Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Proposing an intelligent mesh-smoothing method with graph neural networks*

Zhichao WANG[†1,2], Xinhai CHEN[†‡1,2], Junjun YAN[1,2], Jie LIU[1,2]

[1]*Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha 410073, China*

[2]*Laboratory of Digitizing Software for Frontier Equipment, National University of Defense Technology, National University of Defense Technology, Changsha 410073, China*

[†]E-mail: wangzhichao@nudt.edu.cn; chenxinhai16@nudt.edu.cn

Received Dec. 29, 2023; Revision accepted Apr. 16, 2024; Crosschecked

Unedited

Abstract: In computational fluid dynamics (CFD), mesh-smoothing methods are commonly utilized to refine the mesh quality for achieving high-precision numerical simulations. Specifically, optimization-based smoothing is used for high-quality mesh smoothing, but it incurs significant computational overhead. Pioneer works improve its smoothing efficiency by adopting supervised learning to learn smoothing methods from high-quality meshes. However, they pose difficulty in smoothing the mesh nodes with varying degrees and also require data augmentation to address the node input sequence problem. Additionally, the required labeled high-quality meshes further limit the applicability of the proposed method. In the present paper, we present GMSNet, a lightweight neural network model for intelligent mesh smoothing. GMSNet adopts graph neural networks (GNNs) to extract features of the node's neighbors and outputs the optimal node position. During smoothing, we also introduce a fault-tolerance mechanism to prevent GMSNet from generating negative volume elements. With a lightweight model, GMSNet can effectively smooth mesh nodes with varying degrees and remain unaffected by the order of input data. A novel loss function, MetricLoss, is also developed to eliminate the need for high-quality meshes, which provides a stable and rapid convergence during training. We compare GMSNet with commonly used mesh-smoothing methods on two-dimensional triangle meshes. The experimental results show that the GMSNet achieves outstanding mesh-smoothing performances with 5% model parameters of the previous model, and attains 13.56 times faster than optimization-based smoothing.

Key words: Unstructured mesh; Mesh smoothing; Graph neural network; Optimization-based smoothing
https://doi.org/10.1631/FITEE.2300878 **CLC number:** TP391.4

1 Introduction

With the rapid advancements made in computer technology, computational fluid dynamics (CFD) has emerged as a crucial method for studying the principles of fluid dynamics. Its wide applications span diverse fields, including aerospace, hydraulic engineering, automotive engineering, and biomedicine (Bridgeman et al., 2010; Damjanović et al., 2011; Samstag et al., 2016; Spalart and Venkatakrishnan, 2016). Normally, CFD simulations are performed by discretizing the governing physical equations and subsequently solving large-scale algebraic systems of discretized equations to obtain fluid variables. Discretization, a critical step in CFD, encompasses two key aspects: discretizing the governing physi-

‡ Corresponding author

* This research work was supported in part by the National Key Research and Development Program of China (No.2021YFB0300101), Youth Foundation of National University of Defense Technology (No. ZK2023-11), and National Natural Science Foundation of China (No. 12102467)

ORCID: Zhichao WANG, https://orcid.org/0009-0007-4034-0578

© Zhejiang University Press 2024

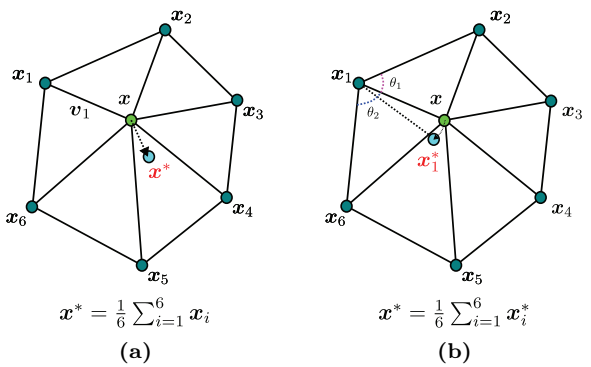


Fig. 1 (a) Laplacian smoothing. The *StarPolygon* of point x is formed by $\mathbf{S}(x) = \{x_1, x_2, \ldots, x_6\}$ as a hexagon. The optimized mesh nodes are then located at the arithmetic average of the coordinates of the nodes in the *StarPolygon*. (b) Angle-based smoothing. The figure shows the optimal position based on x_1. This is achieved by rotating x to the angle bisector of the angle where x_1 is located, resulting in x_1^*. This process is repeated for other points in the mesh, and the final optimized points are obtained through arithmetic averaging.

cal equations and discretizing the computational domain (Moukalled et al., 2016). The latter process, known as mesh generation, plays a fundamental role in CFD. It involves partitioning the computational domain into non-overlapping mesh elements, such as polygons in the two-dimensional region and polyhedra in the three-dimensional region (Baker, 2005). The quality of the generated mesh profoundly impacts the convergence, accuracy, and the efficiency of the numerical simulations. Orthogonality, smoothness, distributivity and density distribution of mesh elements significantly influence the stability and convergence of the solution matrix (Knupp, 2001). Consequently, the quest for high-quality mesh generation remains a vibrant and active area in CFD research. During the advent of practical mesh-generation processes, the initially generated mesh often fails to meet the simulation requirements. To enhance the quality of the mesh, mesh quality improvement techniques are commonly employed, including mesh smoothing (Freitag and Knupp, 2002; Durand et al., 2019; Hai et al., 2021), face-swapping, edge-swapping (Freitag and Ollivier-Gooch, 1997; Prasad, 2018), point insertion/deletion (Escobar et al., 2005; Klingner and Shewchuk, 2008; Guo and Hai, 2021), and other techniques. Among such techniques, mesh-smoothing methods are the most commonly used approach for improving the mesh quality.

Mesh smoothing can be categorized into two main types: heuristic smoothing and optimization-based smoothing (Guo et al., 2022). One representative heuristic method is Laplacian smoothing (Herrmann, 1976; Pan et al., 2020; Sharp and Crane, 2020) (shown in Fig. 1a). In Laplacian smoothing, the mesh node is placed at the arithmetic average of the coordinates of the nodes in the *StarPolygon* (a polyhedron containing the node, as shown in Fig. 1) for smoothing. This method is efficient but may produce negative volume elements when the *StarPolygon* is non-convex. Angle-based smoothing (Zhou and Shimada, 2000; Guo et al., 2020) achieves mesh smoothing by placing the mesh node on the angle bisectors of the nodes of the *StarPolygon* (shown in Fig. 1b). In addition to node-based methods, Centroidal Voronoi tessellation (CVT) smoothing (Du and Gunzburger, 2002; Du and Wang, 2003; Liu et al., 2009) recalculates the Voronoi regions of each node through Lloyd iterations (Lloyd, 1982) and relocates the point to the centroid of the Voronoi region for smoothing. To enhance the efficiency of the Lloyd algorithm, deterministic methods have been proposed for calculating the new point location (Du et al., 1999). Unlike node-based methods, smoothing methods based on element transformation (Vartziotis et al., 2008; Vartziotis and Papadrakakis, 2013; Vartziotis and Wipper, 2019) aim to enhance the mesh quality by applying linear transformations to the mesh elements. In such methods, individual mesh elements can converge to optimal shapes through multiple iterations. Although heuristic-based mesh-smoothing methods are simple and efficient, their optimization capabilities are limited. They may lead to inverted elements, and the smoothness effect heavily relies on the design of the heuristic functions. On the other hand, optimization-based methods (Parthasarathy and Kodiyalam, 1991; Canann et al., 1998; Chen, 2004; Zhang et al., 2009) achieve mesh smoothing by optimizing the mesh quality evaluation metric in the local area. Parthasarathy and Kodiyalam (1991) firstly formulate the mesh-smoothing problem as a constrained optimization problem and utilize iterative optimization algorithms to optimize the mesh node positions. Despite the adoption of different mesh quality evaluation metrics and optimization methods in subsequent works, the optimization-based methods usually require solving optimization problems iteratively for mesh smoothing, resulting

Unedited

in low efficiency.

