Computer-Aided Design ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Tetrahedral mesh improvement using moving mesh smoothing, lazy searching flips, and RBF surface reconstruction☆

Franco Dassi [a], Lennard Kamenski [b,*], Patricio Farrell [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

ARTICLE INFO

Keywords:
Mesh improvement
Mesh quality
Edge flipping
Mesh smoothing
Moving mesh
Radial basis functions

ABSTRACT

Given a tetrahedral mesh and objective functionals measuring the mesh quality which take into account the shape, size, and orientation of the mesh elements, our aim is to improve the mesh quality as much as possible. In this paper, we combine the *moving mesh smoothing*, based on the integration of an ordinary differential equation coming from a given functional, with the *lazy flip* technique, a reversible edge removal algorithm to modify the mesh connectivity. Moreover, we utilize *radial basis function* (RBF) surface reconstruction to improve tetrahedral meshes with curved boundary surfaces. Numerical tests show that the combination of these techniques into a mesh improvement framework achieves results which are comparable and even better than the previously reported ones.

1. Introduction

The key mesh improvement operations considered in this work are *smoothing*, which moves the mesh vertices, *flipping*, which changes the mesh topology without moving the mesh vertices, and a *smooth boundary reconstruction*. Previous work shows that the combination of smoothing and flipping achieves better results than if applied individually [1,2]. In this paper, we combine the recently developed flipping and smoothing methods into one mesh improvement scheme and apply them in combination with a smooth boundary reconstruction via radial basis functions.

Mesh smoothing improves the mesh quality by improving vertex locations, typically through Laplacian smoothing or some optimization-based algorithm. Most commonly used mesh smoothing methods are Laplacian smoothing and its variants [3,4], 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 mesh validity. Alternatives are optimization-based methods that are effective with respect to certain mesh quality measures such as the ratio of the area to the sum of the squared edge lengths [5], the ratio of the volume to a power of the sum of the squared face areas [6], the condition number of the Jacobian matrix of the affine mapping between the reference element and physical elements [7], or various other measures [1,8–10]. Most of the optimization-based methods are local and sequential, combining Gauss–Seidel-type iterations with location optimization problems over each patch. There is also a parallel algorithm that solves a sequence of independent subproblems [11].

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 [12] and employ the recently developed direct geometric discretization [13] 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 [14].

Flipping is the most efficient way to locally improve the mesh quality and it has been extensively addressed in the literature [15,1,16,2]. In the simplest case, 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 into one edge removal operation, which extends the 3-to-2 and 4-to-4 flips. This operation removes the common edge of $n \geq 3$ adjacent tetrahedra by replacing them with $m = 2n - 4$ new tetrahedra (the so-called n-to-m flip). There are at most C_{n-2} possible variants to remove an edge by a n-to-m flip,

☆ This special issue was edited by Scott Canann, Steven J. Owen & Hang Si.

* Corresponding author.
E-mail addresses: franco.dassi@unimib.it (F. Dassi), kamenski@wias-berlin.de (L. Kamenski), farrell@wias-berlin.de (P. Farrell), si@wias-berlin.de (H. Si).

https://doi.org/10.1016/j.cad.2017.11.010
0010-4485/

where $C_n = \frac{(2n)!}{(n+1)!\,n!}$ is the Catalan number. If n is small (e.g., $n < 7$), one can enumerate all possible cases, compute the mesh quality for each case, and then pick the optimal one. Another way is to use 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.

In this paper, we propose the so-called *lazy searching flips*. The key idea is to automatically explore sequences of flips to remove a given edge in the mesh. If a flip sequence leads to a configuration which does not improve the mesh quality, the algorithm reverses this sequence and explores another one (see Section 3 and Figs. 2a to 2c). Once an improvement is found, the algorithms stops the search and returns without exploring the remaining possibilities.

When considering more arbitrary meshes (which may not be piecewise planar), we need to make sure that new nodes are added in a consistent way. To achieve this we use *RBF surface reconstruction* as introduced in [17]. Radial basis functions are a very useful tool in the context of higher-dimensional interpolation as they dispense with the expensive generation of a mesh [18–20]. Here, we will employ them to approximate the underlying continuous surface so that we can project nodes onto it as proposed in [21,22]. This problem turns out to be very challenging for meshes with arbitrary boundary. Hence, we begin with a relatively simple mesh. For more complicated examples we first refine the boundary by using the RBF reconstruction and projection method and then keep the boundary nodes fixed while interior nodes may move.

In this paper, we provide a detailed numerical study of a combination of the MMPDE smoothing with the lazy searching flips and RBF surface reconstruction. More specifically, we compare the results of the whole algorithm with `Stellar` [2], `CGAL` [23] and `mmg3d` [24]. 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 functional (the MMPDE approach). Originally, the method was developed in the continuous setting [25,26]. In this paper, we use its discrete form [13,14,27], 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 $\Omega \subset \mathbb{R}^d$ with $d \geq 1$. Let $\mathcal{T}_h$ denote the simplicial mesh as well as $\#\mathcal{N}_h$ and $\#\mathcal{T}_h$ the numbers of its vertices and elements, respectively. Let K be a generic mesh element and $\hat{K}$ the reference element taken as a regular simplex with volume $|\hat{K}| = 1/\#\mathcal{T}_h$. Further, let F_K' be the Jacobian matrix of the affine mapping $F_K : \hat{K} \to K$ from the reference element $\hat{K}$ to a mesh element K. For notational simplicity, we denote the inverse of the Jacobian by $\mathbb{J}_K$, i.e., $\mathbb{J}_K := (F_K')^{-1}$ (see Fig. 1).

