Finding Hexahedrizations for Small Quadrangulations of the Sphere

KILIAN VERHETSEL, Université catholique de Louvain, Belgique
JEANNE PELLERIN, Université catholique de Louvain, Belgique
JEAN-FRANÇOIS REMACLE, Université catholique de Louvain, Belgique

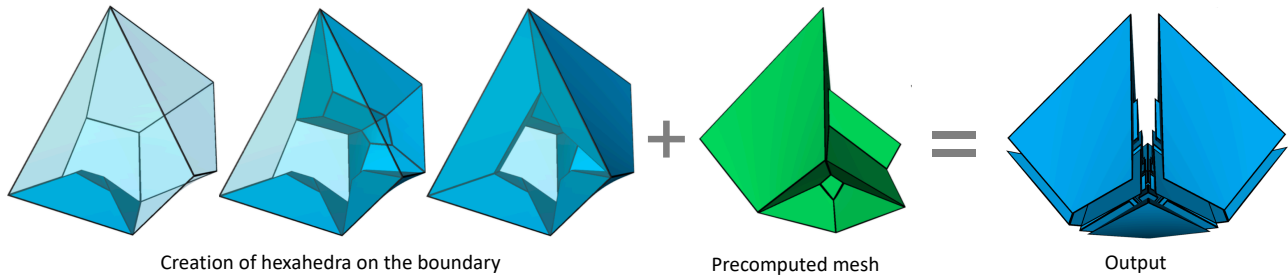Creation of hexahedra on the boundary   Precomputed mesh   Output

Fig. 1. Given a quadrangulation of the topological sphere, our algorithm creates hexahedra on the boundary until the unmeshed cavity matches the boundary of a pre-computed hex mesh that is merged to obtain the final combinatorial hexahedral mesh.

This paper tackles the challenging problem of constrained hexahedral meshing. An algorithm is introduced to build combinatorial hexahedral meshes whose boundary facets exactly match a given quadrangulation of the topological sphere. This algorithm is the first practical solution to the problem. It is able to compute small hexahedral meshes of quadrangulations for which the previously known best solutions could only be built by hand or contained thousands of hexahedra. These challenging quadrangulations include the boundaries of transition templates that are critical for the success of general hexahedral meshing algorithms.

The algorithm proposed in this paper is dedicated to building combinatorial hexahedral meshes of small quadrangulations and ignores the geometrical problem. The key idea of the method is to exploit the equivalence between quad flips in the boundary and the insertion of hexahedra glued to this boundary. The tree of all sequences of flipping operations is explored, searching for a path that transforms the input quadrangulation Q into a new quadrangulation for which a hexahedral mesh is known. When a small hexahedral mesh exists, a sequence transforming Q into the boundary of a cube is found; otherwise, a set of pre-computed hexahedral meshes is used.

A novel approach to deal with the large number of problem symmetries is proposed. Combined with an efficient backtracking search, it allows small shellable hexahedral meshes to be found for all even quadrangulations with up to 20 quadrangles. All 54, 943 such quadrangulations were meshed using

no more than 72 hexahedra. This algorithm is also used to find a construction to fill arbitrary domains, thereby proving that any ball-shaped domain bounded by n quadrangles can be meshed with no more than 78 n hexahedra. This very significantly lowers the previous upper bound of 5396 n.

Authors' addresses: Kilian Verhetsel, Université catholique de Louvain, Avenue Georges Lemaître 4-6, Louvain-la-Neuve, 1348, Belgique, kilian.verhetsel@uclouvain.be; Jeanne Pellerin, Université catholique de Louvain, Avenue Georges Lemaître 4-6, Louvain-la-Neuve, 1348, Belgique, jeanne.pellerin@total.com; Jean-François Remacle, Université catholique de Louvain, Avenue Georges Lemaître 4-6, Louvain-la-Neuve, 1348, Belgique, jean-francois.remacle@uclouvain.be.

1 INTRODUCTION

Volumetric mesh generation is a required step for engineering analysis. Robust algorithms are able to automatically produce a tetrahedral mesh constrained to have a given triangulation as its boundary, e.g. [Si 2015]. However, subdivisions into hexahedra (cube-like cells) are often preferred over tetrahedrizations for their good numerical properties such as a better convergence with fewer elements [Shepherd and Johnson 2008] and faster assembly times [Remacle et al. 2016]. Yet, the hexahedral meshing problem, and more particularly the boundary constrained variant, remains open to this date.

Finding solutions to the boundary constrained hex-meshing problem is crucial for hex-meshing algorithms that use a few simple templates to reduce the complexity of the general meshing problem to a small set of inputs (*e.g.* [Mitchell 1999; Yamakawa and Shimada 2002]). More importantly, hex-dominant mesh generation techniques usually leave small cavities unmeshed [Yamakawa and Shimada 2003] and filling them is one of the missing pieces to the more general problem of all-hex mesh generation.

