applied sciences

MDPI

Article

Simultaneous Smoothing and Untangling of 2D Meshes Based on Explicit Element Geometric Transformation and Element Stitching

Shuli Sun *, Zhihong Gou and Mingguang Geng

LTCS, Department of Mechanics and Engineering Science, College of Engineering, Peking University, Beijing 100871, China; gouzhihong@pku.edu.cn (Z.G.); m.g.geng@pku.edu.cn (M.G.)

* Correspondence: sunsl@mech.pku.edu.cn

Received: 1 July 2020; Accepted: 17 July 2020; Published: 21 July 2020

Abstract: Mesh quality can affect both the accuracy and efficiency of numerical solutions. This paper first proposes a geometry-based smoothing and untangling method for 2D meshes based on explicit element geometric transformation and element stitching. A new explicit element geometric transformation (EEGT) operation for polygonal elements is firstly presented. The transformation, if applied iteratively to an arbitrary polygon (even inverted), will improve its regularity and quality. Then a well-designed element stitching scheme is introduced, which is achieved by carefully choosing appropriate element weights to average the temporary nodes obtained by the above individual element transformation. Based on the explicit element geometric transformation and element stitching, a new mesh smoothing and untangling approach for 2D meshes is proposed. The proper choice of averaging weights for element stitching ensures that the elements can be transitioned smoothly and uniformly throughout the calculation domain. Numerical results show that the proposed method is able to produce high-quality meshes with no inverted elements for highly tangled meshes. Besides, the inherent regularity and fine-grained parallelism make it suitable for implementation on Graphic Processor Unit (GPU).

Keywords: mesh smoothing; mesh untangling; explicit element geometric transformation; parallel algorithm

1. Introduction

Mesh quality has great effects on both the accuracy and efficiency of numerical solutions based on the finite element method and the finite volume method [1–3]. Therefore, meshes with high quality are required for various finite element and finite volume based numerical simulations. For a given geometric model, meshes can be generated using various mesh generation methods [4,5]. Among them Delaunay-based methods are quite popular, since they are well suited for automation, of reasonable complexity, and produce meshes with high quality.

For problems with smooth solutions, a uniform mesh is usually preferred over a nonuniform one because it readily lends itself to efficient solution. Even when adaptivity is required, an isotropic mesh can be preferred for the same reason if it can resolve the solution without using an undue number of mesh points. Anisotropic meshes are favored when there is a need for better alignment of the mesh with certain solution directions, such as those arising due to boundary or interior layers and sharp interfaces. The adaptive moving mesh methods are usually used in these situations, where point locations are dynamically relocated in time while adapting to the evolving structures in the solution [6]. Das et al. [7–13] solved a series of time-dependent partial differential equations (PDEs) using the adaptive moving mesh methods. Their works have rigorous mathematical derivations and elegant

proofs. The comparison results with the methods based on priori meshes show that the adaptive solution on posteriori generated meshes will converge to the exact solution with higher accuracy and reliability. The adaptive moving mesh methods can effectively improve the accuracy of numerical simulations, and the mesh optimization methods studied in this paper try to ensure the feasibility and improve the accuracy of numerical simulations.

However, tangled meshes can occur during mesh generation, mesh optimization, large-scale deformation, moving domains, and mesh morphing. Generally speaking, such meshes cannot be directly used for numerical simulations. Inverted elements should be eliminated before carrying out numerical simulations, since they usually produce invalid solutions [14]. There are many methods for eliminating inverted elements, including remeshing [15], local mesh modification [16], topological transformations [17] and mesh smoothing. Remeshing may not be automatic, especially for models with complex geometries. Local mesh modification can be very effective. However, for a tangled mesh that contains a large number of inverted elements, remeshing the whole computational domain may be a better choice. Topological transformations improve the mesh quality by inserting or removing a node or modifying the node-element connection relationship. Edge flip, edge collapse, and node insertion are three popular topological optimization methods to improve the mesh quality. While remeshing, local mesh modification and topological transformations methods are valuable in mesh untangling, we focus on mesh smoothing or optimization via node-movement in this paper.

Mesh smoothing improves the mesh quality by simply relocating or adjusting node positions while preserving the mesh topology. There exists many node movement methods and the majority of them do not specifically address the mesh untangling problem. For example, Laplace smoothing [18] is widely used to improve mesh quality, where node positions are iteratively updated by the arithmetic mean of neighboring node positions. However, it can lead to a tangled mesh from one that initially is untangled. Therefore, a set of methods known as barrier methods are designed to prevent the initial untangled meshes from becoming tangled. However, these methods can not be used when the initial mesh is tangled.

To address the above issue, there are several methods specialized to untangle an initial tangled mesh, and they generally do not consider the element shape. Therefore, a barrier smoothing technique is needed to obtain a valid and high-quality mesh. Freitag and Plassman tried to untangle the input mesh by maximizing the minimum area/volume of elements within each local patch of the mesh [19]. The system is guaranteed to converge, but not necessarily to an untangled mesh. Knupp used a local non-barrier metric that penalized inverted elements [20] to untangle the mesh. Global optimization of the objective function is recommended when untangling larger and more difficult meshes, as all nodes are updated simultaneously. However, the global optimization strategy is very time-consuming in untangling large-scale meshes. As with the other methods, there is no guarantee of obtaining a valid mesh when using this method. Franks and Knupp proposed two new metrics for simultaneous mesh untangling and smoothing using the target matrix paradigm [21]. This method can effectively eliminate inverted elements and do not require that one specify a parameter as in Reference [20]. The above methods [20,21] are adopted for the well-known mesh quality improvement toolkit (Mesquite) [22].

There are also methods developed for simultaneously eliminating inverted elements and improving the mesh quality. Kim et al. [23] proposed a multiobjective optimization method that combined two or more objective functions into a single objective function, and solved it using a nonlinear conjugate gradient method. Escobar et al. [24,25] proposed a method incorporating a modified quality metric that ensured higher penalty for inverted elements. This method is effective but also does not guarantee an untangled output mesh. Gargallo-Peiró et al. [26] proposed a method optimizing triangular and quadrilateral meshes on parameterized surfaces. The method expresses the quality metrics for planar elements in terms of the parametric coordinates of the nodes, and then relocates the nodes on the parametric space to improve the mesh quality and eliminate the inverted elements in the physical space.

Optimization-based algorithms usually untangle and smooth the input meshes at the expense of higher computational and implementational effort. As a way out of the dilemma between quality improvement effect and computational effort, algorithms that do not need to solve numerical optimization problems have been proposed. Kim et al. [27] proposed a novel approach for simultaneous mesh untangling and smoothing using a Pointer network. This method predicts the approximate solutions for free nodes using a pre-trained network, and there is no need to solve complex numerical optimization problems. A geometry-based algorithm for smoothing 2D/3D meshes is proposed by Vartziotis et al. [28,29], which can produce high quality meshes comparable to those obtained by the optimization-based methods within shorter runtimes. Although their method is designed for mesh smoothing, the method still shows its potential for mesh untangling.

The purpose of this paper is to develop a geometry-based mesh untangling and smoothing approach for 2D meshes. A novel explicit element geometric transformation (EEGT) operation is firstly introduced and analyzed, which can transform an arbitrary initial polygonal element (even inverted) into its regular counterpart if applied iteratively. This is the main driving force behind the proposed approach. In the proposed mesh untangling and smoothing approach, all mesh elements are transformed individually and simultaneously in the first step, then new interior node positions are obtained as weighted means of the scaled transformed element nodes. The proper choice of averaging weights in element stitching ensures that the elements can be transitioned smoothly and uniformly throughout the calculation domain, and the explicit element geometric transformation has strong capability to eliminate inverted elements and improve the mesh quality. To the best of our knowledge, the proposed approach is the first geometry-based framework specially designed for simultaneous mesh untangling and smoothing. Besides, it is also very suitable for parallel implementation, especially on GPU.

In the rest of this paper, a new explicit element geometric transformation operation for arbitrary polygonal elements is introduced in Section 2. Basic properties of the transformation are analyzed and a proof for the convergence of the proposed transformation operation is also given. The proof is based on the use of circulant matrices and the decomposition of a polygon into its eigenpolygons. Then the mesh untangling and smoothing approach for 2D meshes based on explicit element geometric transformation and element stitching is presented in Section 3. Typical numerical examples are tested and comparisons with other methods are given in Section 4. Finally, discussions and limitations are presented in Section 5.

2. Explicit Element Geometric Transformation

In 2D numerical simulations, computational meshes usually consists of triangular and quadrilateral elements. The proposed mesh untangling and smoothing approach is mainly based on a new regularizing element geometric transformation, which can transform an arbitrary initial polygonal element (even inverted) into its regular counterpart if applied iteratively. The transformation of neighbor elements is stitched together to improve the overall mesh quality. In this section, such a transformation for arbitrary polygonal elements will be presented and analyzed.

2.1. Basic Geometric Transformation

The element geometric transformation for smoothing triangular and quadrilateral meshes has been introduced by Sun et al. in Reference [30]. It is based on transforming elements by a specially designed two-step stretching and shrinking operation (SSO). The transformation, if applied iteratively to a triangular or quadrilateral element, will improve its regularity and quality. The purpose of stretching operation is to regularize the element, while the shrinking operation is performed to preserve the element size. The stretching operation proposed by Sun et al. [30] is shown in Figure 1 for a triangular and a quadrilateral element.