Recently, artificial intelligence (AI) methods have also been widely used in mesh-related fields. Most of these research works are devoted to applying AI methods to mesh quality evaluation (Chen et al., 2020, 2021; Wang et al., 2022), mesh density control (Zhang et al., 2020, 2021), mesh generation (Daroya et al., 2020; Papagiannopoulos et al., 2021; Chen et al., 2022), mesh refinement (Bohn and Feischl, 2021; PaszyÅĎski et al., 2021), and mesh adaptation (Fidkowski and Chen, 2021; Wallwork et al., 2022; Wu et al., 2022). However, there is relatively little work on AI-based mesh smoothing. Guo et al. (2022) firstly introduced a supervised learning approach to imitate optimization-based smoothing methods with feedforward neural networks. The proposed model, NN-Smoothing, improves the efficiency of optimization-based smoothing by directly giving the optimal node position. However, feedforward neural networks suffer from fixed-dimensional inputs. This necessitates separate models for mesh nodes with different degrees and data augmentation for different sequences of input nodes, thereby increasing the model's training cost. Moreover, to train the model through supervised learning, high-quality mesh generation also incurs burdensome computational overhead.

To overcome such limitations, we present a novel mesh-smoothing model, GMSNet, based on graph neural networks (GNNs) (Wu et al., 2021) in the present paper. We propose a lightweight and efficient GNN model for learning the process of mesh smoothing. GMSNet avoids the overhead of solving the optimization problem by extracting features of the neighboring nodes of the mesh node to directly output better node position. We show that through the ability of GNNs to handle unstructured data, GMSNet can smooth nodes of varying degrees with a single model and elegantly solves the node input sequence problem without data augmentation. Once trained, it can be applied to smooth the mesh with different shapes. We also propose a shift truncation operation to avoid introducing negative volume elements when smoothing the mesh. Beyond the proposed GMSNet, we introduce a novel loss function, MetricLoss, based on the mesh quality metrics to train the model, which further eliminates the overhead of generating high-quality meshes. We conduct extensive experiments among GMSNet and commonly used mesh-smoothing algorithms on two-dimensional triangular meshes. The experimental results show that our model achieves a speedup of 13.56 times compared with optimization-based smoothing while achieving similar performance, and outstands all the other heuristic smoothing algorithms. The results also indicate that GMSNet can be applied to meshes that were unseen during training. Meanwhile, compared to previous NN-Smoothing model, GMSNet has only 5% of its model parameter, but obtains superior mesh-smoothing performance. We also validate the effectiveness of the proposed MetriLoss with comparative experiments. We summarize our contributions as follows:

1. We propose a lightweight GNN model, GMSNet, for intelligent mesh smoothing. GMSNet can smoothing node with varying degrees and remain unaffected by the data input order. Additionally, we offer a fault-tolerance mechanism, shift truncation, to prevent GMSNet from generating negative volume elements

2. Basing on the mesh quality metrics, we introduce a novel loss function, MetricLoss, to train the model without the necessity for high-quality meshes. MetricLoss exhibits a stable and rapid convergence during model training.

3. We validate the effectiveness of GMSNet through extensive experiments conducted on two-dimensional triangular meshes. The experimental results demonstrate that GMSNet achieves excellent mesh-smoothing performance and significantly outperforms optimization-based smoothing with an average speedup of 13.56 times. Comparative experiments are also conducted to showcase the effectiveness of the proposed MetricLoss.

The remaining parts of the present paper are organized as follows. In Section 2, we introduce commonly used mesh-smoothing methods and applications of neural networks in the field of mesh. In Section 3, we present the proposed model, providing detailed explanations of the mesh data preprocessing, design of the model's architecture, loss function and training method. In Section 4, we conduct experiments to compare the performance of our proposed model with baseline models and discuss the

Unedited

effectiveness of loss function. Finally, in the conclusion section, we summarize the entire paper and propose potential future work that can be explored.

2 Related work

2.1 Heuristic mesh smoothing

Laplacian smoothing is the most commonly used heuristic mesh-smoothing method. It updates the node coordinate to the arithmetic average of nodes in *StarPolygon*. Weighted Laplacian smoothing (Vollmer et al., 1999) introduces additional weights or importance factors to the neighboring nodes or edges during the smoothing process, offering more control over the smoothing effect and preserving specific features of the mesh. Smart Laplacian smoothing (Field, 1988) includes using a check before each node movement to assess whether the operation will improve the mesh quality or not. If the mesh quality does not improve, the movement of the mesh node is skipped, resulting in a more efficient process. Angle-based mesh smoothing achieves smoothness by considering the angles of the mesh nodes. In this method, the node is rotated to align with the angle bisectors of each node in *StarPolygon*. Since the angle bisectors of the various nodes of the *StarPolygon* may not coincide, the final node positions require additional calculation. This can be achieved by calculating the average of the node coordinates or by solving a least squares problem. CVT smoothing positions the node at the centroids of the Voronoi region defined by the mesh node. To enhance the efficiency of the Lloyd algorithm in the solving process, more efficient method has been designed to calculate the centroids, as described below:

$$\boldsymbol{x}_i^* = \frac{1}{|\Omega_i|} \sum_{T_j \in \Omega_i} |T_j| \, C_j. \tag{1}$$

where $\boldsymbol{x}_i^*$ represents the new node position, $|\Omega_i|$ is the total area of the node's *StarPolygon*, $|T_j|$ the area of the j th triangle, and C_j the circumcenter of the j th triangle.

It is worth noting that there is no absolute boundary between heuristic smoothing and optimization-based smoothing. From another perspective, heuristic mesh smoothing can also be viewed as an optimization-based approach. For instance, Laplacian smoothing can be viewed as minimizing the energy function $E = \frac{|\mathbf{S}(\boldsymbol{x})|}{2} \sum_{i=1}^{|\mathbf{S}(\boldsymbol{x})|} |\boldsymbol{v}_i|^2$, where $\boldsymbol{v}_i$ represents the edge from $\boldsymbol{x}$ to $\boldsymbol{x}_i$, and $|\mathbf{S}(\boldsymbol{x})|$ is the number of nodes in the *StarPolygon* (shown in Fig. 1a). Similarly, angle-based mesh smoothing can be seen as minimizing the energy function $E = \frac{|\mathbf{S}(\boldsymbol{x})|}{2} \sum_{i=1}^{2|\mathbf{S}(\boldsymbol{x})|} \theta_i^2$, where θ_i is the angle between the edge from $\boldsymbol{x}$ to $\boldsymbol{x}_i$ and the edge of the polygon (shown in Fig. 1b). The primary advantage of heuristic mesh smoothing lies in its efficiency. However, its smoothing performance often is inferior to that of optimization-based smoothing. Furthermore, the effectiveness of heuristic mesh smoothing heavily relies on the design of the heuristic functions.

