Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 163 (2016) 302 – 314

Procedia Engineering

www.elsevier.com/locate/procedia

25th International Meshing Roundtable

Tetrahedral mesh improvement using moving mesh smoothing and lazy searching flips

Franco Dassi[a,*], Lennard Kamenski[b], Hang Si[b]

[a]Dipartimento di Matematica e Applicazioni, Università degli Studi di Milano Bicocca, Via Cozzi 53, 20125 Milano, Italy
[b]Weierstrass Institute for Applied Analysis and Stochastics, Mohrenstr. 39, 10117 Berlin, Germany

Abstract

We combine the new *moving mesh smoothing*, based on the integration of an ordinary differential equation coming from a given functional, with the new *lazy flip* technique, a reversible edge removal algorithm for local mesh quality improvement. These strategies already provide good mesh improvement on themselves, but their combination achieves astonishing results not reported so far. Provided numerical comparison with some publicly available mesh improving software show that we can obtain final tetrahedral meshes with dihedral angles between 40° and 123°.

Keywords: mesh improvement, moving mesh, edge flipping, mesh quality, mesh smoothing
2010 MSC: 65N50, 65K10

1. Introduction

The two key operations for mesh improving are *smoothing* (which moves the mesh vertices) and *flipping* (which changes mesh topology without moving the mesh vertices). Previous work shows that the combination of these two operations achieves better results than if applied individually [1,2]. In this paper we combine a new smoothing and a new flipping methods to one mesh improvement scheme.

Flips are the most efficient ways to locally improve the mesh quality and they have been extensively addressed in the literature [1–4]. In the most simple cases, the basic flip operations, such as 2-to-3, 3-to-2, and 4-to-4 flips, are applied as long as the mesh quality can be improved. The more effective way is to combine several basic flip operations, such as the edge removal operation, which is an extension of the 3-to-2 and 4-to-4 flips. This operation removes an edge with $n \geq 3$ adjacent tetrahedra and replaces them by $m = 2n - 4$ new tetrahedra (the so-called an n-to-m flip). There are at most C_{n-2} possible cases, where $C_n = \frac{(2n)!}{(n+1)!\,n!}$ is the Catalan number. If n is small (e.g., $n < 7$), one can enumerate all the possible cases, compute the mesh quality for each of the individual cases, and then pick the optimal one. Another way is to use the dynamic programming to find the optimal configuration. However, the number of cases increases exponentially and finding the optimal solution with brute force is very time-consuming.

[*] Corresponding author.
E-mail address: franco.dassi@unimib.it, kamenski@wias-berlin.de, si@wias-berlin.de.

In this paper, we propose the so-called *lazy searching flips*. The key idea is to automatically explore sequences of flips in order to remove a given edge in the mesh. If a sequence of flips leads to a configuration which doesn't improve the mesh quality, the algorithm reverses this sequence and explores another one (see section 3 and Figs 1a to 1c). Once an improvement is found, the algorithms stops the search and returns without exploring the remaining possibilities.

The *lazy searching flips* are accompanied with a smoothing procedure. Mesh smoothing improves the mesh quality by improving vertex locations, typically through Laplacian smoothing or some optimization-based algorithms. Most commonly used mesh smoothing methods are Laplacian smoothing and its variants [5,6], where a vertex is moved to the geometric center of its neighboring vertices. While economic, easy to implement, and often effective, Laplacian smoothing guarantees neither a mesh quality improvement nor the mesh validity.

Alternatives are optimization-based methods that are effective for a variety of mesh quality measures, e.g., for the ratio of the area to the sum of the squared edge lengths [7] or the ratio of the volume to a power of the sum of the squared face areas [8], the condition number of the Jacobian matrix of the affine mapping between the reference element and physical elements [9], or various other measures [1,10–12]. Most of the optimization-based methods are local and sequential, with Gauss-Seidel-type iterations being combined with location optimization problems over each patch. There is also a parallel algorithm that solves a sequence of independent subproblems [13].

In our scheme, we employ the moving mesh PDE (MMPDE) method, defined as the gradient flow equation of a meshing functional (an objective functional in the context of optimization) to move the mesh continuously in time. Such a functional is typically based on error estimation or physical and geometric considerations. Here, we consider a functional based on the equidistribution and alignment conditions [14] and employ the recently developed direct geometric discretization [15] of the underlying meshing functional on simplicial meshes.

Compared to the aforementioned mesh smoothing methods, the considered method has several advantages: it can be easily parallelized, it is based on a continuous functional for which the existence of minimizers is known, the functional controlling the mesh shape and size has a clear geometric meaning, and the nodal mesh velocities are given by a simple analytical matrix form. Moreover, the smoothed mesh will stay valid if it was valid initially [20].

In this paper we provide a detailed numerical study of a combination of the lazy searching flips with the MMPDE smoothing. More specifically, we compare the results of the whole algorithm with `Stellar` [2], `CGAL` [16] and `mmg3d` [17]. We also compare the lazy searching flips and the MMPDE smoothing with the flipping and smoothing procedures provided by `Stellar`.

2. The moving mesh PDE smoothing scheme

The key idea of this smoothing scheme is to move the mesh vertices via a moving mesh equation, which is formulated as the gradient system of an energy function (the MMPDE approach). Originally, the method was developed in the continuous form [18,19]. In this paper, we use its discrete form [15,20,21], for which the mesh vertex velocities are expressed in a simple, analytical matrix form, which makes the implementation more straightforward to parallelize.

2.1. Moving mesh smoothing

Consider a polygonal (polyhedral) domain $\Omega \subset \mathbb{R}^d$, $d \geq 1$, let the simplicial mesh under consideration be $\mathcal{T}_h$, and denote the numbers of its vertices and elements by $\#\mathcal{N}_h$ and $\#\mathcal{T}_h$. Let K be a generic mesh element and $\hat{K}$ the reference element taken as a regular simplex with the volume $|\hat{K}| = 1/\#\mathcal{T}_h$. Further, let F'_K be the Jacobian matrix of the affine mapping $F_K \colon \hat{K} \to K$ from the reference element $\hat{K}$ to a mesh element K. For notational simplicity, we denote the inverse of the Jacobia by $\mathbb{J}_K$, i.e., $\mathbb{J}_K \equiv (F'_K)^{-1}$. Then, the mesh $\mathcal{T}_h$ is uniform if and only if

$$|K| = \frac{|\Omega|}{\#\mathcal{T}_h} \quad \text{and} \quad \frac{1}{d}\,\mathrm{tr}\left(\mathbb{J}_K^T \mathbb{J}_K\right) = \det\left(\mathbb{J}_K^T \mathbb{J}_K\right)^{\frac{1}{d}} \quad \forall K \in \mathcal{T}_h. \tag{1}$$

The first condition requires all elements to have the same size and the second requires all elements to be shaped similarly to $\hat{K}$ (these conditions are the simplified versions of the equidistribution and alignment conditions [19,22]).

The corresponding energy function for which the minimization will result in a mesh satisfying (1) as closely as possible is

$$I_h = \sum_K |K|\, G\left(\mathbb{J}_K, \det \mathbb{J}_K\right) \quad \text{with} \quad G(\mathbb{J}, \det \mathbb{J}) = \theta\left(\mathrm{tr}\left(\mathbb{J}\mathbb{J}^T\right)\right)^{\frac{dp}{2}} + (1 - 2\theta)\, d^{\frac{dp}{2}} (\det \mathbb{J})^p, \tag{2}$$

where $\theta \in (0, 0.5]$ and $p > 1$ are dimensionless parameters (in section 5 we use $\theta = 1/3$ and $p = 3/2$). This is a specific choice and other meshing functionals are possible. An interested reader is referred to [23] for a numerical comparison of meshing functionals for variational mesh adaptation.