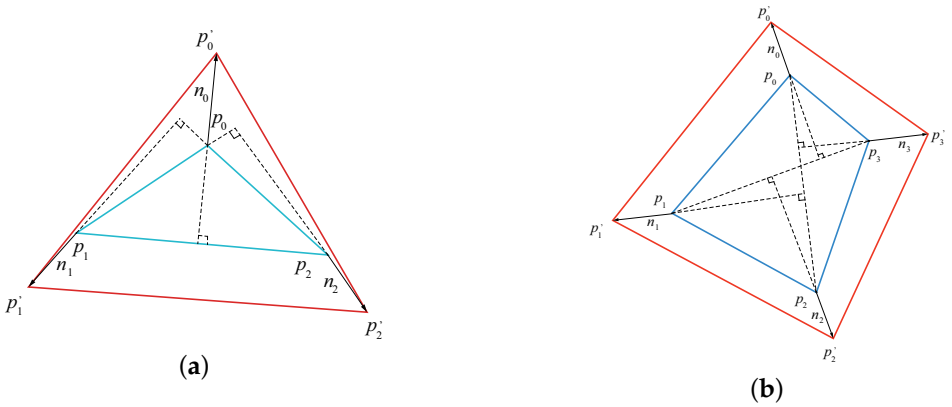


Figure 1. (**a**) The stretching operation of triangular elements in Reference [30]. (**b**) The stretching operation of quadrilateral elements in Reference [30].

The element geometric transformation operation proposed by Sun et al. [30] is mainly used for mesh smoothing, and the regularizing effect for arbitrary triangular and quadrilateral elements remains to be proved in their paper. In this subsection, the element geometric transformation operation will be generalized for polygonal elements with an arbitrary number of nodes, which may even be inverted.

For a given polygonal element with counterclockwise oriented nodes $p_k \in R^2, k \in \{0, 1, ..., n-1\}$, the stretching operation was chosen as a natural extension of that for triangular and quadrilateral elements in Reference [30], that is, just moving node p_k along the outward directed normal of the edge $p_{(k+1)\bmod n}p_{(k+n-1)\bmod n}$. However, numerical tests show that such a stretching operation derived from triangular and quadrilateral elements is not applicable to polygonal elements with more than five nodes. This is illustrated in Figure 2 for an inverted hexagonal element, which is still inverted after applying the transformation in Reference [30] for one hundred times.

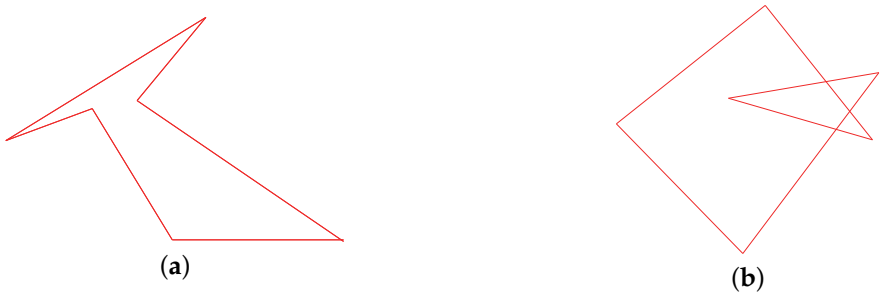


Figure 2. (**a**) Initial inverted hexagonal element. (**b**) Hexagonal element after applying the transformation in Reference [30] for one hundred times.

Inverted elements usually cannot satisfy actual computational requirements, so the simple stretching operation mentioned above needs to be modified and redefined in order to regularize inverted polygonal elements. In the simple stretching operation, new position of the node p_k depends only on its two adjacent nodes $p_{(k+1)\bmod n}$ and $p_{(k+n-1)\bmod n}$, so the simple stretching operation is a kind of local transformation.

The modified stretching operation is conducted by moving node p_k along the direction obtained by combining the outward directed normal of edges $p_{(k+r)\bmod n}p_{(k+n-r)\bmod n}, r \in \{1, 2, ..., ceil(\frac{n-1}{4.0})\}$, where the function $ceil(x)$ returns the smallest integer no less than x. The outward directed normal denoted by n_{kr} of the edge $p_{(k+r)\bmod n}p_{(k+n-r)\bmod n}$ is calculated in the form of components by the formula:

$$\begin{cases} n_{kr}[0] = p_{(k+r)\bmod n}[1] - p_{(k+n-r)\bmod n}[1] \\ n_{kr}[1] = p_{(k+n-r)\bmod n}[0] - p_{(k+r)\bmod n}[0] \end{cases}. \quad (1)$$

It is obvious that the vector n_{kr} is not a unit vector, and its magnitude is the length of the edge $p_{(k+r)\mathrm{mod}n}p_{(k+n-r)\mathrm{mod}n}$. Then the nodes p'_k of transformed element can be obtained by moving the node p_k along the direction defined as:

$$n_k = \sum_{r=1}^{ceil(\frac{n-1}{4.0})} n_{kr}. \tag{2}$$

It can be seen that the position of each node in the modified stretching operation is determined by at least half of the other nodes in the element, which guarantees that the global effect of the modified stretching operation. With a user-specified scaling factor λ, the new node positions of the transformed element can be calculated by:

$$p_k^{'} = p_k + \lambda n_k. \tag{3}$$

The above formula shows that the moving distance of node p_k is proportional to the magnitude of its stretching vector n_k. Therefore, the proposed stretching operation is adaptive to element size.

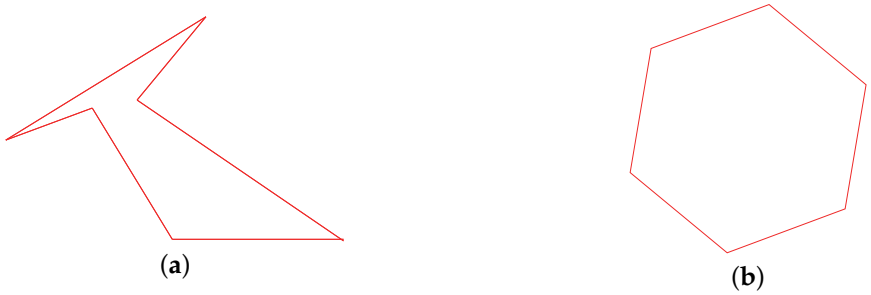
The size of a transformed element after the stretching operation is usually larger than that of the initial element. This can be avoided by scaling the element after stretching operation with respect to the original centroid. Such an operation ensures that the transformed element with more regular shape is close to the initial element and has a small change in element size.

In general, there are two schemes for element scaling, the mean edge length or mean area preserving scaling. The mean edge length preserving scaling scheme is used in this paper for the sake of simplicity. With this scheme, the scaled transformed element with new nodes p_k^s is given by:

$$p_k^s = c + k(p_k{}' - c'), \tag{4}$$

where c and c' represent the initial and transformed element centroid, respectively. The scaling factor k is defined as the mean edge length of the initial element divided by that of the enlarged transformed element.

The stretching and shrinking operation together make up a complete element geometric transformation process. The final node positions of the scaled transformed element can be obtained easily and explicitly with Equations (1)–(4). The geometric transformation process of the same inverted hexagonal element as in Figure 2 using the modified transformation operation is shown in Figure 3. It can be seen that the element becomes a nearly regular hexagonal element after applying the modified transformation for one hundred times, which intuitively illustrates the regularizing capacity of the modified transformation operation. In fact, the modified element geometric transformation operation can successfully transform an arbitrary polygonal element into its regular counterpart after enough iterations, which will be proved in the following subsection.



(**a**) (**b**)

Figure 3. (**a**) Initial inverted hexagonal element. (**b**) Hexagonal element after applying the modified transformation for one hundred times.

2.2. Analysis of the Transformation

In this subsection, the coordinate of node p_k is represented by $p_k = x + iy$. The reason for doing this is to take advantage of the properties of circulant determinants. Using a $n \times n$ matrix A with zero-based indexes $i, j \in \{0, 1, ..., n-1\}$ and a $n \times 1$ vector $p^l = (p_0^l, p_1^l, ..., p_{n-1}^l) \in C^n$ representing the polygonal element with n nodes after l cycles of transformation, the modified stretching operation governed by Equation (3) can be written in the form of $p^{l+1} = Ap^l$, where A is a circulant Hermitian matrix given by:

$$A = \begin{pmatrix} a_0 & a_{n-1} & \cdots & a_2 & a_1 \\ a_1 & a_0 & a_{n-1} & \cdots & a_2 \\ \vdots & a_1 & a_0 & \ddots & \vdots \\ a_{n-2} & \cdots & \ddots & \ddots & a_{n-1} \\ a_{n-1} & a_{n-2} & \cdots & a_1 & a_0 \end{pmatrix} \tag{5}$$

and the component can be written as:

$$a_m = \begin{cases} 1 & if\ m = 0 \\ -\lambda i & if\ n - m = 1, 2, ..., ceil(\frac{n-1}{4.0}) \\ \lambda i & if\ m = 1, 2, ..., ceil(\frac{n-1}{4.0}) \\ 0 & otherwise. \end{cases} \tag{6}$$

In the case of triangular elements this yields:

$$A = \begin{pmatrix} 1 & -\lambda i & \lambda i \\ \lambda i & 1 & -\lambda i \\ -\lambda i & \lambda i & 1 \end{pmatrix} \tag{7}$$

and

$$A = \begin{pmatrix} 1 & -\lambda i & 0 & \lambda i \\ \lambda i & 1 & -\lambda i & 0 \\ 0 & \lambda i & 1 & -\lambda i \\ -\lambda i & 0 & \lambda i & 1 \end{pmatrix} \tag{8}$$

in the case of quadrilateral elements, respectively. With this, the basic properties of the transformation presented in the previous subsection can be analyzed for arbitrary polygonal elements.

