],\ \exists \mathcal{DT}^j(\mathbf{n}^j) \quad \text{(Proposition 1)} \end{aligned} \tag{3}$$

The notation $\mathbf{n}^j$ denotes the sub-array of unknowns corresponding to the local edges incident to the j-th boundary vertex. The unknowns $\mathbf{n}$ are involved in the coupling of the boundary hexahedral configurations. There are also some internal degrees of freedom whenever multiple solutions match a given set of boundary vertex valences $\mathbf{n}^j$. To resolve those internal unknowns, the ideal vertex valence based on the solid angle at the vertex (w_v in Equation (1)) is used.

As for the local problem, the global one (Equation (3)) is guaranteed to have a solution, due to the solution $\forall i \in [1..N_e],\ n_i = 2$ that always satisfies all constraints, and which corresponds to the extrusion inwards of the midpoint-subdivided boundary. This simple extrusion is optimal if the model boundary is smooth enough ($\forall i,\ \alpha_i \approx 180°$).

Considering solutions of (Equation (3)), it is interesting to observe that non grid-like situations (e.g., apex of pyramid) cannot be solved by the local selection of a special boundary hexahedral configuration, but they are resolved by selecting appropriate configurations in a larger neighborhood, which is definitively a non-trivial and non-intuitive task. We develop this aspect in Section 5.2 with specific examples.

For a finer control on the generated boundary layer, further constraints can be added to the problem. The user may for instance prefer a single hexahedral valence along a CAD feature curve $C, e_1, e_2 \in C \implies n_1 = n_2$, or even to prescribe the value of the hexahedral valence along a feature curve C, e.g., $\forall e_i \in C, n_i = 1$. One must however be careful with such additional hard constraints. Would they be in conflict with each other, the global problem (Equation (3)) may no longer have a solution, e.g., Figure 5(a). Our theoretical guarantee of the existence of a solution relies on the existence of the solution ($n_i = 2$ everywhere), which may be overruled by hard constraints. Another way to increase user control on the solution is to fine-tune the objective function of the minimization problem. As we are using a branch and bound search, the only condition on the objective function is boundedness, which allows large freedom on the definition of the objective function.

4.3 Branch and Bound Solver with Constraint Propagation

The highly non-linear *counting constraints* associated with the existence of disk triangulations (Proposition 1) make the global integer programming problem (Equation (3)) hard to solve. The simple brute-force approach that consists in exploring all solutions $\mathbf{n}$ respecting the constraints, and picking the one that minimizes the objective function, is impractical. Branch and bound search, strengthened with constraint propagation, offers a smarter approach to solve such combinatorial optimization problem.

The search space is organized as a tree with the initial full set of candidates at the root, and the branch and bound algorithm explores branches that correspond to subsets of that full set. To avoid enumerating all candidate solutions in a given branch, which would amount to an exhaustive search among all possible solutions, the algorithm keeps track of some bounds on the solution and prune any branch that will provably not give a better solution than the best one known so far. Additionally, the search algorithm can exploit a *constraint propagation* mechanism to narrow down the possible range of each unknown at each branching, and progressively reduces the search space. The above search algorithm is readily available in the constraint programming library *Gecode* [33], which we use to solve (Equation (3)).

Given enough resources (computation time and memory space), the branch and bound search will always find the optimal solution. But in practice, to solve large problems in a reasonable time, it is critical to limit the size of the state space, and to use branching functions adapted to the problem. To set up the *Gecode* solver, we provide it with an integer range $[l_i..u_i]$ for each variable n_i, the complete list of integer constraints built with Proposition 1, an objective function (Equation (3)), and two branching functions, one to select the next variable, and one to select the next value for a given variable. Note that the full set of constraints can be specified *a priori* since *Gecode* supports conditional constraints.

Limit the size of the state space. The integer variables $n_i, i \in [1..N_e]$ can be basically restricted *a priori* to the range $[1..4]$, and this initial range can be further reduced on basis of the local value of the dihedral angle α_i as follows: $\{1, 2\}$ for convex angles (i.e., $\alpha_i < 135°$), $\{2, 3\}$ for flat angles (i.e., $135° < \alpha_i < 225°$) and $\{2, 3, 4\}$ for concave angles (i.e., $225° < \alpha_i$). Excluding valence one for flat and concave angles is a reasonable simplification, as not doing so would lead to hexahedra of very low or even negative quality. It also significantly reduces the number of integer constraints (Proposition 1), as this restriction leaves fewer possibilities for valence one boundary vertices in the disk triangulations.

Branching functions adapted to the problem. Each node in the branch and bound tree is associated with a set of possible candidate solutions for the problem, more precisely in the current case, with a list of possible valences for each boundary edge. The root-node of the tree corresponds to the initial ranges, as described in the previous paragraph. Branching, now, is associated with the

definition of a subset of candidates by setting a particular value to a particular valence. The strategy for searching efficiently the optimum solution in the tree is thus controlled by two branching functions: the variable selection strategy (choose which variable to modify) and the value selection strategy (choose which value to set).

Auxiliary to our selection strategy is the computation, for each boundary edge $e_i \in Q$, of a discrete distance (computed with Dijkstra's algorithm) to the nearest boundary vertex where (rounded) ideal valences are not solutions of the local disk triangulation. During the branch and bound search, the strategy will then be to select variables with the larger distance in priority (farthest edge first), and to set their valence to the value in the current list of possible valences that is the closest to the ideal valence. The current list is the initial list (see the variable range above) from which have been withdrawn the values excluded by the propagation of the constraints associated to the choices made at the other variables in the branch of the tree. This selection strategy intends to consider first solutions that have a larger number of ideal valences. So the first solutions eventually found are likely to be closer to the optimal (geometrically motivated) solution,than to the always existing (topologically trivial) solution. Convergence might however takes some time if many complicated configurations are clustered in the boundary mesh.

4.4 Scalable Solver with Problem Decomposition

A global branch and bound search may be too expensive in practice, especially in terms of memory. In case of large models with numerous non-trivial CAD features, we therefore propose a decomposition of the problem into a set of independent sub-problems of smaller size.

We start with a global initial guess $\mathbf{n}^*$ by rounding off the ideal valences (which are real valued), i.e., $n_i^* = \text{round}(x_i)$. For each boundary vertex $v_j \in Q$, we then check whether or not a triangulation of the disk $\mathcal{DT}^j(\mathbf{n}^{*j})$ exists that fulfills Proposition 1. The vertices with no matching triangulation are flagged, and their incident edges are taken as seeds in the Dijkstra distance computation, which gives for each edge e_i the integer distance d_i to the seeds (using edge-to-edge connectivity for propagation). In order to define sub-domains that could be solved independently, a threshold d_t on the distance is chosen, typically $d_t = 2$. We then identify a number of independent subsets of connected edges whose distance to the seeds is lower than the threshold, i.e., $d_i \in [0, d_t]$. These subsets are sub-domains of smaller size where to apply the above described branch and bound search (Section 4.2). If two sub-domains are connected by a vertex, they are merged. All edges outside the sub-domains have their valence fixed to n_i^*, and act as boundary conditions for the sub-problems.

If the threshold d_t is chosen too low, there might be no solution to the sub-problem. Whenever that happens, the procedure is repeated with an incremented threshold ($d_t \leftarrow d_t + 1$). From a theoretical point of view, with a large enough threshold, the whole model is covered with one domain, and one thus falls back to the initial global problem Section 4.2, for which the existence of a solution is guaranteed. In practice, $d_t = 2$ is almost always sufficient, except for pathologically coarse boundary meshes with multiple non grid-like corners.

It should be noted that, even if this decomposition into sub-problems is guaranteed to work, it may affect the final solution. The solution after decomposition into sub-problems has indeed some valences (on the edges outside the sub-domains) that have been fixed to their ideal value, which may be different from their value if the problem had been globally optimized. Such sub-optimal solutions are however not a problem in practice, and may even be preferable, because the deviation from ideal valences is then really localized around complicated CAD features.