This paper introduces an algorithm that solves the combinatorial constrained hex-meshing problem for small quadrangulation of
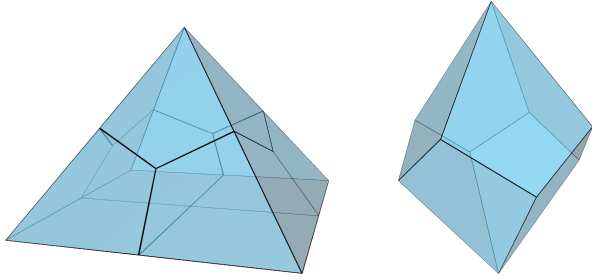
Fig. 2. Schneiders' pyramid [Schneiders 1996] and the tetragonal trapezohedron are two extremely challenging boundaries for combinatorial constrained hexahedral mesh generation.

the sphere (Figure 1). Given a quadrangulation of the sphere Q, it determines a set of combinatorial cubes H such that:

(1) the intersection of any two hexahedra $h_1, h_2 \in H$ is a combinatorial face shared by h_1 and h_2 (*i.e.* the empty set, a vertex, an edge, or a quadrangle);
(2) all quadrangular faces are shared by at most two hexahedra; and
(3) the set of boundary quadrangle faces (adjacent to exactly one hexahedron) is equal to Q.

This is an extremely challenging problem, even when the subsequent problem of finding a geometrical embedding is ignored, and for which no practical method exists. The existence of hexahedral meshes for all even quadrangulation of the topological sphere has been proven by Mitchell [1996], yet seemingly innocuous quadrangulations such as the 16-quadrangle pyramid (Schneider's pyramid) or the 8-quadrangle tetragonal trapezohedron (Figure 2) are notorious failure cases of general purpose meshing methods.

Eppstein [1999a] shows how the interior of a quadrangulated sphere can be meshed with a linear number of hexahedra; this construction was later generalized to all domains which admit hexahedral meshes [Erickson 2014]. Both methods reduce the problem to meshing a few quadrangulated spheres, but neither provide explicit hex meshes for these cases. Previous attempts to mesh these templates have been unsuccessful [Mitchell 2002; Weill and Ledoux 2016].

One method only is able to generate hexahedral meshes for the templates of Eppstein and Erickson [Carbonera and Shepherd 2010]. The drawback is that it requires 5396 n hexahedra to construct a non-degenerate hexahedral mesh of a ball bounded by n quadrangles.

Contributions. The first contribution of this paper is a practical algorithm to build combinatorial hexahedral meshes of reasonable size for small quadrangulation of the sphere. 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 3). 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 4).

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 of this work is to significantly lower the upper bound needed to mesh arbitrary domains. The construction of Erickson 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.

An implementation of all algorithms introduced in this paper is provided as free software and can be found in the supplementary materials or from https://www.hextreme.eu.

2 RELATED WORK

Hexahedral mesh generation is a thriving field of research, with a variety of proposed methods. These include multi-block decomposition methods using frame-field parametrizations [Kowalski et al. 2014; Liu et al. 2018; Lyon et al. 2016; Nieser et al. 2011], hex-dominant meshing methods [Baudouin et al. 2014; Gao et al. 2017; Pellerin et al. 2018; Sokolov et al. 2017; Yamakawa and Shimada 2003], octree-based methods [Ito et al. 2009; Maréchal 2009; Qian and Zhang 2010; Zhang et al. 2012], and polycube-based methods [Fang et al. 2016; Gregson et al. 2011; Han et al. 2011; Yu et al. 2014].

A detailed overview of these various approaches is beyond the scope of this paper, as most methods do not address the problem of generating meshes with a given boundary quadrangular mesh. The rest of this section focuses on methods tackling the constrained problem.

2.1 Existence proofs

Existence theorem for ball inputs. Thurston [1993] and Mitchell [1996] independently showed that a 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. The proof is based on the dual cell complex of quadrangular and hexahedral meshes. The dual complex of a quadrangular mesh is obtained by placing a vertex at the center of each quadrangle and adding edges between the vertices corresponding to adjacent quadrangles. Grouping edges traversing opposite edges of a same quadrangle, the dual complex is interpreted as an arrangement of curves (Figure 3). Similarly, the dual of a hexahedral mesh can be interpreted as an arrangement of surfaces [Murdoch et al. 1997]. 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
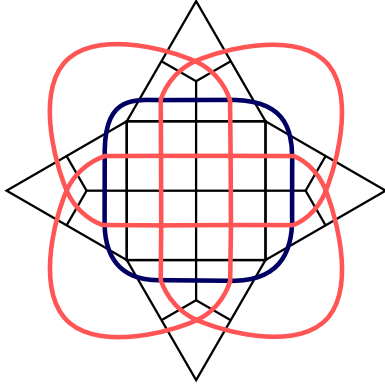
Fig. 3. The dual of a quadrangulated mesh (black edges) can be seen as an arrangement of two curves. The dark blue curve is simple, the red curve intersects itself 8 times. This example mesh is Schneider's pyramid.
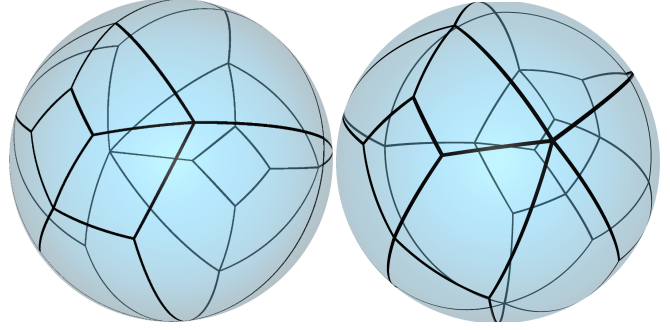

Fig. 4. If hexahedrizations of these two quadrangulated spheres exists, then it is possible to construct hexahedrizations for all other quadrangulated surface [Erickson 2014].