Theorem 1. *The transformation preserves the centroid of a polygonal element, that is* $\frac{1}{n}\sum_{k=0}^{n-1} p_k^{l+1} = \frac{1}{n}\sum_{k=0}^{n-1} p_k^l$.

Proof of Theorem 1.

$$\begin{aligned} \tfrac{1}{n}\sum_{k=0}^{n-1} p_k^{l+1} &= \tfrac{1}{n}\sum_{k=0}^{n-1}\sum_{j=0}^{n-1} A_{kj} p_j^l = \tfrac{1}{n}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} A_{kj} p_j^l \\ &= \tfrac{1}{n}\sum_{j=0}^{n-1}(1 + \sum_{m=1}^{ceil(\frac{n-1}{4.0})}(a_m + a_{n-m}))p_j^l \\ &= \tfrac{1}{n}\sum_{j=0}^{n-1}(1 + \sum_{m=1}^{ceil(\frac{n-1}{4.0})}(\lambda i - \lambda i))p_j^l \\ &= \tfrac{1}{n}\sum_{j=0}^{n-1} p_j^l = \tfrac{1}{n}\sum_{k=0}^{n-1} p_k^l. \quad \square \end{aligned}$$

Since A is a circulant Hermitian matrix, the associated eigenvalues are real-valued. Due to the circulant structure, the eigenvalues $\mu_k, k \in \{0, 1, ..., n-1\}$ of A can be easily obtained by multiplying the discrete Fourier matrix R defined by $R_{u,v} = \omega_{uv} = exp(\frac{2\pi iuv}{n})$ with the transposed first row of

A [31], where i is the imaginary unit. Hence, for arbitrary $n \geq 3$ the corresponding eigenvalues of A are given by:

$$\mu_j = a_0 + a_{n-1}\omega_j + a_{n-2}\omega_j^2 + \cdots + a_1\omega_j^{n-1}, \quad j = 0, 1, ..., n-1. \tag{9}$$

Using the fact that $\omega_j^{n-m} = exp(\frac{2\pi(n-m)ji}{n}) = exp(\frac{-2\pi mji}{n}) = \bar{\omega_j^m}$ and $a_{n-m} = \bar{a_m}$, the eigenvalues can be written as:

$$\mu_j = 1 + \sum_{m=1}^{ceil(\frac{n-1}{4.0})} a_m\omega_j^{n-m} + a_{n-m}\omega_j^m = 1 + 2\lambda \sum_{m=1}^{ceil(\frac{n-1}{4.0})} \sin\frac{2\pi jm}{n}, \quad j = 0, 1, ..., n-1. \tag{10}$$

In particular, the above formula shows that $\mu_0 = 1$ does not depend on the choice of scaling factor λ. Moreover, it holds that $|\mu_1| > |\mu_j|$ is true for each value of j in the set $j \in \{2, 3, ..., n-1\}$.

Theorem 2. *If the transformation parameter $\lambda > 0$, the eigenvalue μ_1 has the largest absolute value among all the eigenvalues of matrix A.*

Proof of Theorem 2. Since $\lambda > 0$, if $\sum_{m=1}^{ceil(\frac{n-1}{4.0})} \sin\frac{2\pi m}{n} = |\sum_{m=1}^{ceil(\frac{n-1}{4.0})} \sin\frac{2\pi m}{n}| \geq |\sum_{m=1}^{ceil(\frac{n-1}{4.0})} \sin\frac{2\pi jm}{n}|$ is true for each value of j in the set $j \in \{2, 3, ..., n-1\}$, we can easily get $|\mu_1| \geq |\mu_j|$.

Notice that $f(n, n-j) = \sum_{m=1}^{ceil(\frac{n-1}{4.0})} \sin\frac{2\pi(n-j)m}{n} = -\sum_{m=1}^{ceil(\frac{n-1}{4.0})} \sin\frac{2\pi jm}{n} = -f(n, j)$, then the only condition needed to be satisfied is that $|\sum_{m=1}^{ceil(\frac{n-1}{4.0})} \sin\frac{2\pi m}{n}| \geq |\sum_{m=1}^{ceil(\frac{n-1}{4.0})} \sin\frac{2\pi jm}{n}|$ is true for each value of j in the set $j \in \{2, 3, ..., ceil(\frac{n-1}{2.0})\}$, which is illustrated in Figure 4 by plotting the absolute value of function $f(n, j)$ for different pair (n, j). Thus, it holds that $|\mu_1| \geq |\mu_j|$ is true for each value of j in the set $j \in \{2, 3, ..., n-1\}$. □

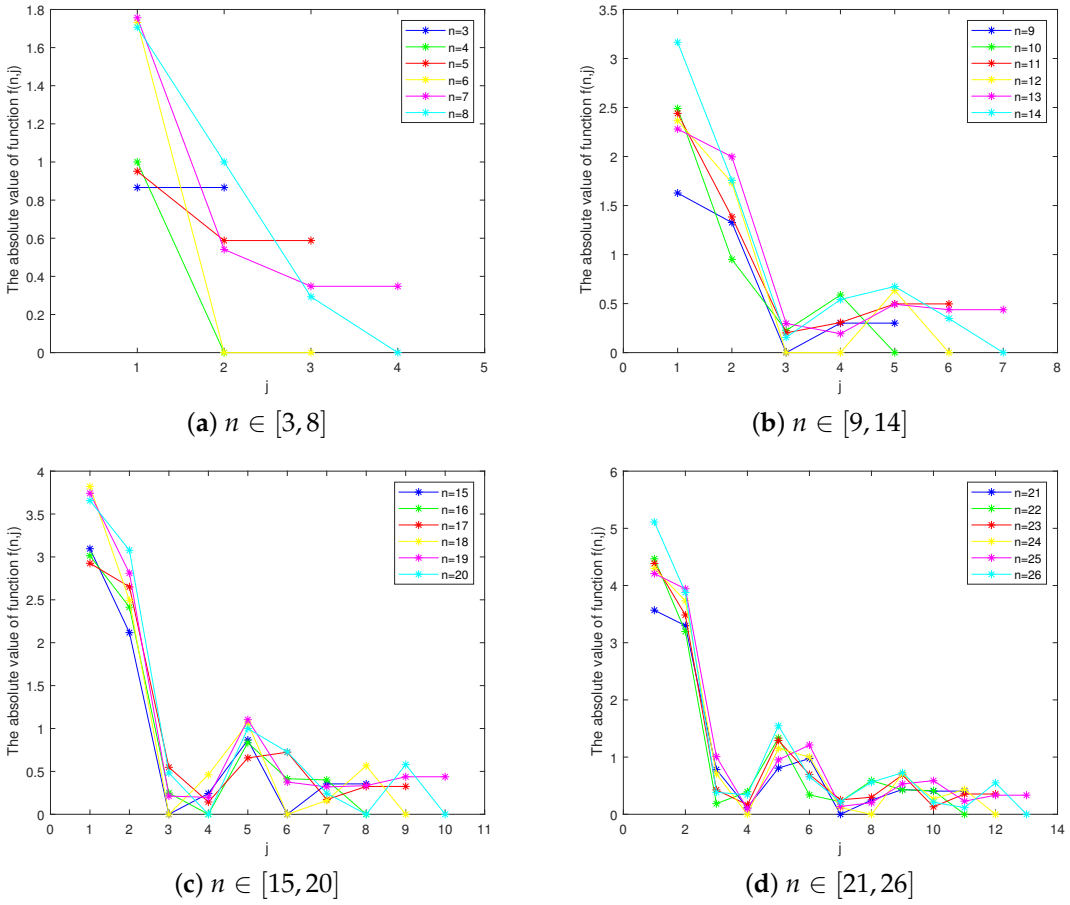
Inspired by the proof in Reference [28], the diagonalization of A will be used to explore the essence of the proposed transformation. Thus, the matrix composed of the associated eigenvectors in the columns, given by $T = 1/\sqrt{n}R$, $R = (\nu_0, \nu_1, ..., \nu_{n-1})$ is used, where the eigenvector ν_j of matrix A is given by $\nu_j = (1, \omega_j, \omega_j^2, ..., \omega_j^{n-1})^T$ [31]. It holds that $A = TDT^*$, where D is the diagonal matrix composed of the eigenvalues of matrix A and T^* is the conjugate transpose of matrix T. With the decomposition of the $n \times n$ identity matrix I in the form of $I = \sum_{k=0}^{n-1} I_k$, where I_k denotes the $n \times n$ matrix with the only nonzero entry $(I_k)_{k,k}$=1, the transformation matrix after l cycles of the modified stretching operation can be written as:

$$A^l = (TDT^*)^l = TD^lT^* = \sum_{k=0}^{n-1} \mu_k^l TI_kT^* = \sum_{k=0}^{n-1} \frac{\mu_k^l}{n} RI_kR^*. \tag{11}$$

Therefore, the corresponding polygonal element after l cycles of the modified stretching operation can be decomposed in the following manner:

$$p^l = A^lp^0 = \sum_{k=0}^{n-1} \mu_k^l \underbrace{\frac{1}{n} RI_kR^*p^0}_{e_k} = \sum_{k=0}^{n-1} \mu_k^l e_k. \tag{12}$$

The so-called eigenpolygons e_k do depend only on the initial polygon element p^0. Examples of such decompositions are depicted in Figure 5 for a triangular, quadrilateral, pentagonal and hexagonal element, where the element nodes are marked with different colors in order to denote the element orientation.



