Weierstraß-Institut

für Angewandte Analysis und Stochastik

Leibniz-Institut im Forschungsverbund Berlin e. V.

Preprint ISSN 2198-5855

Tetrahedral mesh improvement using moving mesh smoothing and lazy searching flips

Franco Dassi, Lennard Kamenski, Hang Si

submitted: June 8, 2016

Weierstrass Institute
Mohrenstr. 39
10117 Berlin
Germany
E-Mail: franco.dassi@wias-berlin.de
lennard.kamenski@wias-berlin.de
hang.si@wias-berlin.de

No. 2270

Berlin 2016

2010 *Mathematics Subject Classification.* 65N50, 65K10.

Key words and phrases. Mesh improvement, moving mesh, edge flipping, mesh quality, mesh smoothing.

The work of Franco Dassi was partially supported under the “Leibniz - DAAD Research Fellowship 2014”. The authors would like to thank Dr. Jeanne Pellerin for her support in computing the examples with MMG3D.

Edited by
Weierstraß-Institut für Angewandte Analysis und Stochastik (WIAS)
Leibniz-Institut im Forschungsverbund Berlin e. V.
Mohrenstraße 39
10117 Berlin
Germany

Fax: +49 30 20372-303
E-Mail: preprint@wias-berlin.de
World Wide Web: http://www.wias-berlin.de/

Abstract

In this paper we combine two new smoothing and flipping techniques. The moving mesh smoothing is based on the integration of an ordinary differential coming from a given functional. The lazy flip technique is a reversible edge removal algorithm to automatically search flips for local quality improvement. On itself, these strategies already provide good mesh improvement, but their combination achieves astonishing results which have not been reported so far. Provided numerical examples show that we can obtain final tetrahedral meshes with dihedral angles between 40° and 123°. We compare the new method with other publicly available mesh improving codes.

1 Introduction

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 the better meshes than if applied individually [11, 25]. In this paper we combine new smoothing and 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 [24, 11, 12, 25]. 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. A more effective way of improving the mesh quality 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 ($n \geq 3$) 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 case, and then pick the optimal one. Another way is to use the dynamic programming technique to automatically find the optimal configuration. However, the cases increase exponentially and finding the optimal solution with brute force is very time-consuming.

In this paper, we propose a mesh quality improvement using edge flips. It automatically searches a valid configuration which improves the worst mesh quality of the initial configuration. Since it finds only a local optimum to gain speed, we call it the *lazy searching flips*.

The key idea to realize the automatic search which is able to reverse the edge flip operation, once it finds out that the current configuration does not improve the mesh quality. This scheme is described in Section 3.

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 [7, 28], where a vertex is moved to the geometric center of its neighboring vertices. While economic, easy to implement, and often effective, Laplacian smoothing does not actually guarantee an improvements of the mesh quality.

Alternatives are optimization-based methods that are effective for a variety of mesh quality measures, for example the ratio of the area to the sum of the squared edge lengths [1], the ratio of the volume to a power of the sum of the squared face areas [30] for tetrahedral meshes, the condition number of the Jacobian matrix of the affine mapping between the reference element and physical elements [10], or various other measures [11, 27, 26, 2] These optimization-based methods are local and sequential (by nature), 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 [9].

In our work, we employ the moving mesh PDE (MMPDE) method, defined as the gradient flow equation of a meshing functional (i.e., 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 [16]. We employ a simple and efficient approach developed recently for the implementation of variational mesh generation, which is based on a direct geometric discretization of the underlying meshing functional on simplicial meshes [18]. Most importantly, for this method, the nodal mesh velocities are expressed in a simple, analytical matrix form, which siplifies the implementation. Compared to existing optimization-based (local) mesh smoothing methods, the considered method has several advantages: 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, mesh velocities have a simple analytical form, and it can be parallelized easily.

In this paper we provide a deep numerical analysis on the proposed mesh improvement scheme, which combines lazy searching flips with the MMPDE smoothing. More specifically, we compare the results of the whole algorithm with some well-known libraries and software, Stellar [25], CGAL [33] and MMG3D [4]. We also provide comparison between our methods and the smoothing and flipping schemes 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 [22, 23]. In this paper, we use its discrete form [18, 19, 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

We 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 the generic element and $\hat{K}$ the reference element taken as a regular simplex with the volume $|\hat{K}| = 1/\#\mathcal{T}_h$. Further, let $\mathbb{J}_K$ be the inverse of the Jacobian matrix of the affine mapping from the mesh element K to the reference element $\hat{K}$. 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 all elements to be shaped similarly to $\hat{K}$ (these conditions are the simplified versions of the equidistribution and alignment conditions [17, 23]).

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

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

where $\theta \in (0, 0.5]$ and $p > 1$ are dimensionless parameters (in section 5 we use $\theta = 1/3$ and $p = 3/2$).[1] I_h is a Riemann sum of a continuous functional for variational mesh adaptation based on equidistribution and alignment [16] 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.

[1] This is a specific choice and other meshing functionals are possible. An interested reader is referred to [20] for a numerical comparison of meshing functionals for variational mesh adaptation.

2.2 Vertex velocities and the mesh movement

The vertex velocities can be computed analytically [18, eq. (39) to (41)] using the scalar-by-matrix differentiation [18, 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 [18, eq. (39) and (41)] as

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

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 $\mathbb{J}_K = \hat{E}E_K^{-1}$). For the considered G we have [18, 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

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

where ω_i is the patch of the vertex $\boldsymbol{x}_i$, 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 [5].

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 [19].