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.

Linear-complexity meshing. The construction of Mitchell can necessitate 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. 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 4) into hexahedra, which is again shown to be possible from Mitchell's proof. Like Eppstein, Erickson does not give an explicit construction of the hexahedral meshes of the buffer cells that are the base of its proofs.

A constructive method. 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-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.

2.2 Constrained hexahedral meshing in practice

The methods in the previous section are impractical, they create far more hexahedra than necessary. In practice, less general methods have been used to obtain smaller meshes.

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. 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 (e.g. Figure 3).

2.3 Searching for hexahedral meshes

Some specific cases have particularly attracted the attention of the research community: Schneiders' pyramid [Schneiders 1995] and the octogonal spindle (Figure 2). The *ad-hoc* constructions for the pyramid proposed Yamakawa and Shimada [2002, 2010] have long been the smallest known solutions. Recently, computer searches have been used to find substantially smaller solutions.

Verhetsel et al. [2018] exhaustively explore the space of all possible hexahedral meshes up to a given number of vertices by considering the possible groups of 8 vertices that can be built without creating an invalid mesh. The search space is usually too large for a solution to be found except in fairly simple cases. Starting from the 88-element solution of Yamakawa and Shimada and successively coarsening small parts of the original mesh, the method of Verhetsel et al. allowed the construction of a 44-element hex mesh of the pyramid.

By considering a smaller search space, Xiang and Liu [2018] construct a mesh of the pyramid with 36 hexahedra by building a shelling of the resulting hexahedral mesh. Starting from a single cube, the algorithm of Xiang and Liu considers all possible ways to add one hexahedron while maintaining a mesh which is both valid and combinatorially equivalent to a topological ball. The process is stopped when the boundary of the mesh matches the target quadrangulation.

3 EXHAUSTIVE SEARCH

3.1 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 1). 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 4, 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 (subsection 3.2). This space is explored in its entirety by considering all possible sequences of quad flips that correspond to valid hexahedral meshes (subsection 3.3). 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 (subsection 3.4).

3.2 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

---

ALGORITHM 1: SEARCH: Enumerate shellable hexahedral meshes

Input: Q: A quadrangulation of the sphere;
H: A partial mesh;
$H_{\max}$: maximum number of hexahedra in a solution;
$V_{\max}$: maximum number of vertices in a solution.
if $H_{\max} = |H|$ **then return**;
else if VISITED-SYMMETRIC-COUNTERPART(H) **then**
  return ;   // subsection 3.4
else if $Q \approx a\ cube$ **then**
  $h \leftarrow$ the hexahedron bounded by Q ;   // subsection 3.3
  if IS-COMPATIBLE(H, h) **then** OUTPUT-SOLUTION($H \cup \{h\}$);
end
foreach *quad flip F* **do**
  $(Q', h) \leftarrow$ PERFORM-FLIP(F, Q) ;   // subsection 3.3
  if NUM-VERTICES($H \cup \{h\}$) $\geq V_{\max}$ **then continue**;
  if IS-COMPATIBLE(H, h) **then**
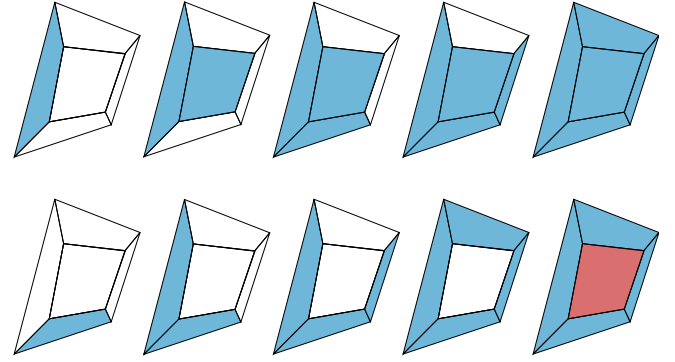   SEARCH($Q', H \cup \{h\}, H_{\max}, V_{\max}$);
  end
end

Fig. 5. (top) a shelling of a quadrangulation; (bottom) not a shelling because a hole is present after inserting the first four quadrangles.

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 5).

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 6). 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 hexahedral meshes admit a shelling order — see for example Furch's ball [Furch 1924]. 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].
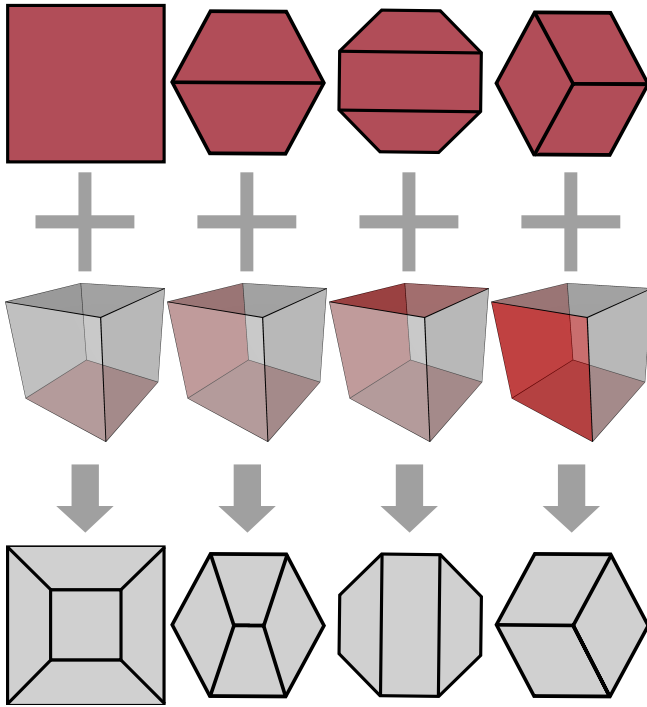
Fig. 6. Equivalence between quad flips and hex creation. Adding one hexahedron glued to the top row red quadrangles, modifies locally the quads and results in the bottom row grey quadrangles. This operation is the key idea of the algorithm to search shellable meshes.