(**a**) $n \in [3, 8]$ (**b**) $n \in [9, 14]$

(**c**) $n \in [15, 20]$ (**d**) $n \in [21, 26]$

Figure 4. The absolute value of function $f(n, j)$ for different pair (n, j).

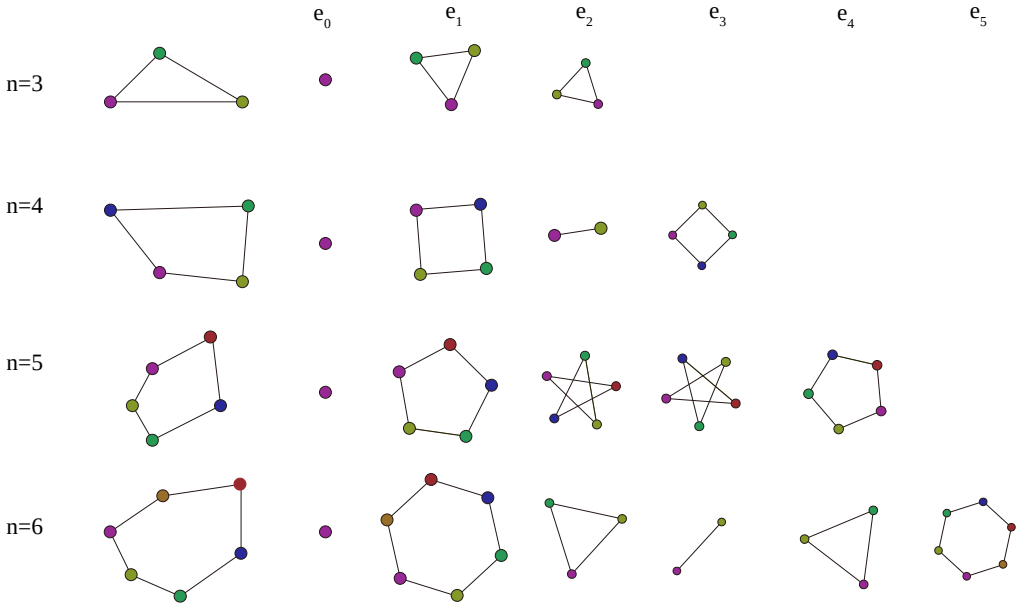


Figure 5. Eigenpolygon decomposition of a triangular, quadrilateral, pentagonal and hexagonal element.

The first two eigenpolygons and their corresponding eigenvalues play an important role in determining the characters of element geometric transformation. Due to $\mu_0 = 1$, the first eigenpolygon is given by:

$$e_0 = \frac{1}{n} R I_0 R^* p^0 = \frac{1}{n} I p^0 = \frac{1}{n} (\sum_{k=0}^{n-1} p_k^0)(1, 1, \cdots, 1)^T, \tag{13}$$

which means all nodes of the polygonal element are placed at the centroid of the initial polygonal element p^0 during the transformation. The second eigenpolygon represents the counterclockwise oriented regular polygonal element, due to the expression:

$$e_1 = \frac{1}{n} R I_1 R^* p^0 = \frac{1}{n} \begin{pmatrix} 1 & \omega_1^1 & \cdots & \omega_1^{n-1} \\ \omega_1^1 & \omega_1^2 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ \omega_1^{n-1} & 1 & \cdots & \omega_1^{n-2} \end{pmatrix} p^0 \tag{14}$$

implies $(e_1)_{(k+1) \bmod n} = \omega_1 (e_1)_k$, $k \in \{0, 1, ..., n-1\}$, which says that node $p_{(k+1) \bmod n}$ of the eigenpolygon e_1 can be obtained by rotating the preceding node p_k $\frac{360}{n}$ degrees around the origin of coordinates. It is not difficult to understand that the remaining eigenpolygons can be generated by the expression $(e_m)_{(k+1) \bmod n} = \omega_m (e_m)_k, k \in \{0, 1, ..., n-1\}$.

The properties above lead to the following crucial result building the foundation for the simultaneous mesh untangling and smoothing approach to be introduced in the next section.

Theorem 3. *The proposed element geometric transformation can regularize arbitrary polygonal elements after enough iterations.*

Proof of Theorem 3. The size of a transformed element can differ significantly from that of the initial element after the stretching operation. This could be easily overcome by scaling the transformed element after each transformation step with respect to the centroid. From a theoretical point of view, using the inverse of μ_1 as scaling factor is the best choice. Of course, the mean edge length preserving scaling scheme is used in practice. The sequence of scaled transformed polygonal elements using the inverse of μ_1 as scaling factor:

$$p_s^l = e_0 + \sum_{k=1}^{n-1} (|\frac{\mu_k}{\mu_1}|)^l e_k = e_0 + e_1 + \sum_{k=2}^{n-1} \underbrace{(|\frac{\mu_k}{\mu_1}|)^l}_{<1} e_k \tag{15}$$

converges to $p_s^\infty = \lim_{l \to \infty} p_s^l = e_0 + e_1$, which represents the counterclockwise oriented regular polygonal element centered at the centroid of the initial polygonal element. □

The speed of convergence for a single polygonal element depends on the ratio $\rho = \max_{2 \le k \le n-1} \{|\mu_k / \mu_1|\} < 1$. The eigenvalues depend only on scaling factor λ, so does the ratio ρ. For triangular elements, it holds that $\mu_1 = 1 + \sqrt{3}\lambda$, $\mu_2 = 1 - \sqrt{3}\lambda$, and it is obvious that ρ achieves a minimum of 0 when λ is equal to $\sqrt{3}/3$. As for quadrilateral elements, it holds that $\mu_1 = 1 + 2\lambda$, $\mu_2 = 1$, $\mu_3 = 1 - 2\lambda$, and it is easy to obtain that ρ achieves a minimum of $1/3$ when λ is equal to 1.

In a word, the optimal scaling factor λ for a single polygonal element can be found through theoretical analysis. Nevertheless, numerical tests are usually needed to find the appropriate scaling factor in practical mesh optimization problems, which may involve elements of different types and sizes.

3. Smoothing and Untangling Approach

The proposed mesh smoothing and untangling approach consists of the following two key steps. First, each element of the input mesh is transformed individually and simultaneously towards its corresponding ideal element by using the transformation introduced in Section 2. In this step, each element is transformed separately without caring about the connection relationship with its neighbors. Therefore, a combination step is needed to stitch the disconnected elements. Here, the element stitching operation is performed by updating the position of each free node by a well-designed weighted means of its corresponding temporary node positions in the scaled transformed elements. The element stitching operation is actually using an explicit formula to approximate the partial derivative of the objective function with respect to the current node position to obtain the displacement of the node.

As the proper choice of a quality metric is important for element stitching operation, the element quality metric used in this paper is first introduced.

3.1. Shape Metrics

There are many different shape metrics to measure the quality of a polygonal element. These shape metrics measure the difference between a given polygonal element and a regular polygonal element from different aspects. The mean ratio (MR) metric [32] is one of the commonly used metric in mesh optimization. For a valid polygonal element, the quality number lies within the interval (0,1]. The quality number is equal to 1 only if the element is a regular polygon, and it is defined to be 0 if the element is inverted or degenerated. Overall mesh quality is assessed with the aid of the minimal and average mesh quality numbers given by:

$$q_{min} = \min q(E_j) \quad and \quad q_{ave} = \frac{1}{N_e} \sum_{j=1}^{N_e} q(E_j), \tag{16}$$

where $q(E_j)$ represents the quality number of element E_j and N_e is the total number of elements.

3.2. Element Stitching Operation

In the local element transformation step, each element is transformed separately without considering the connection relationship with its neighbors. Therefore, global mesh optimization is accomplished by averaging the temporary node positions obtained by the local element transformation. With the element stitching operation, the mesh quality should be improved gradually, and the mesh elements should be transitioned smoothly and uniformly throughout the whole computational domain.