2.2 Optimization-based smoothing

The mesh-smoothing method based on optimization can be formalized as the following problem: Given an initial mesh with node positions $\boldsymbol{x}_i$ and a set of constraints and the objective function f, the goal is to find a new node positions $\boldsymbol{x}_i^*$ that minimizes the objective function f. Mathematically, this can be expressed as:

$$\begin{aligned} \boldsymbol{x}_i^* &= \arg\min_{\boldsymbol{x}_i} f\left(\boldsymbol{x}_i, \mathbf{S}\left(\boldsymbol{x}_i\right)\right), \\ \text{s.t.} \quad & \boldsymbol{x}_i \in \mathcal{X}. \end{aligned} \tag{2}$$

where $\boldsymbol{x}_i^*$ represents the new position of the mesh node $\boldsymbol{x}_i$, f is some mesh quality evaluation function, $\mathcal{X}$ is the feasible set satisfying constraints on $\boldsymbol{x}_i^*$, and $\mathbf{S}(\boldsymbol{x}_i)$ is the set of nodes of the *StarPolygon* of $\boldsymbol{x}_i$. It is important to mention that the input of this function also includes the connectivity between the nodes, which is omitted here for clarity. Different choices of evaluation functions lead to different mesh-smoothing methods, reflecting our emphasis on different mesh qualities. Commonly used evaluation functions include the maximum minimum angle (Xu et al., 2018), aspect ratio, and distort ratio, among others (Parthasarathy and Kodiyalam, 1991). If the function f is differentiable, the optimal point of this constrained optimization problem may be solved by setting the gradient $\nabla f = 0$ to obtain an explicit expression. However, in most cases, it is difficult to derive explicit expressions for $\boldsymbol{x}_i^*$ as in Laplacian smoothing and angle-based smoothing. Iterative methods are often used to solve $\boldsymbol{x}_i^*$, which are of low efficiency. Therefore, developing an efficient way to solve the optimal positions is a problem that needs to be addressed.

Unedited

2.3 Neural networks in mesh-related fields

Inspired by the neurons in the human brain, artificial neural networks are being utilized to learn complex function mappings. They find extensive applications in machine learning (ML) and AI, addressing tasks such as image recognition, speech recognition, natural language processing, and more (Reddy, 1976; LeCun et al., 2015; Pak and Kim, 2017; Chowdhary, 2020). Recently, neural networks have found significant applications in various domains related to meshes. In the realm of mesh quality evaluation, GridNet, a convolutional neural network model, was introduced by Chen et al. (2020), along with the NACA-market dataset, to facilitate the automated evaluation of structured mesh quality. Extending this notion to unstructured meshes, Wang et al. (2022) employed GNNs for mesh quality assessment. In the pursuit of refining the mesh distribution, Zhang et al. (2020) employed an artificial neural network to enhance the conventional mesh-generation software, enabling prediction of the local mesh density throughout the domain. This approach was further expanded to tetrahedral meshes, as demonstrated convincingly through extensive testing (Zhang et al., 2021). In the domain of intelligent mesh generation, Daroya et al. (2020) presented an algorithm that leverages global structural information from point clouds to achieve high-quality mesh reconstruction. Similarly, Papagiannopoulos et al. (2021) harnessed the data extracted from the meshed contours to train neural networks, enabling accurate approximation of the number, placement, and interconnectivity of nodes within the meshing domain. Taking a novel differential approach, Chen et al. (2022) introduced MGNet, an unsupervised neural network methodology for generating structured meshes, yielding promising results. Beyond generation, AI techniques have also made significant contributions to mesh refinement and adaptation. Bohn and Feischl (2021) used recurrent neural networks to learn optimal mesh-refinement algorithms, establishing its prowess as an effective black-box tool for enhancing a wide spectrum of partial differential equations. Enriching variational mesh adaptation, Wu et al. (2022) seamlessly integrated a machine learning regression model to expedite the flow field estimation on the updated meshes. Meanwhile, Wallwork et al. (2022) devised a data-driven, goal-oriented mesh-adaptation strategy, underpinned by a trained neural network, effectively supplanting the computationally expensive error-estimation phase in the adaptation process. Furthermore, Fidkowski and Chen (2021) ingeniously employed artificial neural networks to ascertain optimal anisotropy in computational meshes, yielding enhanced mesh efficiency in comparison to the conventional methods. In terms of mesh smoothing, NN-Smoothing (Guo et al., 2022) imitates optimization-based mesh smoothing using feedforward neural networks, significantly enhancing the efficiency of optimization-based mesh smoothing. However, separate models training and expensive high-quality mesh generation incur significant computational overhead.

In addition to conventional deep learning algorithms, GNNs (Wu et al., 2021) were introduced to enhance the learning capacity of artificial intelligence methods for handling non-structured data. GNNs utilize graph convolutions (Kipf and Welling, 2016) to incorporate the topological connections in the feature learning process on graphs. As meshes can be naturally represented as graph data, GNNs have found extensive applications in various CFD fields, including mesh refinement, flow field simulation, turbulence modeling, and more (Lino et al., 2021; Pfaff et al., 2021; Han et al., 2022; Peng et al., 2022; Song et al., 2022). Therefore, we argue that applying GNNs to mesh smoothing is a promising and effective solution.

Unedited

3 Methodology

3.1 Problem formulation

The mesh-smoothing problem involves enhancing the quality of a mesh by adjusting the positions of its nodes while maintaining the connectivity between them. Mesh-smoothing processes are defined as functions that operate on the mesh, taking the mesh nodes and their connections as inputs and providing new coordinates for the mesh nodes as outputs. For each mesh node, the input of the smoothing function is itself and its *StarPolygon*, which comprises the node's one-ring neighbors. In the present paper, we define the mesh as a node graph. Specifically, given a node $\boldsymbol{x}_0$ and its *StarPolygon*, we represent it with a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ represents the node and nodes in the *StarPoly-*

