Engineering with Computers (2022) 38:4731–4747
https://doi.org/10.1007/s00366-022-01740-4

ORIGINAL ARTICLE

Indirect all-quadrilateral meshing based on bipartite topological labeling

Christos Georgiadis[1] · Maxence Reberol[1] · Jean-François Remacle[1]

Received: 18 January 2022 / Accepted: 31 August 2022 / Published online: 7 October 2022

Abstract
Quadrilateral meshes offer certain advantages compared to triangular ones, such as reduced number of elements and alignment with problem-specific directions. We present a pipeline for the generation of quadrilateral meshes on complex geometries. It is based on two key components: robust surface meshing and efficient indirect conversion of a triangular mesh to an all-quad one. The input is a valid geometric surface mesh, i.e., a triangulation that accurately represents the geometry of the model. A right-angled triangular surface mesh is initially created by continuously modifying the input mesh while always preserving its topological validity. The main advantages of our local mesh modification-based approach are to (i) allow the generation of meshes that are globally aligned with a given direction field and (ii) to reliably handle non-manifold feature edges (in multi-volume models) and small features. The final quadrilateral mesh is obtained by merging pairs of triangles into quadrilaterals. We develop a novel bipartite labeling scheme in order to identify and correct inconsistent pairings. The procedure is based on local operations and is much more efficient than previous global strategies for tri-to-quad conversion. The whole pipeline is tested on a large number of models with diverse characteristics.

Keywords Surface meshing · Quadrilateral meshing · Bipartite labeling · Cross-fields

1 Introduction

Quadrilateral meshes are often preferable to triangular ones for numerical simulations. They have fewer elements for the same number of vertices, they are ideally capable of providing a block-structure and they can provide better alignment to geometric features, as well as to problem-specific features, providing better numerical behavior for specific physical phenomena (a typical example is the demand for structured and aligned boundary layers in Computational Fluid Dynamics). Yet, the automatic generation of quadrilateral meshes is still regarded as a challenging problem in mesh generation. Even though a lot of different approaches exist, there is not to date a conclusive method, analogous to triangular meshing which is considered highly mature and for which there exist robust algorithms based on strong mathematical foundations.

The purpose of this work is to address the problem of generating quadrilateral meshes for complex 3D models. We strive for generality in our approach; our input is simply a triangulation of the model. The triangulation can be an STL representation of the geometry, triangulation of scanned data or a mesh generated from a CAD model with standard meshing techniques (Fig. 1). The input meshes may be of bad quality and contain non-manifold feature edges. Our goal is to design a pipeline satisfying the following design goals: (i) robustness, i.e., guarantee of termination regardless of the complexity/bad quality of input data, (ii) feature preservation, i.e., the persistence of user-defined internal and boundary curves, (iii) high element quality, and (iv) efficiency, i.e., providing a quad meshing algorithm with a running time comparable to or faster than conventional tri-to-quad technologies.

Maxence Reberol and Jean-François Remacle are contributing authors.

✉ Christos Georgiadis
christos.georgiadis@uclouvain.be

Maxence Reberol
maxence.reberol@uclouvain.be

Jean-François Remacle
jean-francois.remacle@uclouvain.be

[1] Institute of Mechanics, Materials and Civil Engineering, Université Catholique de Louvain, Avenue Georges Lemaitre 4, 1348 Louvain-la-Neuve, Belgium

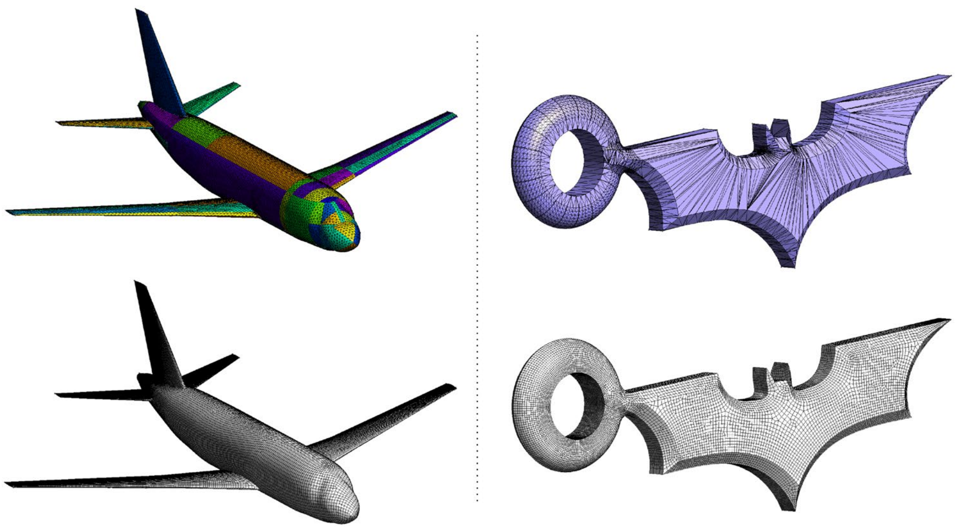

Fig. 1 Quadrilateral meshes produced by our algorithm with input a CAD model (left) and an STL triangulation (right)

To generate a high-quality unstructured quadrilateral mesh that preserves user-defined features, we develop three independent steps: (i) we sample points on the model curves and surfaces, guided from a metric field (cross-field and an associated size-map), (ii) we use local mesh modifications to continuously remove points of the initial triangulation and add the new ones, leading to a right-angled triangular mesh and (iii) we convert the triangular mesh into an all-quad mesh using a novel approach based on a topological labeling scheme.

When it comes to mesh generation, the expected reliability rate of industrial-grade algorithms is essentially 100%. The input data supplied to our meshing algorithms are often noisy. Inputs may be huge, with a wide range of scales. It is not surprising that the most complex/tricky part of our work is related to robustness. We have given special attention to ensure that this mesh generator provides results regardless of the complexity of the input data, as long as it is correct (i.e., a watertight but possibly non-manifold triangulation, in the sense that no folded elements and edge intersections may be present).

In order to ensure reproducibility of this work, the whole implementation will be available in Gmsh, the open-source mesh generator [1]. To demonstrate the robustness of our algorithm, we applied it to a large number of models found in various datasets.

1.1 Related work

Surface meshing Surface mesh generation poses various difficulties related to robustness and efficiency. Of the several methods proposed in the bibliography, we can identify two main categories [2, 3]: (i) *Parametric* approaches, where the surface mesh is generated in the parametric space, and (ii) *Nonparametric (direct)* approaches, where the surface mesh is generated directly in the 3D space.

In *parametric approaches*, the 3D surface is mapped to a 2D parametric space [2, 4–7]. Since the CAD surfaces (typically NURBS patches) have underlying *u*, *v* representation, it can be efficient to generate a mesh in the plane with standard meshing techniques and afterward map it back to the 3D space. Generating planar meshes in the parameter space is a robust approach that is usually able to provide high-quality meshes. Yet, approaches that use the parameter plane are able to consider surfaces that are isomorphic to a punctured disk. Meshing complex models with a parameter space approach do not allow to globally align a mesh with a cross-field, since each discrete patch of a CAD model may be equipped with an independent parametrization and the feature edges that separate those patches are not necessary aligned with the cross-field. Parametrization techniques can also be used to remesh triangulations [8, 9].

Nonparametric (also referred to as *direct surface meshing* in the literature, creating some ambiguity with the terms of *direct/indirect* approaches commonly used in quad meshing) approaches to surface meshing can be based on quad trees [10, 11], advancing front [12, 13] or Delaunay strategies [14, 15]. One of the main advantages of the direct approach is its usefulness for models where an underlying parametrization is not available or when it is degraded. Local mesh modification strategies to remesh models described by STL triangulations are proposed in Refs. [16, 17]. One of the main difficulties of this class of methods is that the manipulation of geometry directly on the 3D space is a challenging task that may lead to geometric or topological ambiguities.

Quadrilateral meshing Initial efforts to automatically generate quadrilateral meshes include *grid-based* and *paving* algorithms. Grid-based methods start with the generation of a background Cartesian or a quad-tree grid with the subsequent snapping of elements to the domain boundaries [18–20]. The paving algorithm, first introduced by Ref. [21] generates quadrilateral elements in an advancing-front fashion, propagating from the boundary to the interior. Both classes of methods suffer from a degraded quadrilateral quality and high node irregularity on specific regions of the domain: the latter on the domain boundaries and the former on the front collisions on the interior. In Ref. [22], a bichromatic Delaunay quadrangulation method is presented, with our current work building upon a similar concept.

On the other hand, quad conversion or indirect methods are based on the merging of pairs of adjacent triangles of an input mesh to quadrilaterals [23, 24]. In Q-Morph [25], triangles are transformed into quadrilaterals with an advancing-front algorithm. Blossom-Quad algorithm [26] computes a perfect matching to optimally pair triangles, while [27, 28] produce meshes better suited for triangle pairing by generating aligned right-angled triangles.

Cross-fields Cross-fields are nowadays commonly used in the context of mesh generation, a line of research stemming from the computer graphics community and global parametrization methods [29]. For quad/hex meshing, cross-fields define preferred orthogonal directions on the domain to guide creation of optimal elements [30–32]. Cross-fields should be as smooth as possible (except at singular points) and aligned with the boundary of the domain.

1.2 Contributions

The quad meshing pipeline that is proposed follows a modular approach, with each of the steps being an independent algorithm that can be re-used in various situations (Fig. 2). As stated before, our main concern is to provide a reliable solution, i.e., we want the quad meshing pipeline to be *resilient to complex/ill-conditioned inputs*.