2.3 Velocity adjustment for the boundary vertices

The vertex velocities need to be modified for the boundary vertices. For example, if $\boldsymbol{x}_i$ is a fixed boundary vertex, then we set its velocity 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 mesh 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 in section 5, the input geometry is a Piecewise Linear Complex (PLC) [29], 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 describe our lazy searching flip algorithm and explain how to remove an edge and how to reverse the removal using flips. Then we present the lazy searching algorithm for mesh improvement.

3.1 Edge Removal and Its Inverse

A basic edge removal algorithm [32] 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 tetrahedra in $\mathcal{T}_h$ sharing $[a, b]$, i.e., the cardinality of A is $|A| = n$ ($n \geq 3$). 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 $A[i]$ 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 lead to the removal of the edge $[a, b]$. Moreover, we also want to reverse the sequence of flips, so that we can return to the original state. In the following, the index i will take values in $\{0, 1, \ldots, n-1\}$, i.e., $i+1$ means $(i+1) \mod n$ and $i-1$ means $(i-1+n) \mod n$.

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 variable b indicating whether the edge is removed or not and an integer m ($3 \leq m \leq n$): if the edge is not removed ($\texttt{done} = \texttt{FALSE}$), 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 "backwards" flips to undo the sequence of flips performed by `flipnm`.

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

Step 1: Return `TRUE` if $n = 3$ and `flip32` is possible for $[a, b]$. Otherwise return `FALSE`.

Step 2 ($n > 3$)**:** For each $i \in \{0, \ldots, n-1\}$ try to remove the face $[a, b, p_i]$ by `flip23`. If it is successfully flipped, then 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 to Figure 1. It then (recursively) calls $\texttt{flipnm}(A[0, \ldots, n-2], \text{level})$. When no face can be removed, go to Step 3.

Step 3 ($n > 3$)**:** If $\texttt{level} > 0$, 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 $\texttt{flipnm}(B[0, \ldots, n_1 - 1], \text{level} - 1)$. If $[x, y]$ is successfully removed, then 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 `flipnm` and the address of the array B (to be able to release the occupied memory later). Then (recursively) call $\texttt{flipnm}(A[0, \ldots, n-2])$. Otherwise, if $[x, y]$ is not removed, call $\texttt{flipnm\_post}(B[0, \ldots, n_1 - 1], m_1)$ 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, we skip flipping faces and edges belonging to the tetrahedra in $A \cap B$ if the array B is allocated, i.e., `flipnm` is called recursively.

In the simplest case, that is, without considering the option to reverse the flips, $\texttt{flipnm\_post}(A[0, \ldots, n-1], m)$ 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:

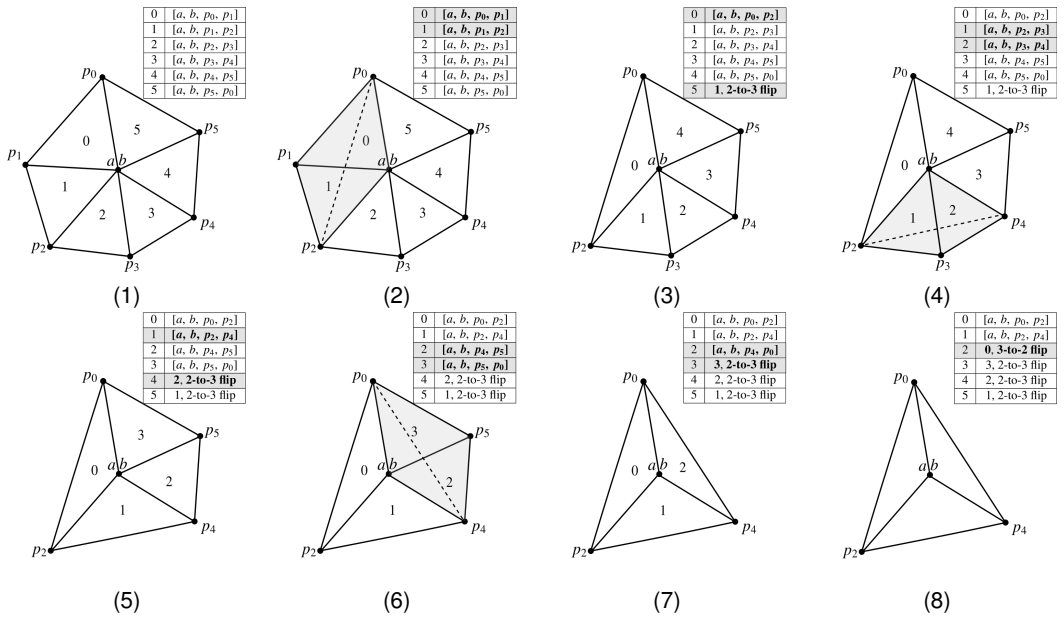

Figure 1: Edge removal by a sequence of flips. In these figures except (8), the edge $[a, b]$ is represented by one vertex centered in the middle. The face $[a, b, p_i]$ is represented by an edge. The array A is attached to each figure to show its current content. (1) There are $n = 5$ tetrahedra sharing at the edge $[a, b]$. In (2) and (3), the face $[a, b, p_1]$ is removed by a 2-to-3 flip. In (4) and (5), a 2-to-3 flip is done on the face $[a, b, p_3]$. In (6) and (7), the face $[a, b, p_5]$ is removed by a 2-to-3 flip. In (8), the edge $[a, b]$ is removed by a 3-to-2 flip.