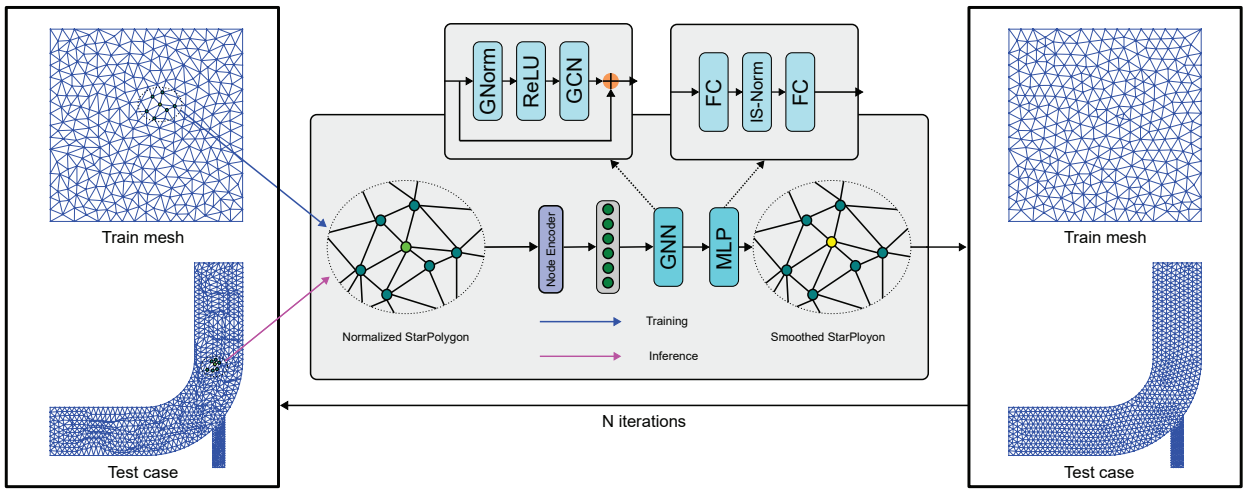


Fig. 2 The architecture of GMSNet. GMSNet smoothes the mesh by calculating the optimized position for each mesh node. The process begins with the normalization of the input *StarPolygon*. Next, feature transformation is performed on the normalized features, and information from the *StarPolygon* nodes is integrated using graph convolution. Finally, the model predicts the optimized node positions through FC layers. In the figure, GNorm represents the GraphNorm operation, and IS-Norm represents the InstanceNorm operation. The model iterates N times over the entire mesh to perform the smoothing operation. After training, GMSNet can be applied to previously unseen meshes, such as the pipe's mesh.

Unedited

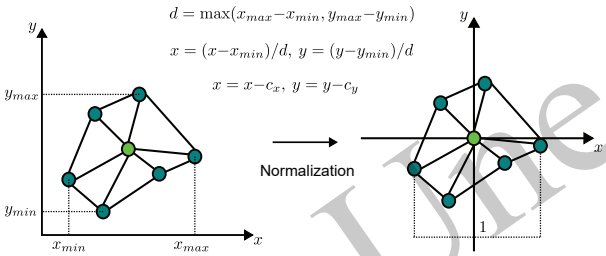


Fig. 3 Data normalization. The *StarPolygon* to be processed is normalized to the center of the coordinate axis. In the figure, d represents the maximum size of the *StarPolygon*, x and y represent the coordinates to be scaled, and c_x and c_y are the normalized coordinates of the target node after the scaling.

gon, $\boldsymbol{x}_i$ is the coordinate of node i in *StarPolygon*, n is the number of nodes in the *StarPolygon*, and $\mathcal{E} = \{(i,j) \mid \text{if } \boldsymbol{x}_i \text{ is connected with } \boldsymbol{x}_j\}$ represents the connections between the nodes. A typical smoothing process is iterative, where at the t th iteration step, the initial node graph is denoted as $\mathcal{G}_t$, and the optimized node position for the center node can be represented as:

$$\boldsymbol{x}_0^{t+1} = \mathcal{F}(\mathcal{G}_t) = \mathcal{F}(\mathcal{V}_t, \mathcal{E}). \tag{3}$$

In the present paper, we employ GNNs to learn the function $\mathcal{F}$ for mesh smoothing.

3.2 GMSNet

3.2.1 Lightweight model design

The mesh-smoothing model based on the GNNs is depicted in Fig. 2. Given a graph consisting of mesh nodes and edges, our goal is to compute a more optimal mesh node position for each mesh node to smoothing the mesh. In the model, we use the coordinate positions as node features, denoted as $\mathbf{X} \in \mathbb{R}^{(n+1)\times 2}$ (n nodes in the *StarPolygon* and one free node to move). The connectivity between the nodes is represented by the adjacency matrix $\mathbf{A} \in \mathbb{R}^{(n+1)\times(n+1)}$ (node index i is omitted for clarity).

To ensure the model's scale invariance, we apply node normalization during processing. The model normalizes the node input using min–max normalization on $\mathbf{X}$, restricting it to the range of 0 and 1, and subsequently performs a translation to center the node around the coordinate origin. After model processing, we employ an affine transformation to map the node positions back to their original scale. Data normalization is shown in Fig. 3. After normalizing the data, we transform the features by a linear layer, which can be expressed as

$$\mathbf{X}_h = \mathrm{Norm}(\mathbf{X})\mathbf{W}_l + \mathbf{b}_l. \tag{4}$$

where Norm represents the normalization operation, $\mathbf{W}_l \in \mathbb{R}^{2\times H}$ and $\mathbf{b}_l$ is the transformation matrix and bias of the linear layer, H is the dimension of the hidden feature, and $\mathbf{X}_h \in \mathbb{R}^{(n+1)\times H}$ is the output of the linear layer. Subsequently, we employ a residual graph convolutional network (GCN) layer (Kipf and Welling, 2016; Li et al., 2019) to compute the features of the hidden layer based on the features of the nodes in the *StarPolygon*. The process involves normalizing the features outputted by the linear layer using GraphNorm (Cai et al., 2021), followed by an activation layer, a convolutional layer, and a summation layer to calculate the hidden features of the nodes. This process can be represented as:

$$\hat{\mathbf{X}}_h = \mathrm{GraphNorm}(\mathbf{X}_h), \tag{5}$$

$$\mathbf{X}_g = \mathrm{GCN}(\mathrm{ReLU}(\hat{\mathbf{X}}_h), \tilde{\mathbf{A}}) + \hat{\mathbf{X}}_h \tag{6}$$

$$= \tilde{\mathbf{A}}[\mathrm{ReLU}(\hat{\mathbf{X}}_h)]\mathbf{W}_g + \hat{\mathbf{X}}_h. \tag{7}$$

where $\tilde{\mathbf{A}}$ is the normalized adjacency matrix[1], ReLU is the activation function, $\mathbf{W}_g$ is the parameter of GCN layer, and $\mathbf{X}_g \in \mathbb{R}^{(n+1)\times H}$ represents the final features output by the residual GCN layer. The final position of the free node is obtained through a two-layer fully connected (FC) neural network with InstanceNorm (Ulyanov et al., 2017). We only need to change the position of the free node , so we extract the feature of the free node and decode it through a multilayer perceptron (MLP). Let i_c be the index of the free node, then the optimized node position is given by:

$$\boldsymbol{x}^* = \mathrm{MLP}(\mathbf{X}_{g[i_c,:]}). \tag{8}$$

The smoothing algorithm should possess the capability to handle nodes with varying degrees and remain unaffected by the order of node inputs. In the NN-Smoothing model, handling nodes with different degrees is achieved by training separate models, while the order of node inputs is addressed through data augmentation by varying the starting node in the ring of *StarPolygon*. However, owing to the permutation invariance property of the GNNs, where the output remains unaffected by permutations in the input order (e.g., in the sum or mean function), GMSNet remains unaffected by changes in the node input sequence. Additionally, it can also smooth the nodes with varying degrees, as there are no restrictions on the number of neighbors a node can have in GNNs.