In this subsection, global indexing will be used to describe the mesh smoothing and untangling approach. Here, the nodes in the k^{th} iteration step of the approach are represented by p_i^k with $i \in \{1, 2, ..., N\}$, where N is the total number of mesh nodes. E_j represents the j^{th} element in the mesh, and $E(i)$ represents the index set of all elements containing node p_i.

At the first step, all elements are transformed simultaneously using the Equations (1)–(4). The scaling factor is set to:

$$\lambda = 0.2(1 - q) \tag{17}$$

for all elements after several attempts, where q is the mean ratio quality number of the corresponding element. A large value of scaling factor λ may increase the risk of inducing inverted elements, so the coefficient 0.2 is used to prevent the element from transforming much too rapidly. Here, an element with poorer quality will undergo a greater degree of transformation in order to improve the convergence speed.

The next step is computing the new position p_i^{k+1} of each node. The elements containing node p_i will produce $|E_i|$ new temporary nodes p_{ij}^k after the simultaneous transformation. The position of node p_i^{k+1} is updated according to the following formula:

$$p_i^{k+1} = p_i^k + \frac{\sum_{j \in E(i)} w_j (p_{ij}^k - p_i^k)}{\sum_{j \in E(i)} w_j}, \tag{18}$$

in which $E(i)$ denotes the index set of elements containing node p_i, p_{ij}^k is the temporary node corresponding to node p_i^k in the scaled transformed element E_j and w_j is the weight of element E_j. A proper choice for element weights should put more emphasis on low-quality elements, especially on inverted elements. In addition, elements closer to the boundary should have greater weights, so that the boundary constraints can be used to gradually eliminate inverted elements, and the mesh elements should be transitioned smoothly and uniformly throughout the whole computational domain. In order to satisfy these requirements, the following weight function is introduced:

$$w_j = f(d_j) g(q(E_j)), \tag{19}$$

where f is the distance function formulated by:

$$f(d_j) = (\frac{\bar{l}}{d_j})^c = (\frac{\bar{l}}{\| c_j - p_{bj} \|})^c \tag{20}$$

and g is the quality function written as:

$$g(q(E_j)) = \begin{cases} 5.0 & if \quad q(E_j) = 0 \\ (1 - q(E_j))^\eta & if \quad q(E_j) > 0. \end{cases} \tag{21}$$

The symbol d_j stands for the Euclidean distance between the centroid c_j of element E_j and the associated boundary node p_{bj} of element E_j. Boundary node that has the shortest distance to the centroid c_j is selected as the associated node p_{bj}. $\bar{l}$ is the average of the mean edge length of all elements in the mesh, and c is a power exponent which is usually set to a value of $c = 2.0$ in order to smooth the distance-decay effect. In this way, all mesh elements are roughly sorted by their distance to the boundaries. As the proposed approach is iterative, this makes the mesh elements transition smoothly and uniformly throughout the whole computational domain. To be honest, it is very time-consuming to update the distance function of all elements at each iteration step. In practice, the distance function value of each element can be updated every few iterations.

According to the expression of quality function, low-quality elements especially inverted elements account for larger weights in determining the final node positions. This is designed to accelerate the convergence speed of the approach and to improve the ability of mesh untangling. Generally speaking, increasing the value of exponent η will lead to an improved worst element quality as well as a slightly lower average element quality. Therefore, $\eta = 2.0$ have been used as a default value to obtain a balanced optimization result.

Note that the default values for $\eta = 2.0$ and $c = 2.0$ usually work well. However, in some cases, they still need to be carefully selected in order to produce a better result.

3.3. Untangling and Smoothing Pipeline

A detailed description of the proposed simultaneous untangling and smoothing approach is given in this subsection. Algorithm 1 provides the pseudo-code of this EEGT-based simultaneous untangling and smoothing approach for isotropic 2D meshes.

The element transformation and node positions updating steps are performed iteratively until the current mesh does not contain inverted elements. In step 3 new node positions are not immediately

updated, since this would influence the computation of subsequent nodes. Therefore, new node positions computation and applying the new coordinates are separated, ensuring that the computation does not depend on the numbering scheme of nodes and elements, so as to obtain reproducible results. This is important for parallelized implementation of the proposed approach.

After the above mesh untangling process (step 2–4), the mesh contains no inverted elements. However, quality of the mesh obtained by the untangling process may be low, and further optimization is needed. As the approach is geometry-driven, the steps element transformation and node positions updating may lead to the generation of inverted elements. Therefore, it has to be checked for every element with its new nodes during the mesh smoothing process (step 5). All mesh elements, which are inverted, get a note, that their nodes have to be reset. After all elements are checked, all nodes p_i^{k+1}, belonging to one or more inverted elements, are reset to their old coordinates, that is, $p_i^{k+1} = p_i^{old}$. These two steps are repeated iteratively until no invalid elements are left. The described method of handling inverted elements during the mesh smoothing process is simple and parallelizable.

Algorithm 1 EEGT-based simultaneous untangling and smoothing of 2D meshes.

1. Pretreatment
   (a) For each node p_i, save the index set $E(i)$ of all mesh elements adjacent to it, and record the boundary nodes;
   (b) Initialize the Boolean variable $Valid = 0$, the number of iterations $Count = 0$, and determine the index set $J_{inverted}$ of all inverted elements. If $J_{inverted} = NULL$, let $Valid = 1$, skip the untangling process (step 2-4) and go to step 5 for mesh smoothing.
2. Simultaneous element transformation
   Perform the following operations for each element E_j:
   (a) Compute the transformed element according to Equation (3) using the scaling factor $\lambda = 0.2(1 - q)$, where q is the quality number of element E_j;
   (b) Scale the transformed element according to Equation (4) resulting in the new temporary nodes p_{ij}^k.
3. Update node positions
   (a) For each inner node p_i^k of the mesh, compute its new position

$$p_i^{k+1} = p_i^k + \frac{\sum_{j \in E(i)} w_j (p_{ij}^k - p_i^k)}{\sum_{j \in E(i)} w_j}, \tag{22}$$

   where w_j is the weight of element E_j;
   (b) Store old node positions $p_i^{old} = p_i^k$ and update all inner nodes.
4. Determine the validity of the current mesh
   Let $Count = Count + 1$, and update the index set $J_{inverted}$:
   (a) If $J_{inverted} = NULL$, let $Valid = 1$, and go to step 5 to further improve mesh quality;
   (b) If $J_{inverted} \neq NULL$ and $Count$ is less than the maximum number of iterations in the untangling process, go to step 2 for the next iteration;
   (c) Otherwise, the proposed algorithm fails to untangle the mesh.
5. Mesh smoothing:
   Keep the boundary nodes fixed, and carry out the simultaneous element transformation, node positions updating and revert nodes of inverted elements to their previous position, until the average mesh quality improvement is below a given tolerance or maximal number of iterations in the smoothing process is reached.

The two main steps of the proposed approach, including element geometric transformation and node positions updating can be easily implemented on GPU due to their fine-grained parallelism. Therefore, a GPU-based implementation is also completed by the authors.

4. Examples

In this section, three tangled meshes will be optimized to evaluate the performance of the proposed approach, and resulting meshes are compared with those obtained by the default untangle wrapper provided in the mesh quality improvement toolkit Mesquite [22]. Mesquite provides three default untanglers based on different metrics, namely Untangle-Beta untangler, Size untangler and Shape-Size untangler. As the three default untanglers in Mesquite are mainly used to eliminate inverted elements, the quality of the mesh after untangling is generally poor, so the Shape Improve Wrapper in Mesquite is used to further improve mesh quality. At last, examples for testing the performance of GPU implementation are given. In this section, the mean ratio (MR) quality metric is also used to compare the quality of meshes obtained by different methods, and the quality number of inverted elements is set to 0.

For the proposed approach, mesh smoothing process (step 5) is terminated if the average mesh quality improvement of two successive meshes is less than 0.001. The described untangling and smoothing approach is implemented with the C++ language and all the tests are executed in an Intel Xeon E5-2637 CPU at 3.50 GHz with 32 GB RAM.

4.1. Quadrilateral Mesh Example

We first consider a square quadrilateral mesh with a hole. The initial mesh is shown in Figure 6a. The mesh has 168 nodes and 140 quadrilateral elements, including 23 inverted elements. There are 56 boundary nodes in the mesh, and they are fixed during the optimization process. The three default untanglers in Mesquite and the proposed approach (EEGT) in this paper are used to optimize the initial tangled mesh. Table 1 shows mesh quality statistics and the number of inverted elements of the initial mesh and output meshes, and Figure 6 shows output meshes obtained by different methods.

Table 1. Mesh quality statistics and the number of inverted elements for quadrilateral mesh example. The mean ratio quality metric is used to measure the mesh quality.