(i) a flag indicating a 2-to-3 flip;

(ii) 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 in three steps:

(1) the position i allows us 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 $[p_{i-1}, p_{i+1}]$;

(2) perform a 3-to-2 flip on these three tetrahedra;

(3) 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 are saved:

(i) a flag indicates that this is a sequence of flips; and

(ii) the original position i i.e., the edge $[a, p_i]$ is flipped;

(iii) 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 that we want to remove 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}, where 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, which is a rooted tree with at most k children at each node (see fig. 2). The root of the tree represents the edge $[a, b]$ to be flipped. Each tree node 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 an inverse of this operation. The tree depth is the parameter `level`.

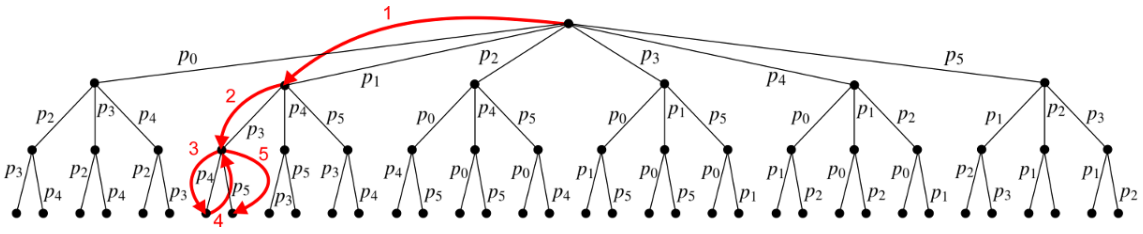

Figure 2: An example of a Lazy Flip search tree to remove the edge $[a, b]$ of the example in fig. 1. On each branch of the tree we report p_i to identify the face $[a, b, p_i]$ which is flipped via a 2-to-3 flip. The search path is highlighted with arrows.

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 only need to check 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.

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. This section describes how we combine the local and global mesh operations described in sections 2 and 3 to achieve this goal.

4.1 Mesh Quality

The way of saying “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 defines a computable quantity $q(K)$ which quantifies how far a tetrahedron K is from the equilateral shape [25, 4, 13, 14, 31, 8]. Here, we take into account the following ones.

- *Aspect Ratio:* This is one of the most classical way 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 height 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 $\theta_{\max,K}$, respectively. An equilateral tetrahedron has $\theta_{\min,K} = \theta_{\max,K} = \arccos(1/3) \approx 70.56°$. If we apply an operation that increases $\theta_{\min,K}$ or decreases $\theta_{\max,K}$ of a given tetrahedron K, we will make K "closer" to the equilateral shape. We underline that this is not a classical quality measure, since we associate two quantities with each tetrahedron and it 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 single target quantity which evaluates the quality of the *whole* mesh so that we can stop the procedure. We define

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

to have an estimate of the quality of the whole mesh. This is a very effective quality measure. Indeed, if we consider a target dihedral angle $\theta_{\lim}$ and we get a mesh $\mathcal{T}_h$ where $Q(\mathcal{T}_h) > \theta_{\lim}$, it is guaranteed that *all* dihedral angles are greater than $\theta_{\lim}$.

4.2 The Scheme

The pseudo-code of the novel mesh improvement procedure is described in Algorithm 1. The inputs are: an initial tetrahedral mesh $\mathcal{T}_h^{ini}$ of a PLC and a target minimum angle limit, $\theta_{\lim}$. The output is a final mesh $\mathcal{T}_h^{fin}$ where all elements have the smallest dihedral angle greater than $\theta_{\lim}$.

Algorithm 1 The mesh improvement proposed in this paper.

IMPROVE($\mathcal{T}_h^{ini}, \theta_{\lim}$)

```
1:  repeat
2:      repeat
3:          repeat
4:              repeat
5:                  repeat
6:                      MMPDE-Based Smoothing
7:                      Lazy Flips
8:                  until no point is moved or no flip is done or Q(T_h) ≥ θ_lim   → smooth and flip
9:              remove too short edges
10:             Lazy Flips
11:         until no edge is contracted or Q(T_h) ≥ θ_lim
12:         split too long edges
13:         Lazy Flips
14:     until no edge is split or Q(T_h) > θ_lim
15:     split bad tetrahedrons
16:     Lazy Flips
17: until no tetrahedron is removed or Q(T_h) > θ_lim                              → main loop
18: change the criterion of the Lazy Flip
19: until no operation is done or Q(T_h) > θ_lim
```

Algorithm 1 consists in 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). Moreover the Lazy Flip is even exploited in the outer loops on both the whole mesh (lines 10, 13 and 16) and on the tetrahedrons involved in the local mesh operations (lines 9, 12 and 15).

One interesting aspect of the Lazy Flip is that it is based on an objective function. In our mesh improvement scheme we exploit even this feature. Indeed, the whole algorithm, (lines 2 to 17) is inside a big "**repeat . . . until**" loop where we change the flipping criterion at each step. In this paper we consider two objective functions:

1 Minimize of the aspect ratio.

2 Maximize $\theta_{\min,K}$ and minimize $\theta_{\max,K}$ simultaneously.

However, the design of the mesh improvement strategy is flexible and it is possible to consider different quality criteria for the Lazy Flip.

Since both the MMPDE Smoothing and the Lazy Flip are the most often used operations in Algorithm 1, they must be cheap from the computational point of view. On the one hand, the algorithm behind the Lazy Flip is complex and requires a lot of effort to be parallelized. On the other hand, the MMPDE Smoothing requires computation of nodal mesh velocities given in a simple, analytical matrix form, which can be parallelized in a straightforward way. We parallelize the MMPDE Smoothing via Open-MPI [3] to increase the speed of the mesh improvement scheme.