The decomposition into sub-problems is very effective as it allows splitting large problems, with potentially hundreds of thousands of integer unknowns, into a series of smaller problems with only

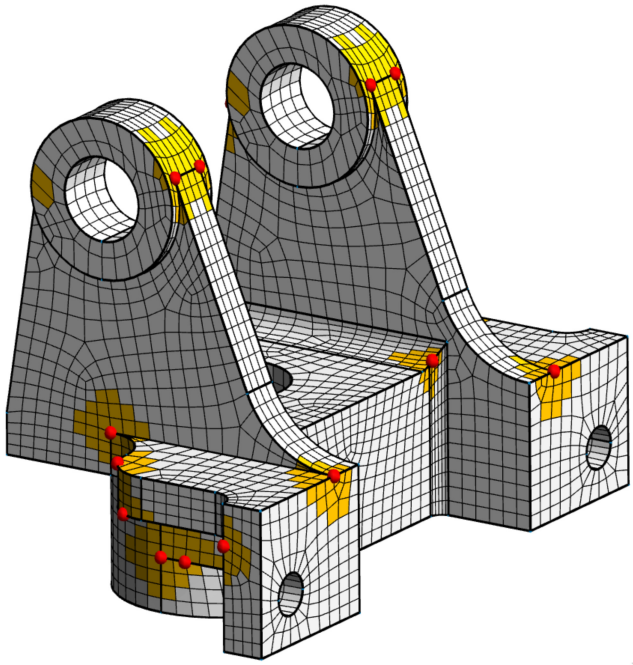


Fig. 9. The global problem for the M3 model (Figure 1) is reduced to 18 sub-problems (regions in yellow), with 30 seeds (vertices indicated in red) where no hexahedral configuration matches the ideal boundary hexahedral valences.

dozens to hundreds of unknowns. An example is shown on Figure 9, where the global problem (10614 integer variables and 5299 triangulations of the disk) is decomposed into 18 sub-problems, with each a number of integer unknowns between 35 and 105. Sub-problems with less than one hundred unknowns are typically solved within a few milliseconds (see Section 5.5).

4.5 Geometry of the Hexahedral Mesh

The solution of the problem Equation (3) (Section 4.2) determines the topology of the hexahedral mesh. Each boundary hexahedral configuration is built with Algorithm 1, and the configurations are glued together by merging the vertices at the interfaces between adjacent configurations (Section 4.1). One has now to determine the geometry of the mesh. Untangling and optimizing an unstructured hexahedral mesh is however a rather hard and computationally expensive task. As this falls outside the scope of this paper, we will not enter into too many details, but simply indicate which techniques work best according to our experience.

Boundary vertices are fixed, and simple heuristics are used to guess initial positions for the interior vertices. Vertices that are the extrusion of boundary vertices in smooth regions are simply placed according to the boundary normals. Along feature curves where the hexahedral boundary layer follows the pattern $\mathbf{n} = (2, c, 2, c), c = 1, 3$ or 4, the vertices are placed at regular angles on planes orthogonal to the feature curve. For the remaining vertices, which are interior vertices associated to irregular configurations, a simple Laplacian smoothing is used, with the previously placed vertices fixed. Except for block-like models, this initial geometry has in general many tangled hexahedra (of negative quality), especially at irregular vertices.

Efficient untangling approaches rely on non-linear optimization techniques, e.g., target-matrix optimization with log-barrier as implemented in Mesquite [1]. In our implementation, we have

finally settled on a more recent technique based on the minimization of the Winslow functional with a special regularization for inverted elements [8]. Results are similar to Mesquite, but faster by at least one order of magnitude. Another advantage of [8] is that untangling and shape optimization/smoothing are performed simultaneously, whereas we had to use another smoother after untangling with Mesquite. To improve performances and help the untangler, small cavities are iteratively built around low quality elements in order to avoid solving a global optimization problem, which would be too expensive for large meshes.

By default, the boundary layer depth is the average of the local surface mesh size. This means that the target shape is the isotropic cube in our mesh geometry optimization objective function [8]. With this approach, the boundary layer may self-intersect in thin regions of the model. Geometric intersections are computed in a post-processing stage, and the boundary layer thickness is locally decreased until there are no intersections, see Section 5.4.

5 DISCUSSIONS AND RESULTS

A strength of the approach proposed in this paper is that the topological part is robust and never fails (Section 5.1). Thanks to the decomposition into sub-problems, solving the integer programming problem is sufficiently fast (Section 5.5) to deal with real-life CAD models. Typically, building the boundary layer (topology plus geometry) is faster than generating the input quadrilateral mesh in the first place. Another valuable benefit of our approach is that the input geometry, including all the feature curves and corners, is strictly respected (Section 5.2). This is essential to numerical simulation, and a clear advantage over other robust hexahedral meshing methods (Section 5.6). On the other hand, the surface-to-volume approach of our technique goes along with a sensitivity to the quality of the boundary mesh, both topological (Section 5.3) and geometrical (Section 5.4).

5.1 Numerical Experiments

Dataset. For validation but also to gather some statistics, our approach has been applied to the 114 CAD models of the MAMBO dataset [16], a dataset manually assembled and curated to benchmark hex meshing algorithms for usage in a CAE industrial workflow. The model complexity varies from low (e.g., cylinder) to medium (e.g., the *M3* model in Figure 1). Many models present non-trivial CAD features, e.g., ski-jump corners, that are common in real-life CAD models but difficult for hex meshing algorithms. Although such models are not very complicated from a CAD point of view, no existing hex meshing method succeeds in meshing them satisfactorily. The techniques that are able to eventually produce a hexahedral mesh either sacrifice the mesh geometry (producing many degenerate hexahedra) or the model geometry (feature curves and corners being not respected).

Setup. In all our numerical experiments, the input quadrilateral meshes have been generated with the quasi-structured quadrilateral mesher [31] available in the open-source meshing software Gmsh [10]. This quad mesher relies on cross-field guidance to generate an initially unstructured quad mesh, which is then topologically optimized with patch-based remeshing. To measure quantitatively the ability of our approach to match the ideal valences x_i prescribed by the geometry of the model, the following energy is evaluated:

$$E(\mathbf{x}, \mathbf{n}) = \sum_{i=1..N_e} (n_i - x_i^*)^2 \tag{4}$$

with $x_i^* = 2$ if $round(x_i) = 2$ and $x_i^* = x_i$ otherwise. The special case $x_i^* = 2$ is used to remove the contribution to the energy of smooth regions where the dihedral angle is not exactly π. Floating-point values x_i are kept for valences different from 2 in order not to introduce spurious

discontinuities, e.g., an intermediate ideal valence like $x_i = 2.51$ is not expected to favor $n_i = 3$ much more than $n_i = 2$ (which rounding-off would do). The geometric quality of the quadrangular and hexahedral elements are estimated with the element-minimum of the **Signed Inverse Condition Number (SICN)** [13], which is a standard quality metric in Gmsh. For practical purposes, this measure is similar to the scaled Jacobian: it is negative for tangled elements, takes values between 0+ to 1 for valid elements, with a maximum for square/cube shapes. The difference is that collapsing edges (tending to 0 but strictly positive) are penalized by the SICN measure, whereas they are not by the scaled Jacobian (because of the "scaled" part). To have a point of reference to analyse the results, the purely extruded hexahedral meshes ($\forall i, n_i = 2$) were also built, with the same input boundary meshes and the same geometry optimization (Section 4.5).

Results. Per-model statistics and figures for the 114 MAMBO models are compiled in the supplementary material, and some results are highlighted in Figure 16. In summary, our method has been able to successfully build a topologically valid all-hexahedral layer for all models. Compared to pure extrusions, the energy (Equation (4)) is decreased by 95% on average, with a minimum at 68%, a maximum at 100% and a median at 98%. Concerning element quality, we observed that the minimum SICN quality was better with our method than with pure extrusion in 94/114 cases, with a minimum at -0.57 and an average at 0.35. The average quality is improved in all cases, with a minimum at 0.64 and an average at 0.92. There are however 5 cases where the minimum SICN quality is negative (inverted hexahedra), but the quality was also negative with pure extrusion in four of these cases. We discuss these *failure to untangle* situations later in Section 5.4. Finally, there are five cases where the purely extruded mesh had a tangled geometry, whereas our boundary layer has a strictly positive quality.

In 69 of the 114 models, there were non grid-like corners in the model for which it was not possible to build a hexahedral mesh matching the ideal valences. To solve these conflicting corners (611 in total), our solver produced 2,717 irregular configurations (final valences different from the ideal ones). There are more irregular configurations in the output than conflicting corners because by definition they cannot be resolved locally and they *leak* on their neighborhood, see Section 5.2 for a more comprehensive discussion.