Methods	Time(s)	q_{ave}	q_{min}	Number of Inverted Elements
Initial mesh	–	0.6347	0	23
Untangle-Beta	0.09	0.6003	0.0207	0
Size	0.41	0.5073	0	4
Shape-Size	0.239	0.5634	0.0953	0
Untangle-Beta + Shape Improve Wrapper	0.525	0.6312	0.2297	0
EEGT	0.006	0.5844	0.2502	0

The Untangle-Beta untangler successfully eliminates all inverted elements in the initial mesh. The average quality of the mesh is also high enough, but the worst quality of mesh elements is only 0.0207. There are many low-quality elements near the left inner boundary, which can be seen in Figure 6b. Such a mesh can not be directly used in numerical simulations, so the Shape Improve Wrapper in Mesquite is used to optimize the output mesh. The resulting mesh is shown in Figure 6f. The average quality of the mesh is improved to 0.6312, and the quality of the worst element is improved to 0.2297. It can be seen from Figure 6f that after further optimization, the distribution of mesh elements is more uniform in the calculation domain. In this example, the Size untangler does not eliminate all inverted elements, and the resulting mesh still contains 4 inverted elements. The Size untangler tries to make the size of the mesh elements uniform while untangling the mesh, which can be seen in Figure 6c. Different form the Size untangler, the Shape-Size untangler considers both the element size and shape, so it can perform mesh untangling and improve the mesh quality at the same time. Figure 6d shows that the optimized mesh of the Shape-Size untangler is better than those obtained by the other two untanglers. Nonetheless, the optimized mesh still has some low-quality elements near the left boundary. Figure 6e shows the optimized mesh obtained by using the proposed approach. Although the average mesh quality is lower than that obtained by the optimization-based method,

the quality of the worst element is higher. In addition, the distribution of elements in the calculation domain is sufficiently uniform.

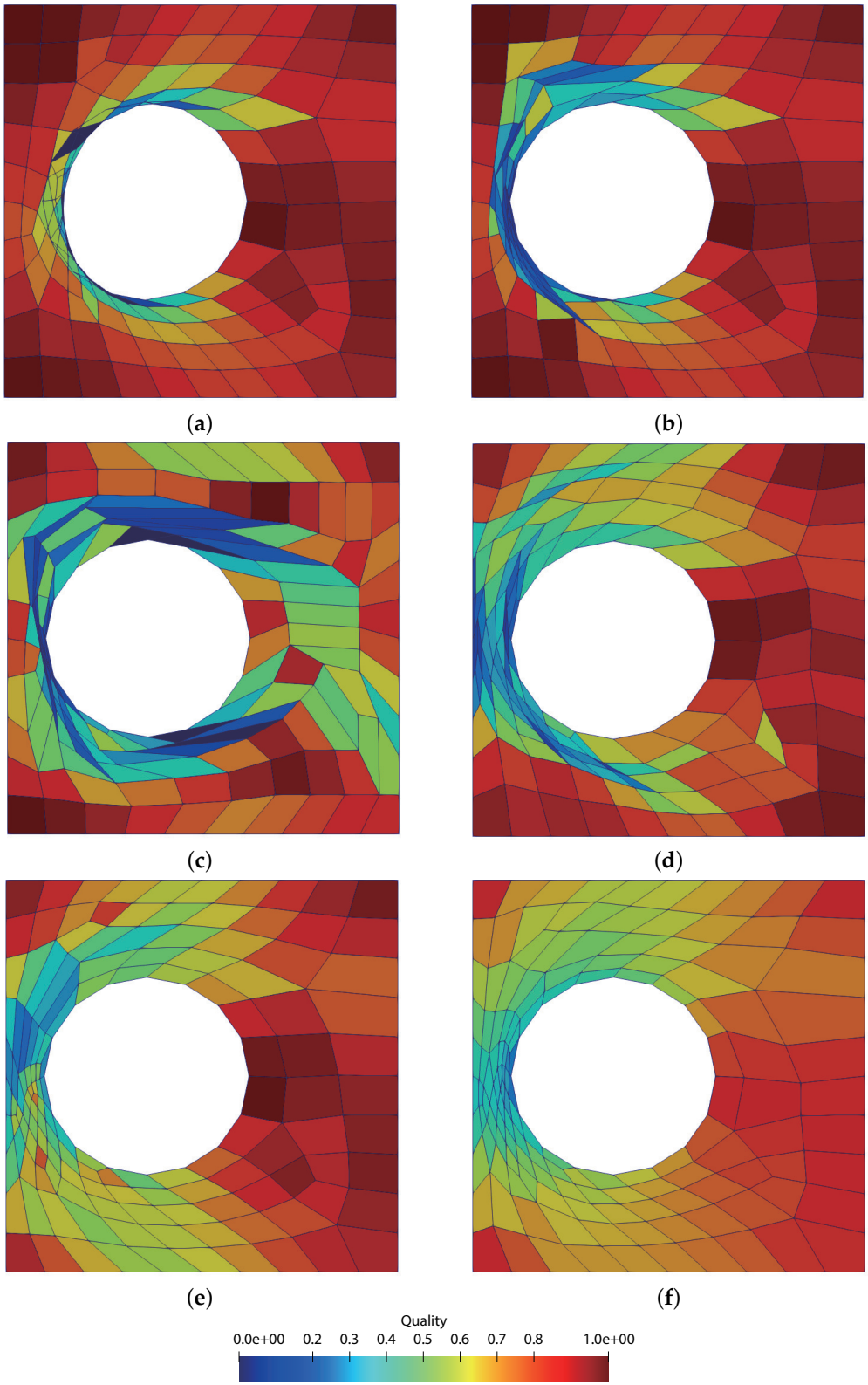


Figure 6. (**a**) Initial quadrilateral mesh with 23 inverted elements. (**b**) Optimized mesh using the Untangle-Beta untangler. (**c**) Optimized mesh using the Size untangler. (**d**) Optimized mesh using the Shape-Size untangler. (**e**) Optimized mesh using the proposed approach. (**f**) Optimized mesh using the Untangle-Beta untangler and the Shape Improve Wrapper.

The proposed approach has obvious advantages over the optimization-based methods in terms of running time. The Size untangler takes 0.09 s to eliminate inverted elements, while the Shape-Size untangler takes 0.239 s. The Shape Improve Wrapper takes extra 0.435 s to smooth the untangled mesh obtained by the Untangle-Beta untangler, so the total running time is 0.525 s for Mesquite to optimize the initial mesh. The proposed approach only takes 0.006 s to untangle and smooth the initial mesh, which shows the high efficiency of this method.

4.2. Mesh Deformation Example

In computational fluid dynamics simulation, the computational mesh should conform to the changing flow domain at each time step during the simulation process. However, mesh deformation methods may lead to the generation of inverted elements for large mesh deformations. Mesh smoothing and untangling algorithms are of great importance for improving the deformation capacity and deformation effect of mesh deformation methods under large deformations.

In this example, an unstructured NACA0012 airfoil mesh with 5892 nodes and 11,363 triangular elements is tested. The size of the square domain is 20×20, while the chord length of airfoil is 1. The airfoil moves 6 units to the both down and right while it rotates 75 degrees counterclockwise around its aerodynamic center, which is a very typical case for mesh deformation. The deformed mesh obtained using the mesh deformation method proposed in Reference [33] is shown in Figure 7a, and the deformation is accomplished in a single step. Due to the large deformation, the deformed mesh has 459 inverted elements. The three untanglers in Mesquite and the proposed approach are used to optimize the deformed mesh. The optimization results are given in Table 2, and the optimized meshes are shown in Figure 7.

None of the three untanglers in Mesquite can untangle the deformed mesh, and the number of inverted elements in the resulting meshes is even larger than that of the initial deformed mesh. Among them, the Size untangler has the worst optimization effect, and some nodes in the output mesh even locate outside the initial boundary.

The proposed approach successfully untangle the deformed mesh. The average and minimal quality of the optimized mesh are 0.8102 and 0.4157, respectively. Unlike the local optimization-based method, the proposed approach can make full use of the boundary constraints to optimize the positions of interior nodes, and element geometric transformation and element stitching operations can improve the mesh quality iteratively during the optimization process.

Of course, there is still room for further optimization of the mesh obtained by our method. Therefore, the Shape Improve Wrapper in Mesquite is used to optimize the mesh obtained by the proposed approach, and the average quality of the mesh is improved from 0.8102 to 0.9006, while the quality of the worst element is reduced to 0.2963. Optimization-based methods usually produce meshes with higher average quality, but they are also more time-consuming. In this example, the Shape Improve Wrapper takes 108.089 s to further optimize the mesh obtained by the proposed approach, while the proposed approach only takes 29.849 s to untangle and smooth the initial deformed mesh.

Table 2. Mesh quality statistics and the number of inverted elements for mesh deformation example. The mean ratio quality metric is used to measure the mesh quality.