Algorithm stagnation. In the proposed mesh improvement strategy the MMPDE Smoothing and the Lazy Flip are the main operations to improve the quality of the mesh. After some iterations both the flipping and the smoothing procedure stagnate, i.e. the mesh $\mathcal{T}_h$ converges to a fixed configuration where no more flips can be done and the smoothing procedure does not 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 $\theta_{\lim}$.

To overcome this difficulty, we apply some simple operations such as edge splitting, edge contraction and tetrahedron splitting when this stagnation occurs, lines 9, 12 and 15 of Algorithm 1. In this way, the algorithm can proceed with both flipping and smoothing towards a mesh satisfying $Q(\mathcal{T}_h) > \theta_{\lim}$. At the moment, we are not interested in optimizing these operations, we exploit them only to overcome the algorithm stagnation.

For the edge contraction and splitting we use the standard edge length criterion. More precisely we compute the average edge length in the actual mesh, l_{ava}, and we contract the edges which are shorter than $0.5\, l_{\text{ava}}$ and split (halve) ones that are longer than $1.5\, l_{\text{ava}}$.

In line 15, we split a tetrahedron K via a standard 1-to-4 flip, where we place the new added point at the barycenter of K [6]. We apply this operation on the tetrahedra with $\theta_{\min,K} < \theta_{\lim}$.

5 Numerical Examples

We compare the new mesh improvement scheme with the aggressive mesh improvement algorithm of Stellar [25], the remeshing procedure of CGAL [33], and MMG3D [4] with the following examples:

- RAND1 and RAND2 are tetrahedral meshes of a cube generated by inserting randomly located vertices both inside and on the boundary of a cube [25],
- LSHAPE is a tetrahedral mesh of an L-shaped PLC generated by `tetgen` [32] without optimizing the minimum dihedral angle (switches `-pa0.019`),
- LENCHALLENGE a tetrahedral mesh of a cube with five holes generated by `tetgen` without optimizing the minimum dihedral angle (switches `-pa0.001`),
- TETGENEXAMPLE is a tetrahedral example mesh of a non-convex PLC with a hole provided with `tetgen`.

We provide the histograms of the dihedral angles of final meshes and report the minimum and the maximum dihedral angles $\theta_{\min,\mathcal{T}_h}$ and $\theta_{\max,\mathcal{T}_h}$. For the better comparison, we also provide the mean dihedral angle $\mu_{\mathcal{T}_h}$ and its standard deviation $\sigma_{\mathcal{T}_h}$ [15].

Comparison on Smoothing and Flipping. To test the effectiveness of the MMPDE based Smoothing technique and the Lazy Flip algorithm, we improve a tetrahedral mesh via only one of these two strategies. Moreover we compare the MMPDE smoothing with Stellar smoothing (fig. 3, left) as well as the Lazy flip ($\texttt{level} = 3$) with Stellar flips (fig. 3, right).

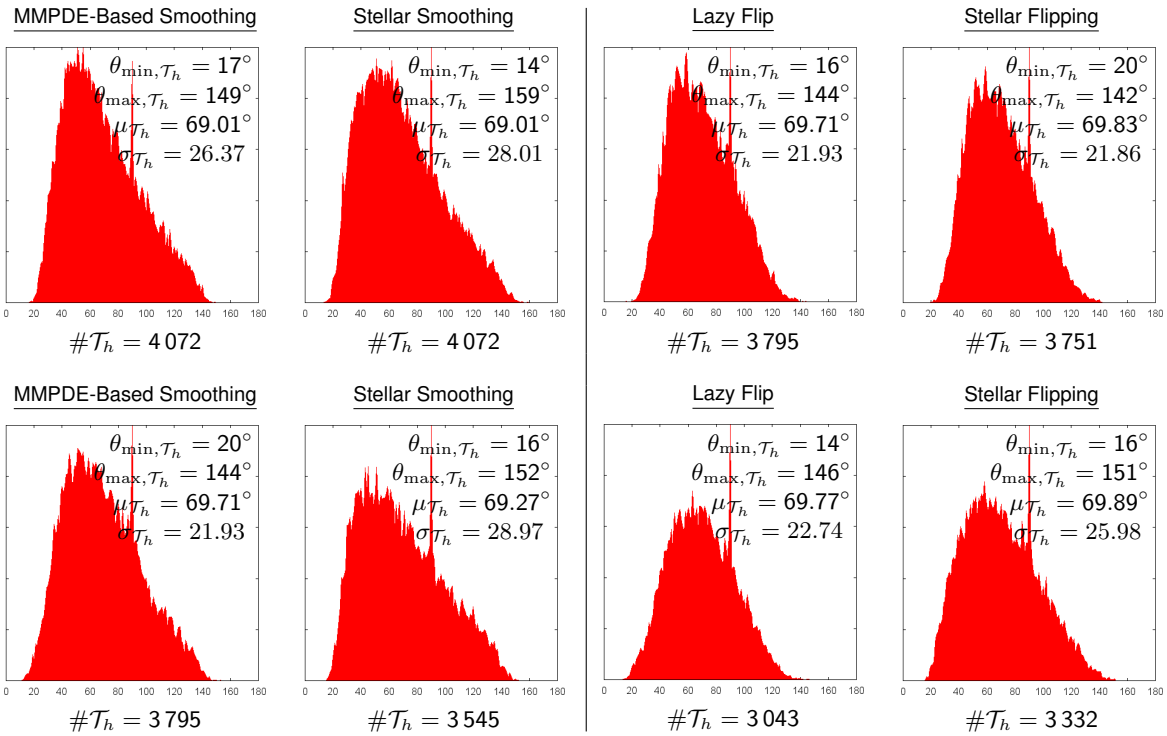

Figure 3: Comparison of smoothing only (MMPDE smoothing vs. Stellar Smoothing, left) and flipping only (Lazy Flip vs. Stellar Flipping, right) for the LSHAPE (first row) and the TETGENEXAMPLE (second row).

