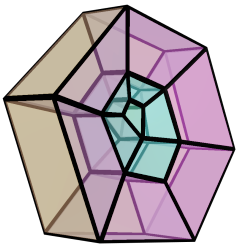
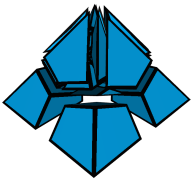
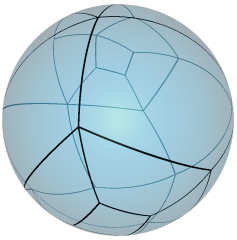
Combinatorial Techniques for Hexahedral Mesh Generation

Kilian Verhetsel

2024

Submitted in partial fulfillment of the requirements for the degree of
Doctor in Engineering Sciences

Advisor
Jean-François Remacle

Jury
David Bommes (Universität Bern)
Julien Hendrickx (UCLouvain)
Bruno Lévy (INRIA)
Scott A. Mitchell (Sandia National Laboratories)
Jeanne Pellerin (TotalEnergies)
Aude Simar (UCLouvain)

UCLouvain

Table of Contents

Abstract

Hexahedral meshes are used in engineering and computer graphics to describe complex geometric shapes by subdividing them into cube-like cells. Hexahedral meshes are widely considered advantageous over tetrahedral meshes, in terms of efficiency or their ability to align elements to relevant geometric features. Nonetheless, they have proven difficult to generate automatically for the wide range of geometric models used in industrial applications.

This thesis uses combinatorial and topological techniques to answer long-standing theoretical questions pertaining to the generation of hexahedral meshes. Namely, search algorithms exploring the space of possible topological meshes are used to find small hexahedral meshes with a given boundary, typically less than 70 hexahedra in the entire mesh. The special case of topological balls is treated separately, using shellings to more efficiently construct hexahedral meshes. These algorithms are fast enough to compute hexahedral meshes for all quadrangulated spheres with up to 20 faces. This yields an explicit construction showing that any quadrangulated sphere with n faces can be filled by a topological mesh containing up to $78n$ hexahedra. This new bound improves previous results requiring up to $5396n$ hexahedra. Furthermore, the indirect generation of hex-dominant meshes by combining tetrahedra into hexahedra is shown to be computationally intractable, justifying the existing use of heuristics for this problem.

These algorithms are also used in the construction of the first geometric meshes with planar faces for two difficult test cases for hexahedral mesh generation: the 8-quadrangle tetragonal trapezohedron, and a 16-quadrangle polyhedron known as Schneiders' pyramid.

Acknowledgements

Merci à Jean-François qui m'a mené sur ce chemin plein de pyramides et autres matroïdes,

Merci à Jeanne pour toutes les conversations et tous ses conseils bienveillants,

Merci à tous les amis eulériens avec qui j'ai passé ces quelques années, Pierre-Alexandre et ses petites croix, Ruiyang toujours plein de gaieté malgré ses domaines malencontreusement décomposés, Amaury qui nous a guidé aux travers ces sinueux sentiers, Céléstin et ses milliards de tétraèdres, Maxence qui nous a aidé à traverser de nombreuses couches limites, Arthur qui s'est si bien adapté à la métrique canadienne qu'il s'y est perdu, et tous les autres habitants plus ou moins incongrus de notre bâtiment bien-aimé. Bonne chance à Alexandre et Jovana dont la quête pour les séparatrices a rendu inséparables. Thank you Christos, who recently, squarely, left us for newer horizons.

蕊丽，你已经回国，我祝你好运研究海洋的奇迹。
Sankou伊伊，跟你花时间很开心，我一直很期待跟你看电影啊，聊个天啊，吃个饭啊，但我最期待的事就是你自己的论文答辩，你自己的成功。

Chapter 1

Introduction

Meshes are the means by which computer systems represent and manipulate geometric objects (Figure 1). Their use is ubiquitous in computer graphics [Baumgart, 1972; Catmull, 1972; Lévy, 2001; Gotsman et al., 2003; Botsch et al., 2007; Sheffer et al., 2007; Zhang et al., 2010], biomedical engineering [Bourdin et al., 2007; Shepherd and Johnson, 2009; Marchandise et al., 2010; Sazonov and

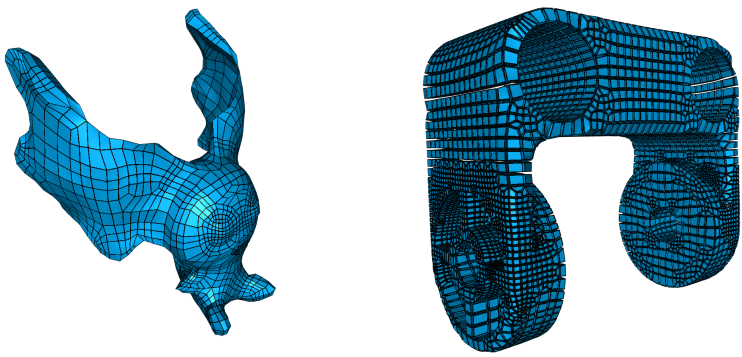


Figure 1: A surface mesh (left) and a volumetric, all-hexahedral mesh (right).

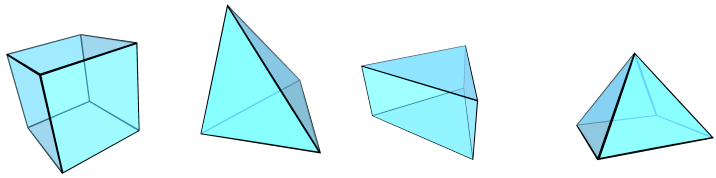


Figure 2: Different cell types used in volumetric meshes. From left to right: a hexahedron, a tetrahedron, a prism and a pyramid.

Nithiarasu, 2011; Bosnjak et al., 2024], and a wide range of other engineering disciplines through the finite-element method [Turner et al., 1956; Wördenweber, 1984; Caendish et al., 1985; Ho-Le, 1988; Tam and Armstrong, 1991; Hughes et al., 2005; Bathe, 2006; Geuzaine and Remacle, 2009]. A mesh allows algorithmic processing of complex shapes by subdividing them into simple *elements*, either polygons or polyhedra [Bern and Plassmann, 2000]. Meshes can be classified according to the kinds of elements they contain. In the case of surface meshes, common element types are triangles and quadrangles. Volumetric meshes are often made up of solid tetrahedra, triangular prisms, square pyramids, and, the focus of this thesis, *hexahedra* (Figure 2). Hexahedra are 6-sided polyhedra bounded by quadrangular faces.

The manuals of software tools that use or generate meshes are rife with recommendations and requirements regarding the quality of the meshes, including ranges of acceptable angles in mesh elements, acceptable aspect ratios for quadrangles [ANSYS, 2021; Altair, 2024b], the location and amount of irregular vertices [Pixar, 2023], as well as restrictions on the proportion of non-quadrangular or non-hexahedral elements [Shimada, 2018]. Indeed, hexahedral meshes offer a variety of advantages over tetrahedral meshes: they allow better allocations of computational resources, can align elements to relevant geometric features [Shimada, 2011], and enable meshes with fewer elements for the same level of accuracy [Chawner et al., 2016] as well as faster processing [Remacle et al., 2016].

In spite of all these advantages, there remain major obstacles to the automatic generation of high-quality hexahedral meshes. A wide variety of approaches have been employed [Tautges et al., 1996; Schneiders, 1996; Murdoch et al., 1997; Nieser et al., 2011; Livesu et al., 2013; Schneiders, 2000; Gao et al., 2019; Li et al., 2012], but none have proven sufficiently robust and versatile as to eliminate the need for manual and semi-manual generation techniques [Zoccheddu et al., 2023; Altair, 2024a; Cubit, 2024]. This is in stark contrast with tetrahedral meshes which can be generated reliably by robust algorithms with strong theoretical guarantees on the quality of the elements [Shewchuk, 1998; Geuzaine and Remacle, 2009; Si, 2015].

It can be tempting to leverage these algorithms by converting a high-quality tetrahedral mesh into a hexahedral mesh. Such *indirect* approaches inherit the level of automation traditionally achieved by tet-meshing software, but fall short in other respects. These techniques are usually only able to generate *hex-dominant* meshes, mixing hexahedra with tetrahedra, prisms and pyramids [Meshkat and Talmor, 2000; Yamakawa and Shimada, 2003; Baudouin et al., 2014; Botella et al., 2016; Pellerin et al., 2018a].

When it comes to all-hexahedral mesh generation, the few theoretical results available primarily concern topological mesh generation, *i.e.* the generation of a mesh with a specific combinatorial boundary, without concerns for the geometric quality of the elements [Thurston, 1993; Mitchell, 1996; Eppstein, 1999a; Bern et al., 2002; Erickson, 2014]. A major source of motivation for this thesis is the case of Schneiders' pyramid [Schneiders, 1995]. It's long been established that it is possible to mesh this polyhedron with topological hexahedra, but known solutions contain many hexahedra of low quality [Yamakawa and Shimada, 2010; Xiang and Liu, 2018] (Figure 3). In particular, it is not clear whether the complex and irregular structure of these meshes is an inherent requirement to achieve the boundary connectivity, or if a much simpler solution has been overlooked. The situation is even more dire in the case of geometric hex-meshing: it is still open which quadrangular meshes can possibly be extended to geometric hexahedral meshes, and what

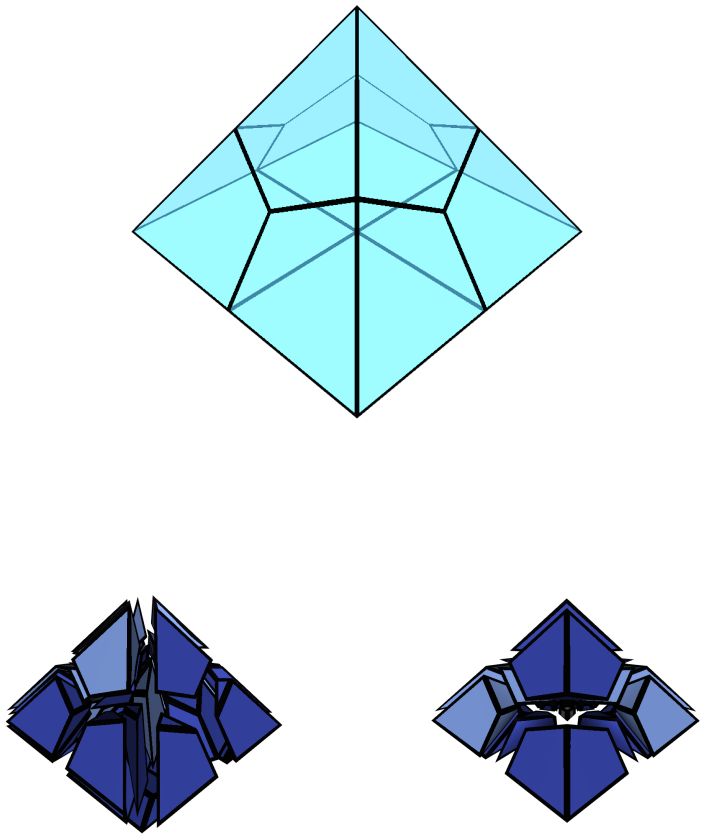


Figure 3: Schneiders' pyramid, a quadrangular mesh with 16 faces. Existing solution shown below (left: 88 hexahedra by Yamakawa and Shimada, right: 36 hexahedra by Xiang and Liu).

restrictions a fixed boundary mesh imposes on the quality of such meshes.

The goal of this work is to build upon those few results and provide a stronger basis for future robust algorithms. Instead of attempting to implement hex-meshing software with the current, limited state of understanding about the problem, this thesis answers or provides new tools to help answer these questions.

1.1 Contributions

As part of this research, several original contributions to the field of mesh generation were made, namely:

1. proving that combining tetrahedra into hexahedra, a technique used in a popular class of indirect hex-dominant meshing algorithms, is NP-Hard, which was previously suspected but unproven [Meshkat and Talmor, 2000] (Chapter 3);

2. an algorithm to exhaustively search for topological hexahedral meshes with a given boundary (Chapter 4), which can be used to locally modify hexahedral meshes (Section 4.2);

3. a more specialized algorithm relying on quad flips to compute topological hexahedral meshes bounded by spheres (Chapter 5), allowing us to construct hexahedral meshes for any quadrangulated sphere with up to 20 faces (Section 5.4) and give a concrete numerical bound on the number of hexahedra needed to mesh spheres of n quadrangles (Section 5.3);

4. focusing on the problem of Schneiders' pyramid, we compute the first *geometric* mesh of this pyramid, *i.e.* one made up of polyhedral cells with planar faces (Chapter 6).

Further related contributions were also published in scientific journals, though they are not the focus of this monograph:

1. An exhaustive enumeration of all 174 subdivisions of a hexahedron into tetrahedra, and a proof that only 171 of them are realizable geometrically, with applications to the generation of hex-dominant meshes [Pellerin et al., 2018b];

2. an algorithm applying combinatorial search to tetrahedral meshes, focusing on methods to efficiently prune the search space of possible triangulations [Marot et al., 2020];

3. a method to generate a single layer of hexahedra on the boundary of a domain, by formulating constraints describing the configurations of adjacent boundary vertices [Reberol et al., 2023].

All of these contributions involve either code to implement new algorithms, or the production of new datasets. These results are made freely available at `https://www.hextreme.eu/download`.

1.2 Publications

This thesis covers works that were previously published in the following articles:

- Kilian Verhetsel, Jeanne Pellerin, and Jean-François Remacle. 2019a. A 44-element mesh of Schneiders' pyramid: Bounding the difficulty of hex-meshing problems. *Computer-Aided Design.* `https://doi.org/10.1016/j.cad.2019.102735` (**Best paper award**.)

- Kilian Verhetsel, Jeanne Pellerin, and Jean-François Remacle. 2019b. Finding hexahedrizations for small quadrangulations of the sphere. *ACM Transactions on Graphics (TOG)* 38, 4 (jul 2019), 53. 0730-0301 `https://doi.org/10.1145/3306346.3323017`

Chapter 2

The Many Facets of Hexahedral Mesh Generation

Due to its industrial applications in a wide range of fields, hexahedral meshing has been studied under many different lenses. We give some context on the concepts of algebraic and computational topology that will be used throughout this work to describe new results,

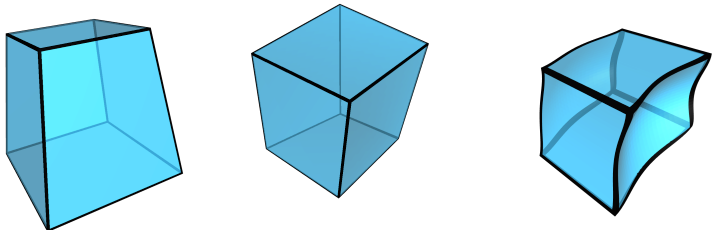


Figure 4: A geometric hexahedron with planar faces (left), a trilinear hexahedron with warped faces (center), and a topological hexahedron (right).

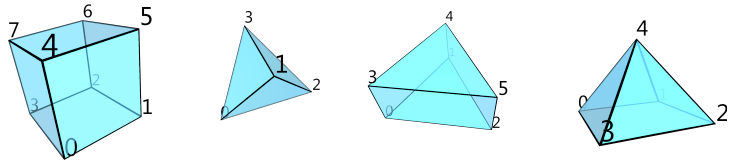


Figure 5: Example of a numbering convention for elements in a mesh.

before reviewing the most common types of hex-meshing methods.

2.1 Background and Definitions

2.1.1 Topological and Geometric Meshes

The main objects discussed throughout this thesis are *meshes*, *volumetric* and *hexahedral* meshes in particular. The properties of meshes are studied through two interrelated branches of mathematics: *topology* and *geometry*.

The topological properties of a mesh can be derived from its combinatorial structure, which we call a *topological* or *combinatorial mesh* (Figure 4). Given a finite set of vertices, we represent surface meshes as a set of polygons, only storing the incidence relations between them, *e.g.* by labeling the vertices and representing polygons as ordered lists of vertex labels. Similarly, we represent volumetric meshes by a set of *cells* (also known as *elements*). Each *cell* is a combinatorial polyhedron, bounded by polygonal facets. Such cells are encoded, for example, by picking a convention to order their vertices (Figure 5). The intersection of any two cells must be exactly one *face* of both cells: the empty set, a vertex, an edge, a polygonal *facet* or the cell itself.

This combinatorial object corresponds to a topological space X using the following construction: each k-dimensional face is represented by a copy of the unit ball $\left\{\mathbf{x} \in \mathbb{R}^k \mid ||\mathbf{x}||^2 \leq 1\right\}$. The k-skeleton $X^{(k)}$ is the disjoint union of these balls. The boundary of

each k-dimensional face (for $k > 1$) is the union of balls of dimension $(k-1)$, corresponding to the boundary facets, and attached to the skeleton $X^{(k-1)}$ using a gluing map from the unit sphere $\left\{\mathbf{x} \in \mathbb{R}^k \mid ||\mathbf{x}||^2 = 1\right\}$ to $X^{(k-1)}$. The topological space X is the union of its skeletons $X = \bigcup_{k=0}^{n} X^{(k)}$. Cell complexes are used extensively in algebraic and computational topology [Munkres, 1984; Edelsbrunner, 2001; Hatcher, 2002; Kaczynski et al., 2003].

A *geometric* mesh is one where each cell is a convex polyhedron, defined as the convex hull of its vertices $\mathbf{x}_1, \ldots, \mathbf{x}_n$:

$$\left\{ \sum_{i=1}^{n} \lambda_i \mathbf{x}_i \;\middle|\; \sum_{i=1}^{n} \lambda_i = 1 \text{ and } \forall i. 0 \leq \lambda_i \leq 1 \right\}$$

Equivalently, convex polyhedra may be defined as the intersection of half-spaces delimited by the planes that contain their facets, *i.e.* they must have planar faces. An extensive discussion on the properties of convex polyhedra and higher-dimensional polytopes is found in [Ziegler, 1995].

A *geometric realization* of a topological mesh assigns coordinates to each vertex such that each cell is a convex polyhedron. A volumetric mesh is *realizable* if it has a geometric realization in $\mathbb{R}^3$.

For finite elements applications, a less restrictive type of geometric mesh is used: *trilinear* hexahedral meshes [Knupp, 1990]. Given coordinates in $\mathbb{R}^3$ for each vertex, a hexahedron whose vertices are $(\mathbf{x}_{000}, \mathbf{x}_{001}, \mathbf{x}_{011}, \mathbf{x}_{010}, \mathbf{x}_{100}, \mathbf{x}_{101}, \mathbf{x}_{111}, \mathbf{x}_{110})$ is defined as the image of a unit cube under a trilinear map $f : [0,1]^3 \to \mathbb{R}^3$, obtained by taking the products of linear functions (denoted F_0 and F_1), weighted by the vertices:

$$f(u,v,w) = \sum_{i=0}^{1} \sum_{j=0}^{1} \sum_{k=0}^{1} F_i(u) F_j(v) F_k(w) \mathbf{x}_{ijk}$$

where

$$F_0(t) = 1 - t$$
$$F_1(t) = t$$

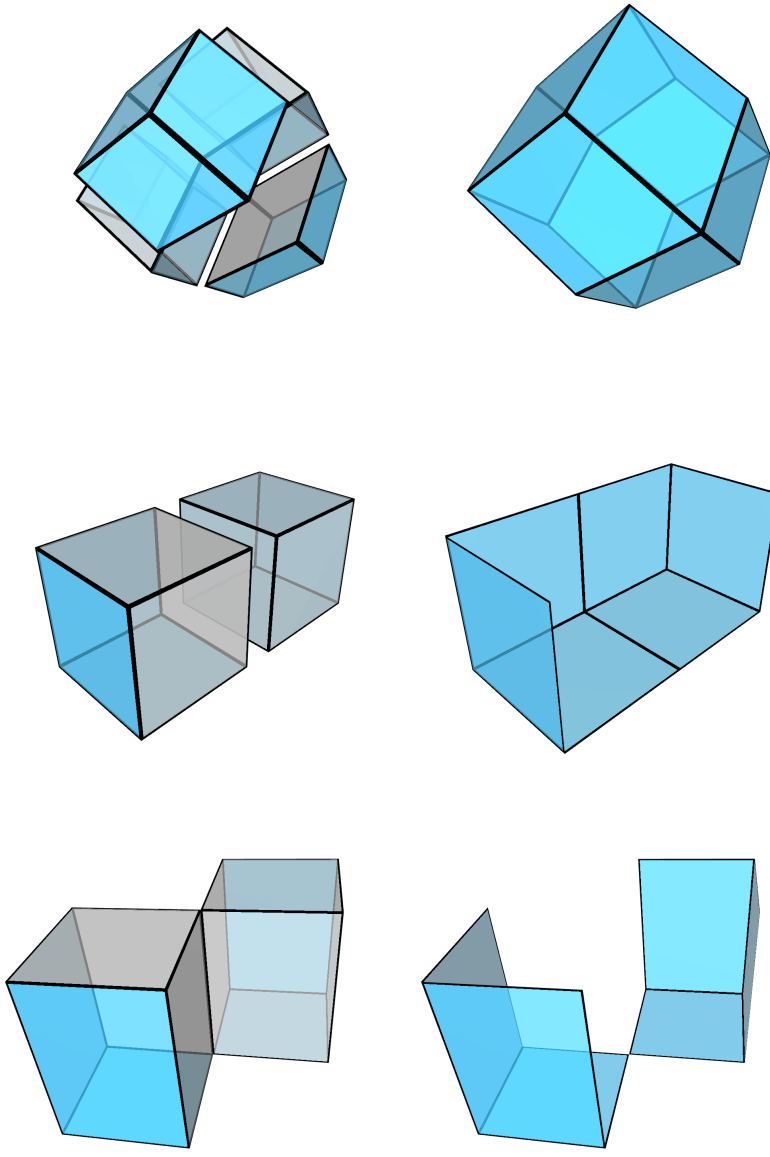


Figure 6: (Left) stars and (right) links of vertices in a manifold (top), on the boundary of a manifold (middle) or in a non-manifold (bottom).

A trilinear mesh is *valid* if every point of the physical domain is the image of precisely one point under the trilinear maps that define the mesh. This corresponds to the Jacobian determinant of f being strictly positive everywhere for every hexahedron (assuming a convention on the orientation of hexahedral elements).

2.1.2 Manifolds

Cell complexes can describe a wide range of spaces. For meshing purposes, the domains that we seek to mesh are usually *manifolds*, or manifolds with boundary. An n-manifold is a space where each vertex v has a neighborhood homeomorphic to an open subset of $\mathbb{R}^n$.

In combinatorial terms, manifolds are characterized in terms of the *links* and *stars* of faces in a cell complex. The *star* of a face f, $\mathrm{St}(f)$ is the set of all cells that contain f, and all of their faces. The *link* of f, $\mathrm{Lk}(f)$, is the set of faces in $\mathrm{St}(f)$ that do not contain f. A combinatorial n-manifold with boundary (for $n \leq 3$) is a topological mesh such that the link of every face of dimension k is homeomorphic to a sphere S^{n-k} (for interior faces) or a ball B^{n-k} (for boundary faces) [Edelsbrunner, 2001; Rourke and Sanderson, 2012]. In particular, the link of each vertex is homeomorphic to S^{n-1} for interior vertices or B^{n-1} for boundary vertices (Figure 6).