I_h is a Riemann sum of a continuous functional for variational mesh adaptation based on equidistribution and alignment [14] and depends on the vertex coordinates $\boldsymbol{x}_i$, $i = 1, \ldots, \#\mathcal{N}_h$. The vertex velocities for the mesh movement are defined as

$$\frac{d\boldsymbol{x}_i}{dt} = -\left(\frac{\partial I_h}{\partial \boldsymbol{x}_i}\right)^T, \quad i = 1, \ldots, \#\mathcal{N}_h, \tag{3}$$

where the derivatives $\frac{d\boldsymbol{x}_i}{dt}$ are considered as row vectors.

2.2. Vertex velocities and the mesh movement

The vertex velocities can be computed analytically [15, Eqs (39) to (41)] using the scalar-by-matrix differentiation [15, Sect. 3.2]. Denote the vertices of K and $\hat{K}$ by $\boldsymbol{x}_i^K$ and $\hat{\boldsymbol{x}}_i$, $i = 0, \ldots, d$, and define the element edge matrices as

$$E_K = [\boldsymbol{x}_1^K - \boldsymbol{x}_0^K, \ldots, \boldsymbol{x}_d^K - \boldsymbol{x}_0^K] \quad \text{and} \quad \hat{E} = [\hat{\boldsymbol{x}}_1 - \hat{\boldsymbol{x}}_0, \ldots, \hat{\boldsymbol{x}}_d - \hat{\boldsymbol{x}}_0] \quad \text{with} \quad \hat{E}E_K^{-1} = \mathbb{J}_K.$$

Then, the local mesh velocities are given element-wise [15, Eqs (39) and (41)] as

$$\begin{bmatrix} (\boldsymbol{v}_1^K)^T \\ \vdots \\ (\boldsymbol{v}_d^K)^T \end{bmatrix} = -GE_K^{-1} + E_K^{-1}\frac{\partial G}{\partial \mathbb{J}}\hat{E}E_K^{-1} + \frac{\partial G}{\partial \det \mathbb{J}}\frac{\det(\hat{E})}{\det(E_K)}E_K^{-1} \quad \text{and} \quad (\boldsymbol{v}_0^K)^T = -\sum_{j=1}^{d}(\boldsymbol{v}_j^K)^T, \tag{4}$$

where G is as in (2) and $\frac{\partial G}{\partial \mathbb{J}}$ and $\frac{\partial G}{\partial \det \mathbb{J}}$ are the derivatives of G with respect to its first and second arguments (evaluated at $\mathbb{J}_K = \hat{E}E_K^{-1}$ and $\det(\mathbb{J}) = \det(\hat{E})/\det(E_K)$). For the considered G we have [15, Example 3.2]

$$\frac{\partial G}{\partial \mathbb{J}} = dp\theta\Big(\mathrm{tr}(\mathbb{J}\mathbb{J}^T)\Big)^{\frac{dp}{2}-1}\mathbb{J}^T \quad \text{and} \quad \frac{\partial G}{\partial \det \mathbb{J}} = p(1-2\theta)d^{\frac{dp}{2}}(\det \mathbb{J})^{p-1}.$$

With the element-wise vertex velocities, the moving mesh equation (3) becomes

$$\frac{d\boldsymbol{x}_i}{dt} = \sum_{K \in \omega_i} |K|\, \boldsymbol{v}_{i_K}^K, \quad i = 1, \ldots, \#\mathcal{N}_h, \tag{5}$$

where ω_i is the patch of the vertex $\boldsymbol{x}_i$ and i_K is the local index of $\boldsymbol{x}_i$ on K.

During smoothing, we use the current vertex locations as the initial position and integrate the equation (5) for a time period (with the proper modification for the boundary vertices, see section 2.3). The connectivity is kept fixed during the smoothing step. In our examples in section 5 we use the explicit Runge-Kutta Dormand-Prince ODE solver [24].

The moving mesh governed by (5) will stay nonsingular if it is nonsingular initially: the minimum height and the minimum volume of the mesh elements will stay bounded from below by a positive number depending only on the initial mesh and the number of the elements [20].

2.3. Velocity adjustment for the boundary vertices

The vertex velocities need to be modified for the boundary vertices. If $\boldsymbol{x}_i$ is a fixed boundary vertex, then its velocity is set to zero: $\frac{\partial \boldsymbol{x}_i}{\partial t} = 0$. If $\boldsymbol{x}_i$ is allowed to move along a boundary curve or a surface represented by the zero level set of a function ϕ, then the vertex velocity $\frac{\partial \boldsymbol{x}_i}{\partial t}$ is modified such that its normal component along the curve (surface) is zero:

$$\nabla\phi(\boldsymbol{x}_i) \cdot \frac{\partial \boldsymbol{x}_i}{\partial t} = 0.$$

In our examples (section 5), the input geometry is a piecewise linear complex (PLC) [25], for which the velocity adjustment is straight forward:

facet vertices:	project the velocity onto the facet plane,
segment vertices:	project the velocity onto the segment line,
corner vertices:	set the velocity to zero.

For a general non-polygonal or non-polyhedral domain, we have to move the vertex and then project it onto the boundary to which it belongs, otherwise it is not guaranteed to be on the domain boundary after the smoothing step.

3. Lazy searching flips

In this section, we explain how to remove an edge and how to reverse the removal using flips. Then we present the lazy searching algorithm for the mesh improvement.

3.1. Edge removal and its inverse

A basic edge removal algorithm [26] performs a sequence of elementary 2-to-3 and 3-to-2 flips. We extend this basic algorithm with the possibility to inverse the flip sequence. The idea is straightforward: the sequence is saved online without using additional memory.

Let $[a, b] \in \mathcal{T}_h$ be an edge with endpoints a and b and $A[0, \ldots, n-1]$ be the array of $n \geq 3$ tetrahedra in $\mathcal{T}_h$ sharing $[a, b]$. For simplicity, we assume that $[a, b]$ is an interior edge of $\mathcal{T}_h$, so that all tetrahedra in A can be ordered cyclically such that the two tetrahedra $A[i \mod n]$ and $A[(i+1) \mod n]$ share a common face. Given such an array A of n tetrahedra, we want to find a sequence of flips that will remove the edge $[a, b]$. Moreover, we also want to reverse this sequence in order to return to the original state. In the following, we will write i instead of $i \mod n$, i.e., the index i will take values in $\{0, 1, \ldots, n-1\}$.

Our edge removal algorithm includes two subroutines

$$[\texttt{done}, m] := \texttt{flipnm}(A[0, \ldots, n-1], \texttt{level}) \quad \text{and} \quad \texttt{flipnm\_post}(A[0, \ldots, n-1], m),$$

with an array A (of length n) of tetrahedra and an integer `level` (maximum recursive level) as an input.

`flipnm` does the "forward" flips to remove the edge $[a, b]$. It returns a boolean value indicating whether the edge is removed or not and an integer m ($3 \leq m \leq n$): if the edge is not removed (`done = FALSE`), m indicates the current size of A (initially, $m := n$).

`flipnm_post` must be called immediately after `flipnm`. It releases the memory allocated in `flipnm` and it can perform the "backward" flips to undo the flip sequence performed by `flipnm`.

The basic subroutine `flipnm`($A[0, \ldots, n-1]$, `level`) consists of the following three steps:

1. Return `TRUE` if $n = 3$ and `flip32` is possible for $[a, b]$ and `FALSE` otherwise.
2. For each $i \in \{0, \ldots, n-1\}$ try to remove the face $[a, b, p_i]$ by `flip23`. If it is successfully flipped, reduce $|A|$ by 1. Update $A[0, \ldots, n-2]$ to contain the current set of tetrahedra at the edge $[a, b]$. Reuse the last entry ($A[n-1]$) to store the information of this `flip23`, refer (see Figure 1c). It then (recursively) calls `flipnm`($A[0, \ldots, n-2]$, level). When no face can be removed, go to Step 3.
3. If `level` > 0, try to remove an edge adjacent to $[a, b]$ by a `flipnm`. For each $i \in \{0, \ldots, n-1\}$ let $[x, y]$ be the edge either $[a, p_i]$ or $[b, p_i]$. Initialize an array $B[0, \ldots, n_1 - 1]$ of $n_1 \geq 3$ tetrahedra sharing $[x, y]$ and call `flipnm`($B[0, \ldots, n_1 - 1]$, level $- 1$). If $[x, y]$ is successfully removed, reduce $|A|$ by 1. Update $A[0, \ldots, n-2]$ to contain the current set of tetrahedra at the edge $[a, b]$. Reuse last entry ($A[n-1]$) to store the information of this `flip`*nm* and the address of the array B (to be able to release the occupied memory later). Then (recursively) call `flipnm`($A[0, \ldots, n-2]$). Otherwise, if $[x, y]$ is not removed, call `flipnm_post`($B[0, \ldots, n_1 - 1], m_1$) to free the memory. Return `FALSE` if no edge can be removed.