Then, the mesh $\mathcal{T}_h$ is uniform if and only if

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

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 [28,26]).

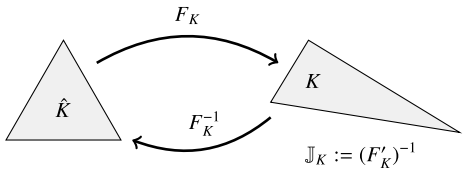

Fig. 1. Reference element $\hat{K}$, mesh element K, and the corresponding mappings F_K and F_K^{-1}.

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

$$I_h = \sum_K |K|\; G\left(\mathbb{J}_K, \det \mathbb{J}_K\right) \tag{2}$$

with

$$G(\mathbb{J}, \det \mathbb{J}) = \theta\left(\operatorname{tr}\left(\mathbb{J}\mathbb{J}^T\right)\right)^{\frac{dp}{2}} + (1-2\theta)\, d^{\frac{dp}{2}} (\det \mathbb{J})^p, \tag{3}$$

where $\theta \in (0, 0.5]$ and $p > 1$ are dimensionless parameters (in Section 6, we use $\theta = 1/3$ and $p = 3/2$). This is a specific choice and other meshing functionals are possible. The interested reader is referred to [29] for a numerical comparison of meshing functionals for variational mesh adaptation.

In Eq. (2), I_h is a Riemann sum of a continuous functional for variational mesh adaptation based on equidistribution and alignment [12] and depends on the vertex coordinates $\boldsymbol{x}_i$, $i = 1, \ldots, \#\mathcal{N}_h$. The corresponding vertex velocities $\boldsymbol{v}_i$ for the mesh movement are defined as

$$\boldsymbol{v}_i := \frac{d\boldsymbol{x}_i}{dt} = -\left(\frac{\partial I_h}{\partial \boldsymbol{x}_i}\right)^T, \quad i = 1, \ldots, \#\mathcal{N}_h, \tag{4}$$

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

2.2. Vertex velocities and the mesh movement

The vertex velocities $\boldsymbol{v}_i$ can be computed analytically [13, Eqs (39) to (41)] using scalar-by-matrix differentiation [13, Sect. 3.2]. Denote the vertices of K and $\hat{K}$ by $\boldsymbol{x}_j^K$ and $\hat{\boldsymbol{x}}_j$, $j = 0, \ldots, d$, and define the element edge matrices as

$$E_K = [\boldsymbol{x}_1^K - \boldsymbol{x}_0^K, \ldots, \boldsymbol{x}_d^K - \boldsymbol{x}_0^K],$$
$$\hat{E} = [\hat{\boldsymbol{x}}_1 - \hat{\boldsymbol{x}}_0, \ldots, \hat{\boldsymbol{x}}_d - \hat{\boldsymbol{x}}_0].$$

Note, that $\hat{E}E_K^{-1} = \mathbb{J}_K$. Then, the local mesh velocities are given element-wise [13, Eqs (39) and (41)] by

$$\begin{bmatrix} (\boldsymbol{v}_1^K)^T \\ \vdots \\ (\boldsymbol{v}_d^K)^T \end{bmatrix} = -G_K E_K^{-1} + E_K^{-1}\frac{\partial G_K}{\partial \mathbb{J}}\hat{E}E_K^{-1} + \frac{\partial G_K}{\partial \det \mathbb{J}}\frac{\det(\hat{E})}{\det(E_K)}E_K^{-1}, \tag{5}$$

$$(\boldsymbol{v}_0^K)^T = -\sum_{j=1}^{d}(\boldsymbol{v}_j^K)^T,$$

where $G_K = G(\mathbb{J}_K, \det \mathbb{J}_K)$ and

$$\frac{\partial G_K}{\partial \mathbb{J}} = \frac{\partial G}{\partial \mathbb{J}}(\mathbb{J}_K) = dp\theta\left(\operatorname{tr}(\mathbb{J}_K\mathbb{J}_K^T)\right)^{\frac{dp}{2}-1}\mathbb{J}_K^T,$$
$$\frac{\partial G_K}{\partial \det \mathbb{J}} = \frac{\partial G}{\partial \det \mathbb{J}}(\det \mathbb{J}_K) = p(1-2\theta)d^{\frac{dp}{2}}(\det \mathbb{J}_K)^{p-1}$$

are the derivatives of G with respect to its first and second argument [13, Example 3.2] evaluated at $\mathbb{J} = \mathbb{J}_K$ and $\det(\mathbb{J}) = \det \mathbb{J}_K$.

Please cite this article in press as: Dassi F., et al. Tetrahedral mesh improvement using moving mesh smoothing, lazy searching flips, and RBF surface reconstruction. Computer-Aided Design (2017), https://doi.org/10.1016/j.cad.2017.11.010.