2.1.3 Dual Meshes

The faces of a topological mesh are partially ordered by a relation $a \preceq b \Leftrightarrow a$ is a face of b. Reinterpreting a mesh by inverting this relation yields its *dual mesh* [Basak, 2010]. Given a surface mesh $\mathcal{M}$, its dual has one vertex for each polygon in $\mathcal{M}$, and one dual edge connecting any two dual vertices corresponding to primal polygons that share an edge. Each vertex of the primal mesh $\mathcal{M}$ then corresponds to a polygonal cell of the new dual mesh. The dual of a volumetric mesh $\mathcal{M}$ is also a volumetric mesh, each vertex of which corresponds to a polyhedral cell of the original mesh. Dual edges

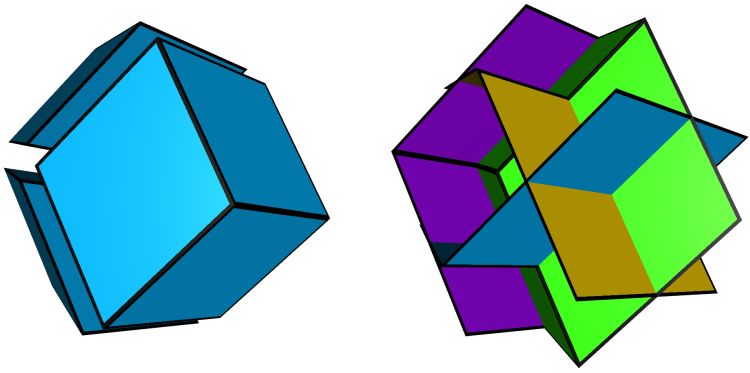


Figure 7: A hexahedral mesh (left) and the corresponding arrangement of dual sheets (right).

connect the vertices corresponding to primal cells that share a facet and dual facets correspond to primal edges.

For quadrangular and hexahedral mesh, we add structure to the dual mesh by grouping the dual edges or polygonal facets into curves or sheets respectively. In a quadrangular mesh, a *dual curve* groups all edges corresponding to opposite edges of a combinatorial quadrangle. In a hexahedral mesh, any two dual polygonal facets that correspond to parallel primal edges of a combinatorial cube are grouped together in a *dual sheet* (Figure 7). The boundary of a dual sheet is either empty, or one or more dual curves of the boundary quadrangular mesh.

2.2 What Domains Are Hex-Meshable

Given a quadrangulation, even with a small number of faces, such as Schneiders' pyramid, it is often difficult to manually find a hexahedral mesh that fills its interior. This prompted the question of characterizing the quadrangular meshes that can be extended to hexahedral meshes.

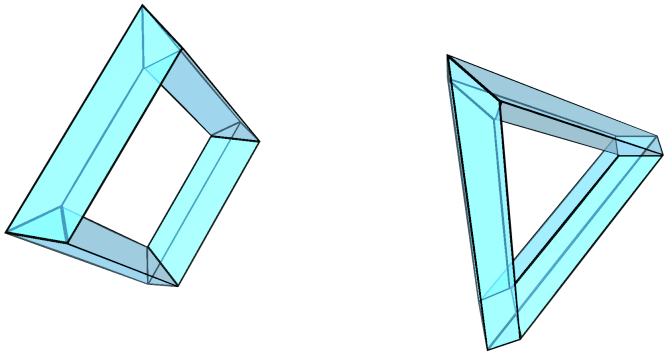


Figure 8: A quadrangulation that cannot be fileld with hexahedra (left) because a cross section of the torus contains a triangle, and one that can be meshed using three hexahedra (right).

For ball domains. Thurston [1993] and Mitchell [1996] independently showed that a topological ball bounded by a quadrangulated sphere can be meshed with hexahedra if and only if the number of quadrangles on the boundary, n, is even. In Mitchell's proof, an arrangement of surfaces bounded by the dual arrangement of curves of the input quadrangulation is first constructed. For curves with an even number of self-intersections (including curves with no self-intersections), a disk is constructed inside the domain and a regular homotopy between a circle and the curve can be used to create a manifold bounded by that curve. Curves with an odd number of self-intersections are paired up arbitrarily. For each pair, a manifold bounded by the two curves is constructed by computing a regular homotopy between the two of them. This arrangement is not in general the dual of a hexahedral mesh, so the next step of the construction is to add new surfaces completely inside the ball until all connectivity requirements of a hexahedral mesh are met.

A linear-complexity solution. The construction of Mitchell requires up to $\Omega(n^2)$ hexahedra where n is the number quadrangles.

Eppstein showed this was the case and proposed a different construction which guarantees the use of $O(n)$ hexahedra [Eppstein, 1999a]. Eppstein's algorithm first subdivides each quadrangle into two triangles, so that a tetrahedral mesh of the interior can be computed. After subdividing each tetrahedron into four hexahedra, a hexahedral mesh is obtained. However, its boundary does not match the initial input quadrangulation. This is solved by inserting *buffer cells*: for each quadrangle, add a cube, and glue one of its face to the original quadrangle; then, subdivide the opposite face into six quadrangles. The six new quadrangles are matched with those obtained from subdividing the original quadrangles during the previous step. The four remaining sides of the buffer cells are carefully subdivided into either two or three quadrangles, so that each buffer cell is bounded by an even number of quadrangles. Mitchell's proof can then be invoked to show that each buffer cell can be subdivided into a finite number of hexahedra.

Generalization to other inputs. Generalizing the previous results, Erickson [2014] gives necessary and sufficient conditions for the existence of a hexahedral mesh of a domain Ω bounded by a quadrangulation Q. The requirement is that every null-homologous subgraph of the input quadrangulation (*i.e.* every subgraph which bounds an embedded surface of Ω) contain an even number of edges (Figure 8). The construction of Erickson is similar to the one proposed by Eppstein, and also starts by computing a tetrahedral mesh of the domain, subdividing it into a hexahedral mesh, and inserting buffer cells to get a complete mesh with the correct boundary. The last step is to subdivide the buffer cells of two different types (Figure 9) into hexahedra, which is again shown to be possible from Mitchell's proof. Neither Eppstein nor Erickson give an explicit construction of the hexahedral meshes of the buffer cells used in the algorithm.

A constructive solution. Carbonera and Shepherd [2010] give the first completely explicit construction. Their algorithm first adds hexahedra inside the domain, guaranteeing that the dual arrangement of the boundary of the remaining region contain no self-

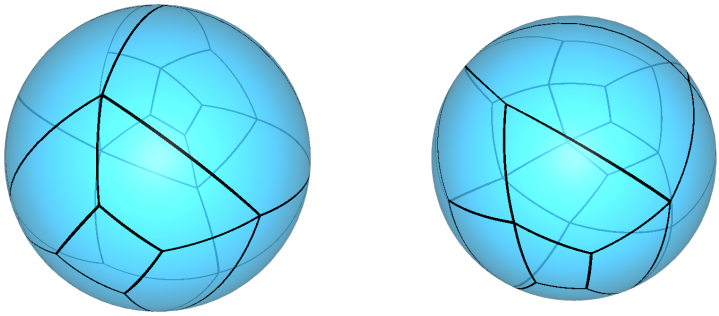


Figure 9: Hexahedrizations of these two quadrangulated spheres enable the construction of hexahedrizations for all other quadrangulated surface that have one.

intersecting curve. Buffer cells are then inserted to transition to a mesh where each quadrangle has been subdivided into four quadrangles. The rest of the domain is then filled using pyramids. A complete hexahedral mesh is obtained after subdividing the pyramids into hexahedra. Given a topological ball bounded by n quadrangles, their construction produces a mesh of 76 n hexahedra. This mesh is degenerate: it contains quadrangles sharing multiple edges and hexahedra sharing multiple faces. A combinatorially valid mesh can be obtained by further refining the mesh [Mitchell and Tautges, 1994]. This method, however, requires as many as 5396 n hexahedra to build a hexhahedral mesh bounded by quadrangulation of size n.

Geometric hex-meshing. It remains open whether all quadrangulations that admit a topological hexahedral mesh also admit a geometric hexahedral mesh. Bern et al. [2002] reduced the problem to finding geometric meshes for all convex polyhedra isomorphic to a warped bicuboid. Schwartz and Ziegler [2004] were nonetheless able to explicitly construct a geometric operator to change the parity of any hexahedral mesh. Their construction relies on building

4-dimensional cubical polytopes with prescribed surfaces in their dual manifolds.

2.3 How Hex-Meshes Are Constructed

Below is a brief summary of the most popular techniques used to generate hexahedral meshes. For a more extensive review of mesh generation techniques, see [Pietroni et al., 2023].

2.3.1 Topological Advancing Fronts

Whisker Weaving. Whisker Weaving has been proposed by Tautges et al. [1996]. The idea is to use a topological advancing front to construct the dual of a hexahedral mesh with a prescribed boundary. The algorithm initially assumes that the final mesh will contain one dual surface for each dual curve of the input quadrangulation. Hexahedra are created inside the domain by creating intersections between three of these sheets, until the entire domain is filled. To choose between the multiple possible operations, heuristics based on geometric information such as the dihedral angle of faces are used. These heuristics are often not enough to completely fill the domain.

Dual cycle elimination. Müller-Hannemann [1999] proposed a method based on *dual cycle eliminations*. At each step of the algorithm, one of the curves of the dual mesh is removed, matching this elimination as the insertion of a layer of hexahedra. The new boundary after removing this cycle bounds the part of the input domain which has not been meshed yet. This process is repeated until the boundary matches that of a single cube. This method succeeds for certain classes of input quadrangulations, but fails for the common cases where the dual contains self-intersecting curves. Kremer et al. [2014] extended this algorithm with heuristics to determine the elimination order of the dual cycles, taking into accounts geometric properties to handle concave objects.

2.3.2 Octree-based Hex-Meshing

Octrees-based methods are a type of automatic hex-meshing algorithm that convert the combinatorial dual of an octree into a hexahedral mesh [Schneiders, 2000; Maréchal, 2009; Gao et al., 2019]. The octree structure is first refined according to the local feature size. A set of fixed templates is used to transform size-transitions present in the octree into a valid hexahedral mesh. The choice of templates directly impacts the quality of the resulting mesh, so they must be designed carefully to avoid poorly shaped hexahedra [Tong et al., 2024].

A major challenge for this family of algorithms is to accurately capture features that are not aligned with the octree. Given a set of sharp features, octree-based methods can project vertices onto the sharp features so that they are preserved in the final mesh. This often requires a high level of refinement in order to capture small features, limiting the practical applicability of those techniques for complex inputs.

2.3.3 Frame fields and Hexahedral Mesh Generation

Frame fields assign three mutually orthogonal directions to each point in space. Lines where this orientation field vanishes are known as singularities. In meshing terms, these lines correspond to edges that are surrounded by a number of hexahedra different from four [Beben, 2020].

Methods based on frame field start by generating a smooth, boundary-aligned orientation field and attempt to extract a block-structured mesh whose singularities match that of the frame field [Nieser et al., 2011; Li et al., 2012; Liu et al., 2018]. Currently, however, there is no guarantee that the singularities in the generaed frame-fields correspond to that of a valid hexahedral mesh. Invalid configurations are generated for most non-trivial models. For example, there can be vertices where only two singularities meet, one of valence three and the second of valence five — which is not a valid configuration for an all-hexahedral mesh. Liu and Bommes [2023]

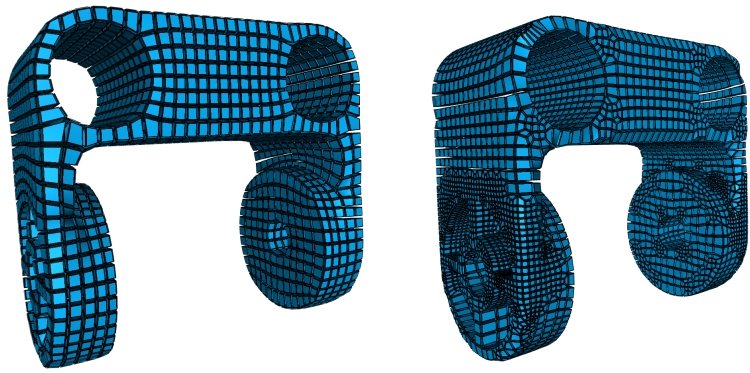


Figure 10: The same domain meshed using PolyCut (left) and an octree-based approach (right).

formulate the problem so as to avoid such invalid configurations, but there is still no guarantee the global orientation field corresponds to a valid hexahedral mesh.

2.3.4 Polycube-Based Methods

Polycubes are solids made up of a finite number of cubes glued face-to-face. This structure makes it easy to subdivide them into a regular arrangement of hexahedra. Polycube-based algorithms use such an arrangement to mesh an object by computing a mapping between it and a polycube [Gregson et al., 2011; Livesu et al., 2013]. If the distortion of this polycube mapping is low enough, then a hexahedral mesh of the polycube can be deformed into a valid mesh of the target model (Figure 10).

Lin et al. [2008] create such polycubes by first segmenting the input model into different parts corresponding to features of the input domain. Each part is then approximated by a coarse polycube chosen from a set of basic primitives. This coarse representation of the model often results in a high distortion mapping, making it

unsuitable for meshing applications.

Gregson et al. [2011] instead deform the input model, so as to align all boundary faces to one of the six axis-aligned directions ($\pm x$, $\pm y$, $\pm z$). The resulting polycube is then meshed with a structured hexhadedral mesh which is mapped back onto the input shape. Polycut [Livesu et al., 2013] also uses a deformation-based approach, but improves the segmentation of the input into axis-aligned patches using a hill climbing algorithm. This leads to a mapping with less distortion while requiring fewer irregular vertices. To reduce artifacts due to the high sensitivity of parametrization methods to the mesh embedding, Mandad et al. [2022] propose an intrinsic formulation. Protais et al. [2022] focus on the robust construction of very coarse meshes from such polycube mappings while still preserving relevant geometric features.

Even though these method allow the construction of high-quality, block-structured hexahedral meshes, the deformation process often results in inverted hexahedra. The regular connectivity of polycube meshes that does not contain singularities also makes it difficult to correctly represent certain boundary features.

2.3.5 Hex-Dominant Mesh Generation

For many models, the all-hexahedral methods mentioned so far fall short of producing a satisfactory hexahedral mesh. Hex-dominant are a more robust alternative thanks to the inclusion of non-hexahedral elements, while still filling most of the volume with well-shaped hexahedra.

H-Morph [Owen, 2001] computes a hex-dominant with a prescribed quadrangular mesh as its boundary. The input is a tetrahedral mesh constrained so that its boundary is a subdivision of the target quad mesh. The algorithm traverses this tetrahedral mesh in an advancing front fashion, applying local transformations to rearrange tetrahedra until they can be combined into hexahedra.

More recent hex-dominant meshing techniques directly create a tetrahedral mesh with its vertices placed so as to facilitate re-

combination. Such a distribution of vertices is obtained by packing rectangular cells [Yamakawa and Shimada, 2003], using the L_p Centroidal Voronoi Tesselation [Botella et al., 2016; Lévy and Liu, 2010], or frontal point insertion [Baudouin et al., 2014]. The tetrahedra are combined into hexahedra by searching for occurrences of the triangulations of a hexahedron [Pellerin et al., 2018b]. This search may rely on an explicit set of templates [Botella et al., 2016; Yamakawa and Shimada, 2003], or by traversing the mesh to look for groups of 12 edges whose connectivity match that of a cube [Pellerin et al., 2018a]. Ray et al. [2018] use an orientation field to guide the construction of a hex-dominant mesh, extracting the hexahedra using standard methods [Nieser et al., 2011]. A quad-dominant mesh of the boundary is computed ahead of time, and hexahedra that self-intersect or intersect the boundary are excluded. The final hex-dominant mesh is obtained by filling the leftover space using a Delaunay triangulation.

The hex-dominant meshes produced by these methods are in general non-conforming, as they may contain hexahedra that share a quadrangular face with two tetrahedra. Non-conformities are usually resolved as a post-processing step. Alternatively, Gao et al. [2017b] generate a conforming mesh directly by allowing a small number of cells with arbitrary polyhedra, in addition to hexahedra, tetrahedra, prisms and pyramids.

2.4 Evaluating and Improving Meshes

A mesh, in order to be useful for any type of physical simulation, needs to be constituted exclusively of valid elements. Validity of a hexahedral element can be tested by using the 27 coefficients of the Bézier decomposition of the trilinear map that defines it. These provide bounds for the Jacobian of the transform. If these bounds are not sufficiently tight to determine whether the element is valid, the hexahedron is recursively subdivided into 8 smaller hexahedra until an unambiguous result is obtained [Johnen et al., 2017]. Marschner et al. [2020] reformulate hexahedron validity as an op-

timization problem, solving it using semidefinite programming, to more efficiently find points where a hexahedron is not locally injective.

These tests are still relatively expensive so simpler conditions are often used as a proxy for the validity of hexahedra. Most commonly, the Jacobian at the eight corners of the hexahedron is tested by checking the orientation of the corresponding corner tetrahedra [Livesu et al., 2015; Knupp, 2001]. It is necessary, but not sufficient, that those eight values be positive for the hexahedron to be valid. Other tests based on the orientations of between 8 and 64 tetrahedra formed by the 8 vertices of the hexahedron have been studied empirically by Ushakova [2011], but none offer a necessary and sufficient condition.

In practice it is not enough for a mesh to be valid in order to serve a practical purpose: poorly-shaped yet valid elements may still yield non-physical results in a simulation. No single value fully captures whether an element is suitable for those simulations, and instead a wide range of metrics are used to evaluate meshes. These metrics include the scaled Jacobian, the ratio between the shortest and longest edges, the solid angles of the corners, measures of the planarity of faces, etc. [Gao et al., 2017a; Motooka et al., 2011].

There are currently no hex-meshing algorithms that guarantee the generation of a valid mesh containing only well-shaped elements. As such, it is common for hexahedral meshing pipelines to include so-called *untangling* and *smoothing* steps to improve the mesh. During untangling, vertices are moved until all elements are valid. The formulation of Marschner et al. [2020] using optimization techniques can also be used to repair hexahedral meshes.

Chapter 3

The Complexity of Indirect Hex-Dominant Meshing

The limitations of tooling used to generate all-hexahedral meshes, combined with the existence of established, industrial software to generate tetrahedral meshes, makes the following idea tempting: first generate a tetrahedral mesh of the domain, then combine the tetrahedra within it to form hexahedra. This is known as an *indirect* approach to hexahedral mesh generation. In general, it may not be possible or practical to combine all tetrahedra of a given mesh into

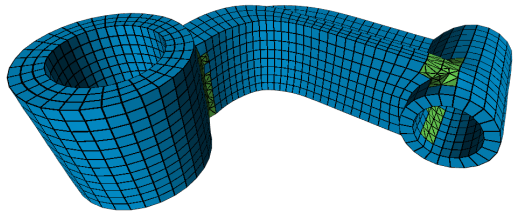


Figure 11: A hex-dominant mesh. Most of the volume is filled with hexahedra, but it also contains tetrahedra.

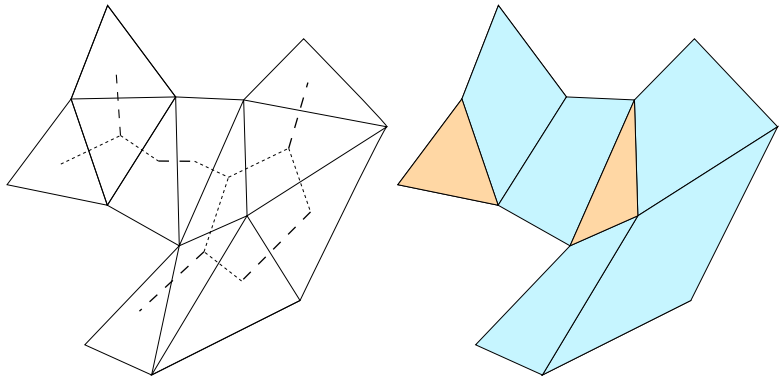


Figure 12: (left) a triangle mesh and its dual graph (dashed edges), recombined into a quad-dominant mesh (right) that corresponds to an edge matching (indicated by bold edges).

hexahedra, so the resulting mesh will still contain leftover tetrahedra as well as other elements such as prisms and pyramids. Such a mesh is known as *hex-dominant* (Figure 11).

However, even in the cases where all tetrahedra can be recombined to form an all hexahedral mesh, existing techniques do not guarantee they will generate such a mesh. Instead of computing the optimal recombined mesh in terms of the number of hexahedra or their quality, indirect hex-dominant meshing is done using approximate methods. This chapter demonstrates that maximizing the number of hexahedra in the recombined mesh is NP-hard, justifying the established use of heuristics to approximate solutions to the problem.

3.1 Problem Statement

Different measures can be used to quantify the quality of a recombined mesh to account for the number of cells, their types (tetrahedra, hexahedra, prisms or pyramids) and their geometric quality. The exact objective function used in practice is often application-

dependent. Throughout this chapter, we will specifically consider the following formulation: given a tetrahedral mesh T and a quality threshold q, find the maximum number n of hexahedra with a minimum scaled Jacobian greater than or equal to q that can be obtained by recombining tetrahedra in T. Any two selected hexahedra must be *compatible*, *i.e.* their intersection must be empty, exactly one vertex, exactly one edge, or exactly one quadrangular face. We allow non-conformities in the mesh, allowing one hexahedron to be adjacent to two tetrahedra. Typically, such invalid connectivity is addressed after recombination as part of post-processing steps, and may be penalized in the objective function.

Consider the analogous problem pertaining to surface meshes: given a triangulation T of a surface, find the maximum number n of quadrangles that can be obtained by combining triangles in the original mesh. This question can be answered in polynomial time by considering the dual graph G of T. The nodes of G are the triangles in T and an arc in G connects two nodes means that the corresponding triangles are adjacent in T. n corresponds precisely to a maximum matching of G (a maximum cardinality set of arcs M in G such that no two arcs in M share one node), hence the problem for surface meshes is solvable in polynomial time (Figure 12) [Remacle et al., 2012; Edmonds, 1965]. Even with a threshold on the quality of combined elements, a solution can still be found in polynomial time by removing from G the arcs that correspond to quadrangles of inadequate quality.

Despite the similarities, the recombination problem for volumetric tetrahedral meshes is more challenging: whilst a quadrangle can only be obtained by merging two triangles that share an edge, there are 174 different patterns made up of tetrahedra that can be recombined into one hexahedron [Pellerin et al., 2018b]. This challenge combined with the additional degree of freedom in volumetric meshes enables us to simulate other NP-Hard problems.