The two main contributions of this work are:

1. **Robust surface meshing**. In our procedure, we follow the idea of Ref. [28] of separating the generation of points and the creation of the elements. The main novelty of this work is our direct approach. In this work, an input triangular mesh is *continuously modified* through robust local mesh modifications. The word continuous is chosen on purpose: each local mesh modification that is performed guarantees the topological integrity of the current triangular mesh. At the end of the remeshing process, most of the points of the initial triangular mesh, and in most cases all those points are removed from the triangulation and replaced by the ones created to accommodate the cross-field and size field characteristics.
2. **Straightforward and efficient all-quad meshing**. We propose a bipartite labeling scheme that propagates topology information on the vertices during point generation. By using this information on the right-angled triangular mesh, we are able to optimally place new Steiner points to fix topological inconsistencies (odd-bounded regions on the interior) and recover a bipartite, all-quad mesh. The approach is very efficient since it converts a global problem to a local one.

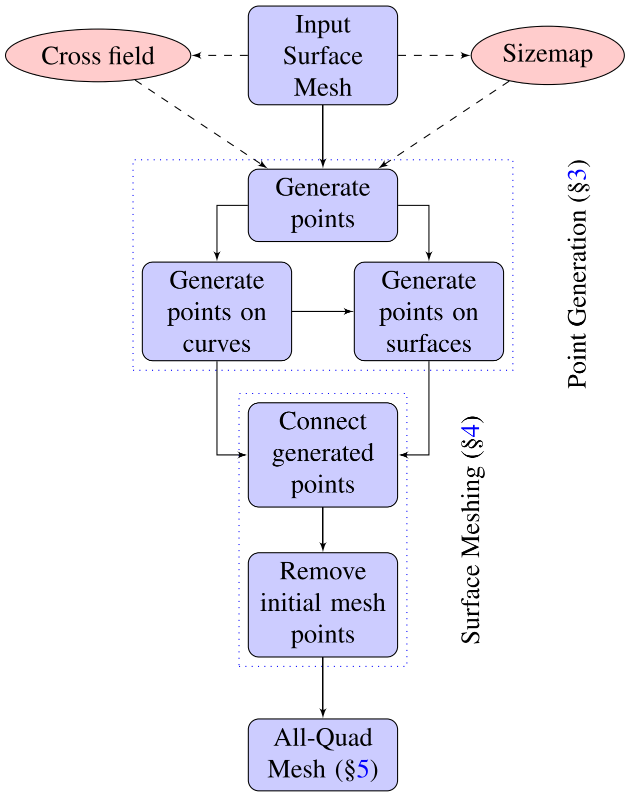

Fig. 2 Schematic outline of the pipeline proposed for quadrilateral meshing

2 Overview

Our algorithm takes as input a watertight, possibly non-manifold triangulation $\mathcal{T}_0$. The input triangulation $\mathcal{T}_0$ is *classified*: the word classified indicates the fact that the triangles are grouped into *colors* and interfaces between colors are considered as *feature edges* that must be conserved in the remeshing process. Internal feature edges may also exist that lie inside a group of triangles with the same color. Note that feature edges can also be detected based on the dihedral angle, given a user-defined threshold. We take into account two special categories of feature edges in $\mathcal{T}_0$: (i) non-manifold feature edges with two or more adjacent surfaces and (ii) internal (embedded) feature edges inside surfaces (e.g., a crack for solid modeling)

All the necessary topological information is available in the initial mesh. Surfaces are bounded by closed feature

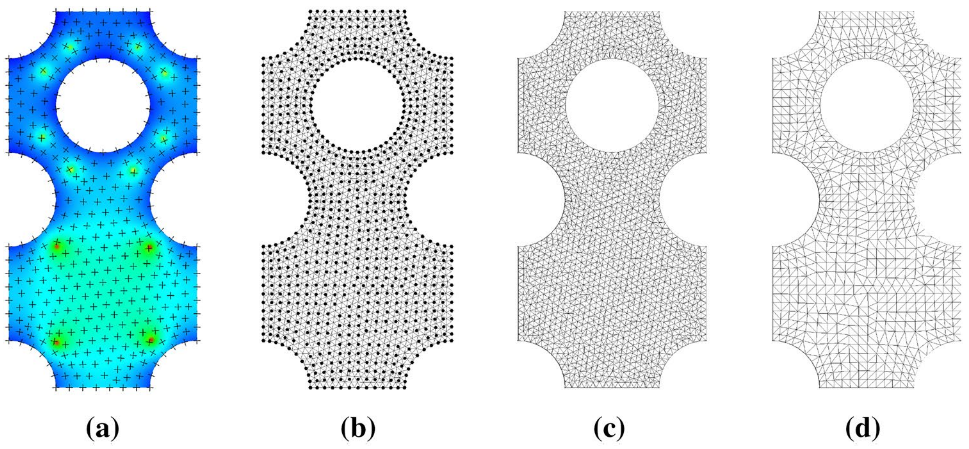

Fig. 3 Surface meshing steps. **a** Cross-field and size field computed on the input mesh. **b** Generated points embedded on the input mesh. **c** Inserting points to the triangulation with local mesh modifications. **d** Right-angled triangular mesh of the generated points

edges and those feature edges by feature points. Consecutive feature edges form *feature curves*. Feature curves must be preserved during the continuous mesh modification process. An important property of our method is the ability to handle the model as a whole and thus take advantage of the global nature of the guiding cross-field. We do not follow a patch-wise approach where we handle each surface independently, followed by a connection of curves to ensure conformity. A half-edge data structure [33] is used to get connectivity information, and an array of boundary edges that define feature curves is stored. We extend this data structure by storing the type of boundary edge (manifold or non-manifold), along with the triangles connected to each (one triangle for open boundary edges, two triangles for manifold edges, and a larger than two number for non-manifold ones). This feature enables us to efficiently treat boundaries during surface meshing.

The scheme provides the flexibility to preserve the topological characteristics of the input mesh, such as 'hard' edges or user-defined feature curves, without relying on extensive *a priori* knowledge of domain characteristics or a complicated feature recognition preprocess [34–37]. Furthermore, by utilizing an appropriate data structure on top of the half-edge one, we can handle non-manifold configurations that may occur in industrial multi-volume CAD models.

The input triangulation $\mathcal{T}_0$ is the geometric model. Two other inputs are required for running the algorithm: (i) a unit cross-field $\mathbf{f}$ and (ii) a size field $h(\mathbf{x})$ that are both used for guiding the point insertion process. A *cross-field* $\mathbf{c}$ is a field defined on a surface $\mathcal{S}$ with values in the quotient space S^1/Q, where S^1 is the circle group and Q is the group of quadrilateral symmetry. It associates to each point of a surface $\mathcal{S}$ to be meshed a cross made of two unit vectors orthogonal to one another in the tangent plane of the surface and their opposites (Fig. 3a). Although $\mathcal{T}_0$, $\mathbf{f}$ and h can be independent, it is beneficial to have (i) a cross-field $\mathbf{f}$ that is aligned with the feature edges of $\mathcal{T}_0$ and (ii) a size field h that takes into account both the local change of direction of the cross-field and small features of the geometry. In this work, $\mathbf{f}$ and h are precomputed using $\mathcal{T}_0$ as support with the algorithms described in Ref. [38].

The transformation of the triangulation $\mathcal{T}_0$ into the final quad mesh $\mathcal{T}_q$ is done in three sequential steps:

1. *Point generation* (Sect. 3) Points of the final mesh $\mathcal{T}_q$ are generated in a frontal fashion starting from the feature curves and guided by both the cross and the size field. This set of points is embedded on the triangles of the base/initial mesh $\mathcal{T}_0$.
2. *Point replacement* (Sect. 4) The initial mesh $\mathcal{T}_0$ is continuously transformed into another triangular mesh $\mathcal{T}_1$ by connecting the newly generated points on $\mathcal{T}_0$ and subsequently removing the initial mesh points, utilizing local mesh modifications.
3. *From triangles to quads* (Sect. 5) Mesh $\mathcal{T}_1$ is transformed into a quad mesh by combining pairs of triangles. Using a binary labeling scheme for the points during frontal generation allows us to instantly extract a valid all-quad topology.

The simple model of Fig. 3 is used as an example to describe those three steps.

3 Point generation

In this work, we take a standard *surface-to-volume* point of view of mesh generation on Gmsh [1] which essentially consists of a bottom-up procedure. Model curves are discretized at first. Mesh edges on the model curves are used as boundaries of model faces and mesh triangles on model faces are used as the boundary of model volumes if they exist.

A set P of points on surfaces is generated using a frontal point propagation algorithm that is similar to Ref. [28]. The main difference with Ref. [28] is that all the operations are performed here directly on $\mathcal{T}_0$ without using a parametric space. The point sampling scheme has been implemented for the special case of the sphere [39]. It is extended here for general surfaces and we reiterate the whole process for completeness. Our frontal approach is enhanced with the use of a cross-field **f** that allows to structure the quad mesh and a size field h that allows taking into account the various feature sizes of a model as well as changes of directions of **f** (Fig. 3a).

3.1 Curve point generation

Each feature curve is uniquely defined from a list of non-intersecting connected edges. Given a size field h defined by a value at each point, we mesh the discrete representation of each curve by following the general guidelines of Ref. [40]. This leads to the set of points $P_c = \{\mathbf{p}_i \mid i = 1, \dots, N_{gc}\}$. It is important to note that at this step, we can control the generation to have an even number of points for each feature curve. This provides us with a topologically necessary condition for all-quadrilateral meshing.

3.2 Surface point generation

Starting now from the N_{gc} generated points on mesh feature curves, we want to spawn a set of points $P_s = \{\mathbf{p}_i \mid i = 1, \dots, N_{gs}\}$ on the surfaces in the directions provided by the cross-field $\mathbf{f}(\mathbf{x})$ and with respect to the underlying size field $h(\mathbf{x})$. The point set P_s, along P_c, will be used to generate a right-angled triangulation $\mathcal{T}_1$ that is well suited for combining triangles into quadrilaterals and form $\mathcal{T}_q$.