Since `flipnm` is called recursively, not every face and edge should be flipped in Steps 2 and 3. In particular, if B is allocated, i.e., `flipnm` is called recursively, we skip flipping faces and edges belonging to the tetrahedra in $A \cap B$.

In the most simple case, that is, without considering the option to reverse the flips, `flipnm_post`$(A[0, \dots, n-1], m)$ simply walks through the array A from $A[m]$ to $A[n-1]$ and checks if there is a saved `flipnm` flip. If so, the saved array address B is extracted and its memory is released.

In Step 2 there are at most $\binom{n}{n-3}/(n-3)!$ different flip sequences, depending on the specific choice of faces in A. Each individual flip sequence is equivalent to a sequence of the n vertices (apexes) in the link of the edge $[a, b]$. We reuse the entries of the array A to store each flip sequence. After a 2-to-3 flip, the number of the tetrahedra in the array A is reduced by 1, then we rearrange these tetrahedra by keeping the original order to the first $n-1$ entries of A. We then use the last entry $A[n-1]$ to store this flip. In particular, the following information is saved:

- a flag indicating a 2-to-3 flip;
- the original position i, meaning that the face $[a, b, p_i]$ is flipped;

This allows the inversion of a particular 2-to-3 flip as follows:

- use the position i to locate the tetrahedra $A[i-1] = [a, b, p_{i-1}, p_{i+1}]$ and get the three tetrahedra $[p_{i-1}, p_{i+1}, a, b]$, $[p_{i-1}, p_{i+1}, b, p_i]$ which share the edge $[p_{i-1}, p_{i+1}]$;
- perform a 3-to-2 flip on these three tetrahedra;
- insert two new tetrahedra into the array A: $A[i-1] = [a, b, p_i, p_{i-1}]$, and $A[i] = [a, b, p_i, p_{i+1}]$.

The above operations allow to reverse any sequence of flips stored in the array A.

In Step 3, if the selected edge $[a, p_i]$ is removed, the sequence of flips to remove $[a, p_i]$ is stored in the array B. We then use the last entry $A[n-1]$ to store this sequence of flips. In particular, the following information is saved:

- a flag indicates that this is a sequence of flips;
- the original position i, i.e., the edge $[a, p_i]$ is flipped;
- the address of array B in which the sequence of flips is stored.

This information allows us to inverse exactly this sequence of flips.

3.2. *Lazy searching flips*

During the mesh improvement process, we want to perform flips to improve the objective mesh quality function. Consider the case of removing an edge in order to improve the local mesh quality. The maximum possible number of flips at an edge is the Catalan number C_{n-2} (n is the size of A). Hence, the direct search for the optimal solution is only meaningful if n is very small. In most situations, an edge may not be flipped if we restrict ourselves to adjacent faces of the edge. Our strategy is to search and perform the flips as long as they can improve the current mesh quality. Our lazy searching scheme is not restricted by the number n and can be extended to adjacent edges.

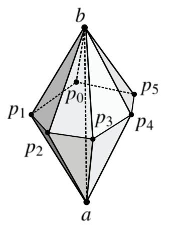
The lazy searching flip scheme is like a walk in a k-ary search tree (a rooted tree with at most k children at each node, see Fig. 1b). The root represents the edge $[a, b]$ to be flipped and each of the tree nodes represents either an adjacent face $[a, b, p_i]$ or an adjacent edge $[a, p_i]$ or $[b, p_i]$ of $[a, b]$. The edges of the tree represent our search paths. In particular, the directed edge from `level` l to $l+1$ represents either a `flip23` or a `flipnm`, and the reversed edge represents the inverse operation. The tree depth is the parameter `level`.

If at `level` > 0 we want to decide if an adjacent face $[a, b, p_i]$ should be flipped, we not only check if $[a, b, p_i]$ is flippable, but also check if this flip improves the local mesh quality. Note that we need to check only two of the three new tetrahedra: $[a, p_{i-1}, p_i, p_{i+1}]$ and $[b, p_{i-1}, p_i, p_{i+1}]$. The tetrahedron $[a, b, p_{i-1}, p_{i+1}]$ will be involved in the later flips, and will be flipped if the edge $[a, b]$ is flipped.

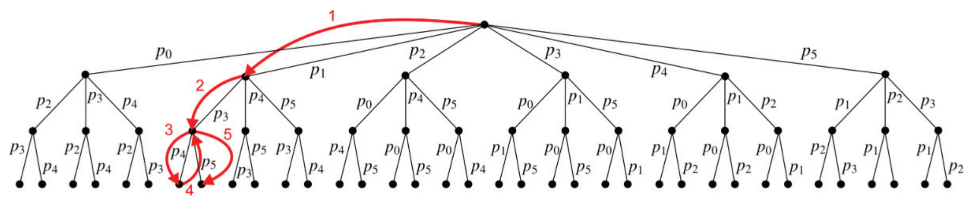
Once an improvement is found, the algorithms stops the search and returns without exploring other possibilities.

4. Mesh improvement strategy

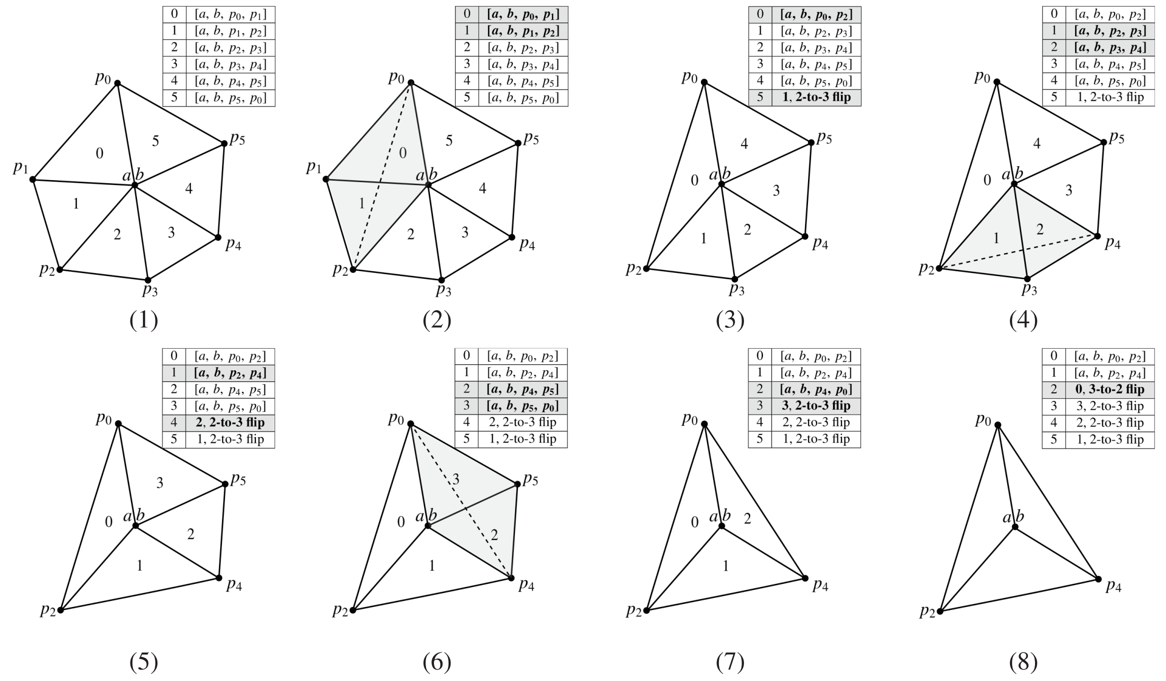
The goal of the proposed algorithm is to obtain a new isotropic mesh whose elements are "as close as possible" to the equilateral one. To achieve this goal we combine the local and global mesh operations described in sections 2 and 3.