[1] $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$,$\hat{\mathbf{D}}$ is the degree matrix of $\hat{\mathbf{A}}$, and $\tilde{\mathbf{A}} = \hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}$

3.2.2 Model training

In the NN-Smoothing method, the model is trained using supervised learning. Labeled high-quality meshes are generated using optimization-based smoothing, which is a time-consuming task. In contrast, the proposed GMSNet directly learns the optimization process for mesh smoothing. This process can be illustrated by comparing it with optimization-based smoothing. In optimization-based smoothing, taking gradient descent as the optimization method and given a mesh quality function f to be optimized, the objective is to find its minimum value. In the k-th iteration, the optimization algorithm updates the position as $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k, \mathbf{S}(\boldsymbol{x}_0))$, where $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k, \mathbf{S}(\boldsymbol{x}_0))$ is the gradient of f and α is the step size. By iteratively updating $\boldsymbol{x}_k$, the function converges to a local or global optimum $\boldsymbol{x}^*$. In contrast, NN-Smoothing directly predicts the optimal point for mesh smoothing, i.e., $\hat{\boldsymbol{x}}^* = \mathrm{NN}\ (\boldsymbol{x}_0, \mathbf{S}(\boldsymbol{x}_0))$. However, this approach requires labeled high-quality meshes generated through the optimization algorithm, which is time-consuming. On the other hand, the GMSNet model does not require labeled data to learn the smoothing process for mesh nodes. Instead, the training process is driven by the quality evaluation metric of the mesh elements, which can be expressed as:

$$\begin{aligned}\mathbf{W}^* &= \arg\min_{\mathbf{W}} \mathcal{L}(\hat{\boldsymbol{x}}^*, \mathbf{S}(\boldsymbol{x}_0)) \\ &= \arg\min_{\mathbf{W}} \mathcal{L}(\mathcal{F}_{\mathbf{W}}(\boldsymbol{x}_0, \mathbf{S}(\boldsymbol{x}_0)), \mathbf{S}(\boldsymbol{x}_0)).\end{aligned} \tag{9}$$

where $\mathcal{L}$ is the loss function constructed using a mesh quality metric, $\mathcal{F}_{\mathbf{W}}$ is learned through the proposed model, and $\mathbf{W}$ is the parameters of the model. The loss function is based on the mesh element evaluation metric, and the model optimizes the positions of the mesh nodes by minimizing this function.

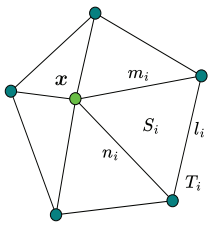
The main distinction between this approach and NN-Smoothing lies in the training data. In the proposed method, there are no high-quality meshes provided for the labels. Instead, the learning process relies solely on minimizing the mesh quality metric function. The primary divergence from optimization-based mesh-smoothing methods is that this approach directly offers optimized positions for mesh nodes without the need to solve an optimiza-

Unedited

Table 1 Comparison among optimization-based smoothing, NN-Smoothing and GMSNet

Method	Speed	Labeled high-quality mesh	Varying node degrees	Node input order
OptimSmoothing[1]	Slow	Not acquiring	Not affected	Not affected
NN-Smoothing	Fast	Acquiring	Training separate models	Performing data augmentation
GMSNet	Fast	Not acquiring	Not affected	Not affected

[1] Optimization-based smoothing.



$$|\mathbf{S}(\boldsymbol{x})| = 5$$

$$\mathcal{L}(\boldsymbol{x}, \mathbf{S}(\boldsymbol{x})) = \frac{1}{|\mathbf{S}(\boldsymbol{x})|} \sum_{i=1}^{|\mathbf{S}(\boldsymbol{x})|} \left(1 - \frac{1}{q_i}\right) = \frac{1}{|\mathbf{S}(\boldsymbol{x})|} \sum_{i=1}^{|\mathbf{S}(\boldsymbol{x})|} \left(1 - \frac{4\sqrt{3}S_i}{m_i^2 + n_i^2 + l_i^2}\right)$$

Fig. 4 MetricLoss. We use the mean transformed aspect ratio as the loss function for model optimization.

tion problem, resulting in a significant improvement in the efficiency of mesh smoothing. A comprehensive comparison of the three aforementioned methods is shown in Table 1.

3.2.3 MetricLoss

In this section, we introduce a loss function based on mesh quality metrics, called MetricLoss, as the optimization objective function for the model. By minimizing this loss function, the model can learn how to smooth the mesh to improve its quality. There are several metrics available to evaluate the quality of mesh elements, including the maximum angle, minimum angle, Jacobian matrix, aspect ratio, and others. In our case, we adopt the aspect ratio $q = \frac{m^2+n^2+l^2}{4\sqrt{3}S}$ to assess the mesh quality, where m, n and l are the edges of the triangle, and S is the area of the triangle. For an equilateral triangle, this value is 1, while for a degenerate triangle, it approaches $+\infty$. However, the range of this metric is too large, which can cause gradient explosion, especially for poor-shaped elements. To address this issue, we normalize this metric's domain to $[0, 1]$ through the function $f(q) = 1 - \frac{1}{q}$. For equilateral triangles, this transformed metric is 0, while for degenerate triangles, it is 1. The MetricLoss is designed as the mean value of the transformed metric of elements in its

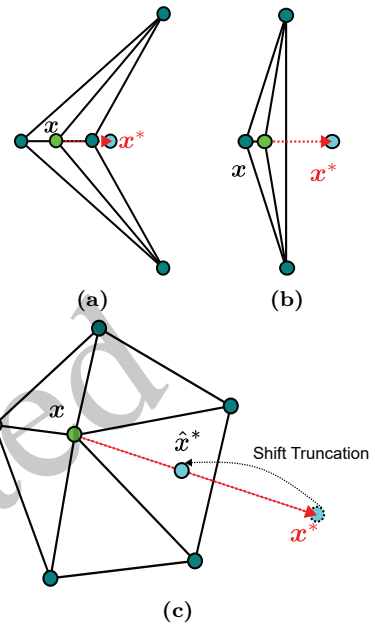


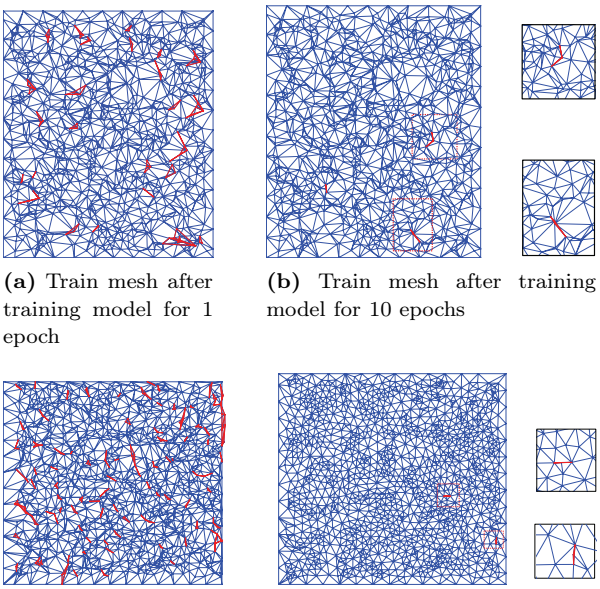
Fig. 5 (a) For non-convex *StarPolygons*, Laplace smoothing may move nodes outside the *StarPolygon*. (b) Computing the circumcenter of triangles is necessary for CVT smoothing. For highly distorted mesh elements, it circumcenter is far away from the *StarPolygon*, resulting in negative volume elements. (c) Shift truncation. In the training and inference phases of the model, we truncate the shift to avoid generating negative volume elements.