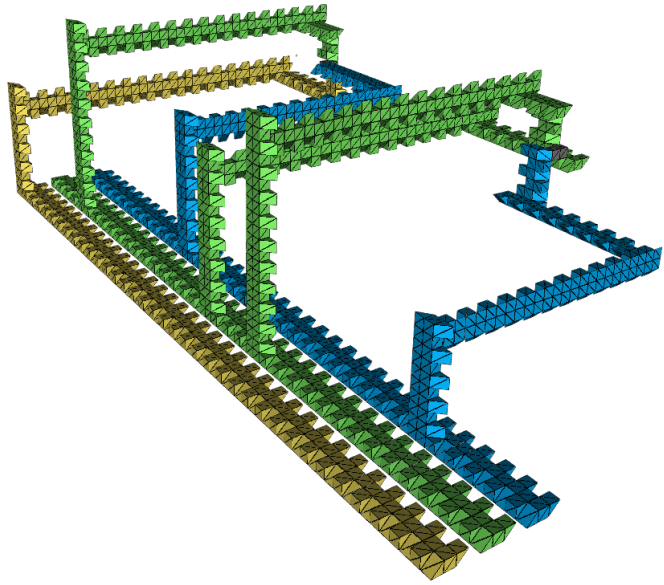


Figure 13: A tetrahedral mesh used to encode the boolean formula $(x \vee y \vee \neg z) \wedge (\neg y \vee z)$. Colors represent the three variables.

3.2 Reduction from 3-SAT

To show that the problem of recombining tetrahedra into hexahedra is NP-hard, we use a reduction from a well-known NP-Complete problem, the Boolean satisfiability problem [Karp, 1972]. Specifically, consider a 3-SAT instance $\mathcal{C}$ written as:

$$\bigwedge_{c \in \mathcal{C}} (c_0 \vee c_1 \vee c_2)$$

A 3-SAT instance is a set of *clauses* where each clause contains 3 *literals*. Each literal is either a boolean variable x_i or its negation $\neg x_i$. $\mathcal{C}$ is *satisfiable* if and only if it is possible to assign either true or false to each variable such that every clause contains at least one literal equal to true.

From any 3-SAT instance $\mathcal{C}$, we compute a tetrahedral mesh T, an integer n and a quality threshold on the scaled Jacobian of the recombined hexahedra q. X is satisfiable if and only if the tetrahedra of T can be recombined into at least n hexahedra with a minimum scaled Jacobian greater than or equal to q.

We construct this mesh in polynomial time. This shows that this formulation of the recombination problem as a decision problem is NP-Hard, *i.e.* as hard as boolean satisfiability and other problems in NP (section 3.2.4). Following typical approaches to formulate such reductions [Aloupis et al., 2015], the construction relies on different *gadgets* used to simulate components of the 3-SAT instance as a tetrahedral mesh:

- each variable is represented as a repeating pattern of tetrahedra that admits two optimal combinations into hexahedra, each corresponding to a value assigned to the variable (section 3.2.1);

- each clause is represented by a clump of tetrahedra that can be combined into one hexahedron in three different ways (section 3.2.2), one for each literal in the clause;

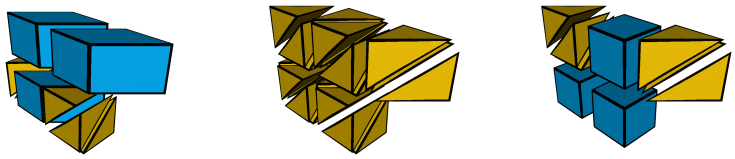


Figure 14: Gadget to encode variables (center), which can be combined into 3 hexahedra in two ways, representing it being set to false (left) or true (right).

- a T-junction device is used to connect variables to clauses and avoid self-intersections in the mesh (section 3.2.3).

3.2.1 Encoding of Boolean Variables

Each variable is represented by stacking up layers of the gadget shown in Figure 14. Each layer can be combined into exactly three hexahedra optimally in two different ways, one to represent true being assigned to a variable, one to represent it being false. When two layers are stacked on top of one another, the optimal combination corresponds to consistently picking one value for the variable. Creating a hex-dominant mesh that mixes hexahedra corresponding to the two different variable states is only possible by using fewer hexahedra per layer or including low-quality hexahedra (Figure 15). A threshold of 0.6 on the minimum scaled Jacobian is used to preclude the latter. This same threshold will be used for all other gadgets we describe.

3.2.2 Encoding of Logical Clauses

Each clause $x \vee y \vee z$ is represented by 10 tetrahedra that can be combined to form one hexahedron in three different ways (Figure 16).

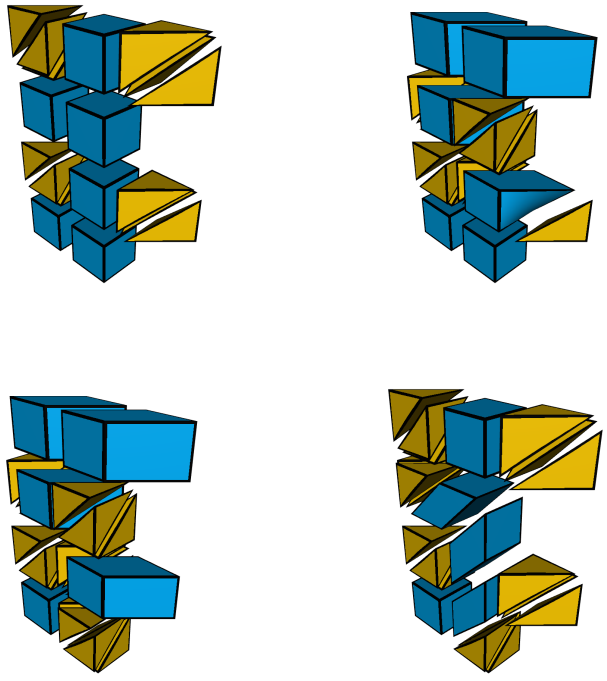


Figure 15: Two consecutive layers recombined optimally (top left), changing the state of a variable using low quality hexahedra (top right) or by skipping part of a layer (bottom left), and using a hexahedron spanning across both layers (bottom right).

Each of the three hexahedra represents selecting a literal (x, y, or z) whose value is true.

To ensure the selected literal is consistent with the variable gadget, the clause gadget must be connected to three variable gadgets, so that the only way to optimally combine both the clause gadget and the variable gadget is if the clause is satisfied.

These constraints are enforced by connecting three of the vertices from the last layer in the variable gadget to the clause. These three vertices are all part of a single quadrangular face in the hexahedra corresponding to the negation of the corresponding literal (*i.e.* the state where x is false if x is included in the clause, and the state where x is true if $\neg x$ is included instead). They are merged with three vertices of a single quadrangular face of the hexahedron used to represent the corresponding literal, while still ensuring that this hexahedron remains compatible with the two hexahedra corresponding to the other two literals.

3.2.3 T-junctions

The gadgets used to represent clauses need to be connected to the last layer of a variable gadget, but each variable needs to be connected to multiple clauses. To allow this, we use a third gadget to create a T-junction in the middle of the stack of layers used to represent variable.

The T-junction gadget is constructed by adding three tetrahedra to a stack of two layers of the variable gadget and attaching another layer in a perpendicular orientation (Figure 18). The resulting tetrahedral mesh admits two optimal recombination into 10-hexahedra, corresponding to the two states a variable can take.

3.2.4 Combining Tetrahedra into Hexahedra is NP-Hard

Any 3-SAT instance $\mathcal{C}$ can be represented by a tetrahedral mesh T using a polynomial number of the gadgets we described (Fig-

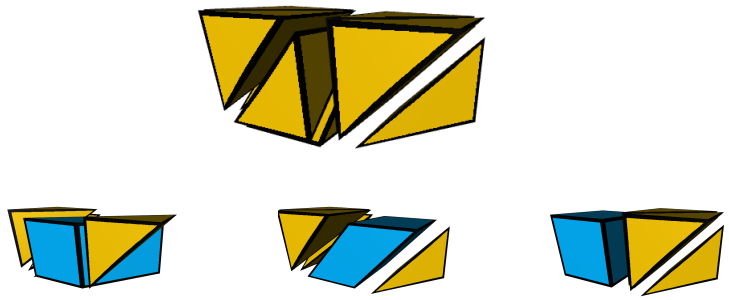


Figure 16: Gadget to encode clauses (top). There are three ways to combine these and form one hexahedron (bottom).

ures 14,16, and 18): each variable x_i is encoded by stacking $O(|\mathcal{C}|)$ layers of the variable gadget. One instance of the clause gadget is used for each clause. For each occurrence of x_i in a clause, two consecutive layers are replaced with a T-junction, and more layers are inserted to connect x_i to the clause. $\mathcal{C}$ is satisfiable if and only if T can be recombined into a hex-dominant mesh containing n hexahedra with a minimum scaled Jacobian of at least 0.6, where n is the sum of the number of hexahedra that each gadget can optimally be combined into.

By construction, any solution to the original 3-SAT instance corresponds to such a hex-dominant mesh. To verify that there is no way to combine T into n hexahedra of sufficient quality when the formula is not satisfiable, consider the incompatibility graph G. The nodes of G are the candidate hexahedra that can be obtained by combining tetrahedra from T. Any two nodes h_i and h_j of G are connected if and only if they are incompatible, either because the corresponding hexahedra geometrically overlap or because they only share part of face. The graph G can be partitioned into one sub-graph for each gadget and one sub-graph for each pair of directly connected gadgets, since no candidate hexahedra span across more than two gadgets.

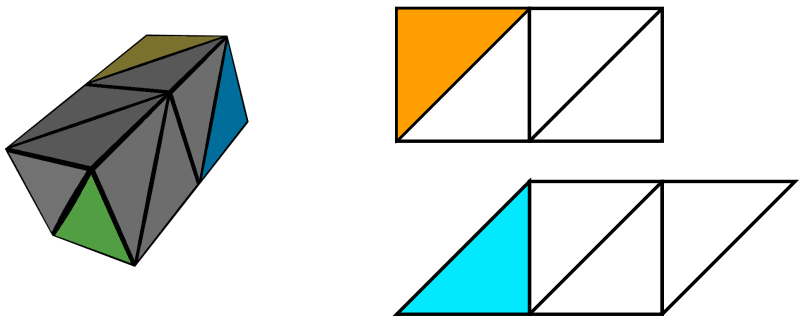


Figure 17: A clause (left) is connected to a variable by attaching a triangle representing one of the three literals (colored) to the corresponding variable gadget, merging it with one of two triangles (right), depending on the sign of the literal.

The maximum number of hexahedra in a hex-dominant mesh obtained from T is at most the sum of the sizes of the maximum *independent sets* of each sub-graph, an *independent set* of a graph being a set of nodes none of which are connected by an arc. This corresponds to ignoring the incompatibility constraints across sub-graphs. To verify that the optimal number of hexahedra can only be achieved if the 3-SAT instance $\mathcal{C}$ admits a solution, we consider the sub-graphs that correspond to all pairs of directly connected gadget G_1 and G_2. There are only a small number of cases used in the construction: the connection between consecutive layers of the variable gadget, the connection between layers and T-junctions, and the 6 different ways a clause can be connected to a variable gadget — corresponding to the 3 literals of each clause, and whether or not any of each of them appears with a negation.

Let G_{12} be the sub-graph of G that contains all candidate hex-

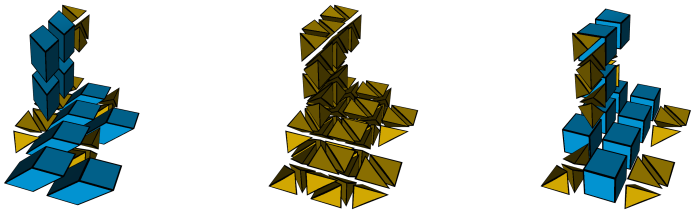


Figure 18: Gadget to encode branches (center). This submesh is inserted in between layers of the variable gadget and can be combined optimally into 10 hexahedra to either match the state of the variable gadget being false (left) or true (right).

ahedra that cross both gadgets. In all cases, we verify that any independent set that contains a hexahedron $h \in G_{12}$ is sub-optimal, so that a better local solution can always be obtained by instead recombining G_1 and G_2 optimally. Since the optimal number of hexahedra n assumes that all gadgets were combined optimally and the optimal hex-dominant mesh for each gadget encodes an assignment for the variables in $\mathcal{C}$, a hex-dominant mesh containing n hexahedra can only be obtained if $\mathcal{C}$ is satisfiable.

This shows our recombination problem is NP-Hard. Because it is in NP (*e.g.* one can construct the incompatibility graph G and compute its maximum independent set), it also follows that it is NP-Complete.

Chapter 4

Searching for Combinatorial Meshes

Given the computational cost of optimal recombination of tetrahedra into hexahedra shown in the previous chapter, these indirect approaches cannot be expected to output all-hexahedral meshes. Nonetheless, a hex-dominant mesh could still be subdivided into an

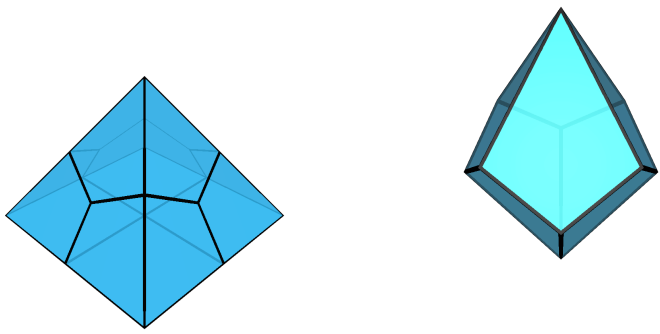


Figure 19: Left: Schneiders' pyramid. Right: The octogonal spindle.

all-hexahedral mesh. If most elements are already hexahedra, this could result in a mesh of suitable quality. One is then confronted with the problem of subdividing pyramids into hexahedra [Schneiders, 1995]. Indeed, there are exceedingly simple quad meshes which do admit hexahedral meshes but finding any solution is extremly difficult (Figure 19). This chapter introduces a new algorithm to search for a combinatorial hexahedral mesh with a given boundary [Verhetsel et al., 2019a]. By ensuring the search is exhaustive, we can either find the smallest hexahedral mesh with the required boundary (Section 4.1), or prove there are no meshes with fewer than a certain number of vertices or hexahedra (Section 4.1.4). Operating on a smaller cavity rather than the entire mesh, we're also able to compute smaller hexahedral meshes even if the input quadrangulation is too complex to find a minimal solution (Section 4.2).

4.1 Enumerating combinatorial hexahedral meshes

In this section, we describe an algorithm that lists all possible hexahedral meshes with a prescribed boundary. We use this algorithm to determine lower bounds for the number of vertices and hexahedra needed to mesh the octagonal spindle and Schneiders' pyramid. It is also the key to the local mesh simplification algorithm we propose in section 4.2.

When discussing the existence of hexahedral meshes or when enumerating the meshes bounded by a given quadrilateral mesh, we first ignore geometric issues and consider *combinatorial hexahedral meshes*. In a combinatorial hexahedral mesh, the hexahedra are represented as sequences of 8 integers, where distinct integers represent distinct vertices. A set of hexahedra defines a valid combinatorial mesh if all pairs of hexahedra are *compatible*: their intersection must be a shared combinatorial face (*i.e.* one of their 8 vertices, 12 edges, or 6 quadrangular facets) or be empty. Each quadrangle is also required to either be on the boundary (i.e. in exactly

one hexahedron), or in the interior of the mesh (i.e. in exactly two hexahedra).

4.1.1 Backtrack search algorithm

Given ∂H, a combinatorial quad-mesh of a sphere or a handlebody, $H_{\max}$ a maximum number of hexahedra, and $V_{\max}$ a maximum number of vertices, our algorithm lists all combinatorial hexahedral meshes H such that:

- the boundary of H is ∂H,
- the number of hexahedra $|H|$ is at most $H_{\max}$,
- the total number of vertices in H is at most $V_{\max}$.

This problem we are solving has similarities with problems commonly encountered in *constraint programming*: (i) efficiently filtering a large set of potential solutions and (ii) managing solutions having multiple equivalent representations. Our implementation adopts concepts and strategies from this field. For a more general study of these problems, we refer the reader to Rossi et al. [2006].

The hexahedra are built one at a time by choosing a sequence of 8 vertices. At each step, all possible candidates for one of the 8 vertices are considered and the algorithm branches for each possibility. Each branch corresponds to the addition of a vertex to the current hexahedron. When a complete solution is determined, or when the search fails (no available candidates to complete a hexahedron), the algorithm backtracks to the previous choice. This process is repeated until all possibilities have been explored. Algorithm 1 corresponds to the exploration of a search tree (Figure 20) where each branching node represents the choice of a vertex, and the leaves represent either solutions or failure points where the algorithm backtracks. The search tree has an exponential size in the maximum number of hexahedra in a solution. This high complexity is managed by pruning branches that cannot contain a solution and by using efficient implementations of all performed operations.

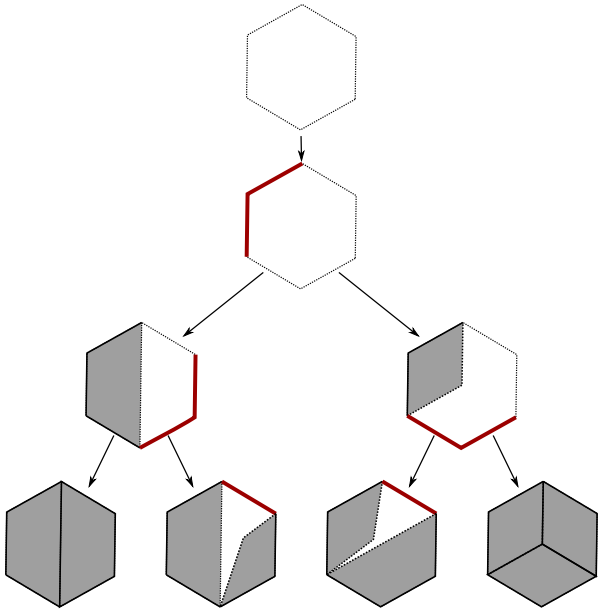


Figure 20: Searching all quadrilateral meshes of a polygon with up to one interior point. The search tree leaves are either valid solutions, or correspond to detected failure points where Algorithm 1 backtracks.

4.1.2 Search-space Reduction Strategies

In this section, we describe the key points of our implementation of Algorithm 1, all of which aim at reducing the search space explored by the algorithm:

- the order in which the hexahedra are constructed is crucial — we use an advancing-front strategy and start the construction of hexahedra from the boundary;
- an efficient filtering algorithm that eliminates candidate vertices that would create incompatible combinatorial hexahedra in the solution;

Algorithm 1 Recursive enumeration of the hexahedral meshes of the interior of ∂H.

Input: ∂H, the boundary; S, a partial solution; $C = (C_1, \ldots, C_8)$, the sets of candidate vertices for the current hexahedron.

```
def search (∂H, S, C):
  if the boundary of S is ∂H:
    Print solution S
  else if |S| = H_max:
    Backtrack
  else:
    C ← filter-candidates(∂H, S, C)
    if |C_1| = ··· = |C_8| = 1:
      S' ← S ∪ {(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8)}
      search (∂H, S', initialize-candidates(S'))
    else if min_{i∈{1,...,8}} |C_i| = 0:
      Backtrack
    else:
      i ← pick-hex-vertex(C)
      for each v ∈ C_i:
        C' ← C
        C'_i ← {v}
        search(∂H, S, C')
```

- a method to manage the high number of symmetries of this problem, *i.e.* different representations of the same meshes;
- the order in which the current hexahedron vertices are selected;
- a strategy to use topological invariants in order to filter out branches that do not contain any suitable solutions.

Advancing-front construction. While the hexahedra of a combinatorial mesh can be arbitrarily reordered, constructing them

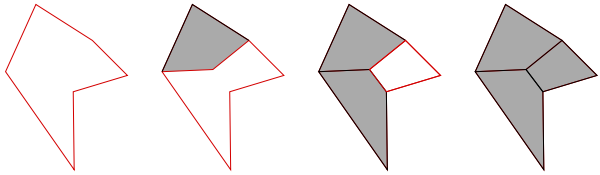


Figure 21: Each new element must share a face with the front of boundary faces (red).

in a specific order makes the algorithm significantly faster. We use a classical advancing front generation strategy and require the hexahedron under construction to share a face with a front of quadrangles. There are then only four vertices needed to complete a hexahedron. The quadrangle front is constituted of the interior facets that are in only one hexahedron, or of boundary facets that are in no hexahedra. At the root of the search tree, it is set to be the boundary ∂H. An interior facet is added to the front after its first appearance in the mesh. The facet is removed from the front when it is added to the partial solution. When the front becomes empty, the boundary of the solution matches the input (Figure 21).

Filtering out candidate vertices. For each of the eight vertices of the hexahedron under construction, we store a set of candidate vertices that could be part of the solution. Some of these vertices would make the current hexahedron incompatible with some already existing hexahedra. Therefore when initiating the construction of a hexahedron, or when adding a vertex to a hexahedron, vertices that cannot be added without creating incompatibilities between the current hexahedron and the already built hexahedra are filtered out. The following rules are used to eliminate candidates:

- the sets of edges, quadrangle diagonals, and interior diagonal of hexahedra are disjoint;
- no two hexahedra may share an interior diagonal;

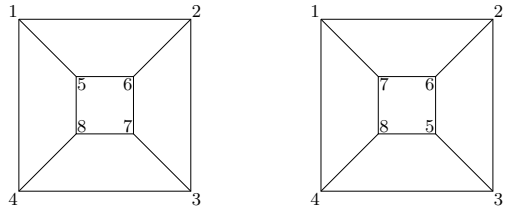


Figure 22: Two of the 4! ways to label the 4 interior vertices of this mesh.

- if one facet diagonal matches an existing quadrangle diagonal, so must the second one;
- all eight vertices must be different.

To enforce these rules, our implementation tracks three sets of vertices for each vertex v: the sets of vertices u such that (u, v) is an edge, the diagonal of a quadrangle, or the diagonal of a hexahedron. These sets are updated whenever a new hexahedron is created.

Because the execution time of the search algorithm blows up as the number of vertices increases, the number of vertices each set contains is always small, making them good candidates for being represented as bit-sets.

Symmetry breaking. Combinatorial meshes are characterized by their large number of symmetries, a major challenge when operating on combinatorial hexahedral meshes. Indeed, a combinatorial hexahedral mesh has many equivalent representations:

1. interior vertices can be relabelled (Figure 22) — for boundary vertices, the algorithm uses the same labels as the input;
2. the hexahedra of the solution can be constructed in a different order (Figure 23);
3. for a given hexahedron, written as an ordered sequence of 8 vertices, there are $1,680 = 8!/24$ ways to reorder these vertices while leaving then hexahedron unchanged (Figure 24).

Algorithm 2 Compute the sets candidate vertices

Input: S, a set of hexahedra.