To conclude this experiment, our approach has demonstrated its robustness and is very effective at matching the ideal hexahedral valences. Interesting and often non-intuitive local solutions are found, even in tricky situations where a human would not be able to find them, or not without considerable effort. The few remaining failures appear to be due to quality issues already present in the input boundary mesh and/or in the input CAD model. In those cases, improving the input is mandatory to obtain a hexahedral layer of good quality.

5.2 Solutions on Non Grid-like Corners

We call grid-like corners configurations where the ideal hexahedral valences can be respected. Non grid-like corners are caused by geometry (e.g., pyramid apex, ski-jump), by non-ideal topology of the boundary mesh at a corner or a ridge (Section 5.3, e.g., Figure 11), or by a combination of both.

Figure 10 illustrates the case of a non grid-like corner. Four convex feature curves are incident to the apex, but no disk triangulation exists with four valence one boundary vertices (Figure 5(a)). It is therefore impossible to match ideal hexahedral valences at the apex of a pyramid. Using our boundary layer solver (Equation (3)), optimal boundary hexahedral valences are computed for three different boundary meshes: the initial Schneiders' pyramid mesh, and two consecutive midpoint subdivisions of it. We can observe that the non grid-like corner also affects the configurations at adjacent vertices, where non-ideal boundary hexahedral configurations must be used as well,

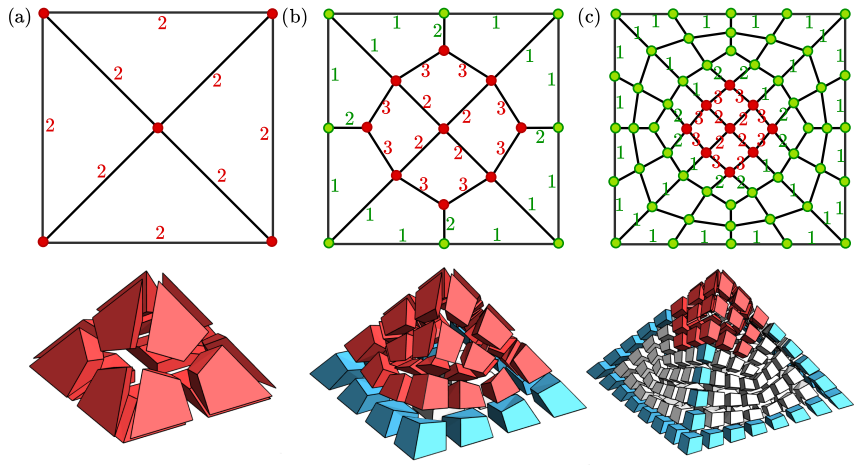


Fig. 10. All-hexahedral boundary layers of the pyramid, for different mesh resolutions (top row). The numbers indicate the edge hexahedral valences found by the integer solver (with valence two implied if no number is shown). Green color is used whenever actual and ideal valences match, and red color otherwise. The bottom row displays the corresponding boundary layers, with white color for regula extrusion, blue color for ideal valence (here valence one on convex feature curves), and red color for irregular solutions.

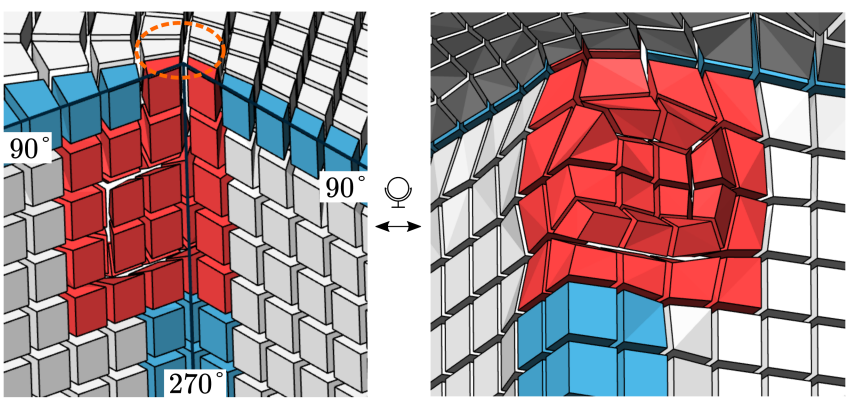


Fig. 11. Irregular hexahedral layer at a corner where the quadrilateral mesh is not ideal: the valence is two on the upper surface instead of three (orange circle). Because of this deviation of the boundary mesh from the ideal case, there exist no hexahedral configuration matching the ideal valences at that corner. The integer solver must find a way to transition from valence three to valence two on the vertical feature curve. The solution found is shown in the picture at the right-hand side, which displays the optimal boundary layer *seen from the interior of the volume.*

as indicated by red dots. The first boundary mesh (Figure 10(a)) is very constrained, as the mesh is very coarse, and the optimal hexahedral layer turns out to be the simple extrusion ($\forall i, n_i = 2$). For the two more refined meshes (Figures 10(b) and (c)), it is interesting to observe that the solver introduces a ring of valence three edges that isolates the non grid-like corner from the rest of the mesh, where ideal configurations (green dots) can then be used.

Figure 11 illustrates the case of a non grid-like corner caused by the boundary mesh. The quad valence at the corner on the top surface is two, whereas it should be three ideally, as the dihedral

angle of the vertical reentrant feature curve is 270°. The integer solver solves this irregularity coming from the boundary mesh and the CAD geometry by inserting a ring of valence three boundary edges surrounding the non grid-like corner (similar to that in the pyramid Figure 10) to transition from three on the lower part the vertical feature curve, to two on the top surface.

According to our experience, such behaviors are typical and the solver is always able to contain non-ideal configurations within a rather small set of boundary vertices, using various transition schemes that may seem counter-intuitive, such as the $\mathbf{n}^j = (1, 3, 2, 3)$ transition configuration introduced at the red nodes adjacent to the apex in the boundary layer of the pyramid (Figures 10(b) and (c)).

Of course, a wealth of different non grid-like configurations are encountered in real-life CAD models, but it can be noted that, even in moderately complicated models like in the MAMBO dataset, it already happened for 66/114 of the models. The introductory figure of this paper, Figure 1, further illustrates this by displaying the zoom-in over four irregular solutions around non grid-like corners of the model *M3*. Each of them is peculiar in the sense that the quadrilateral mesh in the neighborhood of the non grid-like corners is each time different, and the ideal hexahedral valences, depending on the geometry, can be anything from one to four. Complicated configurations are particularly frequent at corners resulting from CAD boolean operations. Things become even harder when several non grid-like corners close to each other must be resolved together. Fortunately, our integer formulation (Equation (3)) is able to deal with all cases in the same generic manner, and always finds the locally optimal solution, or a close one.

5.3 Sensitivity to Boundary Mesh

As our hexahedral layer matches the midpoint subdivision of the input boundary mesh, its topology and geometry are necessarily strongly dependent on this input mesh. In first approximation (this is not exactly true), the boundary mesh defines an upper bound on the hexahedral mesh geometric quality and the topological regularity. This is both a good and a bad thing.

On the positive side, the generated boundary layer is guaranteed to exactly match the boundary geometry. A natural way to improve the hexahedral boundary layers is thus to simply improve the quad mesher, which is a reasonable and reachable goal, in which significant progress is expected in the near future.

But on the other hand, even if our topological formulation is proven robust even in case of *bad* inputs, the topological regularity of the output and the geometry untangling process are not. A high-quality all-hexahedra boundary layers suitable for accurate numerical simulation cannot be expected without having high-quality quadrilateral (or quad-dominant) surface meshes in the first place. Figure 12 shows how irregular the boundary layer can be if it is generated from a triangular boundary mesh (instead of quadrangular). This boundary layer is of no practical use, except that it confirms the robustness of our technique.

We consider that a boundary layer is *good* if the boundary hexahedral valences match their ideal value. It is in general not possible to achieve this at non-grid like corners (Section 5.2), but this regularity requirement should also be considered at the vertices of the boundary mesh that are located on feature curves (Figure 12(b)). In the geometrically-smooth regions of the boundary mesh, on the other hand, irregularity is less an issue, because the boundary layer matching ideal valences is there directly obtained by a simple extrusion.