(a) The initial state.



(b) The lazy flip search tree to remove the edge $[a, b]$. p_i identifies the face $[a, b, p_i]$ which is flipped via a 2-to-3 flip. The search path is highlighted with arrows.



(1)

0	$[a, b, p_0, p_1]$
1	$[a, b, p_1, p_2]$
2	$[a, b, p_2, p_3]$
3	$[a, b, p_3, p_4]$
4	$[a, b, p_4, p_5]$
5	$[a, b, p_5, p_0]$

(2)

0	**$[a, b, p_0, p_1]$**
1	**$[a, b, p_1, p_2]$**
2	$[a, b, p_2, p_3]$
3	$[a, b, p_3, p_4]$
4	$[a, b, p_4, p_5]$
5	$[a, b, p_5, p_0]$

(3)

0	**$[a, b, p_0, p_2]$**
1	$[a, b, p_2, p_3]$
2	$[a, b, p_3, p_4]$
3	$[a, b, p_4, p_5]$
4	$[a, b, p_5, p_0]$
5	**1, 2-to-3 flip**

(4)

0	$[a, b, p_0, p_2]$
1	**$[a, b, p_2, p_3]$**
2	**$[a, b, p_3, p_4]$**
3	$[a, b, p_4, p_5]$
4	$[a, b, p_5, p_0]$
5	1, 2-to-3 flip

(5)

0	$[a, b, p_0, p_2]$
1	**$[a, b, p_2, p_4]$**
2	$[a, b, p_4, p_5]$
3	$[a, b, p_5, p_0]$
4	**2, 2-to-3 flip**
5	1, 2-to-3 flip

(6)

0	$[a, b, p_0, p_2]$
1	$[a, b, p_2, p_4]$
2	**$[a, b, p_4, p_5]$**
3	**$[a, b, p_5, p_0]$**
4	2, 2-to-3 flip
5	1, 2-to-3 flip

(7)

0	$[a, b, p_0, p_2]$
1	$[a, b, p_2, p_4]$
2	**$[a, b, p_4, p_0]$**
3	**3, 2-to-3 flip**
4	2, 2-to-3 flip
5	1, 2-to-3 flip

(8)

0	$[a, b, p_0, p_2]$
1	$[a, b, p_2, p_4]$
2	**0, 3-to-2 flip**
3	3, 2-to-3 flip
4	2, 2-to-3 flip
5	1, 2-to-3 flip

(c) The sequence of flips. The edge $[a, b]$ is represented by one vertex in the center (except (8)). A face $[a, b, p_i]$ is represented by an edge. Array attached to each figure show the current content of A. (1) $n = 5$ tetrahedra share the edge $[a, b]$. In (2) and (3), $[a, b, p_1]$ is removed by a 2-to-3 flip. In (4) and (5), a 2-to-3 flip is done on $[a, b, p_3]$. In (6) and (7), $[a, b, p_5]$ is removed by a 2-to-3 flip. In (8), the edge $[a, b]$ is removed by a 3-to-2 flip.

Figure 1: An example of an edge removal by a sequence of flips.

4.1. Mesh quality

To say "as close as possible to an equilateral tetrahedron" is clear but it is not sufficiently precise from the mathematical point of view. To have a more precise criterion, the majority of the mesh improvement programs define a computable quantity $q(K)$ which quantifies how far a tetrahedron K is from being equilateral [2,17,27–30]. Here, we take into account the following two:

Aspect Ratio: This is one of the most classical ways to evaluate the quality of a tetrahedron. It is defined as

$$q_{ar}(K) := \sqrt{\frac{2}{3}}\frac{L}{h}, \tag{6}$$

where L is the longest edge and h is the shortest altitude of K. $q_{ar}(K) \geq 1$ by construction and an equilateral tetrahedron is characterized by $q_{ar}(K) = 1$.

Min-max Dihedral Angle: For each tetrahedron K we consider both the minimum and the maximum dihedral angles $\theta_{\min,K}$ and $\theta_{\max,K}$. An equilateral tetrahedron has $\theta_{\min,K} = \theta_{\max,K} = \arccos(1/3) \approx 70.56°$. Applying an operation that increases $\theta_{\min,K}$ or decreases $\theta_{\max,K}$ of a given tetrahedron K makes K "closer" to the equilateral shape. Note that this is not a classical quality measure, since we associate two quantities with each tetrahedron, which is one of the novel aspects of the proposed mesh improvement procedure.

These two quality measures refer to a single tetrahedron K of the mesh. However, the design of our mesh improvement scheme requires a quality measure for the *whole* mesh as a stopping criterion. To estimate the quality of the whole mesh we define

$$Q(\mathcal{T}_h) := \min_{K\in\mathcal{T}_h} (\theta_{\min,K}) . \tag{7}$$

This is a very effective quality measure. Indeed, if we consider a target dihedral angle $\theta_{\lim}$ and obtain a mesh $\mathcal{T}_h$ with $Q(\mathcal{T}_h) > \theta_{\lim}$, then *all* dihedral angles are guaranteed to be greater than $\theta_{\lim}$.

4.2. The scheme

The inputs for the mesh improvement algorithm are: a tetrahedral mesh $\mathcal{T}_h^{ini}$ of a PLC and a target minimum angle $\theta_{\lim}$. The output is a mesh $\mathcal{T}_h^{fin}$ with all elements having the smallest dihedral angle greater than $\theta_{\lim}$.

Algorithm 1 The proposed mesh improvement scheme.

```
IMPROVE($\mathcal{T}_h^{ini}$, $\theta_{\lim}$)
 1: repeat
 2:   ┌ repeat
 3:   │   repeat
 4:   │     repeat
 5:   │       ┌ repeat
 6:   │       │   MMPDE-based smoothing
 7:   │       │   Lazy Flips                                            → smooth and flip
 8:   │       └ until no point is moved or no flip is done or $Q(\mathcal{T}_h) \geq \theta_{\lim}$
 9:   │       remove the edges $l_e < 0.5\, l_{ave}$
10:   │       Lazy Flips                                                  → main loop
11:   │     until no edge is contracted or $Q(\mathcal{T}_h) \geq \theta_{\lim}$
12:   │     split the edges $l_e > 1.5\, l_{ave}$
13:   │     Lazy Flips
14:   │   until no edge is split or $Q(\mathcal{T}_h) > \theta_{\lim}$
15:   │   split the tetrahedrons $K$ such that $\theta_{\min,K} < \theta_{\lim}$
16:   │   Lazy Flips
17:   └ until no tetrahedron is removed or $Q(\mathcal{T}_h) > \theta_{\lim}$
18:   change the flip criterion for the Lazy Flip
19: until no operation is done in the main loop or $Q(\mathcal{T}_h) > \theta_{\lim}$
```

The scheme consists of five nested "**repeat . . . until**" loops, whose stopping criterion depends on the operations done inside the loop and $Q(\mathcal{T}_h)$. We apply the MMPDE smoothing and the Lazy Flip in the most internal loop (lines 5 to 8). The Lazy Flip is also exploited in the outer loops both on the whole mesh (lines 10, 13 and 16) and on the tetrahedrons involved in the local operations (lines 9, 12 and 15).

The Lazy Flip is based on an objective function and it is possible to consider several flipping criteria, which makes the design of the scheme flexible. We exploit this feature by using two objective functions and changing the flipping criterion in each iteration of the outer loop (line 18). In this paper we consider the following two (other criteria can be considered as well):

1. maximize $\theta_{\min,K}$ and minimize $\theta_{\max,K}$ simultaneously,
2. minimize of the aspect ratio.