Output: $C = (C_1, \ldots, C_8)$, the sets of candidate vertices for the next hexahedron.

def `initialize-candidates`(S):
  Let (v_1, v_2, v_3, v_4) be some quadrangle that needs to occur in the mesh.
  for each $i \in \{1, \ldots, 4\}$:
    $C_i \leftarrow \{v_i\}$
  for each $i \in \{1, \ldots, 4\}$:
    $C_{4+i} \leftarrow \{0, 1, \ldots, V_{\max} - 1\} \setminus \{v_1, v_2, v_3, v_4\}$
      $\setminus$`interior-diagonals`(v_i)
    for each $j \in \{1, \ldots, 4\}$:
      if $i \neq j$:
        $C_{4+i} \leftarrow C_{4+i} \setminus$
          `interior-diagonals`$(v_j) \setminus$ `edges`(v_j)
      if $i = j + 2 \mod 4$:
        $C_{4+i} \leftarrow C_{4+i} \setminus$ `quad-diagonals`(v_j)
  return $(C_1, \ldots, C_8)$

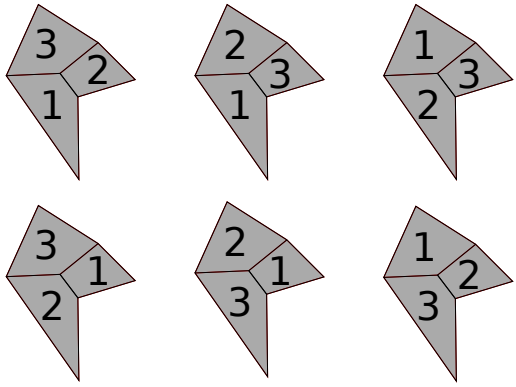


Figure 23: The 3! different ways to number the elements of a 3-element mesh.

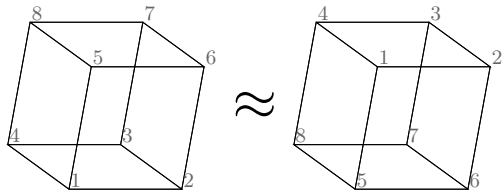


Figure 24: Two combinatorially equivalent hexahedra.

The advancing front strategy defines the order in which the solution hexahedra are constructed (symmetry 2). This also uniquely determines the order of vertices in a hexahedron (symmetry 3). To prevent the relabelling of interior vertices (symmetry 1), we add *value precedence* constraints to our problem. A solution H found by the algorithm can be written as an array of $8|H|$ integers, writing down the vertices of each hexahedron in the order in which they were constructed by the algorithm. In an array, x *precedes* y when the first occurrence of x is before the first occurrence of y. Enforcing a total precedence order on interior vertices, we guarantee that only one of their permutations is a solution.

Using topological properties during the search. The meshes that we are searching for, in addition to being valid combinatorial meshes, need to be meshes of the interior of the input surface. This implies topological requirements that must be met by any solution. These requirements are used not only to filter out branches of the search tree that do not contain any valid meshes, but also to reject combinatorial meshes of different 3-manifolds with the same boundary. Only topological invariants that can efficiently be computed are considered during the search, since no efficient algorithms to recognize 3-manifolds are known, even for the 3-sphere [Schleimer, 2011].

At any point, the partial mesh is maintained to be *oriented*: every quadrangle that appears in two hexahedra must have an opposite orientation in each of them. Whenever one of the faces of the hexahedron under construction is identified with an existing quadrangle because they share a diagonal, the other two vertices are

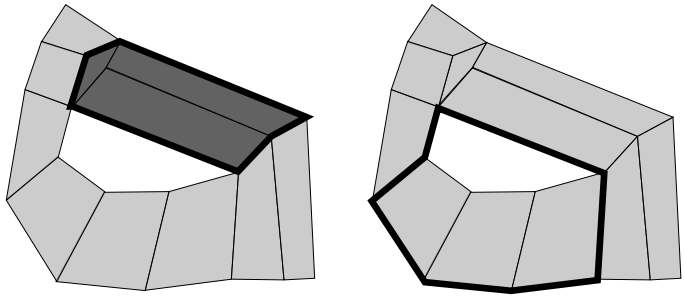


Figure 25: (left) A null-homologous 1-cycle in a quad-mesh, bounding 3 quadrangles; (right) a cycle which is not null-homologous, because it surrounds a hole within the mesh.

selected so that the two quadrangles have opposite orientation.

Meshes of a topological ball, or any hexahedral mesh that could be used as a subset of a mesh of the ball, have a bipartite graph [Eppstein, 1999a]. Let A and B be the two parts of the partition. The sets of candidate vertices are reduced so that edges between two elements of A or of B are never created. The partition is computed initially based on the input quadrangulation, and updated every time a hexahedron is added to the partial mesh.

The last topological property that we use is related to *homology groups*, in particular those computed over $\mathbb{Z}_2$. A detailed introduction to homology computations can be found in [Hatcher, 2002]. We define a k-chain as a set of elements of dimension k — a 1-chain is a set of edges, a 2-chain a set of quadrangles, and a 3-chain a set of hexahedra. The boundary of a k-chain C is the $(k-1)$-chain ∂C whose elements are the faces contained in an odd number of elements of C. A k-chain whose boundary is empty is referred to as a cycle. A k-cycle which is the boundary of a $(k+1)$-chain is called *null-homologous* (Figure 25).

For the domains that we consider, all 2-cycles are null-homologous. Since hexahedra have an even number of faces, any set of hexahedra is bounded by an even number of quadrangles. Therefore, if a

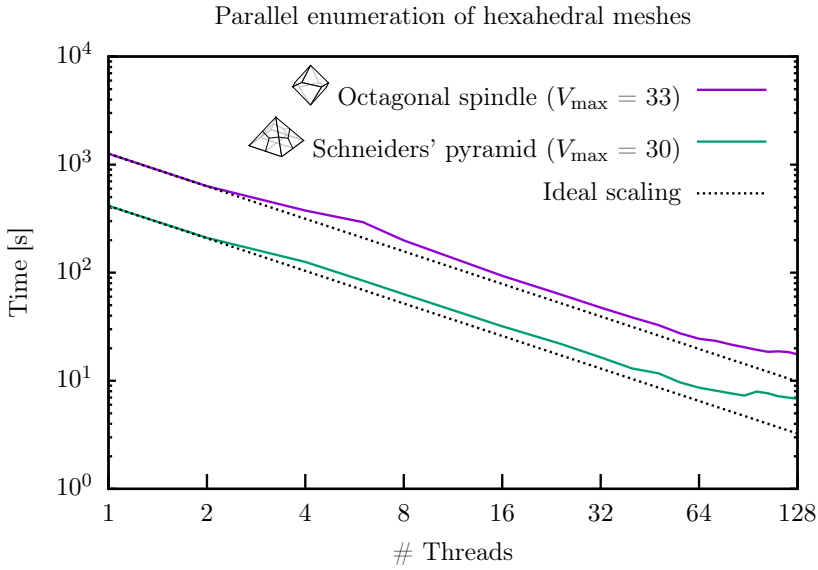


Figure 26: Time to explore a search tree in parallel on a machine with two AMD EPYC 7551 CPUs (32 cores each, 2 threads per core). Using 64 threads, the speed-up is of 48 for Schneiders' pyramid and 52 for the octagonal spindle.

2-cycle containing an odd number of quadrangles is ever created, the search can backtrack, as it will never be possible to create a set of hexahedra bounded by this cycle. We therefore compute a basis for the space of 2-cycles using Gaussian elimination, and verify that every element of the basis contains an even number of quadrangles.

The utility and correctness of these tests were verified by testing the implementation using different filtering rules, and analyzing the topology of the output meshes using Regina [Burton et al., 2023], a standard piece of software to analyze low-dimensional manifolds.

4.1.3 Parallel Search

The exploration of a search tree can be parallelized in a natural way by exploring different subtrees in parallel, making the algorithm much faster on parallel architectures (Figure 26). We use an approach similar to the embarrassingly parallel search of [Régin et al., 2013]. The main challenge to overcome is that some subtrees are multiple orders of magnitude larger than other ones without any possibility to determine which ones ahead of time.

We solve this issue by attributing many subtrees to each worker thread, so that all threads must on average perform the same amount of work. At the start of the search, the tree is explored in a breadth-first manner until a layer with at least kT subproblems is reached, where T is the number of threads and k a constant to control the amount of work per thread (we used $k = 4096$). The nodes of this layer are then explored in parallel by independent worker threads using Algorithm 1.

4.1.4 Lower Bounds for Hex-Meshing Problems

Using Algorithm 1, we computed lower bounds for the number of vertices and hexahedra required to mesh Schneiders' pyramid and the octagonal spindle (Figure 19). The algorithm is run multiple times, and we increment either $V_{\max}$ or $H_{\max}$ between each run. At each step, we verify that no solution was found by the algorithm. The time required to compute these bounds increases exponentially as the bounds become tighter (Figure 27).

The following bounds were proven in this manner:

1. Any hexahedral mesh of Schneiders' pyramid has at least 18 interior vertices and 17 hexahedra.

2. Any hexahedral mesh of the octagonal spindle has at least 29 interior vertices and 21 hexahedra.

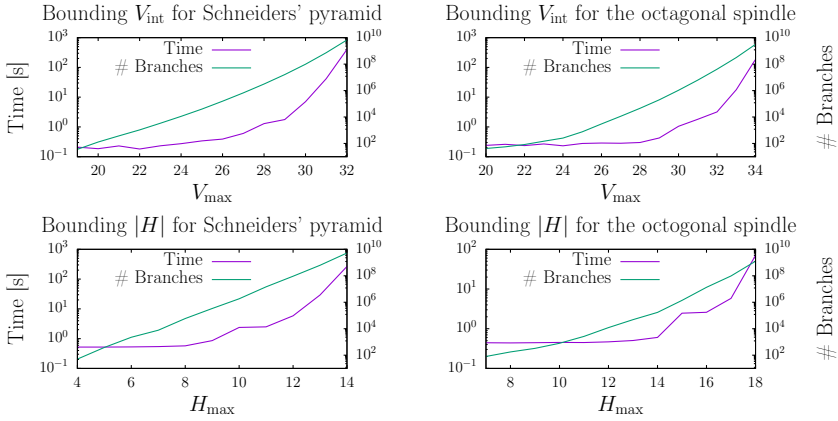


Figure 27: The time to prove lower bounds for the number of interior vertices V_{int} and the number of hexahedra H required to mesh a polyhedron increases exponentially. This is due to the exponential size of the search tree explored by the algorithm.

4.2 Simplifying Hexahedral Meshes

The algorithm described in the previous section can be used to find the smallest hexahedral mesh with a given boundary. In this section, we use this algorithm to improve upper bounds for meshing problems by locally simplifying an input mesh with the desired boundary. By simplification we mean decreasing the number of hexahedra (Figure 28). The realized operations may be viewed as a generalized form of cube flips [Bern et al., 2002] that substitute a set of hexahedra by another set without changing their boundary. However, instead of a finite set of transformations, the algorithm introduced in this section automatically determines them at execution time.

Globally minimizing the number of hexahedra in the mesh is a computationally demanding task. Our algorithm therefore selects a small subset of the mesh, or *cavity*, and focuses on modifying the connectivity of the mesh only within this cavity. Our hexahedral

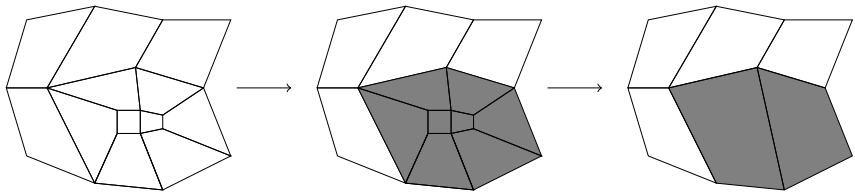


Figure 28: The number of elements in a mesh can be reduced by operating locally on a cavity.

mesh simplification algorithm is based on Algorithm 1. From a geometric hexahedral mesh it outputs a geometric hexahedral mesh whose boundary is strictly identical and which has fewer elements.

The mesh simplification procedure has three main steps:

- the selection of a cavity, the group of hexahedra to simplify, $\mathcal{C}$;
- finding the smallest hexahedral mesh $\mathcal{C}_{\text{min}}$ compatible with the cavity boundary $\partial\mathcal{C}$ and replacing the cavity with this smaller mesh;
- untangling the hexahedra to determine valid coordinates for the mesh vertices.

4.2.1 Cavity Selection

The cavity selection algorithm is a greedy algorithm that starts from a random element of the input hexahedral mesh (Algorithm 3). When the target size, in terms of number of hexahedra, is reached, this process stops. The choice of a target cavity size is a trade-off between the cost of finding the hexahedral meshes of the cavity and the likelihood that the mesh can be simplified by remeshing the cavity. Cavities with many hexahedra are more likely to accept smaller meshes, but the cost of finding the smallest hexahedral mesh $\mathcal{C}_{\text{min}}$ increases exponentially with the number of hexahedra in

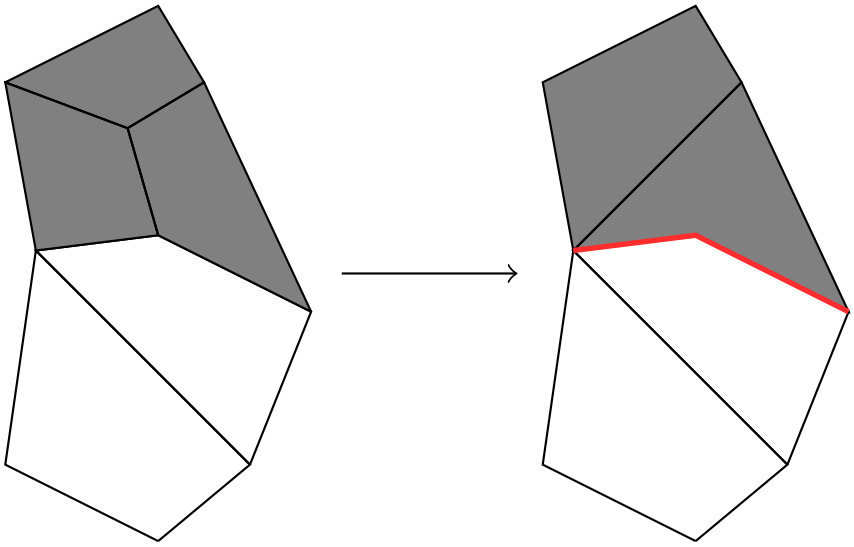


Figure 29: Replacing the cavity with a valid mesh sharing the same boundary still produces an invalid mesh by creating two quadrangles sharing two edges.

the cavity. In practice, we start by considering relatively small cavities containing up to 10 hexahedra, and increase this limit when no improvement is possible. We require the cavity to contain at least 4 interior vertices. Indeed, when there are no interior vertices (e.g. with a stack of hexahedra), it is not possible to remove any hexahedra. As the number of interior vertices increases, so does the likelihood that the cavity can be simplified.

4.2.2 Cavity Remeshing

To find a smaller mesh of the boundary of a cavity $\mathcal{C}$, we first solve the combinatorial problem, i.e. we find the smallest combinatorial hexahedral mesh of $\partial\mathcal{C}$, and then solve the geometric problem of finding valid coordinates for the modified mesh vertices.

Algorithm 3 Recursive enumeration of the hexahedral meshes of the interior of ∂H.

Input: H, the mesh; n, the size of the cavity.
Output: A cavity $\mathcal{C}$ of up to n elements.

```
def select-cavity(H, n):
  h ← a random element of H
  C ← {h}
  while |C| < n:
    h ← a random element of H \ C sharing
      a facet with a hexahedron in C
    C ← C ∪ {h}
  return C
```

The combinatorial problem of finding the smallest mesh of $\partial\mathcal{C}$ is a direct application of Algorithm 1 which enumerates all combinatorial meshes of a given surface. The maximum number of hexahedra H_{max} of the solution is set to a smaller value than $|\mathcal{C}|$. Changing the parity of a hexahedral mesh is known to be a difficult operation [Schwartz and Ziegler, 2004], so we set H_{max} to $|\mathcal{C}| - 2$. We also set the limit to the number of interior vertices V_{max} to one less than the number of interior vertices in $\mathcal{C}$ to accelerate the search.

There is a subtle but important difference between meshing a cavity in an existing mesh and meshing a stand-alone polyhedron: the hexahedra inside the cavity must be compatible with the other elements of the input mesh. An example where new elements from a cavity are not compatible with elements adjacent to the cavity is given in Figure 29. A 3-element cavity is replaced by 2 quadrangles, but one of these two quadrangles shares two edges with an existing element, which is an invalid configuration. To guarantee that the algorithm does not break the mesh validity, the data structures used to filter out inadequate vertex candidates (Section 4.1.2) are modified to take into account the hexahedra that are not part of the cavity.

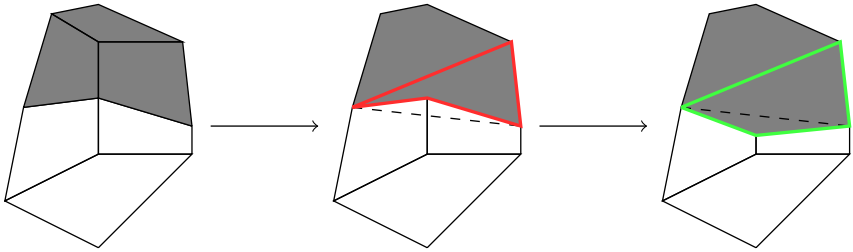


Figure 30: A valid change to the connectivity of the mesh can create a geometrically invalid mesh, which then requires the vertices to be moved.

4.2.3 Untangling

The previous step of the algorithm found a new connectivity for the mesh. The simplified mesh obtained by using this result is not valid in general because the interiors of hexahedra may intersect (Figure 30). To obtain a valid geometric mesh we use the untangling algorithm described by Toulorge et al. [2013]. The vertices are iteratively moved until all hexahedra in the mesh are valid. If the untangling fails, connectivity changes are undone. The validity of the final mesh is evaluated with the method proposed by Johnen et al. [2017].

Chapter 5

Flipping Towards Hexahedral Meshes

In Chapter 4, we introduced an algorithm to compute topological hexahedral meshes with a given boundary. This algorithm was used to improve topological meshes by operating on small cavities. On its own, this search method is not fast enough to compute solutions for cases where no good topological solution is available except if a very small solution exists (fewer than around 15 internal vertices).

To address this limitation, we introduce a faster search algorithm that only searches the space of so-called *shellable* meshes. The algorithm is based on *quad flips*, a set of operations to modify quadrilateral meshes and whose application can be interpreted as the construction of a hexahedron. Given a quadrangulated sphere Q, a hexahedral mesh bounded by Q is built by exploring the space of flipping operations that can be applied to Q. A solution is then obtained by finding a sequence of operations that transforms Q into the boundary of a cube (Section 5.1). When this search space is too large, the algorithm instead searches for a sequence of operations transforming Q into the boundary of any mesh within a library of pre-computed hexahedral meshes (Section 5.2).

This algorithm is used to construct combinatorial hexahedral meshes for all $54,943$ quadrangulations of the sphere with up to 20

quadrangles and which admit a hexahedral mesh. The computed hexahedral meshes contain at most 72 hexahedra.

The last contribution in this chapter is to significantly lower the upper bound on the number of hexahedra needed to mesh arbitrary domains. The construction of Erickson [2014] is made fully explicit by computing hexahedral meshes for its two quadrangulated templates. This proves that an arbitrary ball bounded by n quadrangles can be meshed using only 78 n hexahedra.

5.1 Finding Combinatorial Meshes Using Quad Flips

5.1.1 Overview

Given a quadrangulation of the sphere Q, we describe an algorithm to enumerate all hexahedral meshes bounded by Q and which can be constructed using quad flips (Algorithm 4). To force the algorithm to terminate, the search is limited to meshes with a maximum of $H_{\max}$ hexahedra and a maximum $V_{\max}$ of vertices. In Section 5.2, we then extend the approach to search for hexahedral meshes that are prohibitively large for an exhaustive search of this kind.

The algorithm detailed in this section does not search the entire space of hexahedral meshes, but only the space of so-called *shellable* meshes, which can be explored efficiently using quad flips (Section 5.1.2). This space is explored in its entirety by considering all possible sequences of quad flips that correspond to valid hexahedral meshes (Section 5.1.3). Because many different sequences of flipping operations represent the same mesh, most of this section focuses on how to account for the symmetries of the input quadrangulation, in order to avoid generating different sequences of quad flips corresponding to isomorphic hexahedral meshes (Section 5.1.4).

Algorithm 4 Enumerate shellable hexahedral meshes

def `search`$(Q, H, H_{\max}, V_{\max})$:
  if $H_{\max} = |H|$: **return**
  else if `visited-symmetric-counterpart`(H):
    return *(Section 5.1.4)*
  else if $Q \approx$ a cube:
    $h \leftarrow$ the hexahedron bounded by Q *(Section 5.1.3)*
    if `is-compatible`(H, h): `output-solution`$(H \cup \{h\})$
  for each quad flip F:
    $(Q', h) \leftarrow$ `perform-flip`(F, Q) *(Section 5.1.3)*
    if `num-vertices`$(H \cup \{h\}) > V_{\max}$: **continue**
    if `is-compatible`(H, h):
      `search`$(Q', H \cup \{h\}, H_{\max}, V_{\max})$

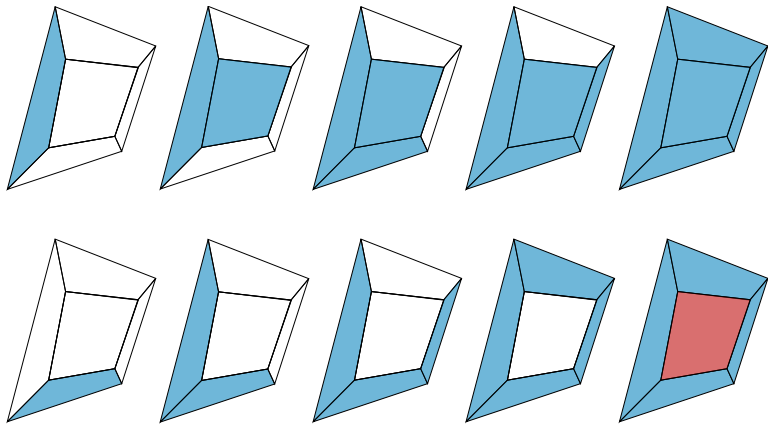


Figure 31: (top) a shelling of a quadrangulation; (bottom) not a shelling because a hole is present after inserting the first four quadrangles.

5.1.2 Shellability and Quad Flips

Our method only considers a specific class of meshes: *shellable hexahedral meshes*. *Shellability* is an important and useful combinatorial concept in the study of polytopes and cell complexes [Ziegler, 1995]. Slightly different notions of shellability are found in the literature. We use that of *pseudo-shellings* [Bern et al., 2002] or *topology-preserving shellings* [Müller-Hannemann, 1999]. This type of shelling is an ordering of the hexahedra $(H_1, H_2, \ldots, H_n)$ of a hexahedral mesh such that any prefix $\bigcup_{0 \leq i < k} H_i$ is homeomorphic to a ball (Figure 31).

This definition implies that any hexahedron H_k must intersect the union of the previous hexahedra in one of six possible patterns. Gluing a hexahedron to one of these patterns modifies the boundary of the mesh locally (Figure 32). The transitions between these patterns are known as *quad flips* or *bubble moves* [Funar, 1999]. These flipping operations are therefore a valuable building block to explore the space of shellable meshes.