As just mentioned, an irregularity of the boundary mesh on a vertex located on a feature curve may lead to a situation for which no ideal configuration exists. For instance, consider a vertex (red dot) on a convex feature curve (materialized in the figure by the two boundary edges with valence one) that has two adjacent quads on one side of the feature curve, and three adjacent quads on the other side (inset figure). Ideal valences (rounded) would be in this case $\mathbf{n}^{j*} = (1^*, 2^*, 1^*, 2^*, 2^*)$,

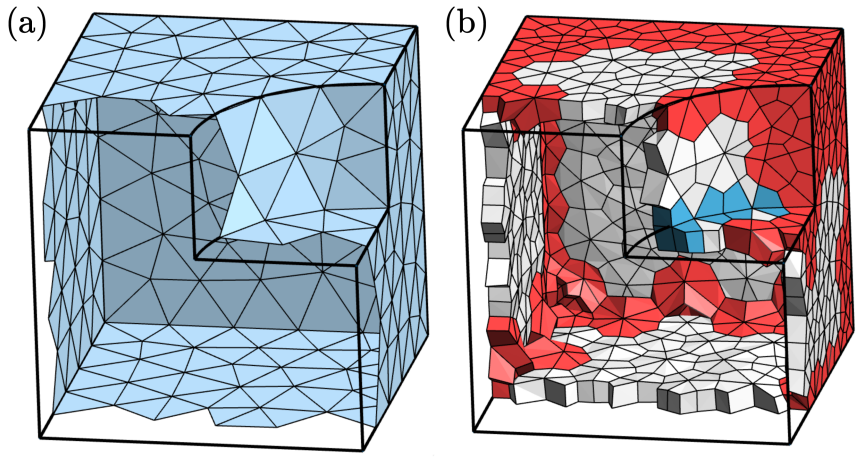
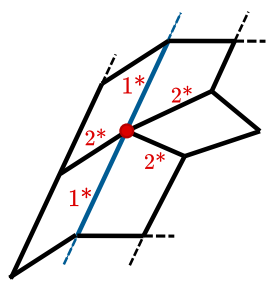


Fig. 12. Extreme case showing the detrimental effect of a triangular boundary mesh (a) (instead of a quadrangular mesh) on the topology of the generated boundary layer (b). As the input topology is irregular on feature curves (most valences are different from four), most boundary hexahedral configurations are irregular around feature curves (red), and the integer programming problem is globally coupled (633 unknowns, 213 disk triangulations).

which is not matched by any triangulation of the disk. A solution is to use a hexahedral valence of three on left, i.e., $\mathbf{n}^j = (1, 3, 1, 2, 2)$, but this would affect the configurations at the other vertices nearby, as in the pyramid apex case. A small irregularity on a feature curve leads thus, as one sees, to a quite irregular hexahedral mesh locally. If there are many such irregular vertices, they can connect together, and the integer programming problem becomes global (Figure 12(b)). In conclusion, it is important to build boundary meshes with as much as possible regular vertices on all feature curves, in order to improve the quality of the hexahedral layer.



5.4 Geometry and Untangling Limitations

A typical failure of the untangling process may occur when several irregular boundary vertices packed in a very acute corner, e.g., ski-jump where the CAD corner angle tends to zero. Consider, for instance, the model *S24* (Figure 13) of the MAMBO dataset (Section 5.1). The difficulty arises in this model from the boolean intersection of a rhomboid and a vertical cylinder, which creates a pair of very thin curvilinear triangle-shaped surfaces. The minimum quality in the hexahedral layer is -0.47, which corresponds to inverted hexahedra in a pinched region (Figure 13(c)). In the initial quad mesh (Figure 13(a)), the minimum quality is low but strictly positive: 0.0046. The issue is not the acute corner in itself, because our method is able to produce geometrically valid hexahedral layers around acute corners (e.g., Figure 1). The reason of the failure is here the unfortunate combination of a pinched CAD geometry with an irregular quadrilateral mesh in the area. The boundary mesh (Figure 13(b)) has at the same time several irregular vertices and highly distorted quadrangles. This results in constraints too strong to be all respected by the geometry untangler, which hence fails to produce valid hexahedra. The untangler algorithm is however not to blame, as it is probably impossible to untangle the mesh under these hard constraints.

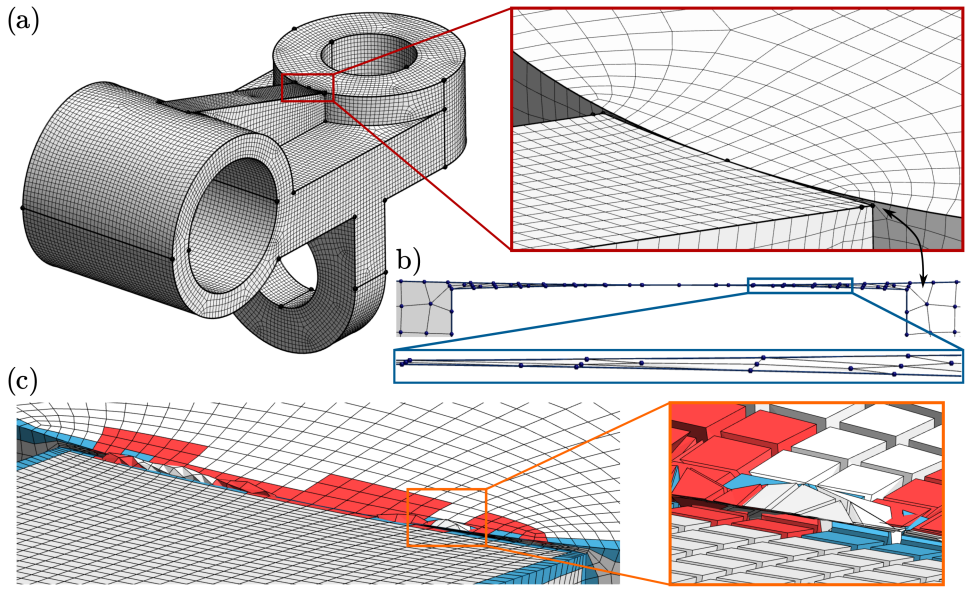


Fig. 13. The model *S24* contains two highly pinched regions (angle ≈ 1°) at the boolean intersection between a rhomboid and a vertical cylinder, as shown in the close-up of (a). In this region, the hexahedral layer (c) contains inverted elements and parts which cross the boundary surface because they are locally pushed outside the volume instead of inwards by the geometry untangler. The untangler is failing because the input quad mesh (a) contains many irregular vertices and highly distorted quads in the pinched region of the cylinder surface, which is isolated and highlighted in (b).

Such situations happen in real-life CAD models, and five examples have been encountered in the MAMBO dataset. However, we think that the problem is to be traced back down to the surface mesher. For reference, with the model *S24*, the minimum quality was worse (*−0.41* instead of *−0.26*) with an extruded hexahedral layer based on the same boundary mesh and the same geometry optimization. Out of experience, we therefore think that very acute corners should be given a specific treatment in quad mesher, to ensure a locally optimal quadrilateral mesh whose topology is that of the classical mid-point subdivision of the triangle into three quadrangles adjacent to an irregular vertex (of triangular valence instead of four), and whose geometry is fitted into the acute corner by shrinking one side of the triangle to a very small length compared to the other two sides.

Another geometric issue is the self-intersection of the boundary layer. Even if all hexahedra are geometrically valid, elements on opposite sides of thin regions may overlap. In the MAMBO dataset, this happens in 21/114 cases. A simple mitigation strategy is to reduce the layer thickness when self-intersections occur. With simple heuristics, such as edge shortening and local surface smoothing around self-intersections, we are able to recover interior surfaces without self-intersections in 111/114 cases, but there is no guarantee. We think that a better approach is to add a tetrahedral mesh of the remaining interior volume (Section 5.7) and to untangle the entire mesh, composed of both the all-hexahedral layer and the interior tetrahedral mesh.

5.5 Performance and Scaling

The most computationally expensive steps of our method are: (a) solving the integer programming problem (Equation (3)), and (b) mesh untangling/smoothing (Section 4.5).