In both test cases the results are comparable with Stellar. Moreover, the new MMPDE smoothing is noticeably better than the smoothing procedure used in Stellar (fig. 3, left): in both examples it achieves larger $\theta_{\min,\mathcal{T}_h}$, smaller $\theta_{\max,\mathcal{T}_h}$, and a smaller standard deviation of the mean dihedral angle.

Mesh Improvement Scheme Comparison. Now we compare the whole isotropic mesh improvement scheme with the aggressive mesh improvement algorithm Stellar [25], the remeshing procedure of CGAL [33], and MMG3D [4] (figs. 4 to 8).

Although all methods provide good results, the new improvement scheme is better: $\theta_{\min,\mathcal{T}_h}$ is larger than if obtained by CGAL or MMG3D and comparable with 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 $\mu_{\mathcal{T}_h}$ and its standard deviation $\sigma_{\mathcal{T}_h}$ is always around 69.6°, i.e., close to the optimal value of $\arccos(1/3) \approx 70.56°$. The standard deviation for the new method is smaller than for other methods. This quantitative consideration becomes clearer from the shape of the histograms in Figures 4 to 8: in our method, the dihedral angle distributions are more concentrated around the mean value in comparison to the distributions provided by Stellar, CGAL and MMG3D.

6 Conclusions

The effectiveness of our mesh improvement algorithm is provided by several examples. In all these examples we obtain better results in terms of distributions of dihedral angles with respect to Stellar, CGAL, and MMG3D.

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, since the MMPDE Smoothing method is based on the moving mesh method which, in tern, allows

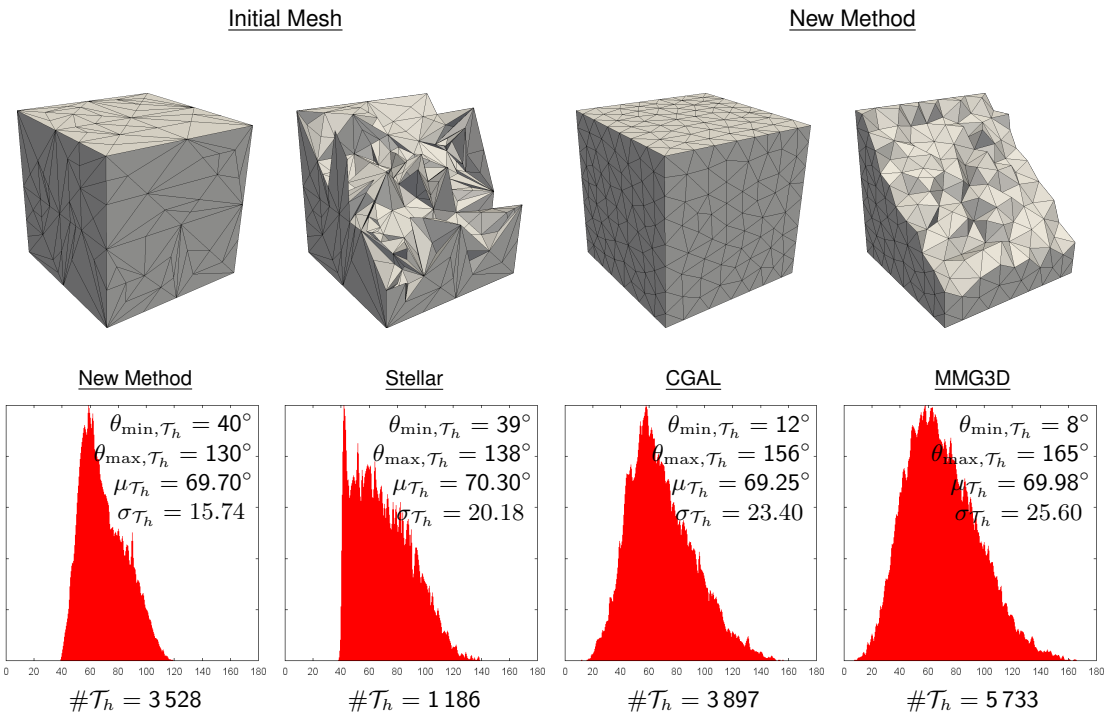

Figure 4: RAND1. Top left: the initial mesh ($\#\mathcal{N}_h = 1\,086$ and $\#\mathcal{T}_h = 5\,104$) and a mesh clip. Top right: the final (optimized) mesh and the same clip. Second row: statistics of the different mesh improvement schemes.