Note that not all combinatorial meshes admit a shelling order (Figure 33) [Lutz, 2003]. Hence, by relying on these flipping operations to build hexahedral meshes, our method is inherently unable to construct certain meshes. Nonetheless, we can guarantee that a solution still exists: all quadrangulations of the sphere with an even number of quadrangles admit a shellable hexahedral mesh [Bern et al., 2002].

5.1.3 Identifying and Performing Flips

For each quadrangulation Q visited during the search, all possible quad flips need to be identified. Each flip corresponds to a different hexahedron that can be inserted in the mesh. The algorithm successively tries adding all of them to the current mesh. Because flips are performed by starting from the target boundary, the hexahedra that are constructed during this process form the reverse of a shelling order. Müller-Hannemann [1999] construct the hexahedra

in the same order, but our method, instead of only considering one mesh, explores the entire tree of possible sequences of quad flips.

The identification of all possible flips is split into two steps: first, the boundary Q is inspected to identify all occurrences of the 6 patterns from Figure 32. Second, those flips that correspond to the insertion of hexahedra that would make the mesh invalid are filtered out.

The hexahedron inserted by performing a flip is obtained by computing the union of the pattern before and after the flip. To determine whether or not this hexahedron is compatible with the mesh constructed so far, an efficient test is devised by considering three relations between the vertices of the mesh:

1. E, the edges of the mesh;
2. D_Q, the diagonals of the quadrangles in the mesh;
3. D_H, the interior diagonals of the hexahedra in the mesh.

These relations are disjoint in any combinatorial hexahedral mesh. For example, if a pair (u, v) is an edge, it is not the diagonal of any quadrangles or hexahedra. This leads to an efficient implementation of the test: simply maintain the three sets E, D_Q, and D_H, represented as bitsets, and verify that, after adding a new hexahedron:

1. the three sets E, D_Q, and D_H remain disjoint;
2. the new quadrangles in the hexahedron share no diagonals with any other quadrangle in the mesh;
3. none of the four interior diagonals of the new hexahedron are an interior diagonal of some other hexahedron.

It is easy to verify that when any two hexahedra share only a vertex, an edge, or a quadrangle, these conditions are met. To verify their sufficiency, consider two hexahedra with an invalid intersection

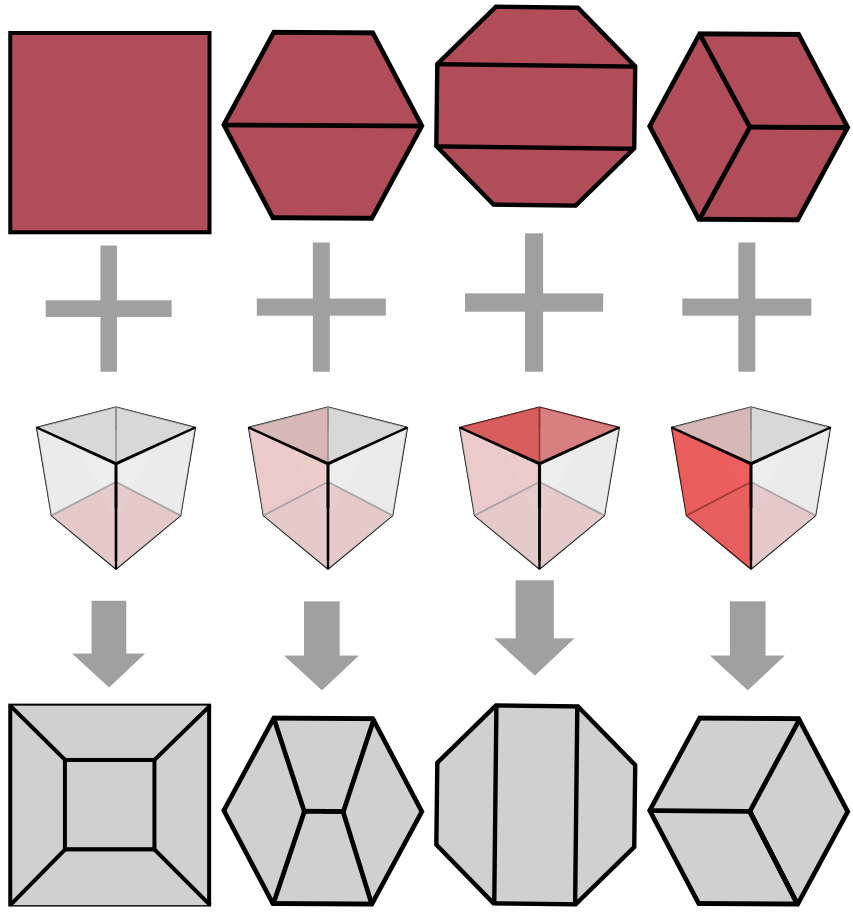


Figure 32: Equivalence between quad flips and hex creation. Adding one hexahedron glued to the red quadrangles (top row), modifies locally the quads and results in the grey quadrangles (bottom row). This operation is the key idea of the algorithm to search shellable meshes.

pattern. If an interior diagonal of one hexahedron is contained in the other hexahedron, one of the rules is always violated: rule 3 is violated if it is also an interior diagonal of the second hexahedron, and rule 1 is violated if it is an edge or the diagonal of a quadrangle. The only remaining cases to consider are those where the shared vertices are part of two distinct quadrangles. In all of those cases, the diagonal of one of those quadrangles appears in the other one. If it appears as an edge, rule 1 is violated; if not, both quadrangles have a shared diagonal, violating rule 2.

The insertion of the last hexahedron requires special treatment. This step does not correspond to a quad flip: when the boundary of the unmeshed region is isomorphic to the boundary of a cube, a hexahedron is inserted to finish the mesh. Detecting whether or not the current boundary corresponds to that of the cube is straightforward: simply verify that the boundary has exactly 6 faces.

5.1.4 Symmetry

There are many distinct sequences of quad flips which represent identical hexahedral meshes. It is thus important to only consider a single representation for each hexahedral mesh constructed during the search, lest most of the computation time be spent generating different representations of equivalent solutions.

One technique commonly used to deal with this type of issue is to define a canonical representation for objects under constructions, so that all those that belong to a given isomorphism class are transformed into the same representative element [Burton, 2011; Brinkmann and McKay, 2007]. A significant portion of the execution time is then spent computing the canonical representations of partial solutions, which may completely change after every operation [Jordan et al., 2018]. The symmetry breaking method used within our algorithm instead compares partial solutions directly, and exploits the tree-shaped structure of the search in order to reuse results from previous computations.

The strategy described in this section is based on *Symmetry*

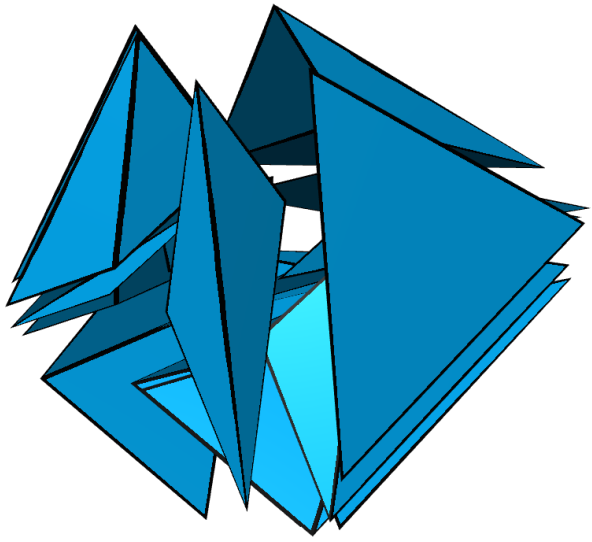


Figure 33: Vertex-minimal example of a non-shellable tetrahedral mesh of the 3-ball, computed by Lutz.

Breaking via Dominance Detection (SBDD) [Fahle et al., 2001]. Consider the search tree explored by the algorithm: its nodes are partial meshes constructed during the search, and edges correspond to the insertion of new hexahedra through quad flips. The objective is to prune from this search tree nodes that correspond to meshes that have already been explored (up to symmetry). This is accomplished using the following steps:

1. first, the automorphism group of the input quadrangulation is pre-computed;
2. then, as the search tree is traversed, fully explored subtrees are encoded into a sequence S;
3. for each new node, we determine whether or not it should be pruned by comparing it against the nodes stored in S.

5.1.4.1 Computing the Automorphism Group

Given a quadrangulation Q, we compute the set of its symmetries, known as its automorphism group. A permutation σ of the vertices of Q is a symmetry if it *preserves* the set of quadrangles: for any quadrangle (a, b, c, d) of Q, its image $(\sigma(a), \sigma(b), \sigma(c), \sigma(d))$ is also quadrangle of Q, and every quadrangle (a, b, c, d) is the image of a quadrangle $(\sigma^{-1}(a), \sigma^{-1}(b), \sigma^{-1}(c), \sigma^{-1}(d))$. Note that the orientations of the quadrangles may be reversed by σ.

Symmetries are computed one at a time, by fixing some quadrangle $q_A \in Q$ and assuming that its image under a symmetry σ is known to be $q_B \in Q$. There are 8 different ways to map the vertices of q_A onto the vertices of q_B, corresponding to the 8 symmetries of a quadrangle. The entire permutation σ is uniquely determined by this part of the map (Figure 34): the quadrangles adjacent to q_A must be the images of the quadrangles adjacent to q_B under σ, and the quadrangles adjacent to those must also be images of each other, and so on, until the whole quadrangulation has been traversed (Algorithm 5). This process is well-defined because each edge is in at most two quadrangles.

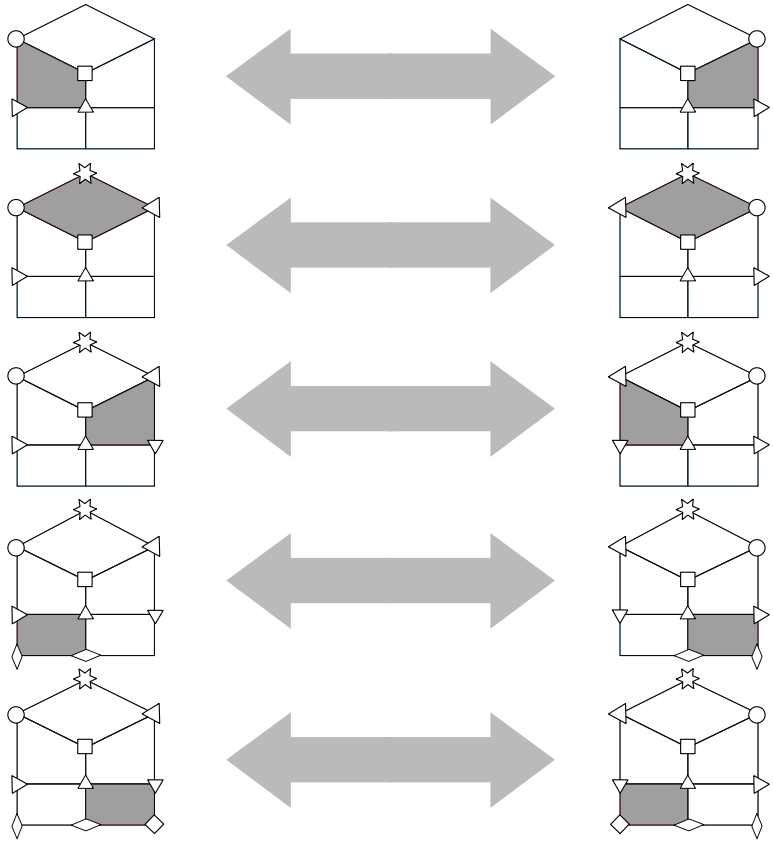


Figure 34: Computation of a symmetry. Starting from the assumption that a quadrangle is the image of some other quadrangle, the mesh is traversed while computing the correspondence between all other vertices.

Algorithm 5 Computes one symmetry from an initial assumption

Input: Q: A quadrangulation of the sphere; a quadrangle $q_0 \in Q$; (x, y, z, w), the image of q_0

Output: The symmetry σ that maps q_0 to (x, y, z, w)

```
def compute-symmetry(Q, q_0, (x, y, z, w)) :
  Initialize σ, mapping q_0 to (x, y, z, w)
  Initialize a queue with the 4 edges of q_0
  visited_A ← {q_0}
  visited_B ← {(x, y, z, w)}
  seen ← {q_0}
  while the queue is not empty:
    Dequeue an edge (a, b)
    q ← the quadrangle containing (a, b)
      and not in visited_A
    q' ← the quadrangle containing σ(a, b)
      and not in visited_B

    for each vertex v in q:
      v' ← the corresponding vertex in q'
      if σ(v) and σ^{-1}(v') are undefined:
        σ(v) ← v' (Extend the map σ)
        σ^{-1}(v') ← v
      else if σ(v) ≠ v' or σ^{-1}(v') ≠ v:
        fail (Stop upon contradiction)
      for each edge e of q:
        o ← the quadrangle on the other side of e
        if o has not been seen before:
          seen ← seen ∪ {o}
          Enqueue e

    visited_A ← visited_A ∪ {q}
    visited_B ← visited_B ∪ {q'}

  return σ
```

The entire set of symmetries is computed by considering all $8|Q|$ possible ways to map an arbitrary quadrangle q_A to any other quadrangle of Q. If an assumption is correct, a symmetry σ is obtained; if not, a contradiction will be reached when trying to construct the symmetry (two vertices mapping onto the same target vertex, or a single vertex with two images under σ). Because q_A must be the image of some quadrangle under any symmetry σ, this process yields the entire automorphism group.

In the worst case, the entire automorphism group is determined in $O(|Q|^2)$ operations. In practice, this quadratic time algorithm outperforms more complex linear time algorithms designed for planar graph isomorphism [Eppstein, 1999b; Colbourn and Booth, 1981] when applied to small quadrangulations, thanks to well-tuned heuristics. In particular, our implementation stops the algorithm as soon as two vertices of different degree are mapped onto one another by the permutation under construction [Brinkmann and McKay, 2007].

Moreover, because this method does not use the planarity of the graph, it is also more general. The only requirement is that the input be a *pseudomanifold*: a combinatorial cell complex in which every facet is contained in at most two distinct cells. Indeed, a variant of this method will be used to compare hexahedral meshes in section 5.1.4.3, by having quadrangles take over the role of edges in Algorithm 5.

5.1.4.2 Encoding the Search Tree

An efficient traversal of the search tree requires the search to stop as soon as the mesh under construction is the symmetric counterpart of a mesh that has previously been constructed. In the previous section, the set of symmetries that need to be considered was determined. This section now focuses on efficiently encoding the set of hexahedral meshes that have been constructed during the search.

Of course, the search tree is exponentially large, making it impossible to store every single mesh that is constructed during the search. Instead, SBDD only stores information about the roots of

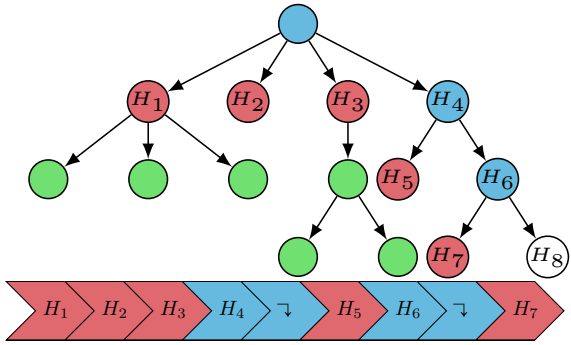


Figure 35: A partially explored search tree and the sequence used to compare the current node (in white) against the previously explored part of the tree. No-goods are shown in red, and partially explored subtrees in blue.

maximal fully explored subtrees, known as *no-goods* [Gent et al., 2006]. The current node should then be pruned if and only if the mesh under construction is the symmetric counterpart of one of the children of one of the no-goods. Note that a no-good is referred as such even if some of its children are solutions, since it is not desirable to compute the symmetric counterparts of those solutions.

No-goods can be stored efficiently thanks to the structure of the search tree. Recall that each node within the search tree corresponds to a partial hexahedral mesh, and each edge corresponds to the insertion of a hexahedron. Nodes with a common ancestor in the tree then share a common set of hexahedra as a prefix, and this prefix only needs to be stored once (Figure 35). Upon visiting a new node, the most recently added hexahedron is inserted in the sequence, followed by a special branching symbol, indicating that the rest of the sequence will encode the children of this node. Upon backtracking, everything up to and including the last branching symbol of the sequence is removed.

5.1.4.3 Dominance Detection and Pruning

The last part of our symmetry breaking method is the test used to prune nodes of the search tree that do not need to be explored because any solution that could be found by doing so has already been found. These nodes are said to be *dominated* by one of the no-goods, *i.e.* they are the symmetric counterpart of one of the children of one of the nodes that have been previously explored and stored in the sequence shown in Figure 35.

Since the search involves exploring exponentially many nodes, this dominance test must be implemented without explicitly comparing the current node against all previously explored nodes. Instead, this test is broken down into two steps: first find a no-good such that all its hexahedra are contained in the current partial solution, then determine if the hexahedra that are in the partial solution but missing from the no-good could be inserted using flipping operations. The sequence S constructed in the previous section is very valuable for this: not only does it save space by factoring out a common prefix, but it also saves time by allowing this prefix to be processed only once.

Let H be the current partial solution. The first step is to search within S for a partial mesh whose hexahedra are a subset of H (Algorithm 6). The process to find such a partial mesh is similar to the algorithm used to compute the automorphism group initially (Section 5.1.4.1). The goal is to construct σ, which maps the vertices of some partial mesh encoded in S to vertices of the current solution H, such that all hexahedra in the no-good are preserved by the map σ. The construction of σ again begins from an initial assumption, namely that the images of all boundary vertices through σ are known. Because boundary quadrangles must be preserved by σ, the set of possible initial assumptions is precisely the automorphism group that was previously computed.

Algorithm 6 is executed once for each element of the automorphism group and consists in a traversal of S during which the map σ is extended. The process ends either upon finding a partial mesh

Algorithm 6 Determine whether or not a partial mesh contains the symmetric counterpart of a previously visited node

Input: S: an encoding of the part of the search tree explored so far (Figure 35); H: a partial mesh; σ: a symmetry of the target boundary.
Output: **true** if H contains the symmetric counterparts of all the hexahedra of a fully explored subtree encoded in S

```
def contains-no-good(S, H, σ):
  seen ← ∅
  for each hexahedron h ∈ S:
    success ← true
    q ← a quadrangle of h whose image through σ is known
    q′ ← σ(q)
    h′ ← a hexahedron in H containing q′ and not in seen

    if there is such a hexahedron h′:
      (σ_old, σ_old^-1) ← (σ, σ^-1)

      for each vertex v of the quadrangle of h opposite to q:
        v′ ← the corresponding vertex in h′
        if σ(v) and σ^-1(v′) are undefined:
          σ(v) ← v′ (Extend the map σ)
          σ^-1(v′) ← v
        else if σ(v) ≠ v′ or σ^-1(v′) ≠ v:
          success ← false
          break (Stop upon contradiction)

      if success:
        seen ← seen ∪ {h′}
      else:
        (σ, σ^-1) ← (σ_old, σ_old^-1)
    else:
      success ← false

    if the symbol after h in S is the branching symbol:
      (All subsequent no-goods contain h.
      The search is aborted if its symmetric
      counterpart is not present.)
      if success is false: return false
    else if success:
      return true (no-good contained in H)
  return false
```

contained in H or upon reaching a contradiction. For each hexahedron h found in S, we attempt to extend σ such that h maps to some hexahedron of the current solution H. Each hexahedron created by a quad flip shares at least one quadrangle with the boundary or with a previously created hexahedron. Because of this, each hexahedron in S has at least one quadrangle whose symmetric counterpart is known. It is therefore possible to search for the hexahedron h' within the current solution H that contains this quadrangle (and has not already been determined to be the symmetric counterpart of another hexahedron).

Clearly, containing all hexahedra from some no-good is a requirement for a node being dominated – all children of the no-good share this common prefix. There could still be cases where none of the children of this no-good contain all the hexahedra that are in the current node. In other words, it may be impossible to find a sequence of quad flips which inserts the missing hexahedra when starting from the no-good. Testing for the existence of such a sequence may appear intractable at first, because shellability is an NP-complete property [Goaoc et al., 2018]. Thankfully, a correct test only needs not to produce any false positives, since false positives are the only reason a part of the search tree would incorrectly get pruned, causing solutions to be missed. Furthermore, because shellable meshes tend to accept many different shelling orders, there is a straightforward algorithm meeting this requirement and which very often computes the correct result: try a small number of permutations (say 10), then give up if no reverse shelling order was found (Algorithm 7).

5.2 Finding Larger Solutions using Pre-Computed Meshes

The exhaustive search described in the previous section can only be used with small limits on the maximum number of hexahedra, because of its exponential execution time. In many cases, finding

Algorithm 7 Determine whether or not a sequence of quad flips can create a given set of hexahedra

Input: Q: a quadrangulation of the sphere; H: a set of hexahedra; M: maximum number of permutations to test.
Output: **true** if a sequence of quad flips was found.

```
def try-reverse-shell(Q, H, M):
  if H = ∅: return true
  for each h ∈ H:
    if h can be added by performing a quad flip
       or Q is a cube:
      Q' ← the boundary after removing h
      if try-reverse-shell(Q', H \ {h}, M):
        return true
      else:
        Increment the number of tested permutations

      if M permutations or more have been tested:
        return false

    return true
```

a complete shelling by searching exhaustively is too difficult: the sequence of flips to construct the smallest solution is too long, and the search tree contains many paths which transform the initial boundary into one which is more difficult to mesh, instead of being closer to a solution.

Instead of searching for a sequence of quad flips that transforms the initial boundary Q into a cube, the key idea for solving larger cases is to stop the algorithm when a known configuration is found (Figure 36). For that purpose, we compute all boundaries that can be shelled with at most n hexahedra (say $n \leq 11$). Using a list of all such boundaries and one of their shellings (Section 5.2.1), this

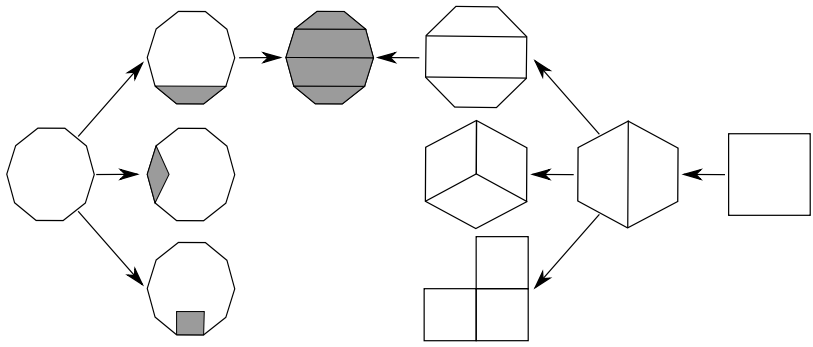


Figure 36: By precomputing quad meshes with up to three quadrangles (right), a search for all shellables meshes of an input polygon (left) can end three steps earlier. This idea generalizes to shellable hexahedral meshes.

variant of the algorithm can efficiently look up boundaries in the list during the search. This allows complete solutions to be constructed from any sequence of flips leading to any of the boundaries in the pre-computed set.