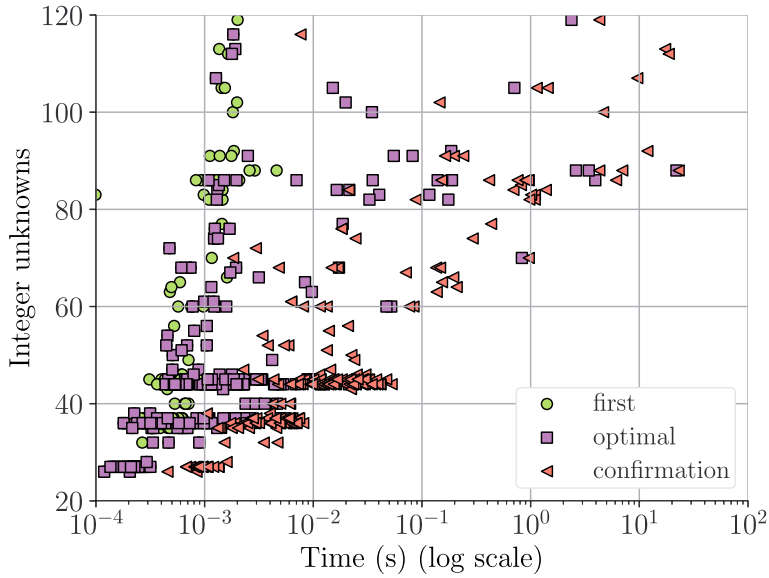


Fig. 14. Performance of the branch-and-bound search on 222 sub-problems found in the MAMBO models. For each sub-problem, we show the elapsed times to get the first solution (green circle) and the last solution (purple square), in relation with the number of integer unknowns. Red triangles indicate the time it took to finish the search, ensuring that the last solution is optimal.

The integer programming problem (Equation (3)) is solved with the branch-and-bound search applied to sub-problems built by problem decomposition (Section 4.4). In the MAMBO models (Section 5.1), the average integer programming sub-problem contains 68 unknowns and 1, 128 integer constraints (built with Proposition 1). The performance is illustrated on Figure 14. We observe that the search is able to find initial solutions very quickly, usually in less than 10 milliseconds. Finding the optimal solution takes a bit longer, but less than 100 milliseconds in general. However, completing the search-through in order to confirm that the solution is indeed optimal can take seconds, or even dozens of seconds. Figure 14 only reports models for which the confirmation of optimality took less than 30 seconds.

Due to our choice of branching functions, first solutions are rapidly close to the optimal solution, and our experience indicates that there is no real need in practice to wait until the end of search. We recommend allocating a comfortable time budget for the initial solution (e.g., 30 seconds), a quite smaller amount of time for the solution improvement (e.g., 3 seconds), and to never wait until completion of the search. With these settings, the optimal solution is found in almost all the cases of Figure 14, albeit without full guarantee.

It is definitely possible to improve the search performance by adding more heuristics in the branching functions, and to use better time budget allocations (e.g., in function of the number of integer unknowns and of integer constraints), but, in practice, we have never felt the necessity to do so. The current strategy is fast enough, i.e., much faster than the other steps of the pipeline (quad meshing, geometry untangling). In the MAMBO dataset examples, solving sub-problems takes on average 0.7 seconds. Detailed timings are available in the supplemental material.

It should be noted that the previous performance considerations assume the decomposition of the large problem into a series of independent sub-problems of moderate size (a few hundred unknowns at most), as explained in Section 4.4. The branch-and-bound search is indeed not directly applicable to a large size model (e.g., hundreds of thousand of unknowns), because the computer

would quickly run out of memory due to the branching. Moreover, at many places, the surface is smooth and the optimal boundary layer is generated straight away by extrusion without the need to solve anything. In these conditions, our approach scales very well and there is no specific limit on the size of the input boundary mesh. Even if it contains dozens of millions of quads, the problem will be reduced to a limited series of sub-problems of modest size, one per non grid-like CAD corner, in first approximation.

The geometrical optimization of the boundary layer is another computationally expensive task of the pipeline, whose cost remains however reasonable if fast heuristics are used to generate the initial geometry (Section 4.5), and if the non-linear optimization [8] is used only in small cavities with low-quality hexahedra. The geometric construction of the hexahedral boundary layer took between 0.4s and 26s per 10k hexahedra, for the MAMBO dataset models, with a median at 1.4s and an average at 2s. Larger values (> 10s) correspond to situations where the mesh could actually not be successfully untangled (Section 5.4), and the solver was stopped after reaching the maximum number of iterations.

All reported timings correspond to computations done with a single thread processor on a 2.8GHz laptop. For better performance, it is straightforward to parallelize the solving of the independent integer programming sub-problems, and also to untangle/smooth the low-quality mesh cavities in parallel.

5.6 Comparison with Related Work

Most existing works that are related to the meshing technique presented in this paper focus on generating an entire hexahedral mesh, which is much more involved than simply generating a single boundary layer. We point out in this section a number of differences that can be observed concerning the generated boundary hexahedra, but the reader should keep in mind that this comparison might be somewhat unfair.

Polycube and frame-field based approaches can only deal with corners where a boundary hexahedral configuration matching the ideal valences exist (i.e., grid-like corners), because they explicitly assume the existence of six orthogonal directions, either for boundary flagging or for frame representation. Due to this fundamental assumption, they fail to respect feature curves at non grid-like corners. Consider for instance how an **Integer Grid Mapping (IGM)** hex meshing algorithm behaves at the apex of a pyramid (Figure 15(a)). Note that a smaller copy of the pyramid in subtracted from the initial one, so that only a layer remains to be meshed instead of the entire pyramid volume. The IGM mesher starts by computing an initial frame-field with [29], then a parameterization with CubeCover [28], and a hexahedral mesh is finally extracted with HexEx [23]. Due to the non grid-like nature of the apex, the generated frame-field topology presents a singular line linking the two apexes (depicted in black in the insert). Consequently, the parameterization is necessarily degenerated and not hex-meshable at that place. The obtained hexahedral mesh is thus completely broken, with missing elements in the middle of the model and degenerate or missing hexahedra at the apex.

On the other hand, grid and octree-based hex meshing techniques are robust, but they do not respect feature curves by construction. Instead, they try to recover them approximatively by projecting (and optimizing) a padding layer. As the hex mesh boundary topology is determined in this case by the interior grid (volume-to-surface), it does not match the topology of the CAD feature curves, and non-grid like corners cannot be recovered exactly. In the example in Figure 15(b), the mesher *Hexotic* [25] is able to successfully mesh the pyramid, but it does so by splitting the initial 4-valent apex into two 3-valent corners, slightly altering thus the model geometry. In this specific example, this is not really an issue because the geometry is very simple and there is ample room to refine the octree-based mesh around the corner, but there are stringent practical limits on the

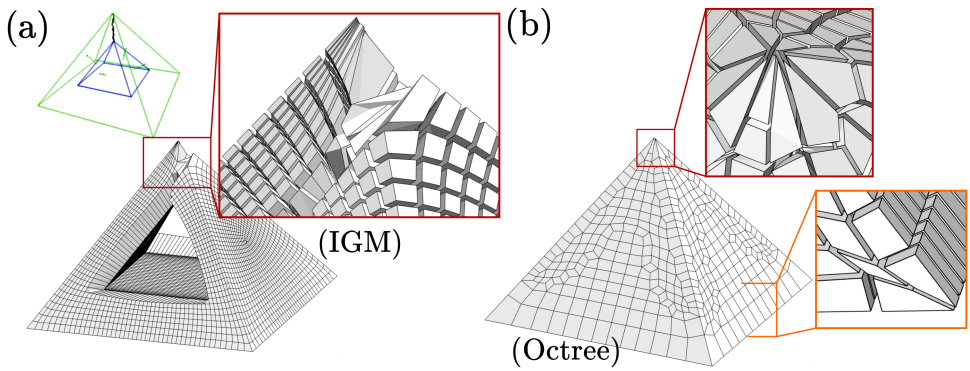


Fig. 15. Results obtained with state-of-the-art hex meshers. In the left (a), a model made of a pyramid minus a smaller pyramid is meshed with a Integer Grid Mapping technique based on frame-field and PolyCube parameterization. The hex mesh is broken, with missing and degenerate hexahedra. In the right (b), the pyramid is meshed with Hexotic [25], a commercial octree-based robust mesher. The hex mesh is valid but does not respect the CAD geometry strictly (apex split in two) and the obtained hexahedral valences are not the ideal ones for the edges on the feature curves.

octree refinement when dealing with real-life CAD models. Another issue with this approach is that the padding layer on the boundary is equivalent to an extrusion, so that the hexahedral valences are not the ideal ones on the feature curves.