The cross-field **f** gives $N_d = 2$ tangent orthogonal directions and their opposites. A priority queue is initially filled with the N_{gc} points ordered along the curves. The point $\mathbf{p}_i$ at the top of the queue then tries to insert 4 new points $\mathbf{p}_{ij}$ in the $j = 1, \dots, 2N_d$ directions defined by $\mathbf{f}\left(\mathbf{p}_i^j\right)$ and at a distance $h(\mathbf{p}_i)$. In order to have points inserted 'by layers,' the priority queue that is chosen is a first-in, first-out queue. Ordering the boundary points along the domain allows smooth propagation on the interior.

Each seed point $\mathbf{p}_i$ tries to spawn $\mathbf{p}_{ij}$, $j = 1, \dots, 2N_d$ neighbor points on $\mathcal{T}_0$. Yet, there is no guarantee that point $\mathbf{p}_{ij}$ is not too close to another point of the queue. Points $\mathbf{p}_{ij}$ are hence filtered. A rectangular exclusion zone is defined by the cross-field orientation and the size field around each vertex $\mathbf{p}_i$ in such a way that any point lying in this zone will not be inserted in the queue. Finally, seed points are removed from the queue, and accepted points are added to the end of the queue as well as in P. The procedure terminates when the queue is empty. Algorithm 1 describes the procedure. Following, we will focus in detail on the two main operations, i.e., the point insertion and the point filtering.

Algorithm 1 Frontal point insertion algorithm.

```
Input: Initial triangulation $\mathcal{T}_0$
       cross field $\mathbf{f}$
       mesh size field function $h(\mathbf{x})$
Output: Array of points $P$
 1: Place boundary points in a queue
 2: while queue is not empty do
 3:     pop the first point $\mathbf{p}_i$ out of the top of the queue
 4:     interpolate $\mathbf{f}$ and $h$ at this point
 5:     for $2N_d$ directions do
 6:         Compute point $\mathbf{p}_{ij}$ by intersecting $\mathcal{T}_0$ with a circle
 7:         Find set of neighboring points $P_f$
 8:         for $\mathbf{p}_f \in P_f$ do
 9:             if $\|\mathbf{p}_{ij} - \mathbf{p}_f\| > \alpha h(\mathbf{p}_{ij})$ then
10:                 add $\mathbf{p}_{ij}$ in $P$
11:                 push $\mathbf{p}_{ij}$ in the back of the queue
12:             else
13:                 delete $\mathbf{p}_{ij}$
14:             end if
15:         end for
16:     end for
17: end while
```

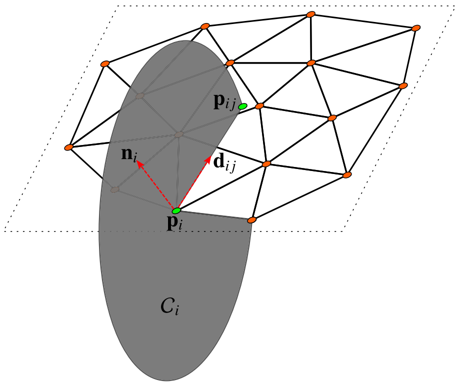

Fig. 4 Computation of point $\mathbf{p}_{ij}$, given a direction $\mathbf{d}_{ij}$, the normal $\mathbf{n}_i$ and the size h_i (the radius of the circle $\mathcal{C}_i$). With green color, we denote the generated points while with red the initial mesh ones. Here the seed point $\mathbf{p}_i$ coincides with a red point (color figure online)

3.2.1 Intersection with triangulation

Assume a point $\mathbf{p}_i$ that lies on one of the triangles of $\mathcal{T}_0$, a direction $\mathbf{d}_{ij} = \mathbf{f}\left(\mathbf{p}_i^j\right)$, i.e., a unit vector tangent to the surface and the mesh size $h_i = h(\mathbf{p}_i)$ at that point. The aim is to create an edge of size h_i. Therefore, the new point $\mathbf{p}_{ij}$ can be computed as the intersection of the triangulated surface $\mathcal{T}_0$ and a circle $\mathcal{C}_i$ with center $\mathbf{p}_i$ and radius h_i. $\mathcal{C}_i$ lies on the plane $\mathcal{P}_i$ that is formed by the direction vector $\mathbf{d}_{ij}$ and the normal to the triangulation at our origin point, $\mathbf{n}_i$ (Fig. 4).

To compute $\mathbf{p}_{ij}$ our goal is to find the intersection point of circle $\mathcal{C}_i$ with the triangulation $\mathcal{T}_0$ (Fig. 4). We start from the triangle of the base mesh $\mathcal{T}_{0i}$ on which $\mathbf{p}_i$ lies. First, we compute the intersection line of the plane $\mathcal{P}_i$ and the plane of the triangle $\mathcal{P}_{\mathcal{T}_{0i}}$. Then, we find the intersection points of this line with the circle $\mathcal{C}_i$ and choose the one that lies in direction $\mathbf{d}_{ij}$. Finally, the barycentric coordinates $\lambda_0, \lambda_1, \lambda_2$ of this point with respect to the current triangle are calculated. In this way, we determine whether the intersection point lies on the triangle, and therefore if we have a successful intersection with this triangle.

In the case where the current triangle is not intersected, we move forward to another triangle. Since we have already computed the barycentric coordinates with respect to the current triangle $\mathcal{T}_{0i}$, we know where on the plane $\mathcal{P}_{\mathcal{T}_{0i}}$ the intersection point lies (Fig. 5, left). The next triangle to be searched for an intersection is thus given from the computed barycentric coordinates and the adjacency information of the input mesh. This procedure continues until a valid intersection point is retrieved.

Essentially, we perform a walk in the triangulation [41] in the desired direction until we obtain the intersection point. Our experience shows that this method is efficient since it utilizes the underlying mesh as a space searching structure. For the same order of magnitude of mesh sizes on input and desired meshes, intersection points are found after a little less than two triangle visits on average.

3.2.2 Filtering procedure

Each point generates $\mathbf{p}_{ij}$ points for $j = 1, \dots, 2N_d$ directions. We have to ensure that new points are not too close to already generated points. Therefore, after each point $\mathbf{p}_{ij}$ is generated, a filtering procedure should follow. To this end, we use RTrees as a spatial search structure [42].

For every candidate point $\mathbf{p}_{ij}$, we define a large enough search box (typically 2 times the mesh size). We find then the set of points $P_f = \{\mathbf{p}_f, k = 1, \dots, n_f\}$ in the vicinity of $\mathbf{p}_{ij}$. Since our objective is to create right-angled triangles, i.e., equilateral triangles in the $\mathcal{L}_\infty$ norm, we compute the distance between the candidate point and its surrounding ones as $\|\mathbf{p}_{ij} - \mathbf{p}_f\|_\infty = \max\{|x_{ij} - x_k|, |y_{ij} - y_k|, |z_{ij} - z_k|\}$ The point is accepted for insertion if condition $\|\mathbf{p}_{ij} - \mathbf{p}_f\|_\infty > \alpha \cdot h(\mathbf{p}_{ij})$ holds for all $\mathbf{p}_f \in P_f$.

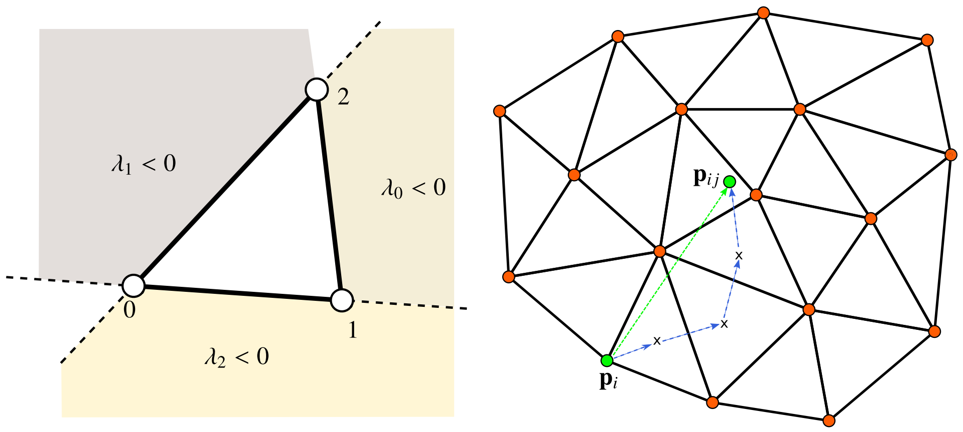

Fig. 5 Indication of the position of intersection point according to barycentric coordinates (left) and computation of point $\mathbf{p}_{ij}$ by walking in the triangulation in specific direction (right)

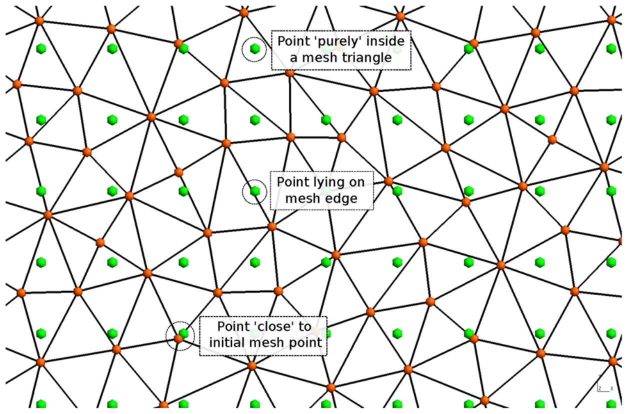
Point 'purely' inside a mesh triangle
Point lying on mesh edge
Point 'close' to initial mesh point

Fig. 6 Generated points (green)—initial mesh (red points to be removed) (color figure online)

4 Surface meshing

The objective now is to create a surface mesh with the set of optimal points $P = \{\mathbf{p}_i \mid i = 1, \dots, N_g\}$ (where $N_g = N_{gc} + N_{gs}$) that have been generated on curves and surfaces. The idea is straightforward: connect the generated points P on the initial mesh $\mathcal{T}_0$, and subsequently remove the initial mesh points (Fig. 6). A similar idea but in a different context has been used in Ref. [43], leading to a quad-dominant mesh.