The stopping criterion is always based on the minimum dihedral angle, $Q(\mathcal{T}_h)$, and the number of operations done.

After a number of iterations both the flipping and the smoothing procedure can stagnate, i.e., the mesh $\mathcal{T}_h$ converges to a fixed configuration where no more flips can be done and the smoothing procedure can't improve the point position anymore. Unfortunately, it is not a priori guaranteed that such a mesh satisfies the constraint on the target minimum dihedral angle θ_{lim}. To overcome this difficulty, we apply edge splitting, edge contraction, and tetrahedron splitting when this stagnation occurs (lines 9, 12 and 15 in Algorithm 1).

For the edge contraction and splitting we use the standard edge length criterion: we compute the average edge length l_{ave} of the actual mesh, contract the edges shorter than $0.5\,l_{\text{ave}}$ (line 9), and split (halve) the ones longer than $1.5\,l_{\text{ave}}$ (line 12). In line 15, we split a tetrahedron K with $\theta_{\min,K} < \theta_{\text{lim}}$ via a standard 1-to-4 flip by placing the new added point at the barycentre of K [32]. In this way, the algorithm can proceed with both flipping and smoothing towards a mesh satisfying $Q(\mathcal{T}_h) > \theta_{\text{lim}}$. At the moment, we are not interested in optimizing these operations, we exploit them only to overcome the algorithm stagnation.

The MMPDE smoothing can be easily parallelized because the computation of nodal velocities (Eqs (4) and (5)) requires local, element-wise computations which are independent from each other; we parallelize it with OpenMP [31] in order to accelerate the mesh improvement scheme. On the other hand, the Lazy Flip may propagate to neighbours and neighbours of neighbours, thus, it is complex and difficult to parallelize; in our tests we use a sequential implementation.

5. Numerical examples

We compare the new mesh improvement scheme with the aggressive mesh improvement algorithm of `Stellar` [2], the remeshing procedure of `CGAL` [16], and `mmg3d` [17] using the following examples:

- Rand1 and Rand2 are tetrahedral meshes of a cube generated by inserting randomly located vertices inside and on the boundary [2] (figs. 3 and 4),
- LShape is a tetrahedral mesh of an L-shaped PLC generated by `TetGen` [26] without optimizing the minimum dihedral angle (switches `-pa0.019`, Fig. 5),
- LenChallenge is a tetrahedral mesh of a cube with five holes generated by `TetGen` without optimizing the minimum dihedral angle (switches `-pa0.001`, Fig. 6),
- TetgenExample is a tetrahedral example mesh of a non-convex PLC with a hole provided with `TetGen` (Fig. 7).

We compare the histograms of the dihedral angles of final meshes, the minimum and the maximum dihedral angles $\theta_{\min,\mathcal{T}_h}$ and $\theta_{\max,\mathcal{T}_h}$, and the mean dihedral angle $\mu_{\mathcal{T}_h}$ and its standard deviation $\sigma_{\mathcal{T}_h}$ [33].

Smoothing and flipping on themselves. We test the effectiveness of the MMPDE smoothing and the Lazy Flip on themselves by using smoothing and flipping separately: we compare the MMPDE smoothing with the `Stellar` smoothing and the Lazy flip (`level` = 3) with `Stellar` flips.

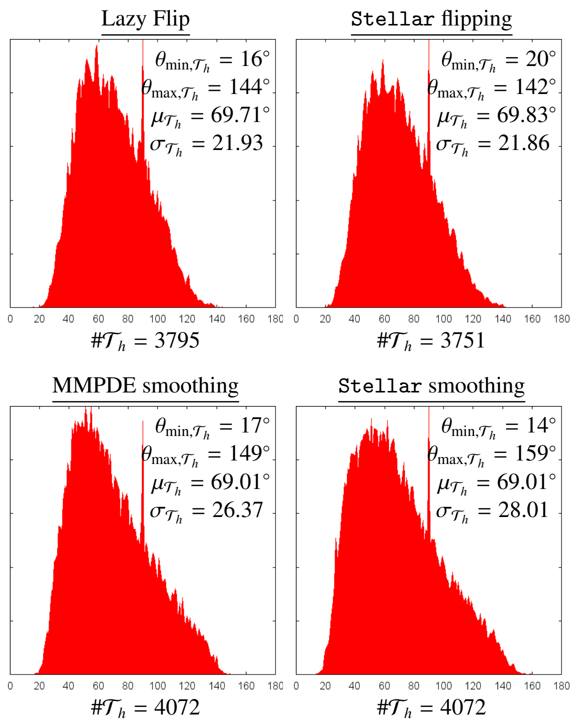
The results of the Lazy Flip are comparable with `Stellar` flips (Fig. 2, first row). However, the MMPDE smoothing is better than its counterpart in `Stellar` (Fig. 2, second row): in both examples it achieves larger $\theta_{\min,\mathcal{T}_h}$, noticeably smaller $\theta_{\max,\mathcal{T}_h}$, and a smaller standard deviation of the mean dihedral angle.

Note also, that the combination of smoothing and flipping produce much better results then if using them separately (cf. Fig. 2a with Fig. 5 and Fig. 2b with Fig. 7b, respectively).

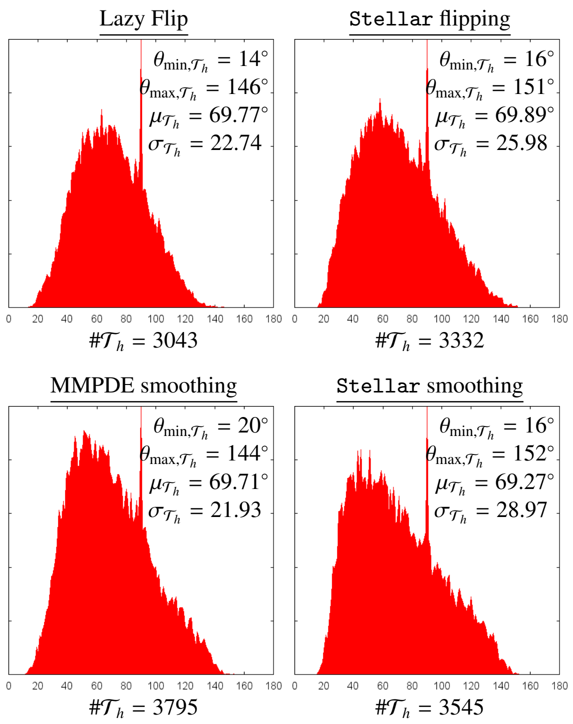
Full scheme. We compare the whole scheme with the aggressive mesh improvement algorithm `Stellar` [2], the remeshing procedure of `CGAL` [16], and `mmg3d` [17] (Figs 3 to 7).

Although all methods provide good results, the new scheme is better: $\theta_{\min,\mathcal{T}_h}$ is larger than the value obtained by `CGAL` or `mmg3d` and comparable to the value obtained by `Stellar`. Moreover, $\theta_{\max,\mathcal{T}_h}$ is smaller than the values obtained by `Stellar`, `CGAL`, or `mmg3d` in all the examples by one.

The mean dihedral angle $\mu_{\mathcal{T}_h}$ is always around 69.6°, which is close to the optimal value of $\arccos(1/3) \approx 70.56°$. The standard deviation $\sigma_{\mathcal{T}_h}$ for the new method is always smaller than for other methods. This quantitative consideration becomes clearer from the shape of the histograms in Figs 3 to 7: for our method, the dihedral angle distributions are more concentrated around the mean value in comparison to the distributions provided by `Stellar`, `CGAL`, and `mmg3d`.



(a) LShape.



(b) TetgenExample.

Figure 2: Comparison of flipping only (first row) and smoothing only (second row) for the LShape and the TetgenExample.