When an existing hexahedral mesh is available (built with the previous methods or by subdividing tetrahedra into four hexahedra), the boundary layer mesh can be improved by inserting fundamental sheets [34], which are dual surfaces parallel to the model boundary but which may also propagate inside the volume at concave ridges. After having identified 11 valid corner configurations, [17, 39] assign hexahedral valences at CAD curves which are compatible with the construction of fundamental sheets. Compared to our approach, their technique is less generic (only 11 cases covered) and the non-grid like corners are not resolved locally but globally because they use a single valence along entire curves. This leads to non-ideal hexahedral valences on large portions of the mesh. Also, the generation of the sheets (by propagation) is not robust and sensitive to the structure of the initial hexahedral mesh.

To the best of our knowledge, our technique is the first robust one which is able to both (a) respect the model geometry (strictly), and (b) produce all-hexahedral boundary layers where the boundary valences are adapted to the geometry.

5.7 Meshing of the Remaining Volume

Starting from the surface mesh Q bounding a volume Ω, we have built a hexahedral layer H. To complete the volume mesh of Ω, it remains to mesh the interior $\Omega_I = \Omega - H$, whose boundary is a watertight quadrilateral mesh Q_I. If Q_I is intersection-free, then we can use standard constrained tetrahedral meshing technique. If Q_I contains self-intersections, we propose a robust topological construction of the interior tetrahedral mesh, but which contains inverted tetrahedra and thus requires further post-processing.

Constrained Delaunay tetrahedral mesh. When there is no tangled hexahedra in H and no overlapping of the boundary layer, Q_I is a geometrically valid quad mesh without self-intersections. We can subdivide each quadrilateral into two triangles and call a constrained Delaunay tetrahedral mesher (e.g., TetGen [35]) to obtain a valid tetrahedral mesh T_I of the interior. Then $M = H + T_I$ is

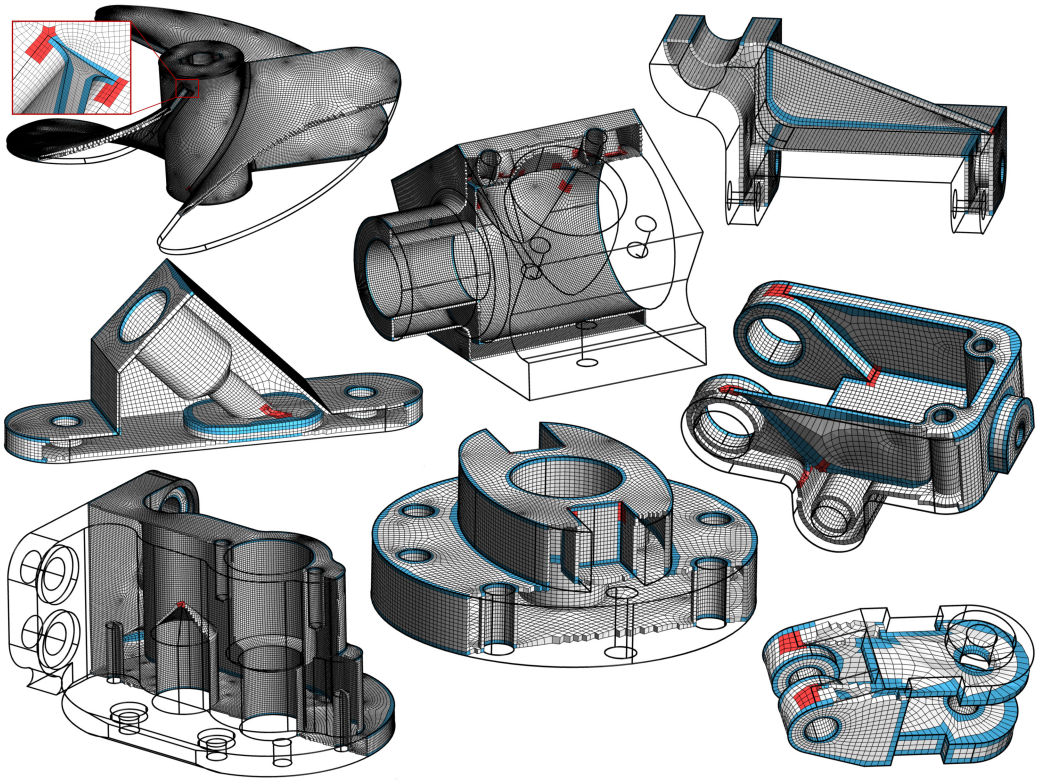


Fig. 16. All-hexahedral boundary layers on models of the MAMBO dataset.

a hex-tet mesh of the model Ω, with non-conforming quad-triangle configurations at the interface between the all-hexahedral boundary layer and the interior tetrahedra. Note that the tetrahedral mesh can be replaced by a hex-dominant one with the following simple procedure: position the interior points according to a 3D frame-field with a frontal insertion algorithm, and combine the tetrahedra into hexahedra.

In the MAMBO dataset, constrained Delaunay tetrahedral meshing of the interior works directly in 89/114 of the cases, as illustrated on Figure 17(a). Four of the failures are due to tangled hexahedra, which induce self-intersections in Q_I, and the remaining twenty-one failures are due to boundary layer overlaps in thin regions. As discussed in Section 5.4, we can use simple heuristics (layer thickness reduction and local smoothing) to recover intersection-free surfaces in 111/114 of the cases and successfully mesh the interior. However, this procedure is not robust and the boundary layer deformation may introduce tangled hexahedra.

Topological tetrahedral padding. When there are self-intersections in Q_I, we propose to use a topological padding to build the interior tetrahedral mesh. The idea is to make a topological transition between Q_I and the midpoint subdivision Q_s of the input surface mesh Q, for which we can have a tet mesh.

As the midpoint subdivision Q_s inherits the intersection-free property of the input Q, we can always build a tetrahedral mesh T_{II} constrained to Q_s with a Delaunay mesher (assuming subdivision of the quads into two triangles).

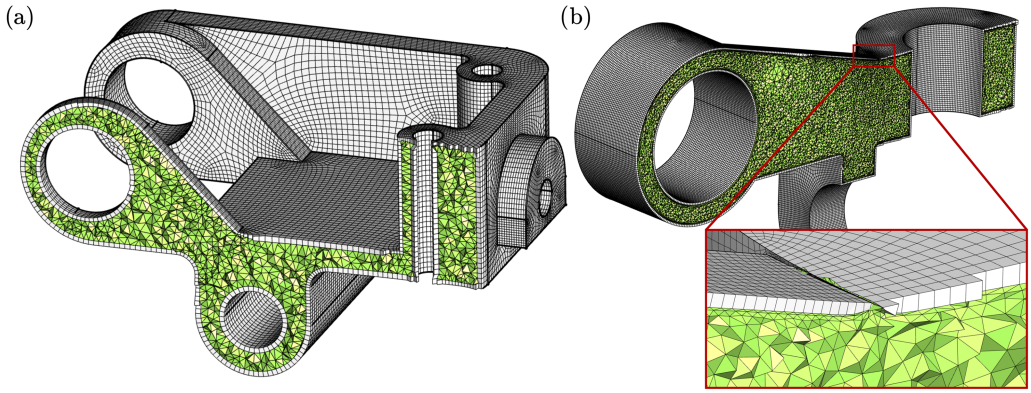


Fig. 17. Hex-dominant meshes with all-hexahedral boundary layer (grey) and interior tetrahedra (green). The interior mesh is constructed either by constrained Delaunay tetrahedralization if the interior quadrilateral mesh is intersection-free (a), or with a topological padding (b). The tetrahedral mesh on the right (b) is built from the tangled hexahedral layer shown on Figure 13 and contains inverted tetrahedra, but is topologically valid.

The previously built hexahedral layer H is a transition between the boundary quad mesh Q_s and the interior quad mesh Q_I. We can build a new transition tetrahedral layer T_T by duplicating the hexahedra in H, inverting them, and subdividing them coherently into tetrahedra [5]. Note that all the tetrahedra in T_T are geometrically inverted (negative volume) in this construction.

By merging the all-hexahedral layer H, the transition tetrahedral layer T_T and the interior tetrahedral mesh T_{II}, we generate a complete volume mesh $M = H+T_T+T_{II}$ of Ω, which is topologically valid but with inverted tetrahedra in T_T, and potentially other inverted elements due to tangled elements in the hexahedral layer or to layer overlaps in thin regions. An example of mesh built with this approach is shown on Figure 17(b). The next step, which we leave for future work, is to untangle the hex-dominant mesh with global geometrical optimization, and topological optimization (e.g., edge swaps) of the interior inverted tetrahedra, e.g., by following the minimal volume principle [4].