3.3 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 (this is one difference with the search method of Xiang and Liu [2018]). 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 6. 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, 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 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.

3.4 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 [Brinkmann and McKay 2007; Burton 2011]. 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 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:

ALGORITHM 2: COMPUTE-SYMMETRY: 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)
Initialize σ, mapping q_0 to (x, y, z, w);
Initialize a queue with the 4 edges of q_0;
VISITED$_A \leftarrow \{q_0\}$;
VISITED$_B \leftarrow \{(x, y, z, w)\}$;
SEEN $\leftarrow \{q_0\}$;
while *the queue is not empty* **do**
   Dequeue an edge (a, b);
   $q \leftarrow$ the quadrangle containing (a, b) and not in VISITED$_A$;
   $q' \leftarrow$ the quadrangle containing $\sigma(a, b)$ and not in VISITED$_B$;
   foreach *vertex v in q* **do**
      $v' \leftarrow$ the corresponding vertex in q';
      if $\sigma(v)$ *is undefined and* $\sigma^{-1}(v')$ *is undefined* **then**
         $\sigma(v) \leftarrow v'$;   // Extend the map σ
         $\sigma^{-1}(v') \leftarrow v$;
      else if $\sigma(v) \neq v'$ *or* $\sigma^{-1}(v') \neq v$ **then**
         fail ;   // Stop upon contradiction
      end
   end
   foreach *edge e of q* **do**
      $o \leftarrow$ the quadrangle on the other side of e;
      if *o has not been seen before* **then**
         SEEN $\leftarrow$ SEEN $\cup \{o\}$;
         Enqueue e;
      end
   end
   VISITED$_A \leftarrow$ VISITED$_A \cup \{q\}$;
   VISITED$_B \leftarrow$ VISITED$_B \cup \{q'\}$;
end
return σ;

---

(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.

3.4.1 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 7): the quadrangles adjacent to q_A must
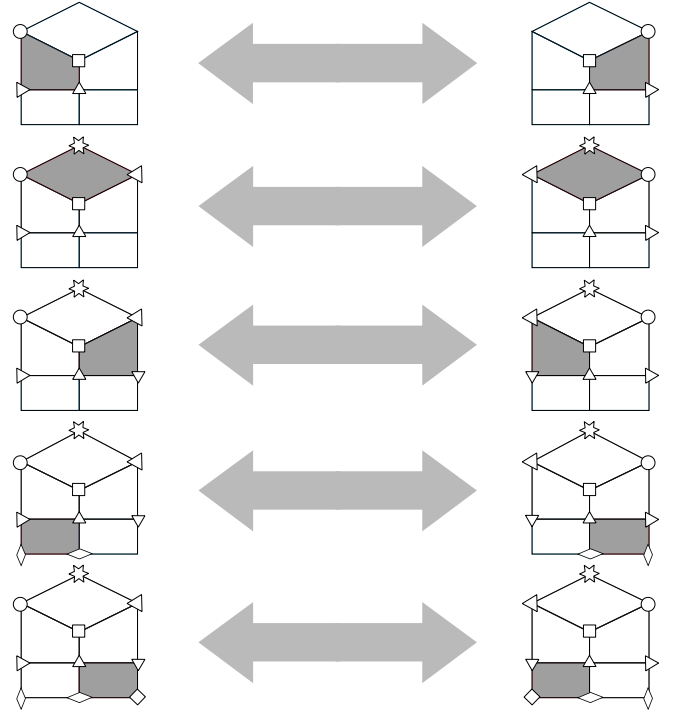

Fig. 7. Computation of a symmetry. Starting from the assumption that a quadrangle is the image of some other quadrangle, the mesh is traversed while computing the correspondence between all other vertices.

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 2). This process is well-defined because each edge is in at most two quadrangles.

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 [Colbourn and Booth 1981; Eppstein 1999b] 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
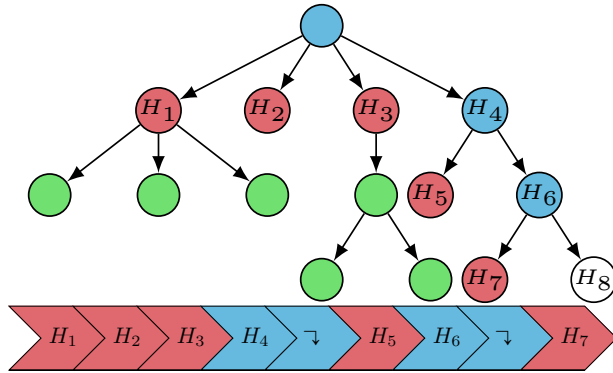
Fig. 8. A partially explored search tree and the sequence used to compare the current node (in white) against the previously explored part of the tree. No-goods are shown in red, and partially explored subtrees in blue.

variant of this method will be used to compare hexahedral meshes in section 3.4.3, by having quadrangles take over the role of edges in algorithm 2.