For the TetgenExample (Fig. 7) we also provide aspect ratio histograms (the results for the other examples are very similar and we omit them). The best aspect ratio distribution is clearly by our method and `Stellar`: for our method, most of the elements in the optimized mesh have aspect ratios from 1.1 to 1.6, followed by the `Stellar` mesh with most of the element aspect ratios in the range from 1.2 to 2.0; `CGAL` and `mmg3d` meshes contain many elements with aspect ratio in the range from 2 to 4. This is in good agreement with the comparison of the dihedral angles, since better dihedral angles lead to better aspect ratios. The reason for the better performance of `Stellar` and our method in comparison to `CGAL` and `mmg3d` is partially due to the fact that `CGAL` and `mmg3d` are meshing and re-meshing tools (fast but lower mesh quality), whereas `Stellar` and the new method are mesh optimization tools (more slowly but better mesh quality).

We also observed that Stellar aggressively removes elements from the mesh during optimization, which results in a much smaller number of mesh elements than in the original input. Our algorithm stays closer to the original number of elements. For example, compare the number of elements for the Rand1 example (Fig. 3): the input mesh has 5104 elements, Stellar's improved mesh has 1186 elements, whereas our method produces 3528 elements.

6. Conclusions

In all provided examples we obtain better results in terms of distributions of dihedral angles with respect to `Stellar`, `CGAL`, and `mmg3d`. Thus, the new mesh improvement algorithm is effective.

There are several possibilities to extend this mesh improvement scheme. One possibility is to apply this method to more general volume domains characterized by curved hulls. A second one is to obtain a more sophisticated way to contract/split edges or tetrahedra, which can improve the performances of both the MMPDE smoothing and the Lazy Flip. Finally, the MMPDE smoothing is based on the moving mesh method and, thus, allows a definition of a metric field. Hence, it can be extended straightforwardly to the adaptive and anisotropic setting.

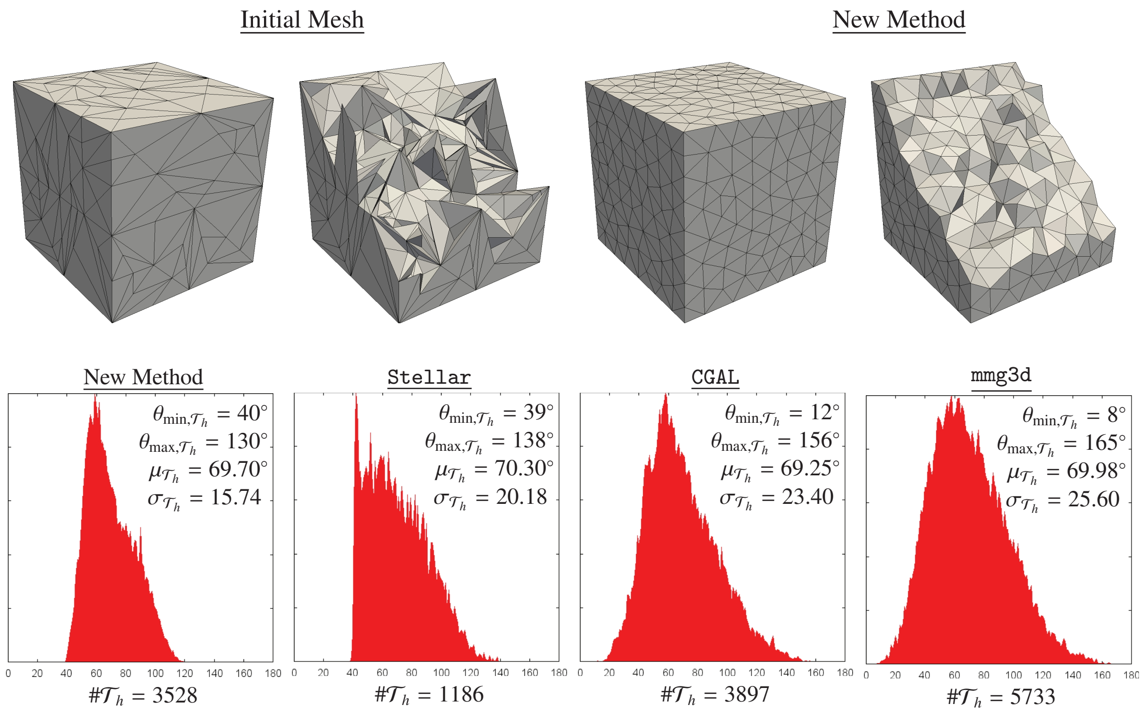


Figure 3: Rand1. The initial mesh with $\#\mathcal{T}_h = 5104$, the final (optimized) mesh, and statistics of dihedral angles for the final meshes.

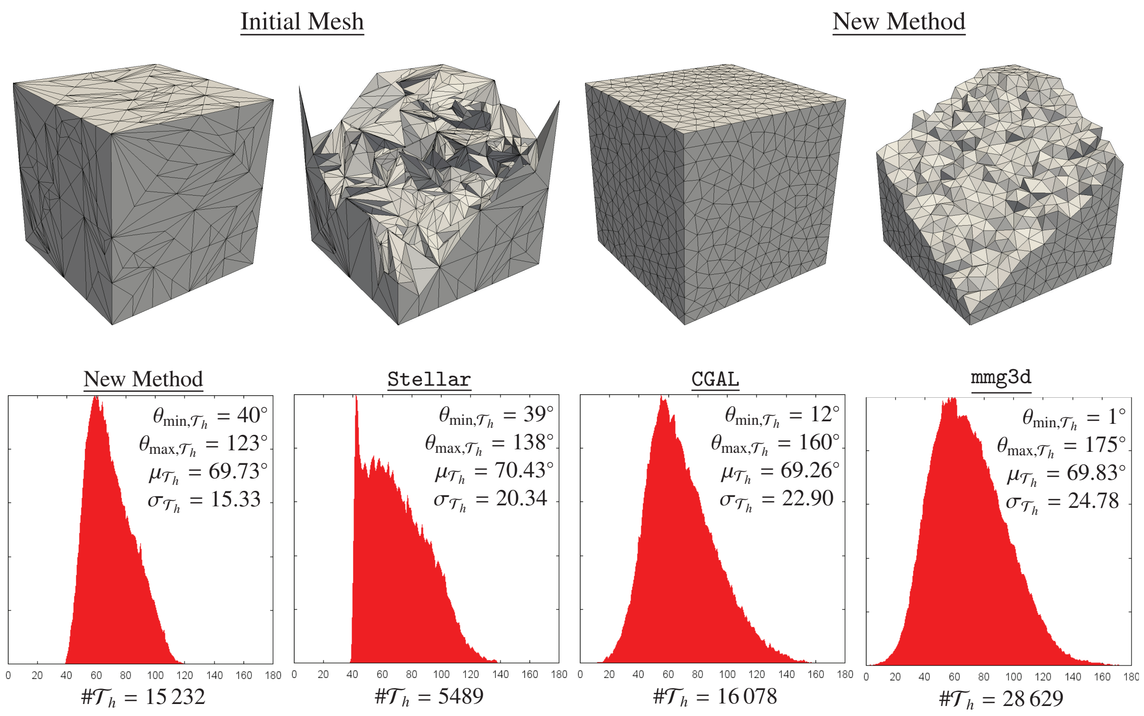


Figure 4: Rand2. The initial mesh with $\#\mathcal{T}_h = 25\,704$, the final (optimized) mesh, and statistics of dihedral angles for the final meshes.