5.2.1 Computing Small Shellable Meshes

Consider the flip graph for quadrangulations of the sphere: its nodes represent quadrangulations of the sphere, and arcs between these nodes represent a flip between two quadrangulations. A breadth-first traversal of this graph starting from the cube and stopped at depth n generates all quadrangulations that can be obtained using a sequence of up to n flips. To deal with cycles in this graph, previously visited quadrangulations are stored in a hash table. The hash value for quadrangulations is constructed from a signature based on the valence of vertices, and the isomorphism test for two quadrangulations is performed using a variation on Algorithm 5 where the two starting quadrangles are part of different quadrangulations.

The signature used by our algorithm is a histogram of the va-

Algorithm 8 Generate small shellable hexahedral meshes

Input: n: maximum size for the generated hexahedral meshes
Output: $\mathcal{H}$: a set of hexahedral shellings with up to n hexahedra in each mesh.

```
def generate-shellings(n):
  S ← ∅
  H ← ∅
  Q ← new-queue()
  enqueue(Q, Cube)
  while Q is not empty:
    H ← dequeue(Q)
    H ← H ∪ {H}
    for each quad flip F:
      (B, h) ← perform-flip (F, ∂H)
      if is-compatible(H, h) ∧ B ∉ S:
        S ← S ∪ {B}
        enqueue(Q, H ∪ {h})

  return H
```

lences of each vertex, followed by the number of edges connecting vertices of valence v_a and valence v_b, for any v_a and v_b where this number is non-zero. While this choice of signature causes collisions, it can be computed quickly and new entries can be inserted without necessarily needing a slower computation to find a unique canonical representation.

Not all quadrangulations generated in this breadth first search admit a shelling with up to n hexahedra: interpreting the flips performed during the traversal of the graph as the insertion of hexahedra, these hexahedra may not all be compatible. By explicitly testing for compatibility while performing the breadth first search (Algorithm 8), we obtain a greedy construction similar to the pro-

$H_{\max}$	# quad meshes	timing
1	1	< 0.1 s
2	2	< 0.1 s
3	5	< 0.1 s
4	17	< 0.1 s
5	74	< 0.1 s
6	489	< 0.1 s
7	4,192	0.12 s
8	42,676	1.78 s
9	476,520	34.418 s
10	5,632,488	14 min 55 s
11	69,043,690	6 h 41 min

Table 5.1: Number of combinatorial quadrangulated boundaries that can be shelled with up to $H_{\max}$ hexahedra. Timings are given for a single thread on an Intel® Core™ i7-7700HQ CPU.

cedure outlined by Xiang and Liu [2018]: the hexahedra that are found are those which admit a shelling such that any prefix is the smallest shellable hexahedral mesh for the corresponding boundary. For large values of n, it is not clear that such a shelling should always exist, but we can verify this property for small values of n. For every quadrangulation found during the breadth-first search but without a hexahedral mesh found by Algorithm 8, Algorithm 4 is used to verify that there is indeed no shellable hexahedral mesh with at most n hexahedra. This test was performed for $n \leq 10$, and no counter-examples were found.

From Algorithm 8, a table of $69,043,690$ boundaries that can be meshed with up to 11 hexahedra is constructed (Table 5.1). These results also coincide with those obtained by Xiang and Liu [2018].

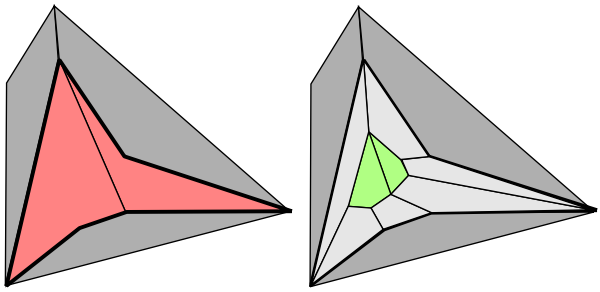


Figure 37: Insertion of a buffer layer to guarantee final mesh validity when using precomputed cavity meshes. (left) Simply adding two quadrangles (red) inside a cavity creates an invalid mesh where pairs of quads share more than one facet; (right) inserting a layer of buffer quads (light gray) allows the use of the same (combinatorial) quads to fill the cavity and produce a valid mesh.

5.2.2 Using the Pre-Computed Table

If at any point during the search, the boundary of the unmeshed region matches one of the pre-computed quadrangulations, the shelling of that quadrangulation is used to finish the meshing of that region.

The idea is to use the shelling computed in the previous section to fill the unmeshed region. Simply combining the two solutions is not always possible: this may produce an invalid mesh where, for example, two hexahedra share multiple quadrangles (Figure 37). Algorithm 4 could be used to compute all shellings of the unmeshed region with up to n hexahedra. If this search finds a shelling compatible with the partial solution constructed so far, a solution can be generated, at the cost of an additional computation.

Even if no such shelling was found, a solution can be constructed from any shelling of the unmeshed region, without performing an additional search or storing multiple hexahedrizations for each boundary: first construct a copy of the boundary of the unmeshed region, then, for each quadrangle of this boundary, create a hexahedron to connect each quadrangle to its copy. The hexahedra that have been

inserted in this manner are guaranteed to be compatible with any hexahedrization of the unmeshed region, allowing a complete mesh to be constructed. When this approach is used, the first solution found by the algorithm is not in general the smallest. However, when the smallest solution contains a large number of hexahedra, this approach can construct solutions in many cases where methods with stronger guarantees fail to find any, because it adds several hexahedra without branching.

Efficient access to the pre-computed table is performed using a binary search. We create an array of all the quadrangulations we found, sorted by their signatures. To find the hexahedral mesh corresponding to a given quadrangulation, its signature is computed and an isomorphism test is performed on all quadrangulations in the table that have the same signature.

5.3 A Constructive Solution for Constrained Hex-Meshing

Only one previous solution to the constrained hexahedral meshing problem gives a completely explicit construction [Carbonera and Shepherd, 2010]. This method requires 5396 n hexahedra to construct a valid mesh bounded by n quadrangles. In the following, we prove that this bound can be lowered to 78 n using the construction proposed by Erickson [2014].

Using our search algorithm (Section 5.2), we found hexahedral meshes for both types of buffer cells that Erickson's construction needs (Figure 38), along with geometric realizations using linear hexahedra obtained by applying existing mesh untangling techniques [Toulorge et al., 2013; Livesu et al., 2015], although the solutions have a very low minimum scaled Jacobian (Table 5.2). The meshes that we found contain 37 and 40 hexahedra. Because gluing multiple buffer cells together as needed by the construction would create a degenerate mesh, a hexahedron is added on each boundary quadrangle. The resulting meshes of the buffer cells have 57 and 62

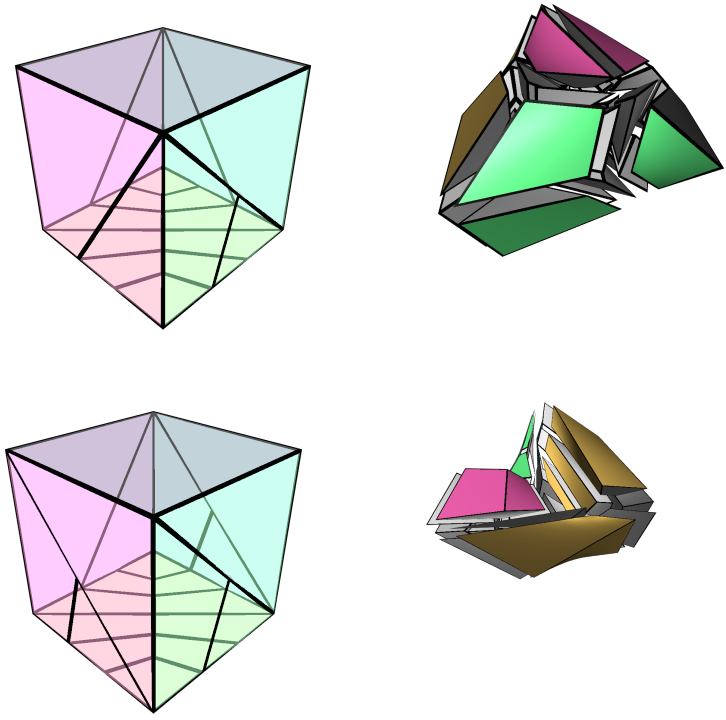


Figure 38: Hexahedrizations of the two types of buffer cubes used to mesh arbitrary domains in the algorithm of Erickson. (top) 37 hexahedra to mesh the 20-quadrangle cell; (bottom) 40 hexahedra to mesh the 22-quadrangle cell. Colors correspond to the different sides of the original cubes (shown on the left).

Template	$\|Q\|$	$\|V_{\text{bnd}}\|$	$\|H\|$	$\|V_{\text{total}}\|$	# edges by valence			Scaled Jacobian		
					3	4	5	min	max	median
Tetragonal trapezohedron	8	10	40	52	40	75	4	0.35	0.42	0.38
Schneiders' pyramid	16	18	36	51	32	62	4	0.12	0.49	0.26
Erickson's buffer cell (1)	20	22	37	53	43	49	4	0.31	0.63	0.42
Erickson's buffer cell (2)	22	24	40	55	44	48	9	0.031	0.45	0.41

Table 5.2: Statistics for the geometric meshes constructed for difficult test cases.

hexahedra respectively, giving the following result:

Let Ω be a compact and connected subset of $\mathbb{R}^3$ bounded by a 2-manifold $\partial\Omega$. Given a quadrangulation Q of $\partial\Omega$, each component of Q containing an even number of quadrangles, and a triangulation T of Ω (splitting each quadrangle of Q into two triangles), if there is a combinatorial hexahedral mesh of Ω bounded by Q, then there is one with no more than $62|Q| + 8|T|$ hexahedra. In particular, if Ω is a ball (hence $\partial\Omega$ is a sphere) and $|Q|$ is even, there is a combinatorial hexahedral mesh bounded by Q with no more than $78|Q|$ hexahedra.

Proof. Follow the construction of Erickson [2014] using the templates that we computed. There is one buffer cell for each boundary quadrangle, and each tetrahedron of the triangulation T is split into 4, 7, or 8 hexahedra. In the worst case, each buffer cell will be meshed with 62 hexahedra, and each tetrahedron will be split into 8 hexahedra.

If Ω is a ball, there is always a triangulation T with $2|Q|$ tetrahedra, obtained by arbitrarily splitting each quadrangle into two triangles, adding a vertex inside the domain, and joining each triangle to this new vertex by a tetrahedron. The bound for this special case is therefore $62|Q| + 8 \times 2|Q| = 78|Q|$.

A similar bound can be obtained for quadrangulations with an odd-number of quadrangles in some of their components. In that case, hexahedra are added to connect pairs of odd components, and the previous result is used to compute the number of hexahedra to mesh the rest of the domain.

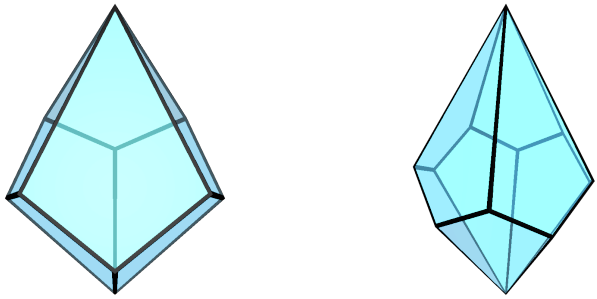


Figure 39: The trapezohedra are a family of polyhedra with two poles, each incident to n faces. They are usually difficult cases to mesh with hexahedra.

5.4 Hexahedrizations for Small Quadrangulations of the Sphere

We used the algorithm described in section 5.2 to compute hexahedrizations for all even quadrangulations of the sphere containing up to 20 quadrangles (Table 5.3). The 54,943 input quadrangulations were generated using `plantri` [Brinkmann et al., 2005]. We pre-computed shellable hexahedral meshes with up to 11 hexahedra. Of the 69,043,690 boundaries that were pre-computed, only 130 are included in the list of inputs. Nonetheless, in about 20% of all instances, the search for a solution terminates almost immediately after loading the set of pre-computed solutions (Figure 40). Only a few additional seconds are enough to find hexahedrizations bounded by most quadrangulations of the sphere. There are however some more difficult cases, requiring over an hour of computation time (Figure 42). The trapezohedron bounded by n faces, obtained by generalizing the tetragonal trapezohedron (Figure 39) is usually among the most difficult cases of a given size, requiring meshes with an intricate internal structure in order to be filled. For example,

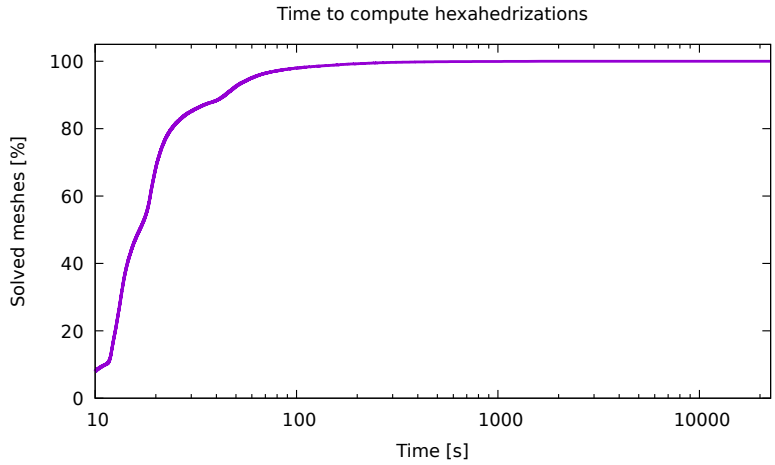


Figure 40: Time to compute hexahedrizations for all quadrangulations of the sphere with up to 20 quadrangles. Run on a machine with two AMD EPYC 7551 CPUs (32 cores per CPU).

the smallest solution found for the 20-face decagonal trapezohedron contained 72 hexahedra, strictly more than any of the other boundaries (Figure 41). Similarly, the 16-face octagonal trapezohedron required 67 hexahedra, with the decagonal trapezohedron being the only boundary for which all solutions found were larger. The trapezohedra are also among the boundaries that require the most time before any solution could be found. The 14-face heptagonal trapezohedron is the second most time consuming input, requiring 2h 50min, and the 20-face decagonal trapezohedron is the third, requiring 2h 43min. In the worst case, shown on Figure 42, it took 6h 15min before a 58-element mesh was found.

5.5 Small Non-Shellable Hexahedral Meshes

Chapter 4 described a generic search method for hexahedral meshes. This chapter has focused solely on the search for shellable meshes.

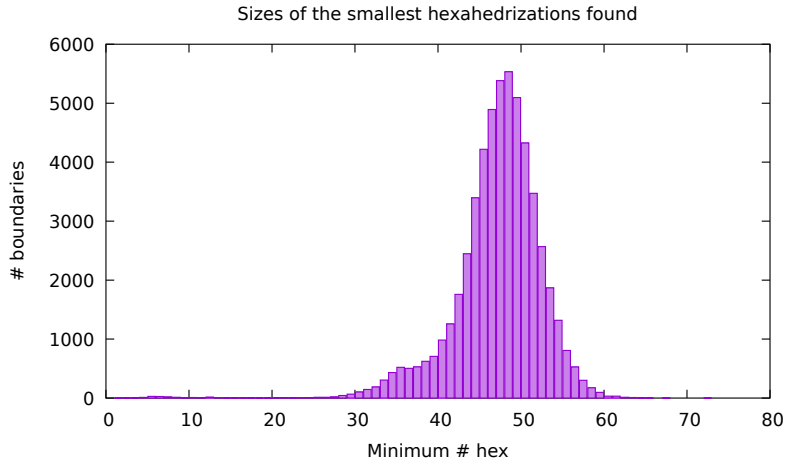


Figure 41: Time to compute hexahedrizations for all quadrangulations of the sphere with up to 20 quadrangles. Run on a machine with two AMD EPYC 7551 CPUs (32 cores per CPU).

Q	$\lvert H\rvert$			$\lvert V_{\text{total}}\rvert$			%edges by valence			
	min	max	med	min	max	med	3	4	5	6
6		1			8			(none)		
8		44			56		48	34	16	2
10	2	58	36	12	64	48	39	45	15	1
12	3	47	43	14	57	52	38	50	11	1
14	3	59	44	16	73	55	38	48	12	1
16	4	67	45	18	77	56	37	51	11	1
18	4	67	46	20	79	58	38	49	12	1
20	5	72	47	22	81	59	38	49	12	1

Table 5.3: Statistics for the combinatorial meshes computed for all even quadrangulations of the sphere with up to 20 quadrangles.

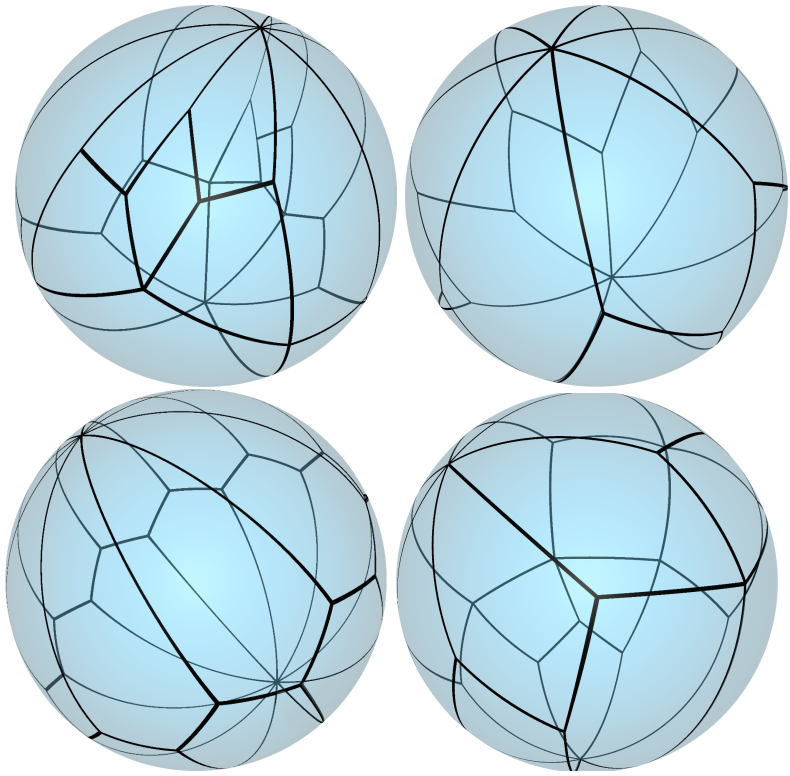


Figure 42: The four most time-consuming quadrangulated spheres to mesh using our method. Each required over an hour of computation time on 64 cores.

By using both algorithms with the same input, we're able to compare their outputs and identify non-shellable meshes: those that are found by the algorithm in Chapter 4, but not by the one described in this chapter.

This method allowed us to find a non-shellable hexahedral mesh with the boundary of a cube, containing a total of 30 vertices and 21 hexahedra. The untangling techniques that we've used so far were unable to produce a valid trilinear realization of this topological mesh.

There were only 14 distinct non-shellable meshes of the cube with at most 23 hexahedra. This justifies the focus on shellings for hexahedral meshing: non-shellable meshes are rare, require an irregular structure, and often do not appear to have a high quality geometric realization.

Chapter 6

A Geometric Mesh of Schneiders' Pyramid

So far we have focused on topological meshes. However, most applications require meshes represented as trilinear or higher-order surfaces with requirements on the quality of the elements depending on the problem domain. A more theoretical question concerns

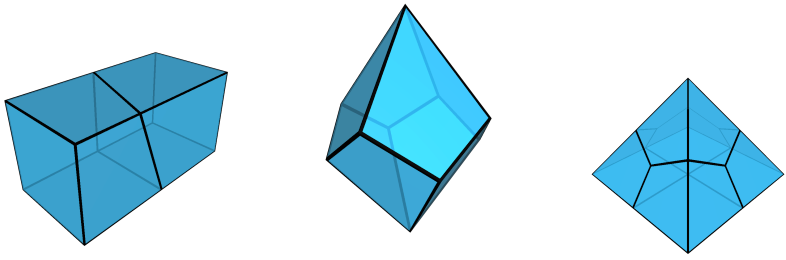


Figure 43: Three cases for which no geometric hexahedral meshes (with planar quadrangular faces) are known: a bicuboid with warped equator, the octagonal spindle, and Schneiders' pyramid. The latter two cases are solved in this chapter.

so-called *geometric* hexahedral meshes, where the hexahedra are convex polyhedra in $\mathbb{R}^3$, bounded by six planar faces. Even for simple input polyhedra, no geometric mesh has been found and the exact requirements the boundary must meet for the existence of a geometric hexahedral mesh remain unknown.

Eppstein [1999a] reduced the problem of geometric hex-meshing to one family of cases: that of bicuboids with a warped equator. Even the 8-quadrangle octagonal spindle (also known as tetragonal trapezohedron) and the closely related 16-quadrangle pyramid have no known hexahedral mesh without warped faces (Figure 43).

However, using the techniques introduced in chapters 4 and 5, we can improve the existing 88-element solution Yamakawa and Shimada [2010] and compute a geometric hexahedral mesh with rational coordinates. First we modify the topology of the mesh, obtaining a solution with only 44-element for the pyramid, containing a 40-element mesh for the trapezohedron (Section 6.1). We then adjust the coordinates of the vertices in the mesh to satisfy the constraint that every face be planar (Section 6.2). The resulting mesh is the first geometric hexahedral mesh of the pyramid, proving that such meshes exist for at least two of the three cases in Figure 43.

6.1 Simplifying a Mesh of Schneiders' Pyramid

We begin by applying the mesh simplification algorithm introduced in Section 4.2 to Yamakawa's mesh of Schneiders' pyramid to reduce the number of elements in the mesh through a sequence of local operations. The resulting hexahedral mesh is valid and contains 66 hexahedra and 63 interior vertices (Figure 44). Table 6.1 shows the sizes of the different cavities simplified by the algorithm. It takes a few minutes to reduce the number of hexahedra in the mesh from 88 down to 66. Figure 45 shows the changes to the connectivity of the mesh performed in two different iterations of the algorithm. The vertices had to be moved to obtain a valid mesh, but the combina-

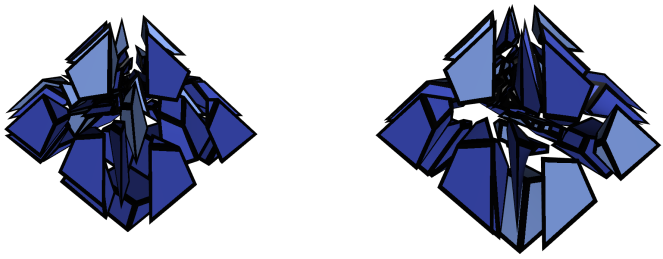


Figure 44: A 66-element mesh of Schneiders' pyramid (right) obtained from improving the topology of the 88-element of mesh Yamakawa and Shimada (left).