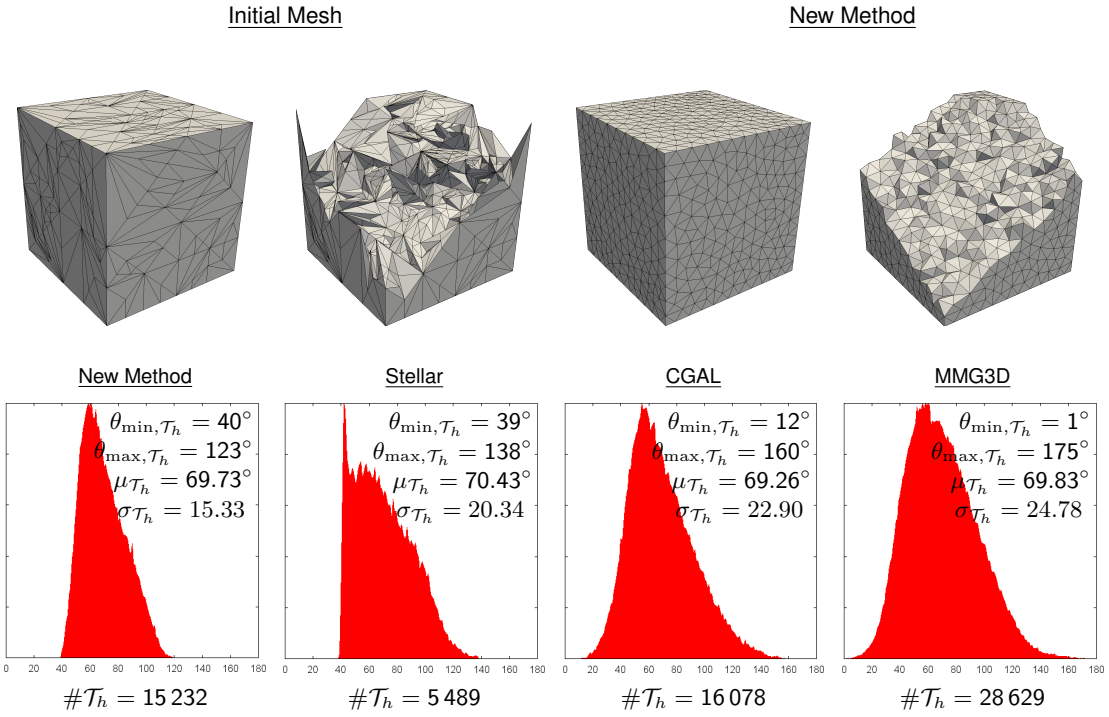

Figure 5: RAND2. Top left: the initial mesh ($\#\mathcal{N}_h = 5\,086$, $\#\mathcal{T}_h = 25\,704$) and a mesh clip. Top right: the final (optimized) mesh and the same clip. Second row: statistics of the different mesh improvement schemes.

a definition of a metric field, the smoothing procedure can be extended straightforwardly to the anisotropic setting.

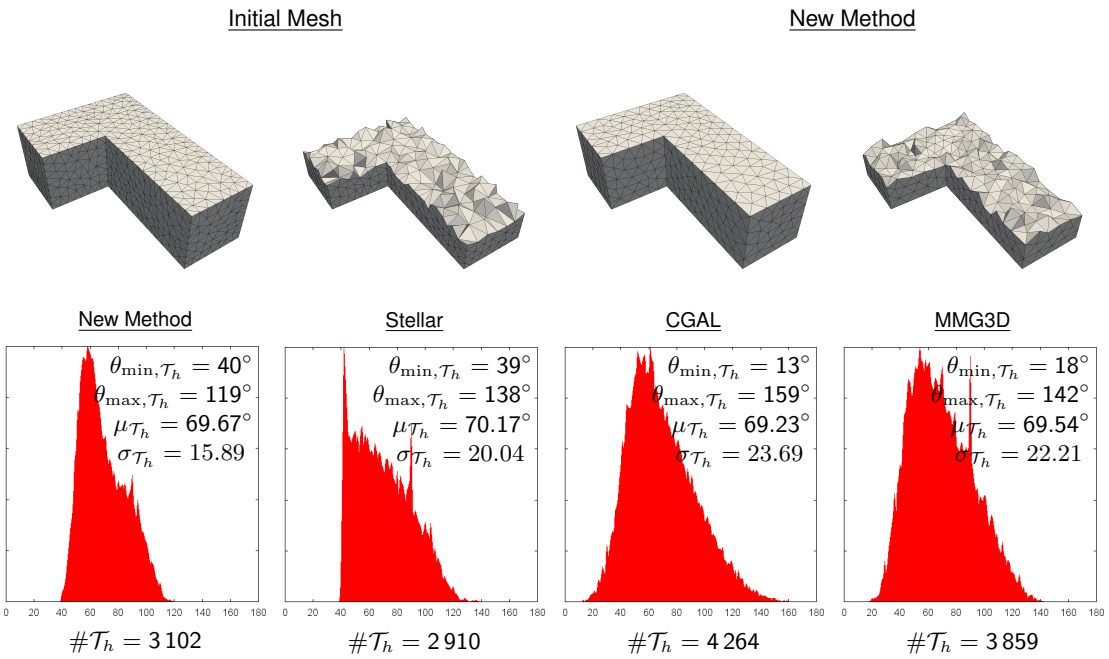

Figure 6: LSHAPE. Top left: the initial mesh ($\#\mathcal{N}_h = 1\,114$, $\#\mathcal{T}_h = 4\,072$) and a mesh clip. Top right: the final (optimized) mesh and the same clip. Second row: statistics of the different mesh improvement schemes.