StarPloygon, which can be expressed as:

$$\mathcal{L}(\boldsymbol{x}, \mathbf{S}(\boldsymbol{x})) = \frac{1}{|\mathbf{S}(\boldsymbol{x})|} \sum_{i=1}^{|\mathbf{S}(\boldsymbol{x})|} (1 - \frac{1}{q_i}) = \frac{1}{|\mathbf{S}(\boldsymbol{x})|} \sum_{i=1}^{|\mathbf{S}(\boldsymbol{x})|} (1 - \frac{4\sqrt{3}S_i}{m_i^2 + n_i^2 + l_i^2}). \tag{10}$$

where $|\mathbf{S}(\boldsymbol{x})|$ is the number of nodes in the *StarPolygon*, m_i, n_i, l_i are the edges of triangle T_i, and q_i is the aspect ratio of T_i (see Fig. 4 for details). We validated the effectiveness of our designed loss function in Section 4.3.

Unedited



(a) Train mesh after training model for 1 epoch

(b) Train mesh after training model for 10 epochs

(c) Test mesh after training model for 1 epoch

(d) Test mesh after training model for 10 epochs

Fig. 6 The essential of shift truncation. The red elements in the figure represent negative volume elements. We show the changes in the number of negative volume elements throughout both the training and prediction phases. The training mesh is shown in Fig. (a) and (b), and the testing mesh is shown in Fig. (c) and (d). It can be seen that although training reduces the number of negative volume element generated, it cannot completely eliminate them.

3.2.4 Shift truncation

In the process of mesh generation and mesh smoothing, it is important to ensure that the generated mesh avoids negative volume elements. However, mesh smoothing algorithms can sometimes lead to the generation of negative volume elements, as depicted in Fig. 5. For instance, in the case of Laplacian smoothing, when dealing with non-convex *StarPolygons*, negative volume elements may arise (as illustrated in Fig. 5a). Similarly, during CVT smoothing, the calculation of the Voronoi centroid for elongated mesh elements can result in shifts that extend far away from the *StarPolygon* region, as shown in Fig. 5b (in the case of degenerate mesh elements, the centroid position may even be at an infinite distance). Additionally, in optimization algorithms, using a uniform step size for different scales of the mesh elements can also lead to the production of negative volume elements, as the optimization process may overshoot the optimal position.

Negative volume elements can also occur during

Algorithm 1 Shift truncation operation

Mesh Node i original position $\boldsymbol{x}_i$, shift $\Delta\boldsymbol{x}_i$, optimized position $\boldsymbol{x}_i^* = \boldsymbol{x}_i + \Delta\boldsymbol{x}_i$
while $\boldsymbol{x}_i^*$ results in negative elements **do**
  $\Delta\boldsymbol{x}_i = 0.5\Delta\boldsymbol{x}_i$
  $\boldsymbol{x}_i^* = \boldsymbol{x}_i + \Delta\boldsymbol{x}_i$
end while

Algorithm 2 Training of GMSNet

Mesh dataset $\mathbf{M} = \{\mathcal{M}_i\}_{i=1}^n$, MetricLoss $\mathcal{L}$
Training epochs N, batch size $\mathcal{B}$, learning rate α
GMSNet parameters $\mathbf{W}$
for $j \leftarrow 1$ to N **do**
  for mesh $\mathcal{M}_i$ in $\mathbf{M}$ **do**
    Sample $\mathcal{B}$ mesh nodes from $\mathcal{M}_i$: $\{\boldsymbol{x}_i\}_{i=1}^{\mathcal{B}}$
    Compute the optimized node position: $\boldsymbol{x}_i^* = \mathrm{GMSNet}(\boldsymbol{x}_i, \mathbf{S}(\boldsymbol{x}_i))$ for $\boldsymbol{x}_i \in \{\boldsymbol{x}_i\}_{i=1}^{\mathcal{B}}$
    Shift truncate $\Delta\boldsymbol{x}_i$ if $\Delta\boldsymbol{x}_i$ results in negative volume elements
  end for
  Update model parameters: $\mathbf{W} \leftarrow \mathbf{W} - \frac{\alpha}{\mathcal{B}}\sum_{i=1}^{\mathcal{B}} \nabla\mathcal{L}(\boldsymbol{x}_i^*, \mathbf{S}(\boldsymbol{x}_i^*))$ // Here we use stochastic gradient descent as the optimization method.
end for

the training and inference stages of neural network-based smoothing algorithms. To show this, we visualized the output mesh after the first epoch of training. It can be observed that a large portion of the node updates resulted in negative volume elements, as shown in Fig. 6a. However, as the model continues training, it learns the smoothing process, and the number of negative volume elements decreases, as depicted in Fig. 6b. We also study the impact of shift truncation in the model predictions. The experimental results are shown in Fig. 6c and 6d. It is evident that for certain highly distorted elements, the model may produce negative volume elements despite being trained for epochs. The generation of negative volume elements does not disrupt the learning process. With the model training, the occurrence of negative volume elements gradually decreases. However, due to the uncertainty of neural networks, although rare, it is still possible for the model to introduce negative volume elements during the inference stage. Hence, a method is required to prevent the occurrence of negative volume elements. The simplest approach is to set the displacements that result in negative volume elements to zero. However, this approach hinders the update of poorly shaped mesh elements, which is precisely the optimization goal of mesh smoothing. Therefore, we have adopted a line search method

Unedited

to handle negative volume elements, as depicted in Fig. 5c and Algorithm 1. We repeatedly half the shift that introduces the negative volume elements until no negative volume element is generated. With the aforementioned method, the training of GMSNet is described in Algorithm 2.

4 Experiments

4.1 Experimental setup

In this section, we conducted a comprehensive performance comparison of the proposed GMSNet with five baseline smoothing methods (algorithms). The baseline methods we evaluated are as follows: Laplacian smoothing (Field, 1988), angle-based smoothing (Zhou and Shimada, 2000), CVT smoothing (Du and Gunzburger, 2002), GETMe smoothing (Vartziotis and Wipper, 2019), optimization-based smoothing, and NN-Smoothing (Guo et al., 2022). Below are the implementation details for each baseline model:

- All models adopted the **sequential updating**, where the mesh nodes are directly updated after each optimization step (as opposed to computing the updates for all nodes and then updating them together).
- All models are implemented entirely in Python, with their sole distinction lying in how they update the positions of mesh nodes.
- Smart Laplacian smoothing was adopted to prevent negative volume elements.
- In the angle-based smoothing, the final node positions were obtained by averaging the optimized positions computed for each angle in the *StarPolygon*.
- The CVT smoothing employed the Eq. (1) to improve the algorithm's performance.
- The element transformation-based smoothing methods, GETMeSmoothing, is implemented in accordance with the algorithms outlined in *The GETMe Mesh Smoothing Framework* (Vartziotis and Wipper, 2019). The parameter configuration utilized in this implementation includes $\theta = \frac{\pi}{3}$, $\lambda = 0$, and $\varrho = 1$. To ensure a fair comparison, a sequential version was employed.
- The optimization-based smoothing used MetricLoss defined in Section 3.2.3 as the objective function and employed Adam (Kingma and Ba, 2015) as the optimizer. Each node was optimized for a maximum of 20 iterations.
- The NN-Smoothing followed the implementation approach in the original paper. For nodes with different degrees, we trained different models to smooth the meshes. Additionally, Laplacian smoothing was used to handle nodes with degrees other than 3, 4, 5, 6, 7, 8, and 9.