Robustness is of crucial interest in our method, since modifying geometric aspects of general surfaces in 3D space is a delicate task. With our approach, the main goal is to preserve the topological integrity of the mesh through each step of the process, while not compromising the accuracy of the geometric representation of the surface.

We can identify two complementary sets of generated points $\mathbf{p}_i$:

i N_{gc} points lying on curves (feature edges)
ii N_{gs} points lying purely on surface triangles

with a corresponding unique parent element (feature edge or mesh triangle, respectively) already stored for each of these points. Correspondingly, there are N_r points from the initial mesh: N_{rc} points of the feature edges and N_{rs} points of the 'interior' surface. This division enables us to perform a bottom-up procedure where topological entities are handled independently (first mesh curves and then surfaces). We can therefore ensure that each step will have a well-defined 'predecessor' mesh to build upon.

4.1 Local mesh modifications

The basic operations utilized to locally modify the mesh follow:

Split triangles

Given a surface point and its parent triangle, split it by replacing it by three triangles. This operation is trivial to implement since it cannot change the geometry or the topology of the mesh.

Split edges

Given a point that lies on a mesh edge, split this edge. For points on boundary edges, we already know the parent edge, while for points on triangles it is easy to compute if the point lies on a triangle edge (given a user-defined threshold value ε). For non-manifold boundary edges, we split all the corresponding triangles connected to this edge (Fig. 7, left). This set of triangles is readily accessible from the extended data structure for boundary edges.

Collapse edges

Remove points from the triangulation by collapsing an edge. Collapsing an edge is not always a valid operation since it can create flipped or degenerate elements. We check the fan of n triangles connected to the vertex in question and choose an edge that can be collapsed. The resulting $n - 2$ triangles should not intersect with neighboring ones. Again, non-manifold edges can be collapsed if all corresponding 'half-fans' pass the validity test (Fig. 7, right).

Swap edges

Given an appropriate quality criterion, swap the edge if the topology is not violated. Obviously, a feature edge cannot be swapped. Edge swaps have a significant role during collapsing of edges, since it is not always feasible to collapse all unnecessary vertices at the first pass. Edge swaps serve at this point to create improved conditions for the next

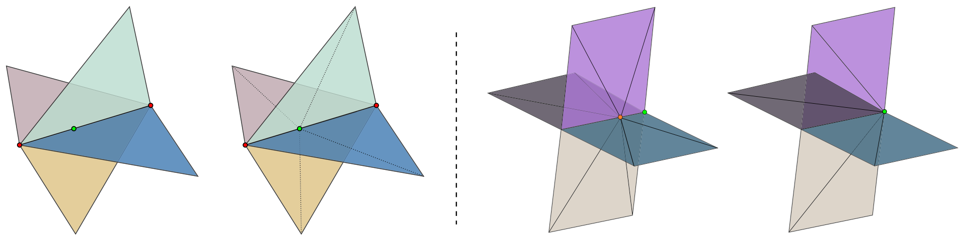

Fig. 7 Splitting (left) and collapsing (right) a non-manifold feature edge

collapsing iteration. Collapsing numerous mesh vertices leads to steep angles; therefore we swap edges if it does not result to bigger dihedral angles. At the final step, swaps can be performed based only on quality criteria.

4.2 Outline

A quality improvement with edge swaps can be performed here, though it is not a necessary condition to continue to the next step.

Algorithm 2 Direct Surface Meshing with Local Modifications.

```
Input: Initial triangulation $\mathcal{T}_0$
       generated set of points $P$ to be triangulated.
Output: New mesh $\mathcal{T}_1$
 1: for $N_{gc}$ points on curves do
 2:     Insert points on mesh curves by splitting feature edges
 3: end for
 4: for $N_{gs}$ points on surfaces do
 5:     Insert point on mesh surfaces by splitting edges or triangles
 6: end for
 7: Swap edges of intermediate mesh $\mathcal{T}_i$
 8: while $N_r \neq 0$ do
 9:     for $N_{rc}$ points on curves do
10:         Collapse initial points on mesh curves
11:     end for
12:     for $N_{rs}$ points on lines do
13:         Collapse initial points on mesh surfaces
14:     end for
15:     Swap edges
16: end while
```

The procedure consists of the following (Algorithm 2). All generated points are flagged to be inserted while all initial points are flagged to be removed. Starting from the model curves, each generated point splits its corresponding parent feature edge, which can be manifold or non-manifold. Each point has a unique parent feature edge and the splitting is done based on its parameter $t \in [0, 1]$, thus defined in an unambiguous way. If a point to be inserted is 'close' (w.r.t to ε) to an initial mesh point, we flag the latter to be preserved instead of the former in order to avoid small-scale geometric ambiguities that may occur. The procedure follows until all points on model curves are inserted into the mesh. Following, points on surfaces are inserted by splitting triangles. These points have a unique parent triangle, and the splitting is based on the barycentric coordinates. If a point is 'close' w.r.t to ε on one of the triangle's edges, we split the edge, and if a point lies 'close' to an initial point, it gets disregarded (as explained for points on lines).

At this stage, we have an 'enhanced' intermediate mesh $\mathcal{T}_i$ containing the initial (red) and generated (green) points (Fig. 3c). This mesh is of low quality, but our interest here is that it represents as accurately as possible the underlying surface and respects the topology of the initial mesh.

We want now to remove the initial mesh points. Starting again from model curves, we iterate for all feature points to be removed and examine if one of the two connected boundary edges can be collapsed. Similarly with the procedure of splitting feature edges, we can collapse non-manifold feature edges without compromising the conformity of the mesh. Subsequently, points on surfaces are removed by collapsing interior edges. Since not all points are guaranteed to be removed in the first pass, the two discrete loops for points are encapsulated in a while loop that terminates when no point to be removed remains or breaks if the remaining 'red' points cannot be removed at all. This is a crucial part since it prevents the algorithm from producing invalid topology. When an initial mesh vertex cannot be collapsed, it can simply remain in the final mesh. In practice, we observe that no more than two iterations are necessary to remove unwanted vertices for most of the models tested. While inserting points in the triangulation by split edge/triangle operators is trivial, robustly implementing the collapsing step proved be a fairly strenuous task for the general case.

5 Bipartite quadrilateral labeling

At this point, we have obtained a mesh $\mathcal{T}_1$ that is right-angled and ready to be transformed to an all-quad mesh by triangle pairing. A necessary condition for an all-quad mesh to exist is to have an even number of edges on the boundary of each surface of the model. In this work, we trivially impose this condition on the feature curves that bound the surfaces during the first step of the pipeline (Sect. 3.1). Dividing each curve into an even number of segments is not sufficient to ensure the existence of a perfect matching, i.e., a triangle pairing that involves all triangles of $\mathcal{T}_1$. This is a typical obstacle of triangle-pairing methods, leading to isolated triangles that cannot be locally removed. A naive way to eliminate those triangles is to perform a Catmull–Clark subdivision [44], converting a quad-dominant mesh to an all-quad at the expense of vastly increasing the number of elements as well as the number of irregular vertices. In Blossom-Quad [26] a more sophisticated solution is proposed by computing a minimum-cost perfect matching of the dual graph to offer a global solution. Still, ensuring a graph to contain at least one perfect matching remains quite complicated.

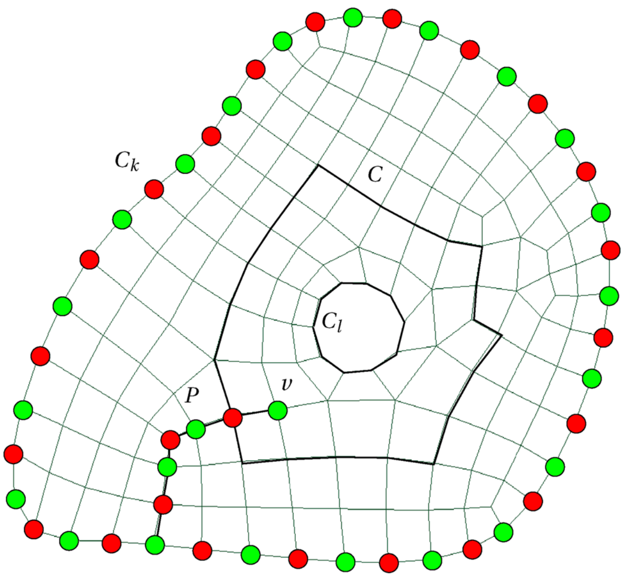

Fig. 8 Every cycle C is even so the graph is bipartite and is 2-colorable. Starting from one 2-colored boundary C_k, it is possible to find the color of any vertex $v \notin C_k$ by 2-coloring any path P between C_k and v (color figure online)

We propose here a new idea for constructing one perfect matching in a triangular mesh: all edges of the matching will be used to combine their two neighboring triangles and form a quad mesh.

Let us first recall some elements of graph theory. A graph labeling is the assignment of labels to the nodes of the graph. A graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called *colors* to the nodes of the graph with the constraint that the graph contains no monochromatic edge, i.e., no edge connecting two nodes with the same color.

A *bipartite graph* $\mathcal{G}$ is a graph that can be 2-colored. An important property of a bipartite graph is that *it does not contain any odd-cycles* (an odd-cycle is a cycle of odd length).

Let us now look at the quadrilateral mesh of a domain Ω as a graph $\mathcal{G}$. We assume here that the boundary $\partial\Omega$ of Ω is divided into n separated sub-boundaries $\partial\Omega_i$, $i = 1 \dots n$, with each of the sub-boundaries $\partial\Omega_i$ forming a boundary cycle C_i in $\mathcal{G}$. We assume that each C_i is an even-cycle. Under these conditions, it is easy to see that $\mathcal{G}$ is bipartite. We consider a cycle C of $\mathcal{G}$: C bounds a quadrilateral mesh and it is known [32] that every quadrilateral mesh has an even number of boundary vertices. If the domain $\mathcal{D}$ that is enclosed by C is simply connected, then C is its only boundary and C is an even-cycle. If $\mathcal{D}$ is not simply connected, its boundary contains other boundary cycles C_l that are by hypothesis even-cycles. Thus, ensuring that every boundary cycle C_k is even is sufficient to ensure that any other cycle C is even as well. Figure 8 illustrates the aforementioned reasoning. It should be noted that not all quad meshes are bipartite: it only holds when every boundary cycle is even.