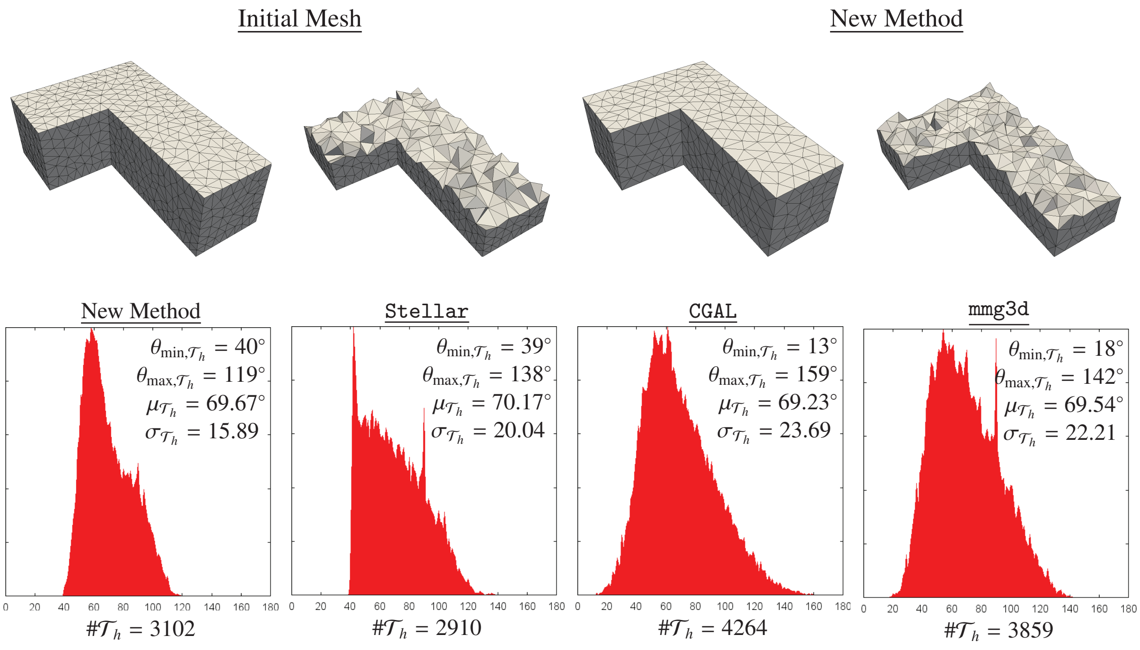


Figure 5: LShape. The initial mesh with $\#\mathcal{T}_h = 4072$, the final (optimized) mesh, and statistics of dihedral angles for the final meshes.

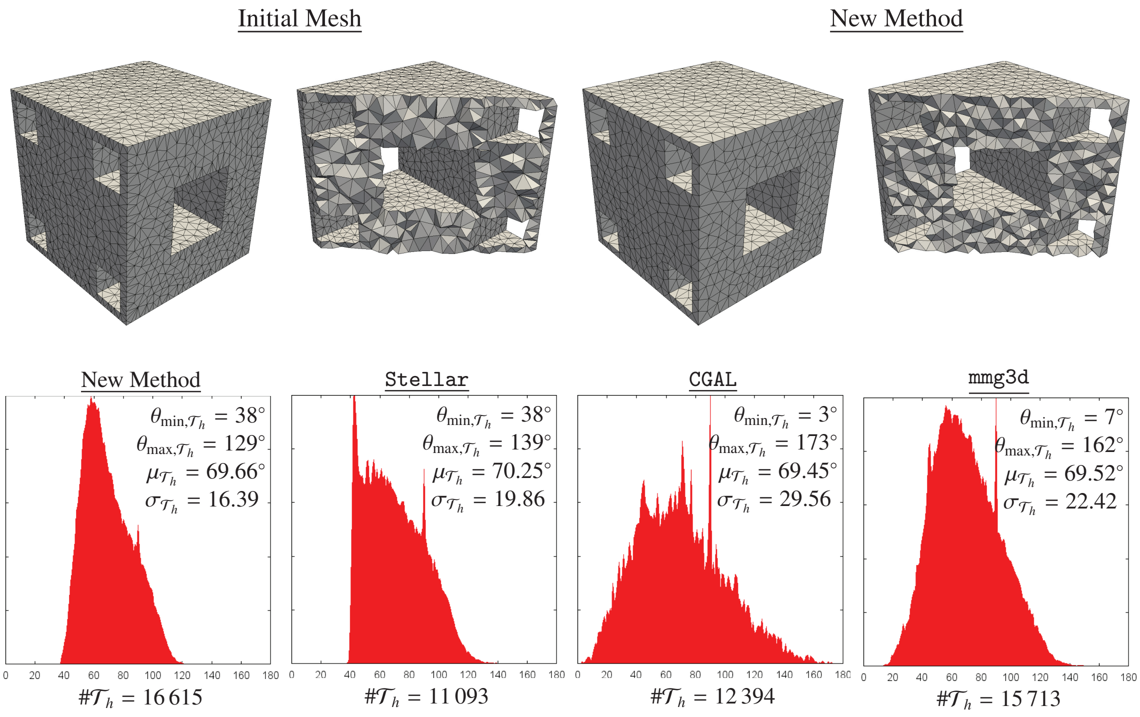
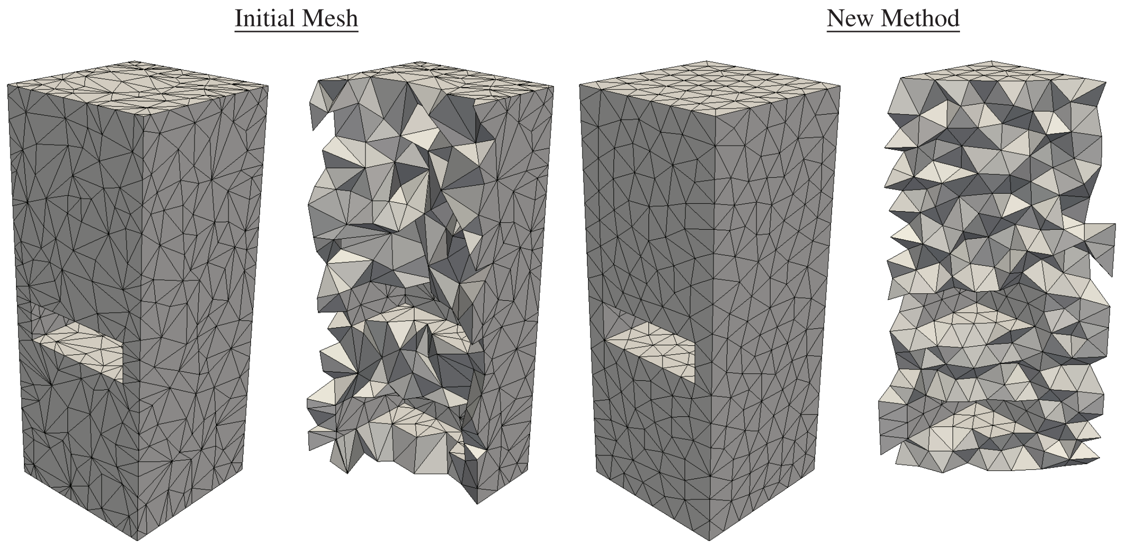
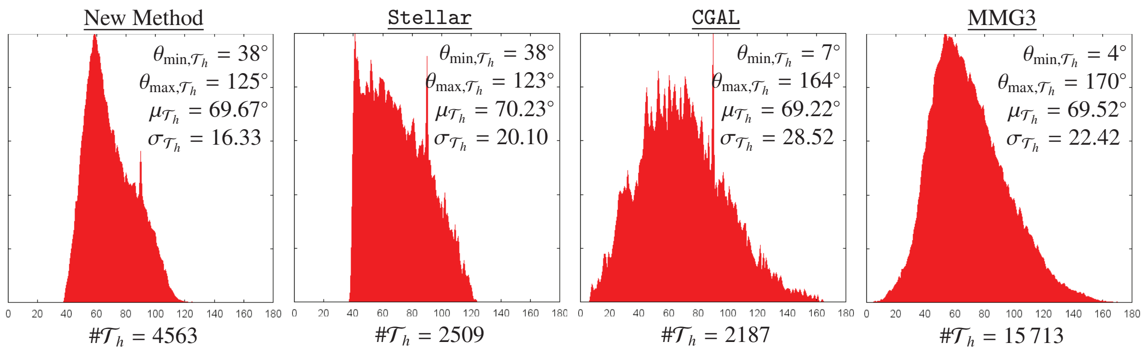