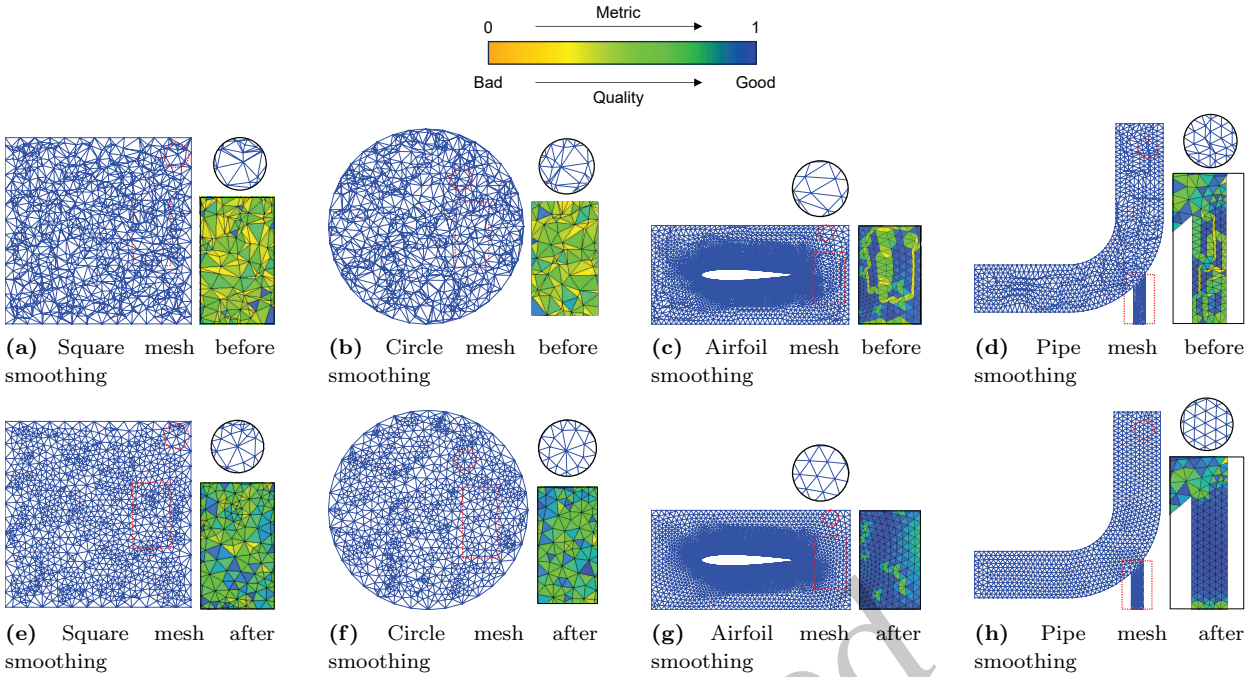
We trained the model solely using two-dimensional triangle meshes, comprising a total of 20 meshes with a square domain. The dataset was split into training, validation, and testing sets in a ratio of 6:2:2. Each mesh has distinct sizes and densities, randomly generated before the training. The mesh nodes are positioned randomly within the geometric domain, and the meshes are generated using Delaunay triangulation (Lee and Schachter, 1980). Figures of mesh examples can be found in **Appendix A.1**.

In the experiment, we utilized the final optimization results obtained from the optimization-based smoothing as the training labels for the NN-Smoothing model. Simultaneously, we trained the GMSNet without incorporating labels during the training process using the same dataset. For both neural network models, we employed Adam (Kingma and Ba, 2015) as the optimizer with an initial learning rate of 1e-2. Throughout the training process, the learning rate was dynamically adjusted based on the performance on the validation set. In each training epoch, we randomly sampled 32 mesh nodes from each mesh to train the models. Despite training with only a partial sampling of mesh nodes, the models converge effectively. NN-Smoothing and GMSNet models were trained on an NVIDIA Tesla P100 GPU. During the testing of model performance, for fair comparison, all the models were run on the Intel Core i7-12700H CPU.

4.2 Experimental results

We conducted tests on four mesh cases to evaluate the model's smoothing performance. To test the performance of the model more thoroughly, we constructed meshes containing highly distorted elements as test cases, as shown in the first row of Fig. 7. For the first two meshes, mesh nodes were generated

Unedited



(a) Square mesh before smoothing **(b)** Circle mesh before smoothing **(c)** Airfoil mesh before smoothing **(d)** Pipe mesh before smoothing

(e) Square mesh after smoothing **(f)** Circle mesh after smoothing **(g)** Airfoil mesh after smoothing **(h)** Pipe mesh after smoothing

Fig. 7 Mesh smoothing results of GMSNet on the test cases. Mesh nodes in meshes (a) and (b) are randomly generated in the domain, and mesh nodes in meshes (c) and (d) are manually adjusted to introduce distorted elements. In the detailed view, we utilize a color scheme to denote the quality of the mesh. Mesh elements range from blue for higher quality to yellow for lower quality, as indicated in the legend. The testing meshes before smoothing are shown in the first row, and the meshes after smoothing are shown in the second row.

Unedited

by uniform sampling within the geometric domain. The latter two meshes were initially created using meshing software to generate high-quality meshes, and then manual adjustments were made to introduce distorted mesh elements. It is worth noting that the meshes we tested are not included in our dataset, which solely consists of randomly generated two-dimensional unstructured meshes with a square domain.

To facilitate a fair comparison among different models, we have implemented the algorithms within the same framework. The variations among the algorithms lie in how they generate the optimized point positions. It is essential to mention that we have employed sequential algorithms in our study. However, for simpler algorithms like Laplacian smoothing, updating all nodes simultaneously is straightforward and fast. We have chosen the minimum angle, maximum angle, and the reciprocal of the aspect ratio as evaluation metrics for mesh quality. We conducted 10 experimental runs for each case, with a maximum of 100 smoothing iterations per experiment. The best-smoothed mesh was selected as the final result based on the weighted quality metric, which is defined as:

$$\hat{q} = \frac{1}{6}[\frac{\alpha_{\text{mean}} + \alpha_{\text{min}} + 120 - \beta_{\text{max}} - \beta_{\text{mean}}}{60} + \left(\frac{1}{q}\right)_{\text{mean}} + \left(\frac{1}{q}\right)_{\text{min}}]. \tag{11}$$

where α is the minimum angle, β is the maximum angle, and q is the aspect ratio. When evaluating mesh elements, a smaller maximum angle and aspect ratio are preferable, while a larger minimum angle is desirable. For an ideal mesh element, the maximum angle, minimum angle, and aspect ratio should be 60°, 60°, and 1, respectively. The algorithm speed is measured in terms of the time taken to process a single mesh node. The results of mesh smoothing using the GMSNet are depicted in the second column of Fig. 7, and a comprehensive comparison of the performances of each algorithm is summarized in Table 2.