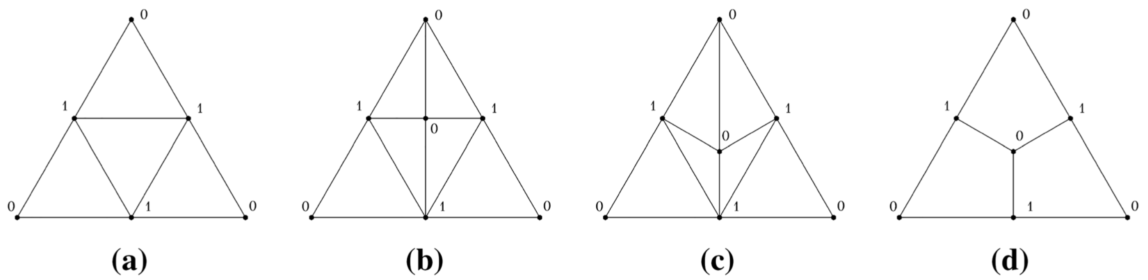

Fig. 9 Bipartite topological correction. **a** Initial labeling on the even-sided boundary. **b** Splitting topologically inconsistent triangle. **c** reposition of vertex according to opposite labeled stencil. **d** recovery of quadrilaterals (defined by same-labeled edges)

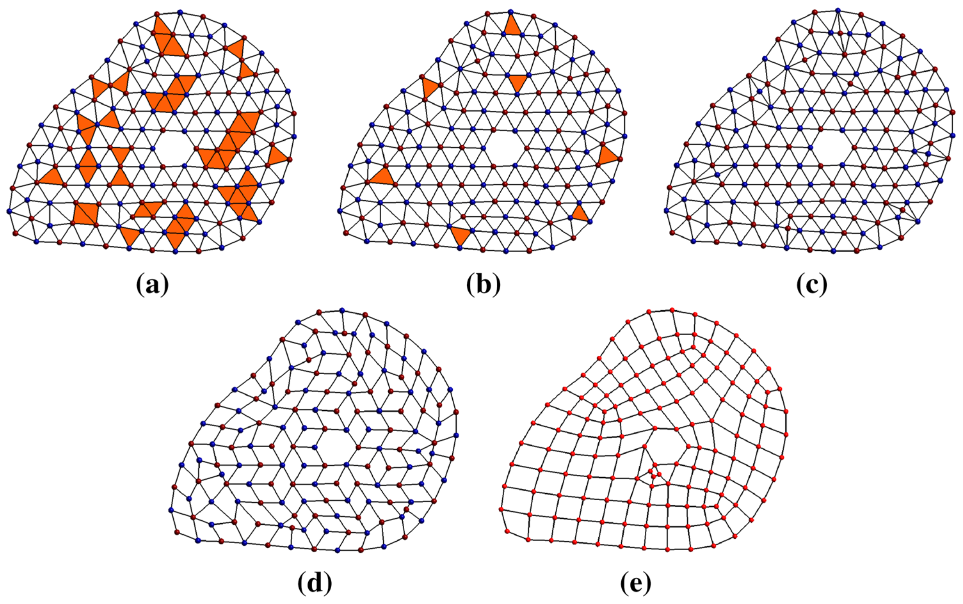

Fig. 10 Different stages of the quadrangulation process. **a** Random labeling of the internal vertices, monochromatic triangles are colored. **b** Local label smoothing. **c** Splitting of remaining isolated monochromatic triangles. **d** Merging triangles using monochromatic edges and obtention of a 2-coloring. **e** Final quad mesh after topological optimization (color figure online)

It should be noted as well that, starting from the 2-coloring of one of the boundary cycle C_k, the 2-coloring of the rest of the graph is unique: the color of any vertex $v \notin C_k$ is found by coloring any path P between any vertex of C_k and v (see Fig. 8).

We consider now a triangulation of Ω and a bipartite labeling of its vertices. The graph of a triangulation is obviously not a bipartite graph since every triangle forms an odd-cycle and therefore it cannot be 2-colored.

Assume that the labeling is done in such a way that there exists no monochromatic triangle (a triangle is monochromatic if its three vertices have the same label). Then, the set of monochromatic edges of the triangulation forms a perfect matching and removing monochromatic edges leads to the desired quadrangulation. The proof is simple: every triangle of the mesh has exactly one monochromatic edge, which means that each triangle will be chosen exactly once for creating a quad: this is by definition a perfect matching. The only possible issue would be the existence of monochromatic edges on $\partial\Omega$. This issue is avoidable if every boundary cycle is even: coloring should be done at first on $\partial\Omega$ and then be propagated inside.

Now the last question: is it possible to avoid monochromatic triangles? The answer is no, at least not without modifying the triangulation. We start by analyzing the simplest possible case of an even triangulation without perfect matching (see Fig. 9a). In this example, all labelings leave the internal triangle monochromatic so there exists no perfect matching in this graph. In other words, merging any of the three possible pairs of triangles would leave two non-connected triangles 'hanging.' In general, picking edges in the graph while constructing a matching may split the graph into disconnected sub-graphs with an odd number of edges on their boundaries, thus not allowing a perfect matching. This condition is a special form of a well-known graph theory theorem from Tutte [45]. By inserting an additional Steiner point of the opposite label (by splitting one edge of the triangle in question) our condition is met (Fig. 9b). A perfect matching actually exists and is composed of all monochromatic edges as demonstrated above (see Fig. 9d).

5.1 Random labeling

We start by 2-coloring the boundary cycles C_k of Ω and assign a random label to all internal vertices. Figure 10a shows this initial random labeling. Random labeling is not the worst-case scenario: it typically produces 25% of monochromatic triangles. Yet, this number is too big because the number of Steiner points that should be added to remove all the monochromatic triangles is sufficiently large to damage the mesh size field.

It is possible to dramatically decrease the number of monochromatic triangles by applying the following *smoothing algorithm*. Each internal vertex is re-labeled if changing its label reduces the number of (its adjacent) monochromatic triangles. The smoothing usually ends with monochromatic triangles that are either isolated or form adjacent pairs (see Fig. 10b). In this specific example, no pairs are observed but they cannot be avoided in the general case.

Then, bipartite topological correction is performed. When monochromatic triangles appear in pairs, their common edge is split. When they are isolated, we choose to split their longest edge, to not degrade the accordance with the size field. This process continues until all monochromatic triangles are eliminated. Figure 10c shows the final labeling without any monochromatic triangle.

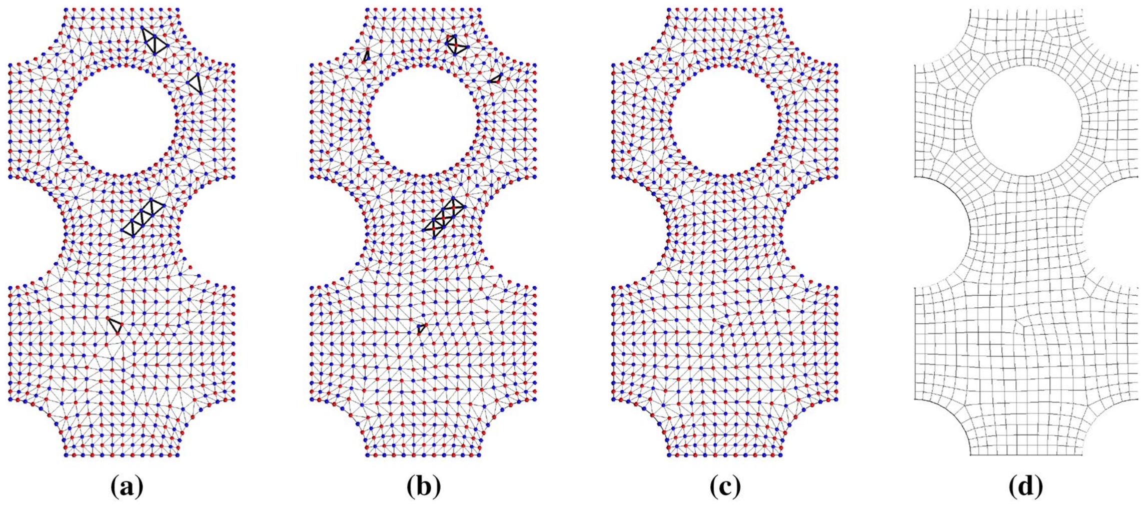

Fig. 11 Bipartite all-quad meshing. **a** Topologically inconsistent triangles. **b** Insert vertices (Steiner points). **c** Reposition of vertices. **d** Bipartite all-quad mesh

The triangular mesh of Fig. 10c can then be transformed into a quadrilateral mesh by removing its monochromatic edges as depicted in Fig. 10d. Mesh of Fig. 10d is not of high quality. We had to use all Gmsh's heavy optimization weaponry to obtain the final mesh of Fig. 10e.

5.2 Cross-field guided labeling

The triangular mesh of Fig. 10a is not suited to be transformed into a quadrilateral mesh; obtaining a good mesh at the end heavily relies on advanced optimization. In Sect. 3.2, we proposed an advancing-front scheme where each point tries to add other points guided by the orthogonal directions provided by a cross-field. The cross-field guided point insertion process produces an excellent labeling scheme: new vertices added by a vertex v have the opposite label of the one of v (since the edges formed between these vertices will be the cross-field aligned ones that we want to have in the final quad mesh). Figure 11a shows the initial labeling after frontal point insertion. As few as 14 monochromatic triangles are present in the triangular mesh, leading to a very small number of Steiner points. More importantly, no Steiner point has been inserted at the vicinity of the boundary, leading to several perfect layers of quads (see Fig. 11d).