3.4.2 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 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 8). 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.

3.4.3 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 8.

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 3). The process to find such a partial mesh is similar to the algorithm used to compute the automorphism group initially (section 3.4.1). 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 3 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 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).

If such a hexahedron h' exists, it must be the symmetric counterpart of the hexahedron h encoded in S, and the map σ is extended accordingly. If h corresponds to a fully explored node (shown in red in Figure 8), the current partial solution H contains the symmetric counterparts of all hexahedra of the corresponding no-good. If, however, h corresponds to a partially explored node (shown in blue in Figure 8), and its symmetric counterpart cannot be found, the traversal ends early because all subsequent partial meshes encoded in S contain h, which is not in the current solution. In all other cases, the traversal of the sequence continues.

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 4).

ALGORITHM 3: CONTAINS-NO-GOOD: 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 8); 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
SEEN $\leftarrow \emptyset$;
foreach *hexahedron* $h \in S$ **do**
    SUCCESS $\leftarrow$ **true**;
    $q \leftarrow$ a quadrangle of h whose image through σ is known;
    $q' \leftarrow \sigma(q)$;
    $h' \leftarrow$ a hexahedron in H containing q' and not in SEEN;
    if *there is such a hexahedron* h' **then**
        $(\sigma_{old}, \sigma_{old}^{-1}) \leftarrow (\sigma, \sigma^{-1})$;
        foreach *vertex* v *of the quadrangle of* h *opposite to* q **do**
            $v' \leftarrow$ the corresponding vertex in h';
            if $\sigma(v)$ *is undefined and* $\sigma^{-1}(v')$ *is undefined* **then**
                $\sigma(v) \leftarrow v'$;   // Extend the map σ
                $\sigma^{-1}(v') \leftarrow v$;
            else if $\sigma(v) \neq v'$ *or* $\sigma^{-1}(v') \neq v$ **then**
                SUCCESS $\leftarrow$ **false**;
                break ;   // Stop upon contradiction
            end
        end
        if SUCCESS **then**
            SEEN $\leftarrow$ SEEN $\cup \{h'\}$
        else
            $(\sigma, \sigma^{-1}) \leftarrow (\sigma_{old}, \sigma_{old}^{-1})$;
        end
    else
        SUCCESS $\leftarrow$ **false**;
    end
    if *the symbol after* h *in* S *is the branching symbol* **then**
        /* All subsequent no-goods contain h. The search
            is aborted if its symmetric counterpart is
            not present.   */
        if SUCCESS *is false* **then return** *false*;
    else if SUCCESS **then**
        return *true* ;   // No-good is contained in H
    end
end
return *false*;

ALGORITHM 4: TRY-REVERSE-SHELL: 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.
if $H = \emptyset$ **then return** *true*;
foreach $h \in H$ **do**
    if *h can be added by performing a quad flip or Q is a cube* **then**
        $Q' \leftarrow$ the boundary after removing h;
        if TRY-REVERSE-SHELL($Q', H \setminus \{h\}, M$) **then**
            return *true*;
        else
            Increment the number of tested permutations;
        end
    end
    if *M permutations or more have been tested* **then**
        return *false*;
    end
end
return *false*;

---

4 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 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. 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 (subsection 4.1), this 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.

4.1 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 two quadrangulations is performed using a variation on algorithm 2 where the two starting quadrangles are part of different quadrangulations.

ALGORITHM 5: GENERATE-SHELLINGS: 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.

$S \leftarrow \emptyset$;
$\mathcal{H} \leftarrow \emptyset$;
$Q \leftarrow$ NEW-QUEUE();
ENQUEUE(Q, CUBE);
while Q *is not empty* **do**
    $H \leftarrow$ DEQUEUE(Q);
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{H\}$;
    if $|H| = n$ **then continue**;
    foreach *quad flip* F **do**
        $(B, h) \leftarrow$ PERFORM-FLIP($F, \partial H$);
        if *IS-COMPATIBLE*(H, h) $\wedge B \notin S$ **then**
            $S \leftarrow S \cup \{B\}$;
            ENQUEUE(Q, $H \cup \{h\}$);
        end
    end
end
return $\mathcal{H}$;

The signature used by our algorithm is a histogram of the valences 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 5), we obtain a greedy construction similar to the procedure 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 5, algorithm 1 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 5, a table of $69,043,690$ boundaries that can be meshed with up to 11 hexahedra is constructed (Table 1).

4.2 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.

Table 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.