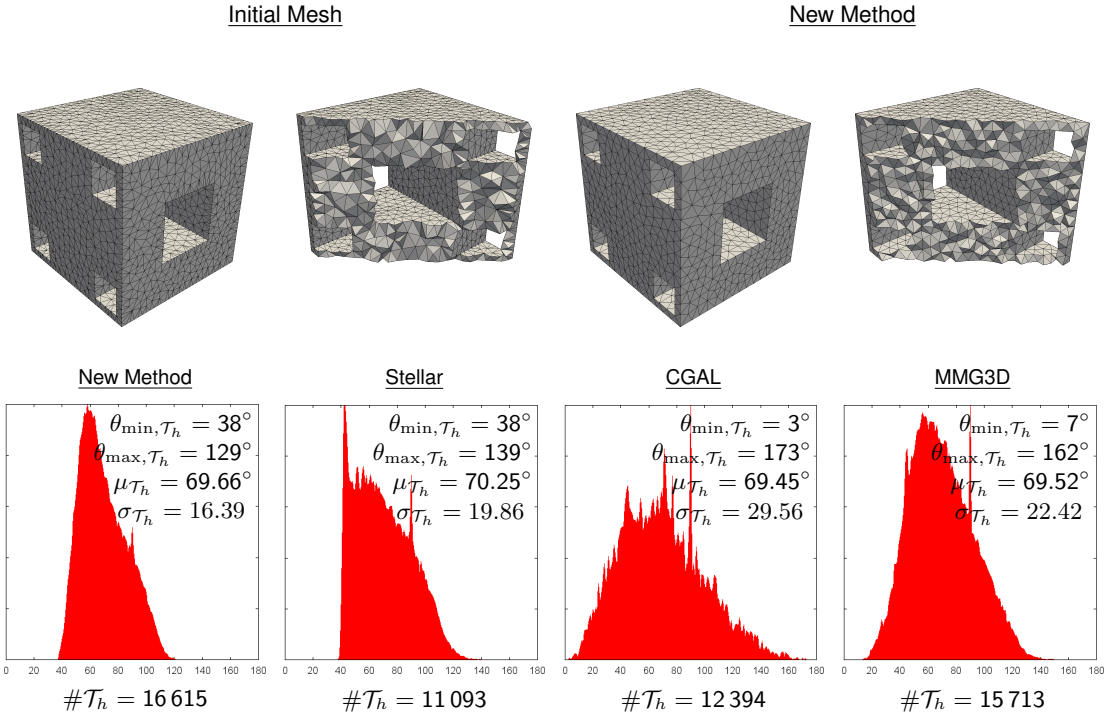

Figure 7: LENCHALLENGE. Top left: the initial mesh ($\#\mathcal{N}_h = 4\,478$, $\#\mathcal{T}_h = 16\,595$) and a mesh clip. Top right: the final (optimized) mesh and the same clip. Second row: statistics of the different mesh improvement schemes.

References

[1] 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'

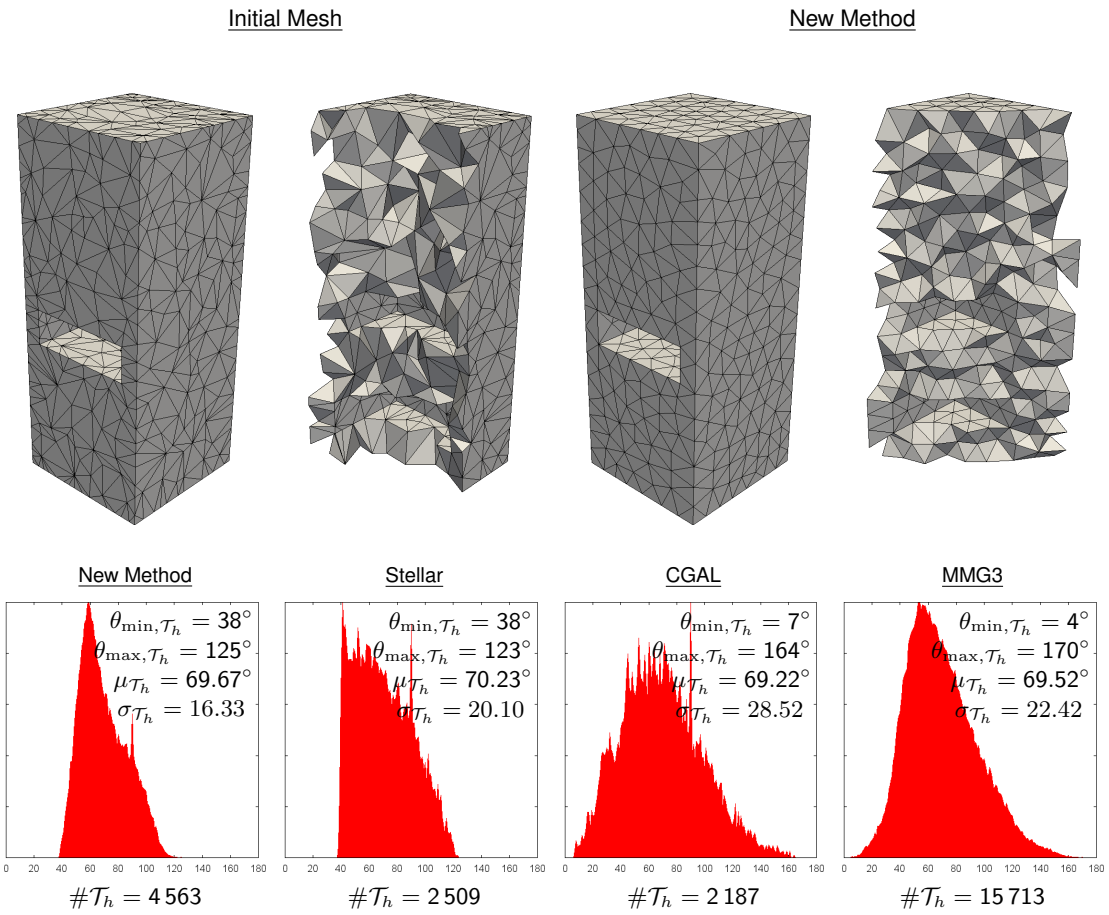

Figure 8: TETGENEXAMPLE. Top left: the initial mesh ($\#\mathcal{N}_h = 1\,456$, $\#\mathcal{T}_h = 3\,545$) and a mesh clip. Top right: the final (optimized) mesh and the same clip. Second row: statistics of the different mesh improvement schemes.

guide 7.0.