6 CONCLUSION AND PERSPECTIVES

The main idea of this article is the writing of an optimization problem in integer variables to find out hexahedral configurations around complex ridges and corners in order to get a high-quality all-hexahedral layer on the boundary, whose topology is adapted to the model geometry. Our proposed method is guaranteed to produce a valid topological volume mesh, both of the all-hexahedral layer (Section 4) and of the remaining interior (Section 5.7). Our surface-to-volume approach accurately respects the model geometry and fits perfectly in a bottom-up meshing workflow, which is common in CAD/CAE. For instance, the meshing of models with multiple volumes and internal surfaces (e.g., bi-material, fluid-structure interaction) is automatically supported.

In most cases, the resulting meshes are geometrically valid (strictly positive Jacobian everywhere) and can be used directly for numerical simulation. When inverted elements are still present in the meshes, further work is required to fix the geometry. For instance, if the surface mesh is too constraining, e.g., extremely low quality quads, it may be necessary to modify the input surface mesh. If the interior mesh has been constructed with topological transition because of self-intersections, and contains inverted tetrahedra, then standard tetrahedral mesh operations such as edge swaps should be used to untangle the interior. Fully addressing the geometrical question is

a significant challenge in itself, but we are confident that it is possible to develop robust methods that guarantee element geometric validity as we can start from a topologically valid volume mesh whose boundary is also geometrically valid.

Iterative layers. An appealing natural extension of this work would be to apply the boundary layer construction iteratively to progressively fill the volume inwards. Although there is no theoretical obstruction, this does not produce interesting meshes because at non-grid like corners, the interior quad mesh is usually more irregular and twisted than the surface boundary mesh (e.g., Figure 1). On the other hand, the layer interior surface is less constrained than the model boundary because we have the freedom to move the vertices. We believe that for a frontal all-hexahedral layering, it would be necessary to add terms related to the interior surface regularity in the global formulation (Equation (3)).

Anisotropic layers. For certain applications such as CFD, it is necessary to have multiple anisotropic layers with increasing and well-controlled thickness. We believe that the best way to extend our work towards such boundary layers is to anisotropically subdivide the existing isotropic layer. As the topological irregularity of the hexahedral mesh (e.g., Figure 2) may lead to undesirable anisotropic meshes, it may be interesting to add regularity terms in our formulation (Equation (3)), e.g., penalizing non-extrusion, or to resort to non-hexahedral elements in specific configurations.

Extension to interior hexahedral configurations. We have developed an integer representation (Proposition 1) of the boundary hexahedral configurations using disk triangulations. Assuming that we could do the same for internal hexahedral configurations with the triangulations of the sphere, and that we would have well placed interior vertices, we could imagine formulating the complete constrained hexahedral meshing problem as a global integer programming problem. Of course this is extremely hypothetical, and it remains to be seen if it is possible or of any practical interest.

APPENDIX

A RECURSIVE DISK BOUNDARY REDUCTION AND INTEGER CONSTRAINTS

The following Python listing describes how to generate the sub-problems of Proposition 1, by performing vertex ablation (Figure 6(a)) recursively.

```
# Example of input:
Ninf = [1, 1, 1, 2, 2]    # minimum of valence range
Off = [0 for v in Ninf]   # offset on valence
Lock = [0 for v in Ninf]  # fixed valence (0 for free)
poly = [Ninf, Off, Lock]  # polygon with valence info

# Generate sub-polygons by recursive cutting
def generateSubPolygons(poly, subpolys):
    subpolys.append(poly)

    N = poly[0] # minimum boundary valence
    O = poly[1] # offset to apply after cuts
    L = poly[2] # fixed boundary valence after cuts

    # Stop the recursion if sub-poly is triangle
    nFree = sum(l == 0 for l in L)
    if nFree == 3:
        return

    # Loop over the free vertices which can be cut
    for i in range(len(N)):
```

```
        if L[i] == 0 and N[i] + O[i] <= 1:
            # Initialize sub-polygon lists
            O2 = O.copy()
            L2 = L.copy()

            # Cut at the i-th vertex
            L2[i] = 1-O2[i] # Lock the vertex valence
            # Decrease valence at cut extremities
            iPrev = getPrevFree(N,L,i)
            iNext = getNextFree(N,L,i)
            O2[iPrev] -= 1
            O2[iNext] -= 1

            # Recursive call
            generateSubPolygons([N,O2,L2],subpolys)
```

Listing 1. Recursive reduction of polygons.

Example of integer constraints. Consider the boundary valences (n_1, n_2, n_3, n_4) of the triangulations of a disk of size four. We assume that the range of each variable n_i is $[1..4]$. To apply the integer constraints of Proposition 1, we start with the above reduction procedure. We obtain five sub-polygons, the initial one and one four each vertex ablation (Figure 6(a)). In each reduced problem, the variables adjacent to the ablated vertex are decreased by one ($n_i^s = n_i - 1,\ n_i^s \in [0..3]$). But $n_i^s = 0$ cannot lead to a valid sub-triangulation, so we can set the range to $n_i^s \in [1..3]$. This correction of the range is equivalent to enforcing the second condition of Proposition 1: no consecutive valence one. The allowed variable ranges in the five sub-polygons are:

$$
\begin{aligned}
&(1) && n_1 \in [1..4], n_2 \in [1..4], n_3 \in [1..4], n_4 \in [1..4] \\
&(2) && n_1 = 1 \text{ and } n_2^{s2} \in [1..3], n_3^{s2} \in [1..4], n_4^{s2} \in [1..3] \\
&(3) && n_2 = 1 \text{ and } n_1^{s3} \in [1..3], n_3^{s3} \in [1..3], n_4^{s3} \in [1..4] \\
&(4) && n_3 = 1 \text{ and } n_1^{s4} \in [1..4], n_2^{s4} \in [1..3], n_4^{s4} \in [1..3] \\
&(5) && n_4 = 1 \text{ and } n_1^{s5} \in [1..3], n_2^{s5} \in [1..4], n_3^{s5} \in [1..3]
\end{aligned}
$$

The reduction is complete because the sub-polygons have only three free variables. Now we can apply the remaining conditions of Proposition 1 to each scenario:

$$
\begin{aligned}
&\#(n_i = 2) \neq 3 \\
&n_1 = 1 \implies \#(n_i^{s2} = 2) \neq 2 \text{ and } \#(n_i^{s2} = 1) \neq 1 \text{ and } \#(n_i^{s2} = 1) \neq 2 \\
&n_2 = 1 \implies \#(n_i^{s3} = 2) \neq 2 \text{ and } \#(n_i^{s3} = 1) \neq 1 \text{ and } \#(n_i^{s3} = 1) \neq 2 \\
&n_3 = 1 \implies \#(n_i^{s4} = 2) \neq 2 \text{ and } \#(n_i^{s4} = 1) \neq 1 \text{ and } \#(n_i^{s4} = 1) \neq 2 \\
&n_4 = 1 \implies \#(n_i^{s5} = 2) \neq 2 \text{ and } \#(n_i^{s5} = 1) \neq 1 \text{ and } \#(n_i^{s5} = 1) \neq 2
\end{aligned}
$$

The library *Gecode* directly supports these conditional and counting constraints, and our C++ setup of the integer solver is available in the supplemental material.

ACKNOWLEDGMENTS

The authors would like to thank Quentin Plazar for his helpful advice on solving the linear integer programming problem.

REFERENCES

[1] Michael L. Brewer, Lori Freitag Diachin, Patrick M. Knupp, Thomas Leurent, and Darryl J. Melander. 2003. The mesquite mesh quality improvement toolkit. In *Proceedings of the 12th International Meshing Roundtable*.

[2] Gunnar Brinkmann, Brendan D. McKay, et al. 2007. Fast generation of planar graphs. *MATCH Commun. Math. Comput. Chem.* 58, 2 (2007), 323–357.

[3] Gianmarco Cherchi, Pierre Alliez, Riccardo Scateni, Max Lyon, and David Bommes. 2019. Selective padding for polycube-based hexahedral meshing. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 580–591.

[4] Thierry Coupez. 2000. Génération de maillage et adaptation de maillage par optimisation locale. *Revue Européenne Des éléments Finis* 9, 4 (2000), 403–423.

[5] Julien Dompierre, Paul Labbé, Marie-Gabrielle Vallet, and Ricardo Camarero. 1999. How to subdivide pyramids, prisms, and hexahedra into tetrahedra. *IMR* 99 (1999), 195.