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

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 9). algorithm 1 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 RESULTS

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].
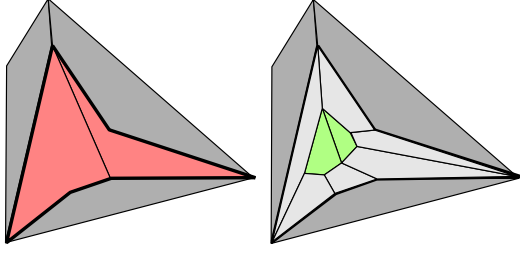
Fig. 9. Principle of the insertion of a buffer layer to guarantee final mesh validity when using precomputed cavity meshes. (left) Simply adding two quadrangles (red) inside a cavity creates an invalid mesh where pairs of quads share more than one facet; (right) inserting a layer of buffer quads (light gray) allows the use of the same (combinatorial) quads to fill the cavity and produce a valid mesh.

Using our search algorithm (section 4), we found hexahedral meshes for both types of buffer cells that Erickson's construction needs, along with geometric realizations using linear hexahedra (Figure 10) obtained by applying existing mesh untangling techniques [Livesu et al. 2015; Toulorge et al. 2013], although the solutions have a very low minimum scaled Jacobian (Table 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 hexahedra respectively, giving the following result:

THEOREM 5.1. *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 5.1 is used to compute the number of hexahedra to mesh the rest of the domain.

Hexahedrizations for small quadrangulations of the sphere. We used the algorithm described in section 4 to compute hexahedrizations for all even quadrangulations of the sphere containing up to 20 quadrangles (Table 3). The 54, 943 input quadrangulations were generated using plantri [Brinkmann et al. 2005]. We precomputed 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 11). 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 13). The trapezohedron bounded by n faces, obtained by generalizing the tetragonal trapezohedron of Figure 2, 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, the smallest solution found for the 20-face decagonal trapezohedron contained 72 hexahedra, strictly more than any of the other boundaries (Figure 12). 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 13, it took 6h 15min before a 58-element mesh was found.

The quadrangulations of Figure 2 are not particularly difficult to mesh using our method. Indeed, most quadrangulations of up to 18 quadrangles require more computation time than those two cases — and even finding the smallest known solutions is orders of magnitude easier than finding any solutions for the cases shown on Figure 13. On a 4-core Intel® Core™ i7-7700HQ CPU, after precomputing a list of shellable meshes with up to 10 hexahedra, the 36-element mesh of Schneiders' pyramid originally found by Xiang and Liu [2018] is found within 3 seconds of search. A 44-element mesh of the tetragonal trapezohedron is found within 4 seconds and it takes 31 seconds to find the smallest known 40-element mesh constructed in [Verhetsel et al. 2018]. Statistics for the geometric realizations found for both meshes are shown on Table 2.

6 CONCLUSIONS

While existence proofs for solutions to the constrained hexahedral meshing problem have long existed, previously known methods to construct hexahedral meshes have required very large meshes even for small quadrangulations; this paper shows that a much lower theoretical bound exists, and provides tools to search for much smaller hexahedral meshes.

Cases previously thought of as difficult and which motivated research about this question are solved in a matter of seconds using our techniques. This research, by allowing hexahedral meshes to be computed for any small quadrangulation of the sphere, opens up a wide array of new possibilities. It is an important step for hex-dominant meshing [Yamakawa and Shimada 2003], as this solves
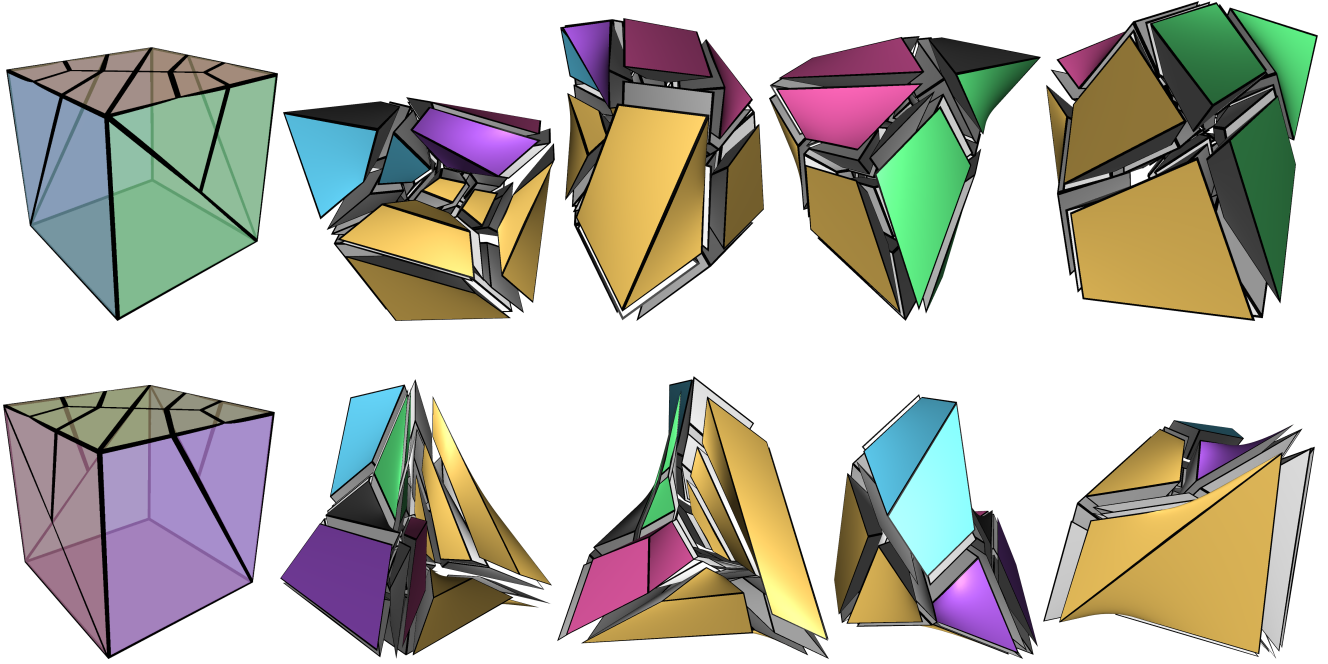
Fig. 10. Hexahedrizations of the two types of buffer cubes used to mesh arbitrary domains in the algorithm of Erickson [2014]. (top) 37 hexahedra to mesh the 20-quadrangle cell; (bottom) 40 hexahedra to mesh the 22-quadrangle cell. Colors correspond to the different sides of the original cubes (shown on the left).