[2] S. A. Canann, J. R. Tristano, and 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.

[3] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[4] C. Dobrzynski. MMG3D: User Guide. Technical Report RT-0422, INRIA, Mar. 2012.

[5] J. R. Dormand and P. J. Prince. A family of embedded Runge-Kutta formulae. *J. Comput. Appl. Math.*, 6(1):19–26, 1980.

[6] H. Edelsbrunner, M. J. Ablowitz, S. H. Davis, E. J. Hinch, A. Iserles, J. Ockendon, and P. J. Olver. *Geometry and Topology for Mesh Generation (Cambridge Monographs on Applied and Computational Mathematics)*. Cambridge University Press, New York, NY, USA, 2006.

[7] D. A. Field. Laplacian smoothing and delaunay triangulations. *Comm. Appl. Num. Meth.*, 4:709–712, 1978.

[8] D. A. Field. Qualitative measures for initial meshes. *International Journal for Numerical Methods in Engineering*, 47(4):887–906, 2000.

[9] L. Freitag, M. Jones, and P. Plassmann. A parallel algorithm for mesh smoothing. *SIAM J. Sci. Comput.*, 20(6):2023–2040, 1999.

[10] L. A. Freitag and P. M. Knupp. Tetrahedral mesh improvement via optimization of the element condition number. *Internat. J. Numer. Methods Engrg.*, 53(6):1377–1391, 2002.

[11] L. A. Freitag and C. Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40(21):3979–4002, 1997.

[12] P. George and H. Borouchaki. Back to edge flips in 3 dimensions, in: Proceedings of the 12th international meshing rountable, September 2003.

[13] C. Geuzaine and J.-F. Remacle. *Gmsh Reference Manual*. http://www.geuz.org/gmsh, 1.12 edition, Aug. 2003.

[14] F. Hecht. New development in freefem++. *J. Numer. Math.*, 20(3-4):251–265, 2012.

[15] R. Hogg, J. McKean, and A. Craig. *Introduction to Mathematical Statistics*. Pearson education international. Pearson Education, 2005.

[16] W. Huang. Variational mesh adaptation: isotropy and equidistribution. *J. Comput. Phys.*, 174(2):903–924, 2001.

[17] W. Huang. Mathematical principles of anisotropic mesh adaptation. *Commun. Comput. Phys.*, 1(2):276–310, 2006.

[18] W. Huang and L. Kamenski. A geometric discretization and a simple implementation for variational mesh generation and adaptation. *J. Comput. Phys.*, 301:322–337, 2015.

[19] W. Huang and L. Kamenski. On the mesh nonsingularity of the moving mesh PDE method, 2015. Submitted.

[20] W. Huang, L. Kamenski, and R. D. Russell. A comparative numerical study of meshing functionals for variational mesh adaptation. *J. Math. Study*, 48(2):168–186, 2015.

[21] W. Huang, L. Kamenski, and H. Si. Mesh smoothing: an MMPDE approach, 2015. Research Notes of the 24th International Meshing Roundtable.

[22] W. Huang, Y. Ren, and R. D. Russell. Moving mesh partial differential equations (MMPDES) based on the equidistribution principle. *SIAM J. Numer. Anal.*, 31(3):709–730, 1994.

[23] W. Huang and R. D. Russell. *Adaptive Moving Mesh Methods*, volume 174 of *Applied Mathematical Sciences*. Springer, New York, 2011.

[24] B. Joe. Construction of three-dimensional improved-quality triangulations using local transformations. *SIAM Journal on Scientific Computing*, 16(6):1292–1307, 1995.

[25] B. M. Klingner and J. R. Shewchuk. Aggressive tetrahedral mesh improvement. In *Proceedings of the 16th International Meshing Roundtable*, pages 3–23, Oct. 2007.

[26] P. M. Knupp. Applications of mesh smoothing: copy, morph, and sweep on unstructured quadrilateral meshes. *Internat. J. Numer. Methods Engrg.*, 45(1):37–45, 1999.

[27] 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.*, 48:401–420, 2000.

[28] S. H. Lo. A new mesh generation scheme for arbitaar planar domains. *Int. J. Numer. Meth. Engng.*, 21:1403–1426, 1985.

[29] G. L. Miller. *Solving Irregularly Structured Problems in Parallel: 5th International Symposium, IRREGULAR'98 Berkeley, California, USA, August 9–11, 1998 Proceedings*, chapter Control volume meshes using sphere packing, pages 128–131. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[30] M. Shephard and M. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *Int. J. Numer. Meth. Engrg.*, 32:709–749, 1991.

[31] J. R. Shewchuk. What is a good linear finite element? Interpolation, conditioning, anisotropy, and quality measures, 2002.

[32] H. Si. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, 41(2):11:1–11:36, 2015.

[33] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.8 edition, 2016.