Methods	Time(s)	q_{ave}	q_{min}	Number of Inverted Elements
Initial mesh	–	0.3243	0	459
Untangle-Beta	5.402	0.1766	0	5533
Size	900.233	0.1355	0	1637
Shape-Size	54.494	0.2829	0	3833
EEGT	29.849	0.8102	0.4157	0
EEGT + Shape Improve Wrapper	108.089	0.9006	0.2963	0

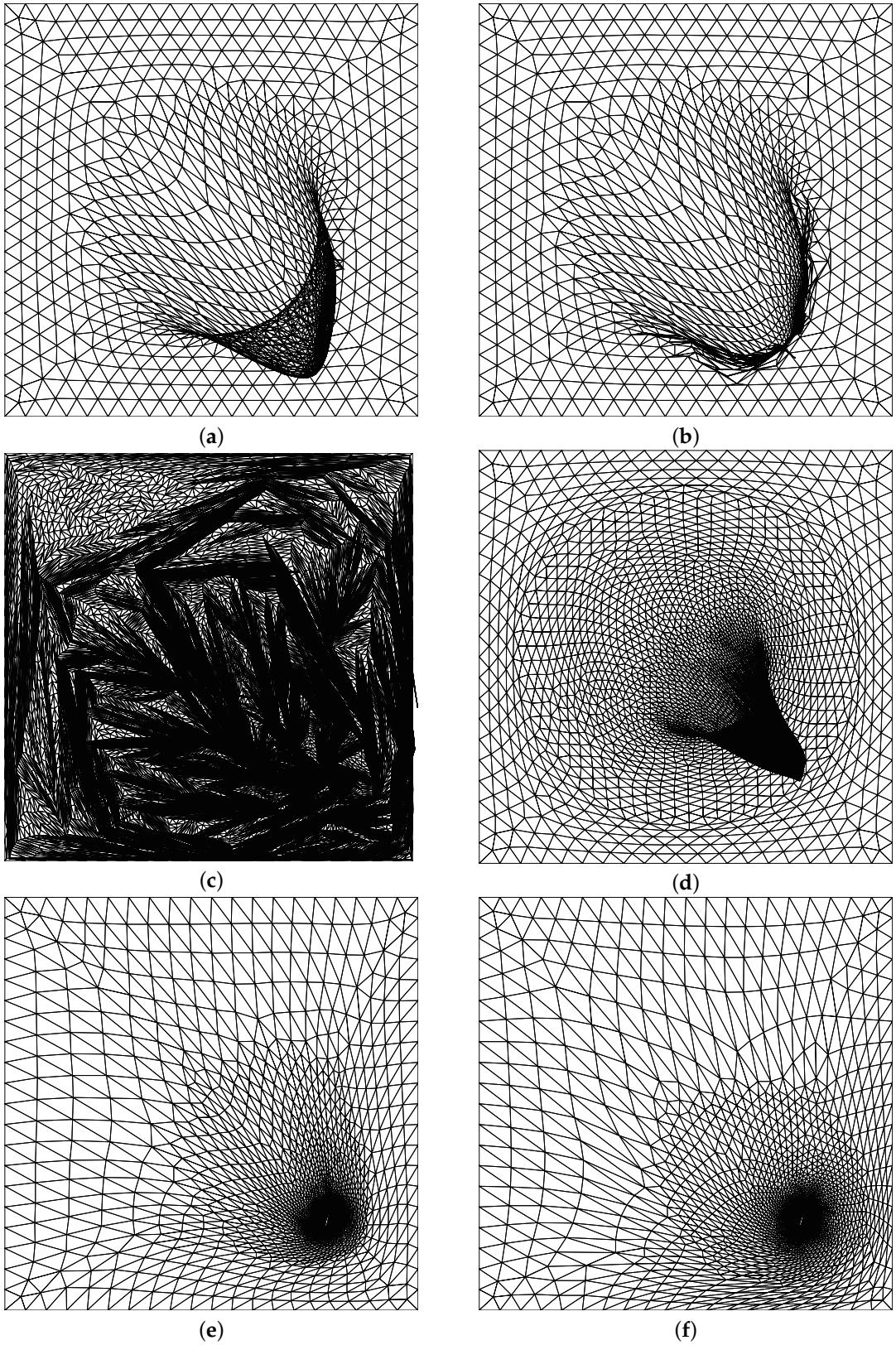


Figure 7. (**a**) Initial deformed mesh with 459 inverted elements. (**b**) Optimized mesh using the Untangle-Beta untangler. (**c**) Optimized mesh using the Size untangler. (**d**) Optimized mesh using the Shape-Size untangler. (**e**) Optimized mesh using the proposed approach. (**f**) Optimized mesh using the proposed approach and the Shape Improve Wrapper.

4.3. Triangular Mesh Example

The final example is a triangular mesh with two inner boundaries. The initial mesh is shown in Figure 8a. The mesh is generated by randomly moving the positions of interior nodes, and it has 5383 nodes and 10083 triangular elements, 4914 of which are inverted. The Size and Shape-Size untanglers in Mesquite and the proposed approach (EEGT) are used to optimize the initial tangled mesh. The Untangle-Beta untangler fails to output the optimized mesh within 1800s, so its optimization result is not given in this example.

The output meshes are shown in Figure 8. Both the Size and Shape-Size untanglers fail to untangle the initial mesh, and the optimized meshes of these two untanglers contain 1825 and 2636 inverted elements, respectively. All of the inverted elements are successfully removed by the proposed approach. Table 3 shows mesh quality statistics and the number of inverted elements of the initial mesh and output meshes of different methods. It can be seen that the proposed approach is effective in eliminating inverted elements and producing meshes with high quality, even for highly tangled meshes.

Table 3. Mesh quality statistics and the number of inverted elements for triangular mesh example. The mean ratio quality metric is used to measure the mesh quality.

Methods	Time(s)	q_{ave}	q_{min}	Number of Inverted Elements
Initial mesh	–	0.2566	0	4914
Size	4.927	0.2043	0	1825
Shape-Size	32.472	0.1729	0	2636
EEGT	37.488	0.9130	0.6461	0

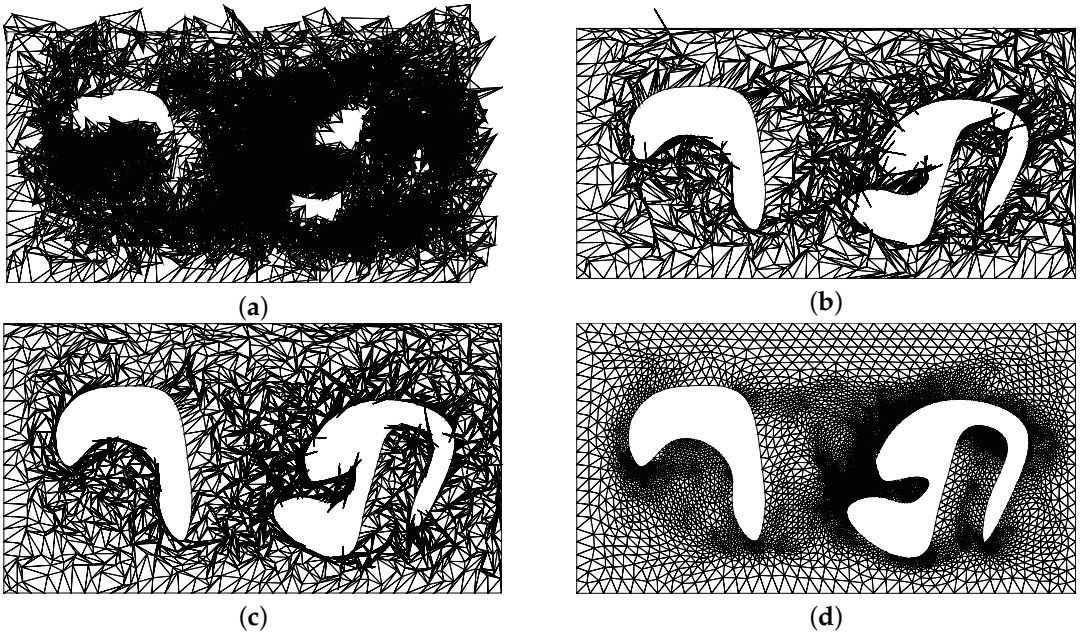


Figure 8. (**a**) Initial triangular mesh with 4914 inverted elements. (**b**) Optimized mesh using the Size untangler. (**c**) Optimized mesh using the Shape-Size untangler. (**d**) Optimized mesh using the proposed approach.

4.4. Implementation on Gpu

In recent years, the programmable Graphic Processor Unit (GPU) has evolved into a highly parallel, multi-threaded processor with tremendous computational horsepower and very high memory bandwidth. More specifically, the GPU is especially well-suited to address problems that can be expressed as data-parallel computations, that is, the same program is executed on many data elements in parallel. Due to fine-grained parallelism, the proposed approach is accelerated by GPU

to demonstrate its potential. The GPU used in this test is NVIDIA Quadro K2200, and its detailed parameters are listed in Table 4. Again, mesh smoothing process is terminated if the average quality numbers of two consecutive meshes deviate less than 0.001. Optimization results on CPU and GPU are given in Table 5. More than 10 times speedup is achieved by GPU implementation in the test platform, which shows the great potential of the proposed approach in solving large-scale mesh optimization problems. With the increase of the number of mesh elements.

Table 4. The detailed parameters of Quadro K2200.

Compute Capability	5.0
Number of stream multiprocessor	5
Cores per stream multiprocessor	128
Warp Size	32
Clock Rate	1.12 GHz
Global Memory	4 GB
Peak FP32 Performance	1.3Tflops

Table 5. Comparison of optimization results on CPU and GPU.