Table 2. Statistics for the geometric meshes computed for the test cases of Figures 2 and 4

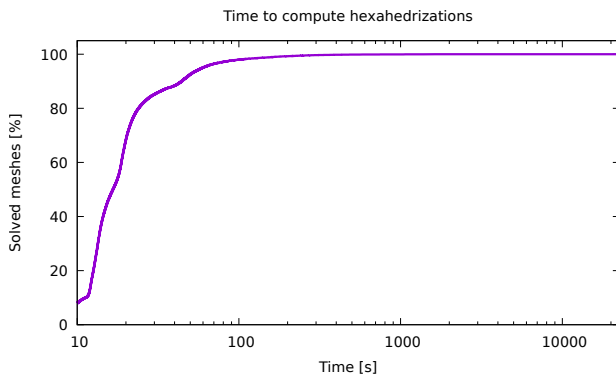| Template	$|Q|$	$|V_{bnd}|$	$|H|$	$|V_{total}|$	# edges per valence			Scaled Jacobian		
					3	4	5	min	max	median
Tetragonal trapezohedron	8	10	40	52	40	75	4	0.35	0.42	0.38
Schneiders' pyramid	16	18	36	51	32	62	4	0.12	0.49	0.26
Erickson's buffer cell (1)	20	22	37	53	43	49	4	0.31	0.63	0.42
Erickson's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

Fig. 11. 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).
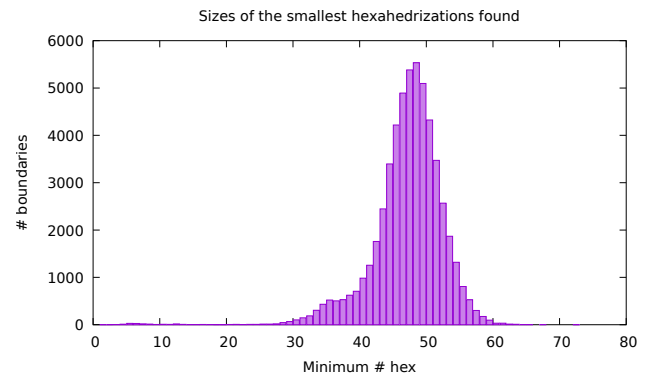

Fig. 12. Sizes of the smallest hexahedrizations found for all quadrangulations of the sphere with up to 20 quadrangles.
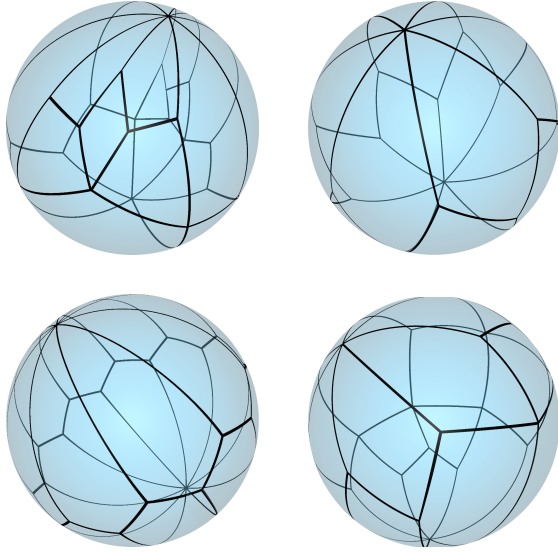
Fig. 13. The four most time-consuming quadrangulated spheres to mesh using our method. Each required over an hour of computation time on 64 cores.

Table 3. Statistics for the combinatorial meshes computed for all even quadrangulations of the sphere with up to 20 quadrangles.

Q	$\|H\|$			$\|V_{\text{total}}\|$			%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

the combinatorial aspect of the problem of filling the cavities that those methods leave. This method also offers new insights into the structure of block decompositions for configurations for which state-of-the-art techniques such as [Liu et al. 2018] fail to generate a valid block structure, by allowing combinatorial meshes to be computed in some of those cases.