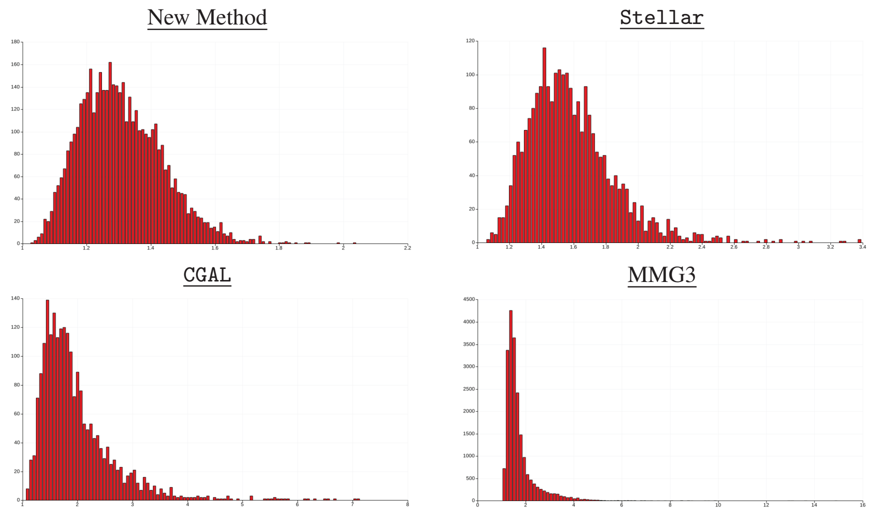
Figure 6: LenChallenge. The initial mesh with $\#\mathcal{T}_h = 16\,595$, the final (optimized) mesh, and statistics of dihedral angles for the final meshes.



(a) The initial mesh with $\#\mathcal{T}_h = 3545$ and the final optimized mesh.



(b) Dihedral angle comparison for the final meshes.



(c) Aspect ratio comparison for the final meshes.

Figure 7: TetgenExample. The initial mesh with $\#\mathcal{T}_h = 3545$, the final (optimized) mesh, and statistics of the dihedral angles and the aspect ratio.

Acknowledgments. The work of Franco Dassi was partially supported by the "Leibniz-DAAD Research Fellowship 2014". The authors are thankful to Jeanne Pellerin for her support in computing the examples with `mmg3d` [17] and the anonymous referees for their valuable comments and suggestions which helped to improve the quality of this paper.

References

[1] L. A. Freitag, C. Ollivier-Gooch, Tetrahedral mesh improvement using swapping and smoothing, International Journal for Numerical Methods in Engineering 40 (1997) 3979–4002.
[2] B. M. Klingner, J. R. Shewchuk, Aggressive tetrahedral mesh improvement, in: Proceedings of the 16th International Meshing Roundtable, 2007, pp. 3–23. URL: `http://graphics.cs.berkeley.edu/papers/Klingner-ATM-2007-10/`.
[3] B. Joe, Construction of three-dimensional improved-quality triangulations using local transformations, SIAM Journal on Scientific Computing 16 (1995) 1292–1307.
[4] P. George, H. Borouchaki, Back to edge flips in 3 dimensions, in: Proceedings of the 12th international meshing rountable, 2003.
[5] D. A. Field, Laplacian smoothing and delaunay triangulations, Comm. Appl. Num. Meth. 4 (1978) 709–712.
[6] S. H. Lo, A new mesh generation scheme for arbitaar planar domains, Int. J. Numer. Meth. Engng. 21 (1985) 1403–1426.
[7] R. E. Bank, PLTMG: a software package for solving elliptic partial differential equations, volume 15 of *Frontiers in Applied Mathematics*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1994. Users' guide 7.0.
[8] M. Shephard, M. Georges, Automatic three-dimensional mesh generation by the finite octree technique, Int. J. Numer. Meth. Engrg. 32 (1991) 709–749.
[9] L. A. Freitag, P. M. Knupp, Tetrahedral mesh improvement via optimization of the element condition number, Internat. J. Numer. Methods Engrg. 53 (2002) 1377–1391.
[10] P. M. Knupp, Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part I – A framework for surface mesh optimization, Internat. J. Numer. Methods Engrg. 48 (2000) 401–420.
[11] P. M. Knupp, Applications of mesh smoothing: copy, morph, and sweep on unstructured quadrilateral meshes, Internat. J. Numer. Methods Engrg. 45 (1999) 37–45.
[12] S. A. Canann, J. R. Tristano, M. L. Staten., An approach to combined laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes, in: Proceedings of the 7th International Meshing Rountable, 1998.
[13] L. Freitag, M. Jones, P. Plassmann, A parallel algorithm for mesh smoothing, SIAM J. Sci. Comput. 20 (1999) 2023–2040.
[14] W. Huang, Variational mesh adaptation: isotropy and equidistribution, J. Comput. Phys. 174 (2001) 903–924.
[15] W. Huang, L. Kamenski, A geometric discretization and a simple implementation for variational mesh generation and adaptation, J. Comput. Phys. 301 (2015) 322–337.
[16] The CGAL Project, CGAL User and Reference Manual, 4.8 ed., CGAL Editorial Board, 2016. URL: `http://doc.cgal.org/latest/Manual`.
[17] C. Dobrzynski, MMG3D: User Guide, Technical Report RT-0422, INRIA, 2012. URL: `https://hal.inria.fr/hal-00681813`.
[18] W. Huang, Y. Ren, R. D. Russell, Moving mesh partial differential equations (MMPDES) based on the equidistribution principle, SIAM J. Numer. Anal. 31 (1994) 709–730.
[19] W. Huang, R. D. Russell, Adaptive Moving Mesh Methods, volume 174 of *Applied Mathematical Sciences*, Springer, New York, 2011.
[20] W. Huang, L. Kamenski, On the mesh nonsingularity of the moving mesh PDE method, 2015. `arXiv:1512.04971`, submitted.
[21] W. Huang, L. Kamenski, H. Si, Mesh smoothing: an MMPDE approach, 2015. Research Notes of the 24th International Meshing Roundtable.
[22] W. Huang, Mathematical principles of anisotropic mesh adaptation, Commun. Comput. Phys. 1 (2006) 276–310.
[23] W. Huang, L. Kamenski, R. D. Russell, A comparative numerical study of meshing functionals for variational mesh adaptation, J. Math. Study 48 (2015) 168–186.
[24] J. R. Dormand, P. J. Prince, A family of embedded Runge-Kutta formulae, J. Comput. Appl. Math. 6 (1980) 19–26.
[25] G. L. Miller, Solving Irregularly Structured Problems in Parallel: 5th International Symposium, IRREGULAR'98 Berkeley, California, USA, August 9–11, 1998 Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, pp. 128–131. doi:`10.1007/BFb0018533`.
[26] H. Si, Tetgen, a delaunay-based quality tetrahedral mesh generator, ACM Trans. Math. Softw. 41 (2015) 11:1–11:36.
[27] C. Geuzaine, J.-F. Remacle, Gmsh Reference Manual, 1.12 ed., http://www.geuz.org/gmsh, 2003.
[28] F. Hecht, New development in freefem++, J. Numer. Math. 20 (2012) 251–265.
[29] J. R. Shewchuk, What is a good linear finite element? Interpolation, conditioning, anisotropy, and quality measures, 2002.
[30] D. A. Field, Qualitative measures for initial meshes, International Journal for Numerical Methods in Engineering 47 (2000) 887–906.
[31] L. Dagum, R. Menon, Openmp: an industry standard api for shared-memory programming, Computational Science & Engineering, IEEE 5 (1998) 46–55.
[32] H. Edelsbrunner, M. J. Ablowitz, S. H. Davis, E. J. Hinch, A. Iserles, J. Ockendon, P. J. Olver, Geometry and Topology for Mesh Generation (Cambridge Monographs on Applied and Computational Mathematics), Cambridge University Press, New York, NY, USA, 2006.
[33] R. Hogg, J. McKean, A. Craig, Introduction to Mathematical Statistics, Pearson education international, Pearson Education, 2005.