Initial Mesh		**Initial Cavity**			**Remeshed Cavity**		**New Mesh**	
#hex	**#vert.**	**#hex**	**#vert.**	**#bd. facets**	**#hex**	**#vert.**	**#hex**	**#vert**
88	105	8	23	18	6	21	86	103
86	103	8	23	18	6	21	84	101
84	101	8	23	18	6	21	82	99
82	99	14	33	24	8	27	76	93
76	93	6	16	10	2	12	72	89
72	89	18	40	30	12	32	66	81

Table 6.1: Cavity remeshing operations performed by our hex-mesh simplification algorithm on Yamakawa's 88-element mesh of Schneiders' pyramid.

torial boundary remains the same. For example, for the second pair of cavities in the figure, the same 30 facets can be seen before and after the remeshing operation: there is a central facet, surrounded by a ring of five quadrangles, followed by three rings of six quadrangles, followed by one more ring of five quadrangles surrounding a single face.

The next step of our construction is to use the 72-element mesh constructed in one of the intermediate steps detailed in Table 6.1 to

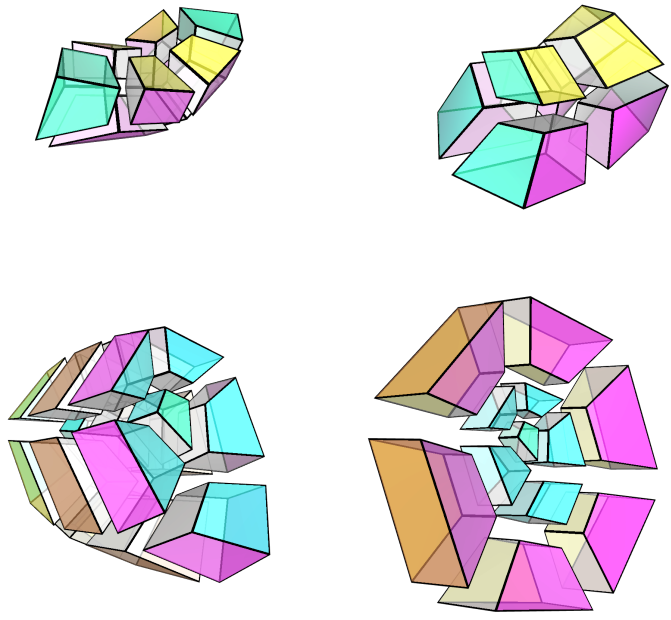


Figure 45: (top) Removal of two hexahedra from Schneiders' pyramid; (bottom) removal of six hexahedra. The initial cavity (left) and the remeshed cavity (right) have the same combinatorial boundary (top: 18 facets; bottom: 30 facets). Colors highlight the correspondence between faces.

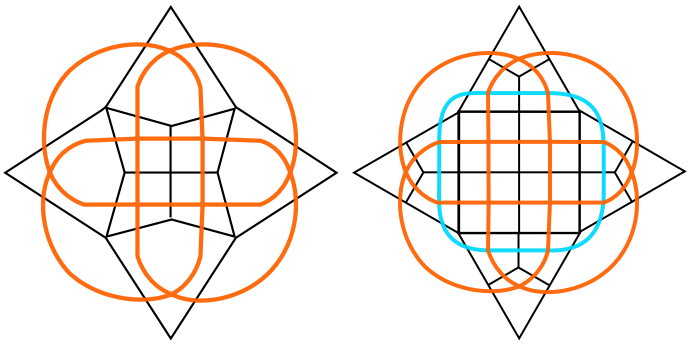


Figure 46: The octagonal spindle (left) can be used to construct Schneiders' pyramid (right) by only adding one layer of quadrangles (cyan curve).

create a 40-element mesh of the octagonal spindle. Indeed, Schneiders' pyramid and the octagonal spindle are related by their dual graphs: replace each quadrangle by a vertex, and create an edge between each pair of adjacent quadrangles. If the dual edges that traverse opposite edges of a quadrangle in the primal are grouped together, a simple arrangement of curve is obtained. The dual of Schneiders' pyramid is obtained by adding one curve to the dual of the octagonal spindle (Figure 46).

This relationship determines a method to create hexahedral meshes of the octagonal spindle from certain meshes of Schneiders' pyramid. The dual of a hexahedral mesh is a simple arrangement of surfaces [Murdoch et al., 1997], where each surface is bounded by zero, one, or multiple curves of the dual arrangement of the boundary quad mesh (Figure 47). If the dual of a mesh of Schneiders' pyramid contains a surface which is bounded only by the curve present in Schneiders' pyramid but not in the octagonal spindle, that surface can be removed by collapsing all the edges that it traverses [Borden et al., 2002]. The resulting mesh is a hexahedral mesh of the octagonal spindle.

In general, this operation may produce a degenerate mesh, with

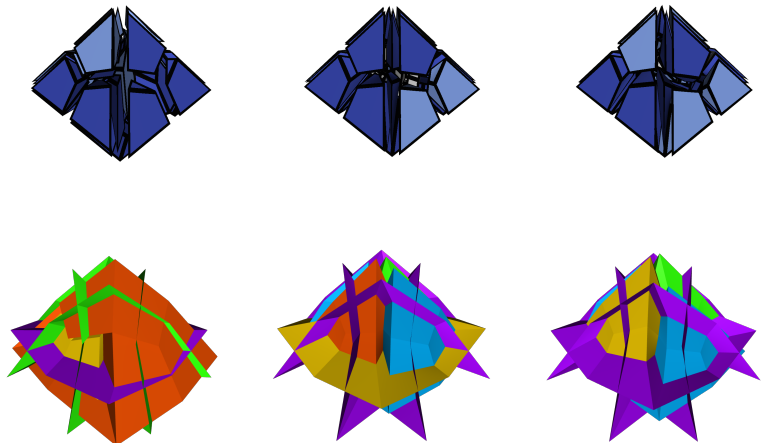


Figure 47: The mesh of Schneiders' pyramid by Yamakawa and Shimada (left), and our 72-element (center) and 66-element (right) meshes constructed from it. The first two meshes have two distinct dual surfaces for the two dual curves of the boundary (bottom row), but they were merged in the 66-element mesh.

hexahedra sharing multiple quadrangles or quadrangles sharing multiple edges. This is what prevented the construction of a mesh of the octagonal spindle directly from the 88-element mesh of Yamakawa and Shimada [2010]. Applying this procedure to our 72-element mesh, however, we obtain a new mesh the octagonal spindle, with 40 hexahedra and 42 interior vertices (Figure 49). This is the smallest known mesh of the octagonal spindle.

Finally, we add 4 hexahedra to our mesh of the spindle, reintroducing the missing layer of hexahedra to obtain a 44-element mesh of Schneiders' pyramid. (Figure 48). This trilinear mesh serves as our starting point for the construction of a geometric mesh.

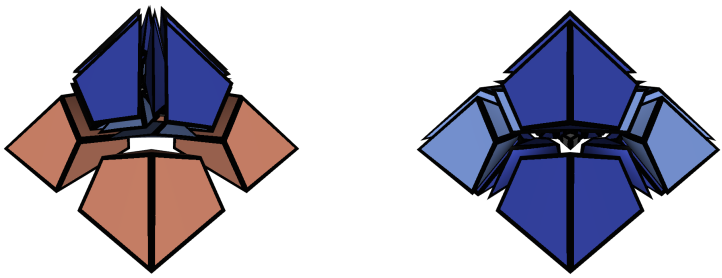


Figure 48: Comparison of our 44-element mesh of Schneiders' pyramid (left) with the smallest known 36-element solution (right). Both admit two planar symmetries.

6.2 The First Geometric Mesh of Schneiders' Pyramid

There are multiple meshes of the pyramid using trilinear hexahedra. That is, find coordinates for every vertex such that, for each hexahedron in the topological mesh, the convex hull of the 8 vertices indeed has 6 quadrangular faces. In theory, it is possible to determine if such a system of equation has a solution symbolically by formulating it using the existential theory of real numbers Tarski [1948]. In practice, however, this approach proves far too expensive. Instead, we opt to search for an exact numerical solution by making a few assumptions:

1. there is a geometric mesh *close* to one of our trilinear meshes, *i.e.* vertices only need to be adjusted by a small amount;

2. this geometric mesh can be represented using rational coordinates;

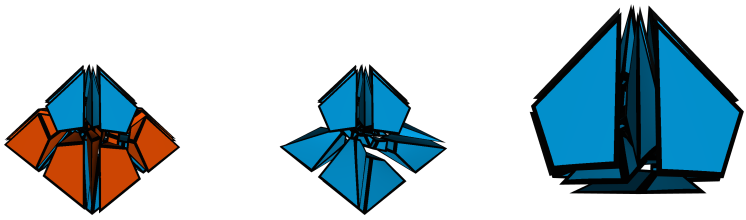


Figure 49: A layer of hexahedra in a 72-element mesh of Schneiders' pyramid (left) is removed (center). After merging the endpoints of each removed edge, a mesh of the octagonal spindle is obtained (right).

3. the geometric mesh *preserves* the same symmetries as our trilinear mesh. A *symmetry* of a combinatorial mesh M is a permutation σ of its vertices such that h is a hexahedron of M if and only if $\sigma(h)$ is also a hexahedron of M (possibly up to a reversal of orientation). A geometric or trilinear mesh *preserves* a combinatorial symmetry if for every such symmetry σ there is an affine function $f : \mathbb{R}^3 \to \mathbb{R}^3$ such that h is a hexahedron of the geometric or trilinear mesh if and only if the vertices of $\sigma(h)$ are the images of the corresponding vertices of h under f. Our mesh of the pyramid has two reflections as symmetries, $(x, y, z) \to (-x, y, z)$ and $(x, y, z) \to (x, y, -z)$. Its linear symmetry group also includes their composition and the identity.

In general, it is not the case that a solution meeting those requirements exist, even if a geometric mesh does exist. For example, the realization of a combinatorial mesh may require irrational numbers [Richter-Gebert and Ziegler, 1995] or not be able to preserve all combinatorial symmetries. However, these three assumptions greatly reduce the number of parameters needed for a solution, as well as the computational cost to manipulate them.

6.2.1 Initial Numerical Solution

The trilinear embedding of the meshes available to us were constructed to maximize quality measures used to evaluate meshes in finite element applications. The faces are non-planar because these measures do not seek to minimize the extent to which faces are warped.

Using SCIP [Bolusani et al., 2024], we compute a numerical mesh ensuring the volume of the tetrahedron spanned by the vertices $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$ of each face is within a small tolerance ϵ of zero:

$$-\epsilon \leq \frac{1}{6}[(\mathbf{b}-\mathbf{a}) \times (\mathbf{c}- \ \mathbf{a})] \cdot (\mathbf{d}-\mathbf{a}) \leq \epsilon$$

All boundary vertices have the coordinates fixed using an embedding of the boundary polyhedron. This process informs the choice for a combinatorial mesh to extend to a geometric mesh: whereas this process finds hexahedral mesh with volumes for the faces with $\epsilon \leq 10^{-8}$ from our 44-element mesh, it is not able to find such solutions for the 36-element mesh of the pyramid originally found by Xiang and Liu [2018]. This suggests it may not be possible to extend the 36-element mesh to a geometric mesh.

6.2.2 Constructing an Exact Geometric Mesh

From this approximate numerical solution, we construct a new mesh where every face will be planar. We keep track of the set F of vertices whose coordinates in the geometric mesh have been determined. These exact coordinates are computed by applying a sequence of the following three operations:

1. If there is a hexahedron h containing a vertex v such that the 6 vertices that share a quadrangle of h with v are in F, compute v by intersecting the 3 planes that contain it and add it to F.

2. If there is a quadrangle q containing a vertex v such that the other 3 vertices of q are in F, compute v by projecting it onto the plane of the remaining vertices and add it to F.

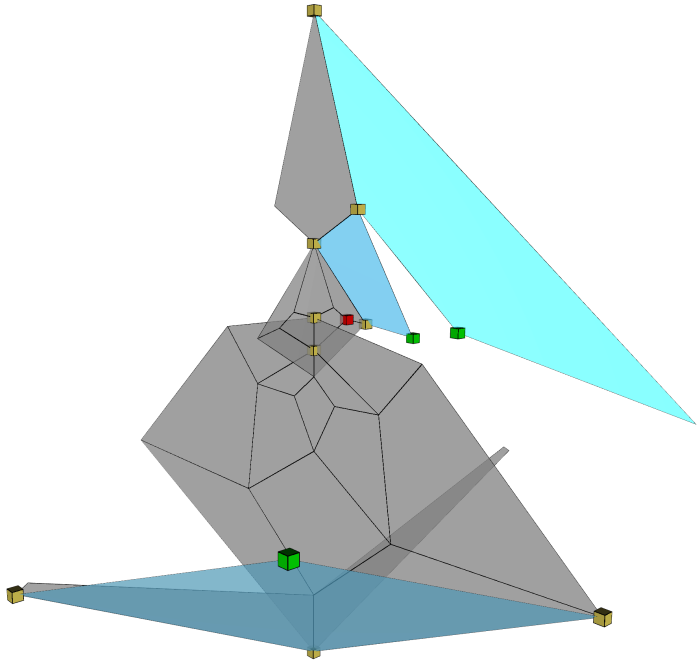


Figure 50: Construction of the geometric mesh. Vertices used from the quantized numerical solution are shown in yellow. Green vertices were projected onto a plane determined by other vertices (shown in blue). One vertex shown in red was manually adjusted. All other vertices are determined by symmetry or by intersecting three planes.

3. If none of those are possible, directly use a rational approximation of the coordinates of v in the numerical solution and add it to F. For this operator, the vertices located on the symmetry planes (*i.e.* the fixed points of the symmetries) are prioritized, since they have fewer degrees of freedom. In particular the vertices at the intersection of all symmetry planes are of the form $(0, y, 0)$ and only have one degree of freedom.

After adding a vertex v to F, its symmetric counterparts under each symmetry σ are also added to F, with their coordinates computed from the image of v under the corresponding geometric symmetry. Due to the high number of planarity constraints across the mesh, the location of all vertices become completely constrained after defining only a small number of them.

Each operation tends to increase the size of the fractions used to represent rational vertices. To keep them from growing too large, making each consecutive operation slower, the coordinates in the numerical solution are initially quantized to the nearest multiple of $\frac{1}{256}$ (except for the boundary vertices so that the boundary faces remain planar).

We manually find a sequence of operations that yields a mesh with all faces planar by construction, using 10 vertices from the quantized mesh (operation 3), 3 projections of a vertex onto a quadrangle (operation 2) and computing the other vertices with the first operation whenever possible (Figure 50). The resulting mesh may still include flipped hexahedra depending on the initial vertices, despite all faces being planar. Changing the location of one vertex by a small amount in the initial numerical solution and performing these operations again results in the first geometric mesh of Schneiders' pyramid, as well as the octagonal spindle (Figure 51).

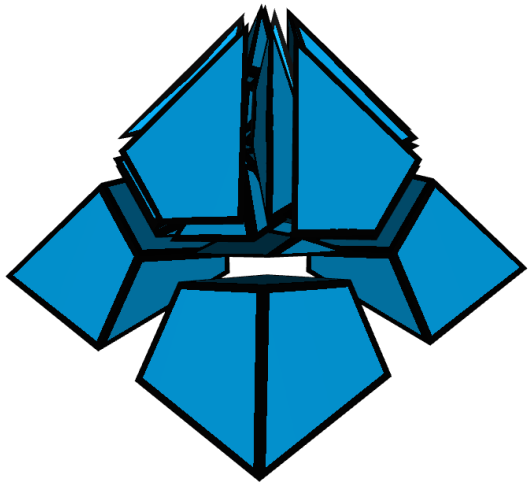


Figure 51: A geometric hexahedral mesh of Schneiders' pyramid.

Chapter 7

Conclusion

Topological meshing is an aspect of hexahedral mesh generation that had been investigated from a theoretical perspective, in the form of algorithm too impractical to implement in practice. This thesis demonstrated much smaller bounds on the size of hexahedral meshes than those found in the literature. Combinatorial search techniques further enabled us to find topological meshes for all quadrangulations of the sphere with up to 20 faces.

Importantly, these algorithms also demonstrate that there are no small hexahedral meshes for many simple boundary configurations, including Schneiders' pyramid. This has implication for many hexahedral meshing techniques. Consider any method that starts from the boundary and fills the domain using an advancing front, or any meshing algorithm that ends up enclosing a cavity with hexahedra at some point during the meshing process. These techniques are susceptible to encounter situations where the cavity they create requires a non-trivial combinatorial structure. Even if a valid topological mesh is found, computing valid trilinear mesh may be impractical. To robustly handle difficult cases, advancing fronts methods must either include mechanisms to avoid difficult-to-mesh boundaries, or to find a correct topological mesh even when such configurations arise.

Throughout this work, we've primarily discussed quadrangular

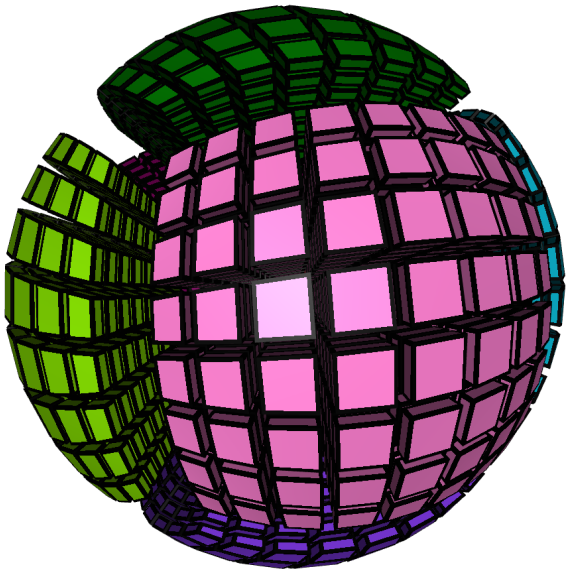


Figure 52: A block-structured mesh contains coarse-grained blocks (shown using colors), each subdivided into a regular grid of hexahedra.

meshes with a small number of faces, orders of magnitudes fewer than the meshes used in industrial applications. Nonetheless, we note two ways such combinatorial techniques may find concrete applications:

1. Block-structured meshes may contain a large number of elements, but are subdivided into a relatively small number of blocks, each containing a regular grid of hexahedra or quadrangles (Figure 52). Combinatorial techniques can be used to manipulate the mesh at the level of blocks.

2. Multiple meshing techniques rely on manually designed hex-

ahedral templates, for instance octree-based techniques use them extensively to represent size transitions. Our techniques automate the process of creating such templates, enabling techniques that rely on a wider range of templates.

Indeed, such combinatorial-first techniques have already found practical applications to generate boundary layers of hexahedral meshes [Reberol et al., 2023]. In order to successfully implement these techniques for a broader range of meshing tasks, more robust smoothing, untangling and mesh repair algorithms will be required. To deal with more complex cases, geometric information may also be used to guide topological and combinatorial algorithms towards suitable mesh structures.

Another important area of future research is geometric hex-meshing. Even if trilinear or higher order elements remain the focus for mesh generation in practice, the simpler geometric setting of convex polyhedral meshes may offer new insights into the link between the combinatorial structure of a mesh and its geometric quality. The tetragonal trapezohedron is one of several test cases for which no geometric mesh could be found for a long time. Yet, the methods shown in this thesis show that it in fact does admit one. Finding a geometric mesh for all bicuboids would prove that the criteria for geometric and topological meshing are the same. The enumeration of topological meshes may be part of a proof of this statement, or a useful tool to narrow down a geometric obstruction to the construction of hexahedral meshes.

Hexahedral meshing is an appealing goal with many industrial applications. This has motivated the continuous development of a wide range of mesh generation techniques, in spite of the many challenges that have stood for decades. The integration of methods from other fields, including combinatorial methods, is a promising path towards finally overcoming these obstacles.

Bibliography

Greg Aloupis, Erik D. Demaine, Alan Guo, and Giovanni Viglietta. 2015. Classic Nintendo games are (computationally) hard. *Theor. Comput. Sci.* 586 (2015), 135–160. https://doi.org/10.1016/J.TCS.2015.02.037

Altair. 2024a. HyperMesh Documentation: Map Meshing. https://help.altair.com/hwdesktop/hwx/topics/pre_processing/meshing/meshing_solid_map_unity_c.htm

Altair. 2024b. HyperMesh Documentation: Review Elements by Element Criteria. https://help.altair.com/hwdesktop/hwx/topics/pre_processing/meshing/2d_element_quality_by_criteria_review_t.htm

ANSYS. 2021. Introduction to ANSYS Meshing: Mesh Quality & Advanced Topics. Lecture. https://featips.com/wp-content/uploads/2021/05/Mesh-Intro_16.0_L07_Mesh_Quality_and_Advanced_Topics.pdf

Tathagata Basak. 2010. Combinatorial cell complexes and Poincaré duality. *Geom. Dedicata* 147 (2010), 357–387. https://doi.org/10.1007/s10711-010-9458-y

Klaus-Jürgen Bathe. 2006. *Finite element procedures*. Klaus-Jurgen Bathe.

Tristan Carrier Baudouin, Jean-Francois Remacle, Emilie Marchandise, Francois Henrotte, and Christophe Geuzaine. 2014. A

frontal approach to hex-dominant mesh generation. *Advanced Modeling and Simulation in Engineering Sciences* 1, 1 (2014), 1. http://amses-journal.springeropen.com/articles/10.1186/2213-7467-1-8

Bruce G. Baumgart. 1972. Winged edge polyhedron representation. https://doi.org/10.21236/ad0755141

Piotr Beben. 2020. Topology of frame field design for hex meshing. *arXiv preprint arXiv:2011.05276* (2020).

Marshall W. Bern, David Eppstein, and Jeff Erickson. 2002. Flipping Cubical Meshes. *Eng. Comput. (Lond.)* 18, 3 (2002), 173–187. https://doi.org/10.1007/s003660200016

Marshall W. Bern and Paul E. Plassmann. 2000. Mesh Generation. In *Handbook of Computational Geometry*, Jörg-Rüdiger Sack and Jorge Urrutia (Eds.). North Holland / Elsevier, 291–332. https://doi.org/10.1016/B978-044482537-7/50007-3

Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Lixing Xu. 2024. *The SCIP Optimization Suite 9.0.* Technical Report. Optimization Online. https://optimization-online.org/2024/02/the-scip-optimization-suite-9-0/

Michael J. Borden, Steven E. Benzley, and Jason F. Shepherd. 2002. Hexahedral Sheet Extraction.. In *IMR*. 147–152.

Domagoj Bosnjak, Antonio Pepe, Richard Schussnig, Dieter Schmalstieg, and Thomas-Peter Fries. 2024. Higher-order block-

structured hex meshing of tubular structures. *Eng. Comput.* 40, 2 (2024), 931–951. https://doi.org/10.1007/S00366-023-01834-7

Arnaud Botella, Bruno Lévy, and Guillaume Caumon. 2016. Indirect unstructured hex-dominant mesh generation using tetrahedra recombination. *Computational Geosciences* 20, 3 (2016), 437–451.