[6] Jeff Erickson. 2014. Efficiently hex-meshing things with topology. *Discrete & Computational Geometry* 52, 3 (2014), 427–449. https://doi.org/10.1007/s00454-014-9624-3

[7] Xifeng Gao, Hanxiao Shen, and Daniele Panozzo. 2019. Feature preserving octree-based hexahedral meshing. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 135–149.

[8] Vladimir Garanzha, Igor Kaporin, Liudmila Kudryavtseva, François Protais, Nicolas Ray, and Dmitry Sokolov. 2021. Foldover-free maps in 50 lines of code. *arXiv preprint arXiv:2102.03069* (2021).

[9] Rao V. Garimella and Mark S. Shephard. 2000. Boundary layer mesh generation for viscous flow simulations. *Internat. J. Numer. Methods Engrg.* 49, 1-2 (2000), 193–218. https://doi.org/10.1002/1097-0207(20000910/20)49:1/2<193::aid-nme929>3.0.co;2-r

[10] Christophe Geuzaine and Jean-François Remacle. 2009. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *Internat. J. Numer. Methods Engrg.* 79, 11 (2009), 1309–1331. https://doi.org/10.1002/nme.2579

[11] Jin Huang, Yiying Tong, Hongyu Wei, and Hujun Bao. 2011. Boundary aligned smooth 3D cross-frame field. *ACM Transactions on Graphics* 30, 6 (2011), 1–8. https://doi.org/10.1145/2070781.2024177

[12] Rajeev Jain and Timothy J. Tautges. 2014. PostBL: Post-mesh boundary layer mesh generation tool. In *Proceedings of the 22nd International Meshing Roundtable*. Springer International Publishing, 331–348. https://doi.org/10.1007/978-3-319-02335-9_19

[13] Patrick M. Knupp. 2000. Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part II—A framework for volume mesh optimization and the condition number of the Jacobian matrix. *International Journal for Numerical Methods in Engineering* 48, 8 (2000), 1165–1185.

[14] Nicolas Kowalski, Franck Ledoux, Matthew L. Staten, and Steve J. Owen. 2011. Fun sheet matching: Towards automatic block decomposition for hexahedral meshes. *Engineering with Computers* 28, 3 (2011), 241–253. https://doi.org/10.1007/s00366-010-0207-5

[15] Michael Kremer, David Bommes, Isaak Lim, and Leif Kobbelt. 2014. Advanced automatic hexahedral mesh generation from surface quad meshes. In *Proceedings of the 22nd International Meshing Roundtable*. Springer International Publishing, 147–164. https://doi.org/10.1007/978-3-319-02335-9_9

[16] Franck Ledoux. 2020. MAMBO Dataset. (2020). https://gitlab.com/franck.ledoux/mambo.

[17] Franck Ledoux, Nicolas Le Goff, Steve J. Owen, Matthew L. Staten, and Jean-Christophe Weill. 2013. A constraint-based system to ensure the preservation of sharp geometric features in hexahedral meshes. In *Proceedings of the 21st International Meshing Roundtable*. Springer Berlin, 315–332. https://doi.org/10.1007/978-3-642-33573-0_19

[18] Franck Ledoux and Jean-Christophe Weill. 2008. An extension of the reliable Whisker Weaving algorithm. In *Proceedings of the 16th International Meshing Roundtable*. Springer, 215–232.

[19] Heng Liu, Paul Zhang, Edward Chien, Justin Solomon, and David Bommes. 2018. Singularity-constrained octahedral fields for hexahedral meshing. *ACM Transactions on Graphics* 37, 4 (2018), 1–17. https://doi.org/10.1145/3197517.3201344

[20] Marco Livesu, Luca Pitzalis, and Gianmarco Cherchi. 2021. Optimal dual schemes for adaptive grid based hexmeshing. *arXiv preprint arXiv:2103.07745* (2021).

[21] Marco Livesu, Nicholas Vining, Alla Sheffer, James Gregson, and Riccardo Scateni. 2013. PolyCut. *ACM Transactions on Graphics* 32, 6 (2013), 1–12. https://doi.org/10.1145/2508363.2508388

[22] Adrien Loseille and Rainald Löhner. 2013. Robust boundary layer mesh generation. In *Proceedings of the 21st International Meshing Roundtable*. Springer Berlin, 493–511. https://doi.org/10.1007/978-3-642-33573-0_29

[23] Max Lyon, David Bommes, and Leif Kobbelt. 2016. HexEx. *ACM Transactions on Graphics* 35, 4 (2016), 1–11. https://doi.org/10.1145/2897824.2925976

[24] David L. Marcum, Frederic Alauzet, and Adrien Loseille. 2017. On a robust boundary layer mesh generation process. In *55th AIAA Aerospace Sciences Meeting*. American Institute of Aeronautics and Astronautics. https://doi.org/10.2514/6.2017-0585

[25] Loïc Maréchal. 2009. Advances in octree-based all-hexahedral mesh generation: Handling sharp features. In *Proceedings of the 18th International Meshing Roundtable*. Springer Berlin, 65–84. https://doi.org/10.1007/978-3-642-04319-2_5
[26] Loïc Maréchal. 2016. All hexahedral boundary layers generation. *Procedia Engineering* 163 (2016), 5–19. https://doi.org/10.1016/j.proeng.2016.11.007
[27] M. Müller-Hannemann. 1999. Hexahedral mesh generation by successive dual cycle elimination. *Engineering with Computers* 15, 3 (1999), 269–279. https://doi.org/10.1007/s003660050022
[28] Matthias Nieser, Ulrich Reitebuch, and Konrad Polthier. 2011. CubeCover–parameterization of 3D volumes. In *Computer Graphics Forum*, Vol. 30. Wiley Online Library, 1397–1406.
[29] Nicolas Ray, Dmitry Sokolov, and Bruno Lévy. 2016. Practical 3D frame field generation. *ACM Transactions on Graphics* 35, 6 (2016), 1–9. https://doi.org/10.1145/2980179.2982408
[30] Maxence Reberol, Alexandre Chemin, and Jean-François Remacle. 2019. Multiple approaches to frame field correction for CAD models. *International Meshing Roundtable* (2019).
[31] Maxence Reberol, Christos Georgiadis, and Jean-François Remacle. 2021. Quasi-structured quadrilateral meshing in Gmsh–a robust pipeline for complex CAD models. *arXiv preprint arXiv:2103.04652* (2021).
[32] R. Schneiders. 1996. A grid-based algorithm for the generation of hexahedral element meshes. *Engineering with Computers* 12, 3-4 (1996), 168–177. https://doi.org/10.1007/bf01198732
[33] Christian Schulte, Mikael Lagerkvist, and Guido Tack. 2006. Gecode. Software Download and Online Material at the Website: http://www.gecode.org (2006), 11–13.
[34] Jason F. Shepherd. 2007. *Topologic and Geometric Constraint-based Hexahedral Mesh Generation*, Vol. 68.
[35] Hang Si. 2015. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Software* 41, 2 (2015), 1–36. https://doi.org/10.1145/2629697
[36] T. J. Tautges, T. Blacker, and S. A. Mitchell. 1996. The Whisker Weaving algorithm: A connectivity-based method for constructing all-hexahedral finite element meshes. *Internat. J. Numer. Methods Engrg.* 39, 19 (1996), 3327–3349. https://doi.org/10.1002/(sici)1097-0207(19961015)39:19<3327::aid-nme2>3.0.co;2-h
[37] Ko-Foa Tchon, Charles Hirsch, Robert Schneiders, Ko-Foa Tchon, Charles Hirsch, and Robert Schneiders. 1997. Octree-based hexahedral mesh generation for viscous flow simulations. In *13th Computational Fluid Dynamics Conference*. American Institute of Aeronautics and Astronautics, 195–196. https://doi.org/10.2514/6.1997-1980
[38] Kilian Verhetsel, Jeanne Pellerin, and Jean-François Remacle. 2019. Finding hexahedrizations for small quadrangulations of the sphere. *ACM Transactions on Graphics* 38, 4 (2019), 1–13. https://doi.org/10.1145/3306346.3323017
[39] Rui Wang, Zhihao Zheng, and Shuming Gao. 2020. Stream surface-supported fundamental sheets insertion toward high-quality hex meshing. *Journal of Computing and Information Science in Engineering* 20, 6 (2020). https://doi.org/10.1115/1.4047239

Received 15 September 2021; revised 23 August 2022; accepted 3 November 2022