Meshes	Platform	Number of Elements	Number of Inverted Elements	Time(s)
Quadrilateral mesh example	CPU GPU	140	23	0.006 4.184×10^{-4}
Mesh deformation example	CPU GPU	11,363	459	29.849 0.817
Triangular mesh example	CPU GPU	10,085	4914	37.488 1.199

5. Conclusions and Future Work

A novel and efficient geometry-based approach for simultaneously untangling and smoothing isotropic 2D meshes has been proposed for the first time in this paper. It has the following features:

(1) The proposed approach is based on a new explicit element geometric transformation operation, which can transform an arbitrary polygonal element into its regular counterpart if applied iteratively. It is more reliable and efficient, and its convergence is proved by theoretical analysis.

(2) The proposed approach involves two main steps, namely independent element geometric transformation and node positions updating. It is very suitable for parallel implementation, especially on GPU.

(3) The proposed approach has a strong ability to eliminate inverted elements, and the optimized mesh elements can be transitioned smoothly and uniformly throughout the whole computational domain.

Limitations. The proposed approach can fail on meshes with extreme difference in element size. It may also fail when the initial positions of interior nodes are fairly random. Both situations may lead to the failure of eliminating all inverted elements, because the node positions updating may lead to worse node coordinates. As the element geometric transformation aims at transforming a polygonal element into its regular counterpart, the proposed approach based on the regularizing element transformation cannot be used to optimize anisotropic meshes.

Future Work. Further study will be focused on the extension of the proposed approach to volume meshes and anisotropic meshes. Beyond that, it can be combined with topology modification techniques and local mesh regeneration techniques to form a more reliable method for untangling and smoothing of 2D meshes.

Author Contributions: Conceptualization, S.S. and Z.G.; methodology, S.S. and Z.G.; software, Z.G.; theoretical analysis, Z.G. and M.G.; writing—original draft preparation, Z.G.; writing—review and editing, S.S., Z.G. and M.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Key R&D Program of China grant number 2017YFC0803300 and 2018YFC0809700.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MR	Mean ratio
EEGT	Explicit element geometric transformation
Mesquite	Mesh quality improvement toolkit
GPU	Graphics processing unit
PDEs	Partial differential equations

References

1. Berzins, M. Solution-based mesh quality indicators for triangular and tetrahedral meshes. *Int. J. Comput. Geom. Appl.* **2000**, *10*, 333–346. [CrossRef]
2. Berzins, M. Mesh quality: A function of geometry, error estimates or both? *Eng. Comput.* **1999**, *15*, 236–247. [CrossRef]
3. Batdorf, M.; Freitag, L.; Ollivier-Gooch, C.; Batdorf, M.; Freitag, L.; Ollivier-Gooch, C. Computational study of the effect of unstructured mesh quality on solution efficiency. In Proceedings of the 13th Annual AIAA Computational Fluid Dynamics Conference, Snowmass Village, CO, USA, 29 June–2 July 1997; pp. 1–11.
4. Shewchuk, J.R. Delaunay refinement algorithms for triangular mesh generation. *Comput. Geom.* **2002**, *22*, 21–74. [CrossRef]
5. Cheng, S.W.; Dey, T.K.; Shewchuk, J. *Delaunay Mesh Generation*; CRC Press: Boca Raton, FL, USA, 2012.
6. Huang, W.; Russell, R.D. *Adaptive Moving Mesh Methods*; Springer Science & Business Media: Berlin, Germany, 2010; Volume 174.
7. Chandru, M.; Das, P.; Ramos, H. Numerical treatment of two-parameter singularly perturbed parabolic convection diffusion problems with non-smooth data. *Math. Methods Appl. Sci.* **2018**, *41*, 5359–5387. [CrossRef]
8. Chandru, M.; Prabha, T.; Das, P.; Shanthi, V. A numerical method for solving boundary and interior layers dominated parabolic problems with discontinuous convection coefficient and source terms. *Differ. Equ. Dyn. Syst.* **2019**, *27*, 91–112. [CrossRef]
9. Das, P. An a posteriori based convergence analysis for a nonlinear singularly perturbed system of delay differential equations on an adaptive mesh. *Numer. Algorithms* **2019**, *81*, 465–487. [CrossRef]
10. Das, P.; Rana, S.; Vigo-Aguiar, J. Higher order accurate approximations on equidistributed meshes for boundary layer originated mixed type reaction diffusion systems with multiple scale nature. *Appl. Numer. Math.* **2020**, *148*, 79–97. [CrossRef]
11. Das, P. Comparison of a priori and a posteriori meshes for singularly perturbed nonlinear parameterized problems. *J. Comput. Appl. Math.* **2015**, *290*, 16–25. [CrossRef]
12. Das, P. A higher order difference method for singularly perturbed parabolic partial differential equations. *J. Differ. Equ. Appl.* **2018**, *24*, 452–477. [CrossRef]
13. Das, P.; Mehrmann, V. Numerical solution of singularly perturbed convection-diffusion-reaction problems with two small parameters. *BIT Numer. Math.* **2016**, *56*, 51–76. [CrossRef]
14. Danczyk, J.; Suresh, K. Finite element analysis over tangled simplicial meshes: Theory and implementation. *Finite Elem. Anal. Des.* **2013**, *70*, 57–67. [CrossRef]
15. Peraire, J.; Vahdati, M.; Morgan, K.; Zienkiewicz, O.C. Adaptive remeshing for compressible flow computations. *J. Comput. Phys.* **1987**, *72*, 449–466. [CrossRef]
16. Kossaczkỳ, I. A recursive approach to local mesh refinement in two and three dimensions. *J. Comput. Appl. Math.* **1994**, *55*, 275–288. [CrossRef]

17. Kim, J. An Iterative Mesh Untangling Algorithm Using Edge Flip. *Math. Probl. Eng.* **2017**, *2017*, 2953736. [CrossRef]
18. Field, D.A. Laplacian smoothing and Delaunay triangulations. *Commun. Appl. Numer. Methods* **1988**, *4*, 709–712. [CrossRef]
19. Freitag, L.A.; Plassmann, P. Local optimization-based simplicial mesh untangling and improvement. *Int. J. Numer. Methods Eng.* **2000**, *49*, 109–125. [CrossRef]
20. Knupp, P.M. Hexahedral and tetrahedral mesh untangling. *Eng. Comput.* **2001**, *17*, 261–268. [CrossRef]
21. Franks, J.W.; Knupp, P.M. A new strategy for untangling 2D meshes via node-movement. In *CSRI Summer Proceedings*; Sandia National Laboratories: Albuquerque, NM, USA, 2010; pp. 152–165.
22. Brewer, M.L.; Diachin, L.F.; Knupp, P.M.; Leurent, T.; Melander, D.J. The Mesquite Mesh Quality Improvement Toolkit. In Proceedings of the 12th International Meshing Roundtable, Santa Fe, NM, USA, 14–17 September 2003; pp. 239–250.
23. Kim, J.; Panitanarak, T.; Shontz, S.M. A multiobjective mesh optimization framework for mesh quality improvement and mesh untangling. *Int. J. Numer. Methods Eng.* **2013**, *94*, 20–42. [CrossRef]
24. Escobar, J.M.; Rodrıguez, E.; Montenegro, R.; Montero, G.; González-Yuste, J.M. Simultaneous untangling and smoothing of tetrahedral meshes. *Comput. Methods Appl. Mech. Eng.* **2003**, *192*, 2775–2787. [CrossRef]
25. Escobar, J.; Montero, G.; Montenegro, R.; Rodríguez, E. An algebraic method for smoothing surface triangulations on a local parametric space. *Int. J. Numer. Methods Eng.* **2006**, *66*, 740–760. [CrossRef]
26. Gargallo-Peiró, A.; Roca, X.; Sarrate, J. A surface mesh smoothing and untangling method independent of the CAD parameterization. *Comput. Mech.* **2014**, *53*, 587–609. [CrossRef]
27. Kim, J.; Choi, J.; Kang, W. A Data-Driven Approach for Simultaneous Mesh Untangling and Smoothing Using Pointer Networks. *IEEE Access* **2020**, *8*, 70329–70342. [CrossRef]
28. Vartziotis, D.; Wipper, J. The geometric element transformation method for mixed mesh smoothing. *Eng. Comput.* **2009**, *25*, 287–301. [CrossRef]
29. Vartziotis, D.; Wipper, J. Fast smoothing of mixed volume meshes based on the effective geometric element transformation method. *Comput. Methods Appl. Mech. Eng.* **2012**, *201*, 65–81. [CrossRef]
30. Sun, S.; Zhang, M.; Gou, Z. Smoothing algorithm for planar and surface mesh based on element geometric deformation. *Math. Probl. Eng.* **2015**, *2015*, 435648. [CrossRef]
31. Davis, P.J. *Circulant Matrices*; American Mathematical Soc.: Providence, RI, USA, 2013.
32. Knupp, P.M. Algebraic mesh quality metrics. *SIAM J. Sci. Comput.* **2001**, *23*, 193–218. [CrossRef]
33. Sun, S.; Lv, S.; Yuan, Y.; Yuan, M. Mesh deformation method based on mean value coordinates interpolation. *Acta Mech. Solida Sin.* **2016**, *29*, 1–12. [CrossRef]