Mario Botsch, Mark Pauly, Leif Kobbelt, Pierre Alliez, Bruno Lévy, Stephan Bischoff, and Christian Rössl. 2007. Geometric modeling based on polygonal meshes. In *International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2007, San Diego, California, USA, August 5-9, 2007, Courses*, Sara McMains and Peter-Pike Sloan (Eds.). ACM, 1. https://doi.org/10.1145/1281500.1281640

Xavier Bourdin, Xavier Trosseille, Philippe Petit, and Philippe Beillas. 2007. Comparison of tetrahedral and hexahedral meshes for organ finite element modeling: an application to kidney impact. In *20th International technical conference on the enhanced safety of vehicle, Lyon.*

Gunnar Brinkmann, Sam Greenberg, Catherine S. Greenhill, Brendan D. McKay, Robin Thomas, and Paul Wollan. 2005. Generation of simple quadrangulations of the sphere. *Discrete Mathematics* 305, 1-3 (2005), 33–54. https://doi.org/10.1016/j.disc.2005.10.005

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

Benjamin A. Burton. 2011. The pachner graph and the simplification of 3-sphere triangulations. In *Proceedings of the 27th Symposium on Computational Geometry*. 153–162. https://doi.org/10.1145/1998196.1998220

Benjamin A. Burton, Ryan Budney, William Pettersson, et al. 1999–2023. Regina: Software for low-dimensional topology. http://regina-normal.github.io/

James C Caendish, David A Field, and William H Frey. 1985. An apporach to automatic three-dimensional finite element mesh generation. *International journal for numerical methods in engineering* 21, 2 (1985), 329–347.

Carlos D. Carbonera and Jason F. Shepherd. 2010. A constructive approach to constrained hexahedral mesh generation. *Eng. Comput. (Lond.)* 26, 4 (2010), 341–350. https://doi.org/10.1007/s00366-009-0168-8

Edwin E. Catmull. 1972. A system for computer generated movies. In *Proceedings of the ACM annual conference, ACM 1972, Boston, MA, USA, August 1972, Volume 1*, John J. Donovan and Rosemary Shields (Eds.). ACM, 422–431. https://doi.org/10.1145/800193.569952

John R Chawner, John Dannenhoffer, and Nigel J Taylor. 2016. Geometry, mesh generation, and the CFD 2030 vision. In *46th AIAA Fluid Dynamics Conference*. 3485.

Charles J. Colbourn and Kellogg S. Booth. 1981. Linear time automorphism algorithms for trees, interval graphs, and planar graphs. *SIAM J. Comput.* 10, 1 (1981), 203–225. https://doi.org/10.1137/0210015

Cubit. 2024. CUBIT 16.16 User Documentation. https://cubit.sandia.gov/files/cubit/16.16/help_manual/Printed_Documentation/Cubit_16.16_User_Documentation.pdf

Herbert Edelsbrunner. 2001. *Geometry and Topology for Mesh Generation*. Cambridge monographs on applied and computational mathematics, Vol. 7. Cambridge University Press.

Jack Edmonds. 1965. Paths, trees, and flowers. *Canadian Journal of mathematics* 17 (1965), 449–467.

David Eppstein. 1999a. Linear complexity hexahedral mesh generation. *Computational Geometry* 12, 1-2 (1999), 3–16. https://doi.org/10.1016/S0925-7721(98)00032-7

David Eppstein. 1999b. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.* 3, 3 (1999). http://www.cs.brown.edu/publications/jgaa/accepted/99/Eppstein99.3.3.pdf

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

Torsten Fahle, Stefan Schamberger, and Meinolf Sellmann. 2001. Symmetry Breaking. In *Principles and Practice of Constraint Programming*. 93–107. https://doi.org/10.1007/3-540-45578-7_7

Louis Funar. 1999. Cubulations mod bubble moves. *Contemp. Math.* 233 (1999), 29–44.

Xifeng Gao, Jin Huang, Kaoji Xu, Zherong Pan, Zhigang Deng, and Guoning Chen. 2017a. Evaluating Hex-mesh Quality Metrics via Correlation Analysis. *Comput. Graph. Forum* 36, 5 (2017), 105–116. https://doi.org/10.1111/cgf.13249

Xifeng Gao, Wenzel Jakob, Marco Tarini, and Daniele Panozzo. 2017b. Robust hex-dominant mesh generation using field-guided polyhedral agglomeration. *ACM Trans. Graph.* 36, 4 (2017), 114:1–114:13. https://doi.org/10.1145/3072959.3073676

Xifeng Gao, Hanxiao Shen, and Daniele Panozzo. 2019. Feature Preserving Octree-Based Hexahedral Meshing. *Comput. Graph. Forum* 38, 5 (2019), 135–149. https://doi.org/10.1111/cgf.13795

Ian P. Gent, Karen E. Petrie, and Jean-François Puget. 2006. Symmetry in Constraint Programming. In *Handbook of Constraint Programming*. 329–376. https://doi.org/10.1016/S1574-6526(06)80014-3

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 (Sep 2009), 1309–1331. https://doi.org/10.1002/nme.2579

Xavier Goaoc, Pavel Paták, Zuzana Patáková, Martin Tancer, and Uli Wagner. 2018. Shellability is NP-Complete. In *Proceedings of the 34th Symposium on Computational Geometry*. 41:1–41:15. https://doi.org/10.4230/LIPIcs.SoCG.2018.41

Craig Gotsman, Xianfeng Gu, and Alla Sheffer. 2003. Fundamentals of spherical parameterization for 3D meshes. *ACM Trans. Graph.* 22, 3 (2003), 358–363. https://doi.org/10.1145/882262.882276

James Gregson, Alla Sheffer, and Eugene Zhang. 2011. All-Hex Mesh Generation via Volumetric PolyCube Deformation. *Computer Graphics Forum* 30, 5 (Aug. 2011), 1407–1416. https://doi.org/10.1111/j.1467-8659.2011.02015.x

Allen Hatcher. 2002. *Algebraic topology.* Cambridge University Press, Cambridge. xii+544 pages.

K Ho-Le. 1988. Finite element mesh generation methods: a review and classification. *Computer-aided design* 20, 1 (1988), 27–38.

Thomas JR Hughes, John A Cottrell, and Yuri Bazilevs. 2005. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering* 194, 39-41 (2005), 4135–4195.

Amaury Johnen, Jean-Christophe Weill, and Jean-François Remacle. 2017. Robust and efficient validation of the linear

hexahedral element. *Procedia Engineering* 203 (2017), 271–283. https://doi.org/10.1016/j.proeng.2017.09.809

Charles Jordan, Michael Joswig, and Lars Kastner. 2018. Parallel Enumeration of Triangulations. *Electr. J. Comb.* 25, 3 (2018), P3.6. http://www.combinatorics.org/ojs/index.php/eljc/article/view/v25i3p6

Tomasz Kaczynski, Konstantin Mischaikow, and Marian Mrozek. 2003. Computing homology. Vol. 5. 233–256. http://projecteuclid.org/euclid.hha/1088453326 Algebraic topological methods in computer science (Stanford, CA, 2001).

Richard M. Karp. 1972. Reducibility Among Combinatorial Problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA (The IBM Research Symposia Series)*, Raymond E. Miller and James W. Thatcher (Eds.). Plenum Press, New York, 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9

Patrick M Knupp. 1990. On the invertibility of the isoparametric map. *Computer Methods in Applied Mechanics and Engineering* 78, 3 (1990), 313–329.

Patrick M. Knupp. 2001. Hexahedral and Tetrahedral Mesh Untangling. *Engineering with Computers* 17, 3 (Oct. 2001), 261–268. https://doi.org/10.1007/s003660170006

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, 147–164.

Bruno Lévy. 2001. Constrained texture mapping for polygonal meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2001,*

Los Angeles, California, USA, August 12-17, 2001, Lynn Pocock (Ed.). ACM, 417–424. https://doi.org/10.1145/383259.383308

Bruno Lévy and Yang Liu. 2010. L_p Centroidal Voronoi Tessellation and its applications. *ACM Trans. Graph.* 29, 4 (2010), 119:1–119:11. https://doi.org/10.1145/1778765.1778856

Yufei Li, Yang Liu, Weiwei Xu, Wenping Wang, and Baining Guo. 2012. All-hex meshing using singularity-restricted field. *ACM Trans. Graph.* 31, 6 (2012), 177:1–177:11. https://doi.org/10.1145/2366145.2366196

Juncong Lin, Xiaogang Jin, Zhengwen Fan, and Charlie C. L. Wang. 2008. Automatic PolyCube-Maps. In *Advances in Geometric Modeling and Processing, 5th International Conference, GMP 2008, Hangzhou, China, April 23-25, 2008. Proceedings.* 3–16. https://doi.org/10.1007/978-3-540-79246-8_1

Heng Liu and David Bommes. 2023. Locally Meshable Frame Fields. *ACM Trans. Graph.* 42, 4 (2023), 112:1–112:20. https://doi.org/10.1145/3592457

Heng Liu, Paul Zhang, Edward Chien, Justin Solomon, and David Bommes. 2018. Singularity-constrained octahedral fields for hexahedral meshing. *ACM Trans. Graph.* 37, 4 (2018), 93:1–93:17. https://doi.org/10.1145/3197517.3201344

Marco Livesu, Alla Sheffer, Nicholas Vining, and Marco Tarini. 2015. Practical Hex-mesh Optimization via Edge-cone Rectification. *ACM Trans. Graph.* 34, 4 (July 2015), 141:1–141:11. https://doi.org/10.1145/2766905

Marco Livesu, Nicholas Vining, Alla Sheffer, James Gregson, and Riccardo Scateni. 2013. PolyCut: monotone graph-cuts for PolyCube base-complex construction. *ACM Trans. Graph.* 32, 6 (2013), 171:1–171:12. https://doi.org/10.1145/2508363.2508388

Frank H. Lutz. 2003. A Vertex-Minimal Non-Shellable Simplicial 3-Ball with 9 Vertices and 18 Facets EG-Models Home. http://www.eg-models.de/models/Simplicial_Manifolds/2003.05.004/_preview.html

Manish Mandad, Ruizhi Chen, David Bommes, and Marcel Campen. 2022. Intrinsic mixed-integer polycubes for hexahedral meshing. *Comput. Aided Geom. Des.* 94 (2022), 102078. https://doi.org/10.1016/J.CAGD.2022.102078

Emilie Marchandise, Gaëtan Compère, Marie Willemet, Gaëtan Bricteux, Christophe Geuzaine, and J-F Remacle. 2010. Quality meshing based on STL triangulations for biomedical simulations. *International Journal for Numerical Methods in Biomedical Engineering* 26, 7 (2010), 876–889.

Loïc Maréchal. 2009. Advances in Octree-Based All-Hexahedral Mesh Generation: Handling Sharp Features. In *Proceedings of the 18th International Meshing Roundtable, IMR 2009, October 25-28, 2009, Salt Lake City, UT, USA*. 65–84. https://doi.org/10.1007/978-3-642-04319-2_5

Célestin Marot, Kilian Verhetsel, and Jean-François Remacle. 2020. Reviving the search for optimal tetrahedralizations. *Proceedings of the 28th International Meshing Roundtable. Zenodo, Buffalo, New York, USA* (2020).

Zoë Marschner, David R. Palmer, Paul Zhang, and Justin Solomon. 2020. Hexahedral Mesh Repair via Sum-of-Squares Relaxation. *Comput. Graph. Forum* 39, 5 (2020), 133–147. https://doi.org/10.1111/CGF.14074

Sia Meshkat and Dafna Talmor. 2000. Generating a mixed mesh of hexahedra, pentahedra and tetrahedra from an underlying tetrahedral mesh. *Internat. J. Numer. Methods Engrg.* 49, 1-2 (Sept. 2000), 17–30. https://doi.org/10.1002/1097-0207(20000910/20)49:1/2<17::AID-NME920>3.0.CO;2-U

Scott A Mitchell. 1996. A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volume. In *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 465–476.

Scott A. Mitchell and Timothy J. Tautges. 1994. Pillowing doublets: refining a mesh to ensure that faces share at most one edge. In *Proceedings of the 4th International Meshing Roundtable*. 231–240.

Yuichiro Motooka, So Noguchi, and Hajime Igarashi. 2011. Evaluation of hexahedral mesh quality for finite element method in electromagnetics. In *Materials Science Forum*, Vol. 670. Trans Tech Publications, 318–324.

Matthias Müller-Hannemann. 1999. Hexahedral mesh generation by successive dual cycle elimination. *Eng. Comput. (Lond.)* 15, 3 (1999), 269–279.

James R. Munkres. 1984. *Elements of algebraic topology*. Addison-Wesley Publishing Company, Menlo Park, CA. ix+454 pages.

Peter Murdoch, Steven Benzley, Ted Blacker, and Scott A. Mitchell. 1997. The spatial twist continuum: a connectivity based method for representing all-hexahedral finite element meshes. *Finite Elem. Anal. Des.* 28, 2 (1997), 137–149. https://doi.org/10.1016/S0168-874X(97)81956-7

Matthias Nieser, Ulrich Reitebuch, and Konrad Polthier. 2011. CubeCover - Parameterization of 3D Volumes. *Comput. Graph. Forum* 30, 5 (2011), 1397–1406. https://doi.org/10.1111/j.1467-8659.2011.02014.x

Steven J. Owen. 2001. Hex-dominant mesh generation using 3D constrained triangulation. *Computer-Aided Design* 33, 3 (2001), 211–220. https://doi.org/10.1016/S0010-4485(00)00121-4

Jeanne Pellerin, Amaury Johnen, Kilian Verhetsel, and Jean-François Remacle. 2018a. Identifying combinations of tetrahedra into hexahedra: A vertex based strategy. *Computer-Aided Design* 105 (2018). https://doi.org/10.1016/j.cad.2018.05.004

Jeanne Pellerin, Kilian Verhetsel, and Jean-François Remacle. 2018b. There are 174 subdivisions of the hexahedron into tetrahedra. *ACM Trans. Graph.* 37, 6 (2018), 266:1–266:9. https://doi.org/10.1145/3272127.3275037

Nico Pietroni, Marcel Campen, Alla Sheffer, Gianmarco Cherchi, David Bommes, Xifeng Gao, Riccardo Scateni, Franck Ledoux, Jean-François Remacle, and Marco Livesu. 2023. Hex-Mesh Generation and Processing: A Survey. *ACM Trans. Graph.* 42, 2 (2023), 16:1–16:44. https://doi.org/10.1145/3554920

Pixar. 2023. OpenSubDiv 3.6.0 Documentation: Modeling Tips. https://graphics.pixar.com/opensubdiv/docs/mod_notes.html

François Protais, Maxence Reberol, Nicolas Ray, Etienne Corman, Franck Ledoux, and Dmitry Sokolov. 2022. Robust Quantization for Polycube Maps. *Comput. Aided Des.* 150 (2022), 103321. https://doi.org/10.1016/J.CAD.2022.103321

Nicolas Ray, Dmitry Sokolov, Maxence Reberol, Franck Ledoux, and Bruno Lévy. 2018. Hex-dominant meshing: Mind the gap! *Computer-Aided Design* 102 (2018), 94–103. https://doi.org/10.1016/j.cad.2018.04.012

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 (2023), 2:1–2:32. https://doi.org/10.1145/3577196

Jean-Charles Régin, Mohamed Rezgui, and Arnaud Malapert. 2013. Embarrassingly parallel search. In *International Conference on*

Principles and Practice of Constraint Programming. Springer, 596–610.

Jean-Francois Remacle, Rajesh Gandham, and Tim Warburton. 2016. GPU accelerated spectral finite elements on all-hex meshes. *J. Comput. Physics* 324 (Nov. 2016), 246–257.

J-F Remacle, Jonathan Lambrechts, Bruno Seny, Emilie Marchandise, Amaury Johnen, and C Geuzainet. 2012. Blossom-Quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm. *International journal for numerical methods in engineering* 89, 9 (2012), 1102–1119.

Jürgen Richter-Gebert and Günter M. Ziegler. 1995. Realization spaces of 4-polytopes are universal. *Bull. Amer. Math. Soc. (N.S.)* 32, 4 (1995), 403–412. `https://doi.org/10.1090/S0273-0979-1995-00604-X`

Francesca Rossi, Peter Van Beek, and Toby Walsh. 2006. *Handbook of constraint programming*. Elsevier.

Colin P Rourke and Brian Joseph Sanderson. 2012. *Introduction to piecewise-linear topology*. Springer Science & Business Media.

Igor Sazonov and Perumal Nithiarasu. 2011. Semi-automatic surface and volume mesh generation for subject-specific biomedical geometries. *International Journal for Numerical Methods in Biomedical Engineering* 28, 1 (2011), 133–157.

Saul Schleimer. 2011. Sphere recognition lies in NP. In *Low-dimensional and symplectic topology*. Proc. Sympos. Pure Math., Vol. 82. Amer. Math. Soc., Providence, RI, 183–213. `https://doi.org/10.1090/pspum/082/2768660`

Robert Schneiders. 1995. Open problem. `https://www.robertschneiders.de/meshgeneration/open.html`

Robert Schneiders. 1996. A grid-based algorithm for the generation of hexahedral element meshes. *Engineering with computers* 12, 3-4 (1996), 168–177.

Robert Schneiders. 2000. Octree-Based Hexahedral Mesh Generation. *Int. J. Comput. Geometry Appl.* 10, 4 (2000), 383–398. https://doi.org/10.1142/S021819590000022X

Alexander Schwartz and Günter M. Ziegler. 2004. Construction techniques for cubical complexes, odd cubical 4-polytopes, and prescribed dual manifolds. *Experimental Mathematics* 13, 4 (2004), 385–413.

Alla Sheffer, Emil Praun, Kenneth Rose, et al. 2007. Mesh parameterization methods and their applications. *Foundations and Trends® in Computer Graphics and Vision* 2, 2 (2007), 105–171.

Jason F. Shepherd and Chris R. Johnson. 2009. Hexahedral mesh generation for biomedical models in SCIRun. *Eng. Comput.* 25, 1 (2009), 97–114. https://doi.org/10.1007/S00366-008-0108-Z

Jonathan Richard Shewchuk. 1998. Tetrahedral Mesh Generation by Delaunay Refinement. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry, Minneapolis, Minnesota, USA, June 7-10, 1998*, Ravi Janardan (Ed.). ACM, 86–95. https://doi.org/10.1145/276884.276894

Kenji Shimada. 2011. Current Issues and Trends in Meshing and Geometric Processing for Computational Engineering Analyses. *J. Comput. Inf. Sci. Eng.* 11, 2 (2011). https://doi.org/10.1115/1.3593414

Kenji Shimada. 2018. Mesh Generation — Fundamental Issues and Emerging Applications. https://cmu.app.box.com/s/fqx4tbetklypt89e6kb4zkujxugn5coh

Hang Si. 2015. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw.* 41, 2 (feb 2015), 11:1–11:36. https://doi.org/10.1145/2629697

T. K. H. Tam and Cecil G. Armstrong. 1991. 2D finite element mesh generation by medial axis subdivision. *Advances in engineering software and workstations* 13, 5-6 (1991), 313–324.

Alfred Tarski. 1948. *A Decision Method for Elementary Algebra and Geometry.* The Rand Corporation, Santa Monica, CA. iii+60 pages.

Timothy J. Tautges, Ted Blacker, and Scott 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/%28SICI%291097-0207%2819961015%2939%3A19<3327%3A%3AAID-NME2>3.0.CO%3B2-H

William P. Thurston. 1993. Hexahedral decomposition of polyhedra. Posting to sci.math. http://www.ics.uci.edu/~eppstein/gina/Thurston-hexahedra.html

Hua Tong, Eni Halilaj, and Yongjie Jessica Zhang. 2024. HybridOctree_Hex: Hybrid octree-based adaptive all-hexahedral mesh generation with Jacobian control. *J. Comput. Sci.* 78 (2024), 102278. https://doi.org/10.1016/J.JOCS.2024.102278

Thomas Toulorge, Christophe Geuzaine, Jean-François Remacle, and Jonathan Lambrechts. 2013. Robust untangling of curvilinear meshes. *J. Comput. Physics* 254 (2013), 8–26. https://doi.org/10.1016/j.jcp.2013.07.022

M Jon Turner, Ray W Clough, Harold C Martin, and LJ Topp. 1956. Stiffness and deflection analysis of complex structures. *journal of the Aeronautical Sciences* 23, 9 (1956), 805–823.

Olga V. Ushakova. 2011. Nondegeneracy tests for hexahedral cells. *Computer Methods in Applied Mechanics and Engineering* 200, 17–20 (apr 2011), 1649–1658. https://doi.org/10.1016/j.cma.2011.01.014

Kilian Verhetsel, Jeanne Pellerin, and Jean-François Remacle. 2019a. A 44-element mesh of Schneiders' pyramid: Bounding the difficulty of hex-meshing problems. *Computer-Aided Design*. https://doi.org/10.1016/j.cad.2019.102735

Kilian Verhetsel, Jeanne Pellerin, and Jean-François Remacle. 2019b. Finding hexahedrizations for small quadrangulations of the sphere. *ACM Transactions on Graphics (TOG)* 38, 4 (jul 2019), 53. https://doi.org/10.1145/3306346.3323017

B Wördenweber. 1984. Finite element mesh generation. *Computer-Aided Design* 16, 5 (1984), 285–291.

Shang Xiang and Jianfei Liu. 2018. A 36-Element Solution To Schneiders' Pyramid Hex-Meshing Problem And A Parity-Changing Template For Hex-Mesh Revision. *arXiv preprint arXiv:1807.09415* (2018).

Soji Yamakawa and Kenji Shimada. 2003. Fully-automated hex-dominant mesh generation with directionality control via packing rectangular solid cells. *Internat. J. Numer. Methods Engrg.* 57, 15 (2003), 2099–2129. https://doi.org/10.1002/nme.754 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.754

Soji Yamakawa and Kenji Shimada. 2010. 88-Element solution to Schneiders' pyramid hex-meshing problem. *International Journal for Numerical Methods in Biomedical Engineering* 26 (2010), 1700–1712. https://doi.org/10.1002/cnm.1256

Hao Zhang, Oliver van Kaick, and Ramsay Dyer. 2010. Spectral Mesh Processing. *Comput. Graph. Forum* 29, 6 (2010), 1865–1894. https://doi.org/10.1111/J.1467-8659.2010.01655.X

Günter M. Ziegler. 1995. *Lectures on polytopes*. Graduate Texts in Mathematics, Vol. 152. Springer-Verlag, New York. https://doi.org/10.1007/978-1-4613-8431-1

F. Zoccheddu, Enrico Gobbetti, Marco Livesu, Nico Pietroni, and Gianmarco Cherchi. 2023. HexBox: Interactive Box Modeling of Hexahedral Meshes. *Comput. Graph. Forum* 42, 5 (2023), i–viii. https://doi.org/10.1111/CGF.14899