ACKNOWLEDGMENTS

REFERENCES

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

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

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

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

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

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

Xianzhong Fang, Weiwei Xu, Hujun Bao, and Jin Huang. 2016. All-hex meshing using closed-form induced polycube. *ACM Trans. Graph.* 35, 4 (2016), 124:1–124:9. https://doi.org/10.1145/2897824.2925957

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

Robert Furch. 1924. Zur grundlegung der kombinatorischen topologie. *Abh. Math. Sem. Univ. Hamburg* 3, 1 (1924), 69–88. https://doi.org/10.1007/BF02954617

Xifeng Gao, Wenzel Jakob, Marco Tarini, and Daniele Panozzo. 2017. 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

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

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

James Gregson, Alla Sheffer, and Eugene Zhang. 2011. All-hex mesh generation via volumetric polycube deformation. *Comput. Graph. Forum* 30, 5 (2011), 1407–1416. https://doi.org/10.1111/j.1467-8659.2011.02015.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.02015.x

Shuchu Han, Jiazhi Xia, and Ying He. 2011. Constructing hexahedral shell meshes via volumetric polycube maps. *Computer-Aided Design* 43, 10 (2011), 1222–1233. https://doi.org/10.1016/j.cad.2011.06.023

Yasushi Ito, Alan M. Shih, and Bharat K. Soni. 2009. Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates. *Internat. J. Numer. Methods Engrg.* 77, 13 (2009), 1809–1833. https://doi.org/10.1002/nme.2470 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.2470

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

N. Kowalski, F. Ledoux, and P. Frey. 2014. Block-structured Hexahedral Meshes for CAD Models Using 3D Frame Fields. *Procedia Engineering* 82 (2014), 59–71. https://doi.org/10.1016/j.proeng.2014.10.373

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 (2015), 141:1–141:11. https://doi.org/10.1145/2766905

Max Lyon, David Bommes, and Leif Kobbelt. 2016. HexEx: robust hexahedral mesh extraction. *ACM Trans. Graph.* 35, 4 (2016), 123:1–123:11. https://doi.org/10.1145/2897824.2925976

Loïc Maréchal. 2009. Advances in octree-based all-hexahedral mesh generation: Handling sharp features. In *Proceedings of the 18th International Meshing Roundtable*. 65–84. https://doi.org/10.1007/978-3-642-04319-2_5

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. 1999. The all-hex geode-template for conforming a diced tetrahedral mesh to any diced hexahedral mesh. *Eng. Comput. (Lond.)* 15, 3 (1999), 228–235.

Scott A. Mitchell. 2002. A Technical History of Hexahedral Mesh Generation. https://www.sandia.gov/~samitch/_assets/documents/pptx/hex_mesh_history_samitch.ppt. Retrieved January 2, 2019.

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.

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

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

Jeanne Pellerin, Amaury Johnen, Kilian Verhetsel, and Jean-François Remacle. 2018. 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

Jin Qian and Yongjie Zhang. 2010. Sharp feature preservation in octree-based hexahedral mesh generation for CAD assembly models. In *Proceedings of the 19th International Meshing Roundtable*. 243–262. https://doi.org/10.1007/978-3-642-15414-0_15

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.

Robert Schneiders. 1995. Open problem. http://www.robertschneiders.de/meshgeneration/open.html. Retrieved January 2, 2019.

Robert Schneiders. 1996. A grid-based algorithm for the generation of hexahedral element meshes. *Eng. Comput. (Lond.)* 12, 3–4 (Sep 1996), 168–177. https://doi.org/10.1007/BF01198732

Jason F. Shepherd and Chris R. Johnson. 2008. Hexahedral mesh generation constraints. *Eng. Comput. (Lond.)* 24, 3 (Sept. 2008), 195–213. https://doi.org/10.1007/s00366-008-0091-4

Hang Si. 2015. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.* 41, 2 (2015), 11:1–11:36. https://doi.org/10.1145/2629697

Dmitry Sokolov, Nicolas Ray, Lionel Untereiner, and Bruno Lévy. 2017. Hexahedral-dominant meshing. *ACM Trans. Graph.* 36, 4 (2017). https://doi.org/10.1145/3072959.3126827

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/(SICI)1097-0207(19961015)39:19<3327::AID-NME2>3.3.CO;2-8

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

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

Kilian Verhetsel, Jeanne Pellerin, and Jean-François Remacle. 2018. A 44-element mesh of Schneiders' pyramid: Bounding the difficulty of hex-meshing problems. In *Proceedings of the 27th International Meshing Roundtable*. Springer.

Jean-Christophe Weill and Franck Ledoux. 2016. Towards an automatic and reliable hexahedral meshing. http://tetrahedron.montefiore.ulg.ac.be/weill.pdf. Retrieved January 2, 2019.

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. 2002. HEXHOOP: modular templates for converting a hex-dominant mesh to an all-hex mesh. *Eng. Comput. (Lond.)* 18, 3 (2002), 211–228. https://doi.org/10.1007/s003660200019

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, 12 (2010), 1700–1712.

Wuyi Yu, Kang Zhang, Shenghua Wan, and Xin Li. 2014. Optimizing polycube domain construction for hexahedral remeshing. *Computer-Aided Design* 46 (2014), 58–68. https://doi.org/10.1016/j.cad.2013.08.018

Yongjie Zhang, Xinghua Liang, and Guoliang Xu. 2012. A robust 2-refinement algorithm in octree and rhombic dodecahedral tree based all-hexahedral mesh generation. In *Proceedings of the 21st International Meshing Roundtable*. 155–172. https://doi.org/10.1007/978-3-642-33573-0_10

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