By inserting the Steiner points, we have constructed the underlying topological connectivity of an all-quad mesh (Fig. 11b). Mesh vertices can be repositioned at the center of the stencil of opposite labeled neighbors (essentially a single iteration Laplacian smoothing, e.g., Fig. 11c). In the case of non-planar surfaces, the repositioned vertices are projected onto the original triangulation. We can then trivially extract an all-quad mesh, since each pair of triangles whose common edge has two same labels defines a quadrilateral (Fig. 11d). At this point, the advancing labeling scheme is converted to the true 2-coloring of the final bipartite quadrilateral mesh. Another way to see this is that the set of same-colored edges is perpendicular to the set of the edges of the perfect matching of the dual. It is interesting to note that forming the quads this way before inserting the additional vertices would lead to a quad-dominant mesh with non-quad polygonal regions that are even-sided due to the bipartite condition.

Finally, it must be noted that our algorithm is straightforward, and linear in time compared to the computation of an optimal perfect matching, which is quadratic in time and very complex. For the sake of completeness, the algorithm is presented on Algorithm 3.

Algorithm 3 All-Quad Surface Meshing.

```
Input: (Right-Angled) Triangulated Surface $\mathcal{T}_1$
       binary label for each vertex $v_i \in \mathcal{T}_1$
Output: All-quad mesh $\mathcal{T}_q$
1:  for $N_t$ triangles do
2:      Flag triangles with 3 vertices of same label
3:  end for
4:  for $N_e$ edges do
5:      if adjacent triangles $t_i$ and $t_j$ are flagged then
6:          Split edge $e_{ij}$
7:      end if
8:  end for
9:  for the rest (isolated) flagged triangles do
10:     Split longest edge of the triangle
11: end for
12: for $N_e$ edges do
13:     if edge vertices have the same label then
14:         Form a quadrilateral from the two adjacent triangles of the edge
15:     end if
16: end for
```

5.3 Mesh optimization

The algorithm described is always able by construction to produce a topological quad mesh $\mathcal{T}_q$. Moreover, $\mathcal{T}_q$ is also of high geometric quality (measured by the scaled Jacobian Q [46]) since it follows the cross-field directions.

In few cases, especially when the requested size field is much coarser than the geometric characteristics of the model, we may encounter minor defects where elements with $Q \leq 0$ may occur, such as doublet quads (two quads connected only by one vertex) or flat quads on the boundary (since boundary vertices cannot be repositioned). Both cases can be locally handled, the former by merging the doublet into one quad and the latter by inserting additional vertices and thus ‘pushing’ the boundary irregular vertex inside the domain.

The only smoothing procedure used in this work is the Laplacian smoothing described in Sect. 5.2 for the repositioning of vertices. Further and more suited smoothing (e.g., Winslow smoothing [47]) can be performed on the final quadrilateral mesh.

The next step in our developments will be to complement our approach with the more sophisticated optimization procedure proposed in [38] that takes as input the quad mesh $\mathcal{T}_q$, the cross-field $\mathbf{f}$ as well as the size field h. The main features of this optimization process are:

- All vertices of high valence are eliminated.
- All boundary vertices have their optimal valence in such a way that we can create a boundary layer mesh without effort.
- Isolated irregular vertices corresponding to the singularities of the cross-field are preserved.
- Advanced vertex relocation schemes are performed to obtain a geometry regular quad mesh.

6 Results

To demonstrate the capabilities of our methodology, we have tested the whole pipeline on a variety of cases with diverse characteristics (Fig. 12). One of our main interests is to be able to produce meshes on any input, regardless of its complexity (Fig. 13). Reviewing the relevant literature, we observe that the majority of the works present results on relatively simple models. We work on large model databases (that are recently becoming the common standard for validation purposes on bulk numbers of models), and we aim to have a close to 100% success rate on them. The models are obtained from the following sources: the ABC [48], MAMBO [49] and Thingi10k [50] datasets, and the supplementary test data of LoopyCuts software [51].

We present statistics for all the models from MAMBO and LoopyCuts, since all their models are directly suited for meshing (Table 1). MAMBO contains three categories of CAD models: basic, simple, and medium, of which we use the latter two since they have more complex features of interest. LoopyCuts models are in general simpler models in terms of surface complexity (smooth surfaces, absence of very small features) oriented mostly for computer graphics research.

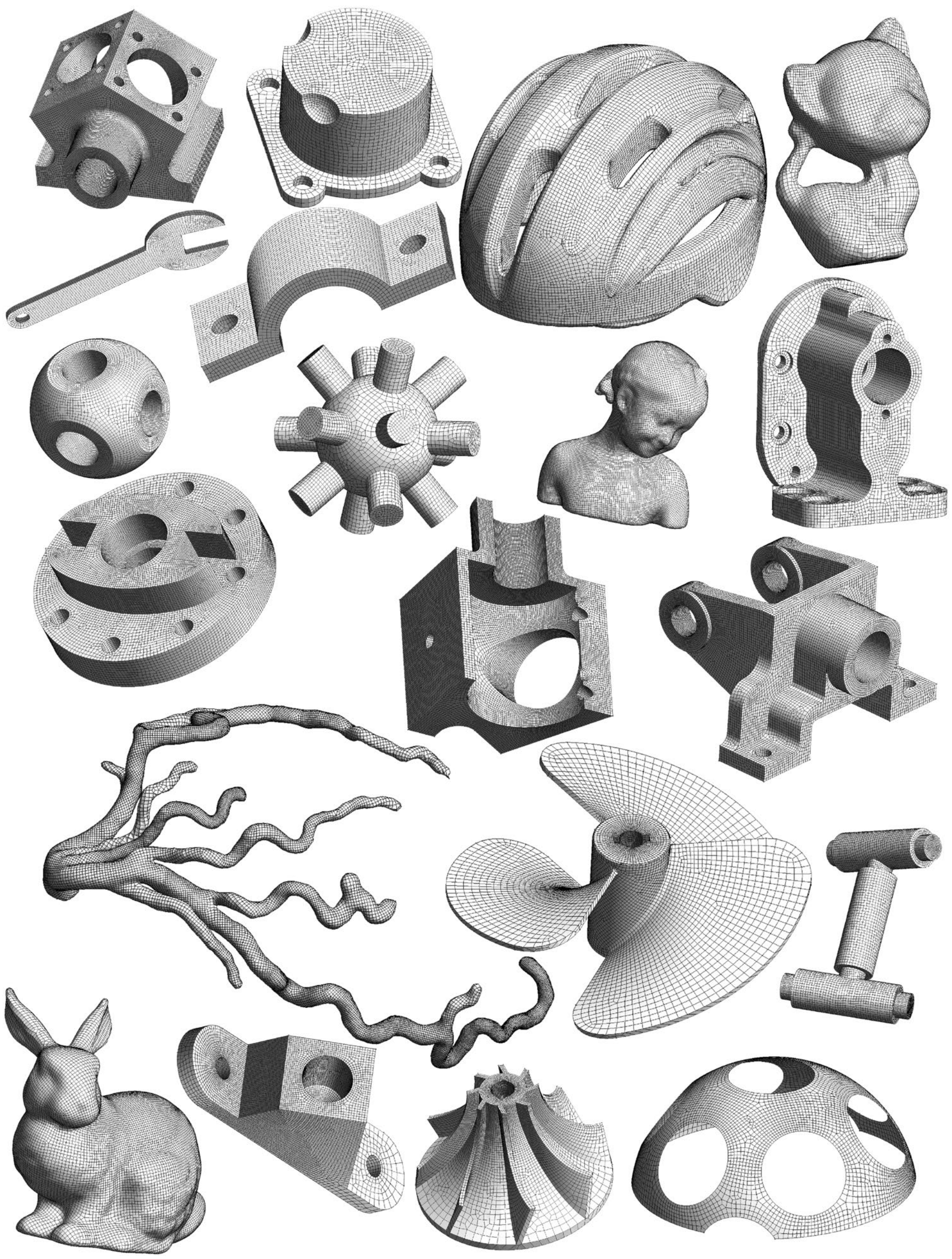
Fig. 12 All-quad meshes on various models obtained from the following sources: ABC [48], MAMBO [49], Thingi10k [50] and LoopyCuts [51]

A desired property of our mesh generation process is to keep user intervention at a minimum. Input parameters are a threshold value ε to suppress potential discrepancies during local mesh modifications and the angle threshold to detect hard edges. The angle threshold ε may be disregarded, and the algorithm can initiate from a random edge of the initial mesh instead of the model curves. We keep in mind that this may result in low-quality meshing on hard edges.

Cross-fields are precomputed and considered input to the algorithm, or they can be taken as the unit axis directions for practical purposes. Size fields can be a uniform input size or computed from the curvature and geometric characteristics of the input mesh. Both cross and size fields impact considerably the output mesh quality, and we can observe that the directionality of the elements matches the input cross-field directions. It must be noted that our point generation module can have any kind of metric input. For example, we can use an anisotropic metric or a prescribed user-defined size field that is 'incompatible' with the computed cross-field. In Fig. 14 (right), a size field of sinusoidal form is prescribed and we can observe that our algorithm can successfully handle it at the cost of lower element quality and an excess of irregular vertices to accommodate for the size transitions.

Our pipeline can succesfully produce a quad mesh in all the models that can be processed by GMSH for the initial

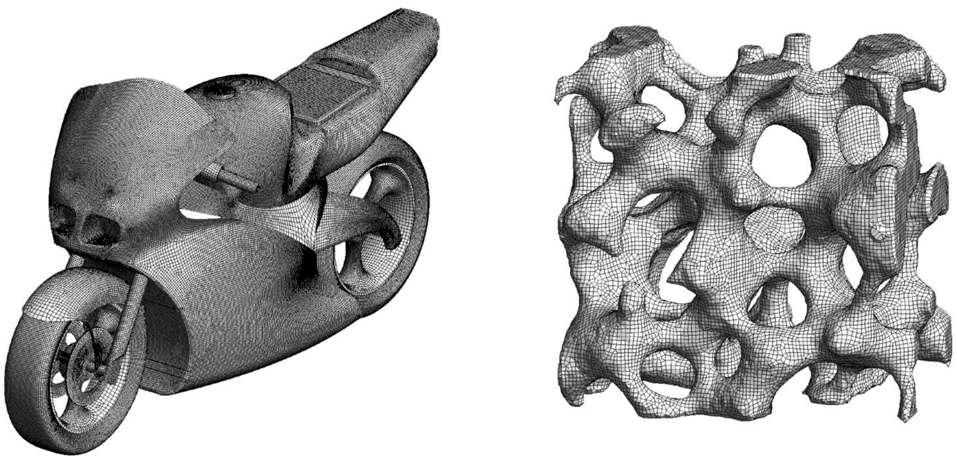
Fig. 13 Left: Quadrilateral mesh on a B-rep model of a motorbike. Right: Quadrilateral mesh from a triangulated surface of scanned data

triangulation. One of the most challenging aspects of our methodology is the removal of the initial mesh points by collapsing and testing multiple 'extreme' cases on that is an ongoing process. As desired, non-manifold features are well handled and conformity is preserved (Fig. 14, left). The final conversion to an all-quad mesh proves to be straightforward and efficient, given that the input respects the fundamental topological condition (i.e., even number of edges on each curve). The quad meshes exhibit high quality Q, computed as the scaled Jacobian (Table 1). Minimum quality over all LoopyCuts models is 0.22 with the vast majority of the models having over 0.50. On the other hand, we observe similar outputs on the MAMBO models except of a few cases where very thin sliver regions result to almost flat quad elements with $Q \simeq 0$. These kind of features are common in 'real-life' models and element quality there could be improved with further optimization. The output meshes shown here consist of the 'raw' quadrilateral meshes, without heavy further optimization (an example of further optimizing a quad mesh is presented in Fig. 15).

One of our main interests is the computational efficiency of the method. The majority of operations are local and the algorithms of linear complexity. The most time-consuming parts of our pipeline is the (inevitable) use of spatial search structures (involved in filtering and also in projection of points onto the triangulation during surface meshing and all-quad conversion). The performance of the algorithm is heavily dependent on the input mesh, specifically its number of triangles and colored surfaces. More triangles lead to more spatial searches during point generation and filtering as well as more local mesh modifications during surface meshing. Spatial searches and projections are the most

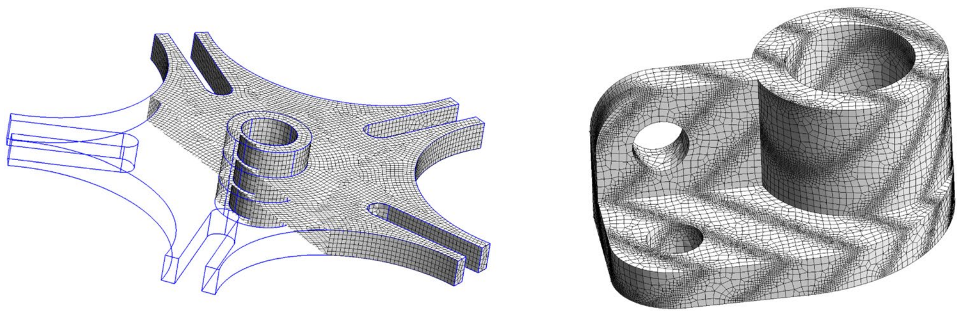
Fig. 14 Left: CAD model with multiple volumes, leading to non-manifold feature curves. Right: A quadrilateral mesh with a prescribed sinusoidal non-uniform size field

Table 1 Average quality and timings values for MAMBO and LoopyCuts models

Dataset (#)	N_q	Q	$t_{\text{points}}(s)$	$t_{\text{surf}}(s)$	$t_{\text{quad}}(s)$	$t_{\text{total}}(s)$
MAMBO (39)	23730	0.988	0.907	0.723	0.366	2.112
LoopyCuts (70)	12748	0.985	0.305	0.351	0.197	0.953

N_q is the average number of quadrilaterals of the output meshes. The quality metric Q is the scaled Jacobian and t_{points}, t_{surf} and t_{quad} are the timings in seconds for the three pipeline steps, respectively: point generation, surface meshing and quad conversion

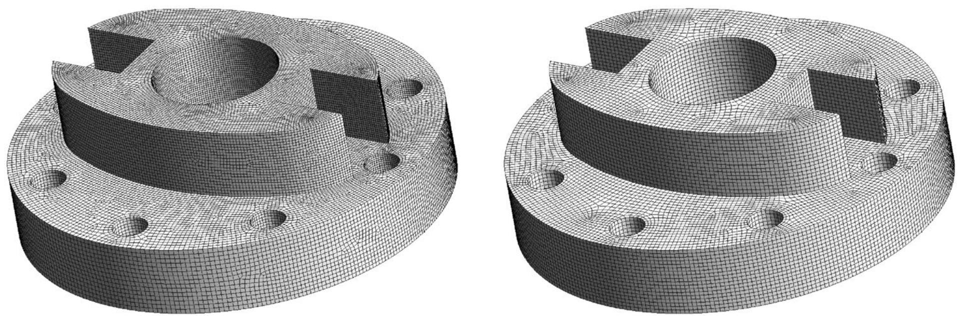
Fig. 15 Left: unstructured quad mesh produced with our pipeline, where we can observe the excess of irregular vertices. Right: quasi-structured quad mesh obtained with the topological optimization of Ref. [38]

important bottlenecks on the performance of our pipeline and we consider that we can speed up with more efficient procedures. In Table 1, we present the average time for each part of our pipeline, using a single-thread on a laptop with 2.6 GHz CPU.

7 Conclusions

We have presented a method for the creation of quadrilateral surface meshes on general complex geometries. One of the important aspects of our pipeline is its modular nature. Each step can be independently used, for example, a more efficient point sampling method could be used, or a typical CAD meshing algorithm to generate the mesh with these points.

The surface meshing algorithm proposed is used to produce right-angled triangles, but it can be repurposed as a remeshing tool for triangulations, given appropriate input (i.e., an 'asterisk' field [32, 39]). The proposed two-step method for surface meshing aims at balancing the trade-off between output quality and robustness. One of its strengths is the ability to handle the model as a whole and thus create cross-aligned elements globally. While producing high-quality elements in this aspect, we always respect the topology and succeed at meshing small-scale and non-manifold features as shown in the results demonstrated (Sect. 6).

As stated, scaled cross-fields are beneficial for quad mesh generation. We have mainly used here size fields obtained from the scaling of cross-fields. In practice though, a user would commonly prefer prescribing a size field that is a better fit for the problem in hand, and unfortunately this is currently an open problem. One of the advantages of our approach is that our pipeline can produce meshes regardless of the topological integrity of the input metric, for example with a user-defined size field that is incompatible with the computed cross-field (Fig. 14, left). Nevertheless, adapting a direction field to an input size field would vastly improve the results. Regarding highly anisotropic metrics (for example for the generation of structured boundary layer meshes), it is straightforward to extend the point generation part of our pipeline for such a use. Nevertheless, we consider the triangulation on such a metric a much more difficult task (where the aim is having structured layers of anisotropic triangles to be combined). We believe that the best strategy would be to create structured layers of boundary quadrilaterals and then subdivide them.

A novel method to obtain an all-quad mesh from any quad-dominant mesh with minimal, if any, post-processing clean-up is introduced. By using a bipartite labeling scheme, we simplify the global treatment required for quad meshing to a set of localized operations that can be performed on a quad-dominant mesh. We are thus enabled to produce always a high-quality quad mesh, regardless of the complexity of the model. Our meshes can be further optimized with a strategy to remove the majority of irregular vertices, presented on [38].

We are currently in the process of incorporating this quad surface mesher as a component of a general high-quality hex-dominant meshing pipeline, where it will be used as the volume bounding surface. An interesting open problem to this end is if hexahedral element generation should be constrained by a quadrilateral mesh, and if yes, what would be its optimal characteristics. Besides that, we are investigating ways to take advantage of the topological labels used here for quadrilateral elements to the generation of hexahedra, possibly with a labeling scheme that can codify more topological information.

Declarations

Conflict of interest This study was carried out in the framework of the research project 'Hextreme,' funded by the European Research Council (ERC-2015-AdG-694020) and hosted at the Université catholique de Louvain. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Geuzaine C, Remacle J-F (2009) Gmsh: a 3-D finite element mesh generator with built-in pre- and post-processing facilities. Int J Numer Methods Eng 79(11):1309–1331. https://doi.org/10.1002/nme.2579
2. Borouchaki H, Laug P, George P-L (2000) Parametric surface meshing using a combined advancing-front generalized Delaunay

approach. Int J Numer Methods Eng 49(1–2):233–259. https://doi.org/10.1002/1097-0207(20000910/20)49:1/2h233::AID-NME931i3.0.CO;2-G
3. Lo S (2015) Finite element mesh generation
4. Lo SH (1988) Finite element mesh generation over curved surfaces. Comput Struct 29(5):731–742. https://doi.org/10.1016/0045-7949(88)90341-0
5. Sheng X, Hirsch BE (1992) Triangulation of trimmed surfaces in parametric space. Comput Aided Des 24(8):437–444. https://doi.org/10.1016/0010-4485(92)90011-X
6. Shimada K, Gossard DC (1998) Automatic triangular mesh generation of trimmed parametric surfaces for finite element analysis. Comput Aided Geom Des 15(3):199–222. https://doi.org/10.1016/S0167-8396(97)00037-X
7. Floater MS, Hormann K (2005) Surface parameterization: a tutorial and survey. In: Dodgson NA, Floater MS, Sabin MA (eds) Advances in multiresolution for geometric modelling. Springer-Verlag, Berlin/Heidelberg, pp 157–186. https://doi.org/10.1007/3-540-26808-1_9
8. Remacle J-F, Geuzaine C, Compère G, Marchandise E (2010) High-quality surface remeshing using harmonic maps. Int J Numer Methods Eng 83(4):403–425. arXiv:10.1002/nme.2824. https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.2824. https://doi.org/10.1002/nme.2824
9. Beaufort P-A, Geuzaine C, Remacle J-F (2020) Automatic surface mesh generation for discrete models—a complete and automatic pipeline based on reparametrization. J Comput Phys 417:109575 arXiv:2001.02542. https://doi.org/10.1016/j.jcp.2020.109575
10. Shephard MS, Georges MK (1991) Automatic three-dimensional mesh generation by the finite octree technique. Int J Numer Methods Eng 32(4):709–749. https://doi.org/10.1002/nme.1620320406
11. Frey PJ, Marechal L (1998) Fast adaptive quadtree mesh generation. In: Proceedings of the seventh international meshing roundtable, pp 211–224
12. Löhner R, Parikh P (1988) Generation of three-dimensional unstructured grids by the advancing-front method. Int J Numer Methods Fluids 8(10):1135–1149. https://doi.org/10.1002/fld.1650081003
13. Lau TS, Lo SH (1996) Finite element mesh generation over analytical curved surfaces. Comput Struct 59(2):301–309. https://doi.org/10.1016/0045-7949(95)00261-8
14. Baker TJ (1989) Automatic mesh generation for complex three-dimensional regions using a constrained Delaunay triangulation. Eng Comput 5(3–4):161–175. https://doi.org/10.1007/BF02274210
15. Borouchaki H, George PL, Hecht F, Laug P, Saltel E (1997) Delaunay mesh generation governed by metric specifications Part I. Algorithms. Finite Elem Anal Des 25(1–2):61–83. https://doi.org/10.1016/S0168-874X(96)00057-1
16. Wang D, Hassan O, Morgan K, Weatherill N (2006) EQSM: an efficient high quality surface grid generation method based on remeshing. Comput Methods Appl Mech Eng 195(41–43):5621–5633. https://doi.org/10.1016/j.cma.2005.10.028
17. Béchet E, Cuilliere J-C, Trochu F (2002) Generation of a finite element MESH from stereolithography (STL) files. Comput Aided Des 34(1):1–17. https://doi.org/10.1016/S0010-4485(00)00146-9
18. Baehmann PL, Wittchen SL, Shephard M, Grice KR, Yerry M (1987) Robust, geometrically based, automatic two-dimensional mesh generation. Int J Numer Methods Eng 24:1043–1078
19. Schneiders R (1996) A grid-based algorithm for the generation of hexahedral element meshes. Eng Comput 12(3–4):168–177. https://doi.org/10.1007/BF01198732
20. Frey P, Marechal L (2000) Fast adaptive quadtree mesh generation. In: Proceedings of the 7th international meshing roundtable
21. Blacker TD, Stephenson MB (1991) Paving: a new approach to automated quadrilateral mesh generation. Int J Numer Methods Eng 32(4):811–847. https://doi.org/10.1002/nme.1620320410
22. Mitchell SA, Mohammed MA, Mahmoud AH, Ebeida MS (2014) Delaunay quadrangulation by two-coloring vertices. Proc Eng 82:364–376. https://doi.org/10.1016/j.proeng.2014.10.397
23. Lee CK, Lo SH (1994) A new scheme for the generation of a graded quadrilateral mesh. Comput Struct 52(5):847–857. https://doi.org/10.1016/0045-7949(94)90070-1
24. Borouchaki H, Frey PJ (1998) Adaptive triangular-quadrilateral mesh generation. Int J Numer Methods Eng 41(5):915–934. https://doi.org/10.1002/(SICI)1097-0207(19980315)41:5h915::AID-NME318i3.0.CO;2-Y
25. Owen S, Staten M, Canann S, Saigal S (1999) Q-Morph: an indirect approach to advancing front quad meshing. Int J Numer Methods Eng 44(9):1317–1340. https://doi.org/10.1002/(SICI)1097-0207(19990330)44:9h1317::AID-NME532i3.0.CO;2-N
26. Remacle J-F, Lambrechts J, Seny B, Marchandise E, Johnen A, Geuzainet C (2012) Blossom-Quad: a non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm. Int J Numer Methods Eng 89(9):1102–1119. https://doi.org/10.1002/nme.3279
27. Remacle J-F, Henrotte F, Carrier-Baudouin T, Béchet E, Marchandise E, Geuzaine C, Mouton T (2013) A frontal Delaunay quad mesh generator using the L ∞ norm. Int J Numer Methods Eng 94(5):494–512. https://doi.org/10.1002/nme.4458
28. Baudouin TC, Remacle J-F, Marchandise É, Henrotte F, Geuzaine C (2014) A frontal approach to hex-dominant mesh generation. Adv Model Simul Eng Sci 1(1):8–8. https://doi.org/10.1186/2213-7467-1-8
29. Bommes D, Lévy B, Pietroni N, Puppo E, Silva C, Tarini M, Zorin D (2013) Quad-mesh generation and processing: a survey: quad-mesh generation and processing. Comput Graphics Forum 32(6):51–76. https://doi.org/10.1111/cgf.12014
30. Ray N, Li WC, Lévy B, Sheffer A, Alliez P (2006) Periodic global parameterization. ACM Trans Graph 25(4):1460–1485. https://doi.org/10.1145/1183287.1183297
31. Palacios J, Zhang E (2007) Rotational symmetry field design on surfaces. ACM Trans Graph 26(3):55. https://doi.org/10.1145/1276377.1276446
32. Beaufort P-A, Lambrechts J, Henrotte F, Geuzaine C, Remacle J-F (2017) Computing cross fields A PDE approach based on the Ginzburg-Landau theory. In: 26th International meshing roundtable, vol. 203, IMR26, 18-21 September 2017, Barcelona, Spain pp 219–231. https://doi.org/10.1016/j.proeng.2017.09.799
33. Kettner L (1998) Designing a data structure for polyhedral surfaces. In: Proceedings of the fourteenth annual symposium on computational geometry. SCG '98, pp 146–154. ACM, New York. https://doi.org/10.1145/276884.276901
34. Shamir A (2008) A survey on mesh segmentation techniques. Comput Graph Forum 27(6):1539–1556. https://doi.org/10.1111/j.1467-8659.2007.01103.x
35. Vieira M, Shimada K (2005) Surface mesh segmentation and smooth surface extraction through region growing. Comput Aided Geometr Des 22(8):771–792. https://doi.org/10.1016/j.cagd.2005.03.006
36. Sunil VB, Pande SS (2008) Automatic recognition of features from freeform surface CAD models. Comput Aided Des 40(4):502–517. https://doi.org/10.1016/j.cad.2008.01.006
37. Thakur A, Banerjee AG, Gupta SK (2009) A survey of CAD model simplification techniques for physics-based simulation applications. Comput Aided Des 41(2):65–80. https://doi.org/10.1016/j.cad.2008.11.009
38. Reberol M, Georgiadis C, Remacle J-F (2021) Quasi-structured quadrilateral meshing in Gmsh—a robust pipeline for complex CAD models. arXiv:2103.04652 [cs]. arXiv:2103.04652 [cs]

39. Georgiadis C, Beaufort P-A, Lambrechts J, Remacle J-F (2017) High quality mesh generation using cross and asterisk fields: application on coastal domains. In: 26th International meshing roundtable, research notes, Barcelona
40. Frey PJ, George PL (2008) Mesh generation: application to finite elements, 2nd edn. Wiley, London
41. Devillers O, Pion S, Teillaud M (2001) Walking in a triangulation. In: Proceedings of the seventeenth annual symposium on computational geometry. SCG '01, pp 106–114. Association for Computing Machinery, New York. https://doi.org/10.1145/378583.378643
42. Beckmann N, Kriegel H-P, Schneider R, Seeger B (1990) The R*-tree: an efficient and robust access method for points and rectangles. ACM SIGMOD Record 19(2):322–331. https://doi.org/10.1145/93605.98741
43. Ray N, Sokolov D, Reberol M, Ledoux F, Lévy B (2018) Hex-dominant meshing: mind the gap! Comput Aided Des 102:94–103. https://doi.org/10.1016/j.cad.2018.04.012
44. Catmull E, Clark J (1978) Recursively generated B-spline surfaces on arbitrary topological meshes. Comput Aided Des 10(6):350–355. https://doi.org/10.1016/0010-4485(78)90110-0
45. Bondy JA, Murty USR (1976) Graph theory with applications, p 264. Macmillan London, London
46. Stimpson CJ, St NM, Ernst CD, St NM, Pébay PP, Thompson D (2007) The verdict geometric quality library. Technical Report SAND2007-1751, SANDIA
47. Knupp PM (1999) Winslow smoothing on two-dimensional unstructured meshes. Eng Comput 15(3):263–268. https://doi.org/10.1007/s003660050021
48. Koch S, Matveev A, Jiang Z, Williams F, Artemov A, Burnaev E, Alexa M, Zorin D, Panozzo D (2019) ABC: a big CAD model dataset for geometric deep learning. In: The IEEE conference on computer vision and pattern recognition (CVPR)
49. Ledoux F MAMBO Dataset. https://gitlab.com/franck.ledoux/mambo
50. Zhou Q, Jacobson A (2016) Thingi10K: a dataset of 10,000 3D-Printing models. arXiv preprint arXiv:1605.04797. arXiv:1605.04797
51. Livesu M, Pietroni N, Puppo E, Sheffer A, Cignoni P (2020) LoopyCuts: practical feature-preserving block decomposition for strongly hex-dominant meshing. ACM Transactions on Graphics. https://doi.org/10.1145/3386569.3392472