From Fig. 7, it can be observed that for all four test cases, our proposed model significantly improves the mesh element quality. It is worth noting that GMSNet successfully smoothes mesh cases that are unseen during training (including circle mesh, airfoil mesh ,and pipe mesh). For meshes with rea-

Table 2 Performance of mesh-smoothing algorithms.

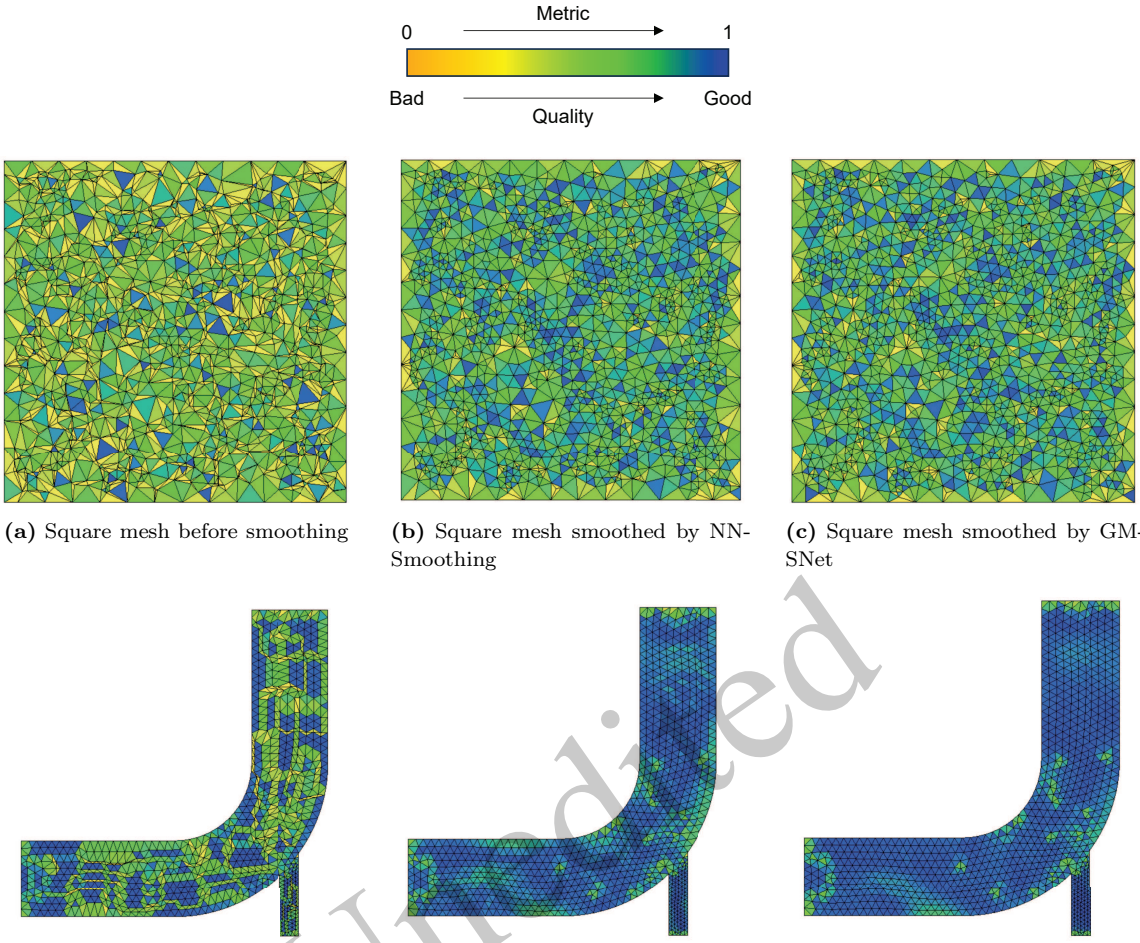
Mesh	Algorithm	Min. Angle		Max. Angle		$\frac{1}{\text{Aspect ratio}}$		s/per node
		Min	Mean	Max	Mean	Min	Mean	
Square	Origin	0.04	29.65	179.82	95.09	0.00	0.58	-
	LaplacianSmoothing	12.38	**43.97**	147.67	**79.44**	0.25	**0.79**	1.28E-04
	AngleSmoothing	10.52	41.26	148.6	81.76	0.22	0.76	5.66E-04
	CVTSmoothing	2.55	40.88	170.6	84.04	0.05	0.75	1.17E-03
	GETMeSmoothing	13.30	36.30	146.88	87.63	0.25	0.69	2.10E-03
	OptimSmoothing	**16.33**	43.94	**136.99**	80.7	**0.32**	**0.79**	9.27E-03
	NN-Smoothing	9.14	43.58	158.97	79.79	0.16	**0.79**	4.67E-04
	GMSNet	13.99	43.81	151.39	79.81	0.22	**0.79**	6.39E-04
Circle	Origin	0.29	30.38	179.05	94.60	0.01	0.59	-
	LaplacianSmoothing	1.43	42.86	174.53	**81.19**	0.03	**0.78**	1.31E-04
	AngleSmoothing	1.95	39.47	173.97	84.23	0.04	0.73	5.95E-04
	CVTSmoothing	3.19	39.98	172.98	85.59	0.05	0.73	1.20E-03
	GETMeSmoothing	1.17	36.83	172.90	88.08	0.03	0.69	2.18E-03
	OptimSmoothing	**6.52**	**43.31**	**154.66**	81.9	**0.15**	**0.78**	8.78E-03
	NN-Smoothing	1.26	41.1	176.04	83.86	0.03	0.75	4.83E-04
	GMSNet	2.2	43	169.21	81.29	0.05	**0.78**	6.58E-04
Airfoil	Origin	0.25	53.25	178.91	67.84	0.01	0.92	-
	LaplacianSmoothing	26.36	**54.91**	111.02	65.78	0.51	**0.94**	1.34E-04
	AngleSmoothing	20.26	53.49	118.76	66.64	0.42	0.93	6.11E-04
	CVTSmoothing	25.49	52.08	115.79	68.19	0.48	0.91	1.22E-03
	GETMeSmoothing	**36.02**	52.90	**95.78**	67.89	**0.65**	0.92	2.24E-03
	OptimSmoothing	30.53	54.9	108.85	**65.72**	0.54	**0.94**	8.40E-03
	NN-Smoothing	27.69	54.06	113.06	66.57	0.51	0.93	4.88E-04
	GMSNet	27.17	54.5	110.47	66.08	0.52	**0.94**	6.71E-04
Pipe	Origin	4.10	46.28	170.48	76.81	0.07	0.82	-
	LaplacianSmoothing	27.91	**56.55**	112.45	**63.93**	0.52	**0.96**	1.30E-04
	AngleSmoothing	24.28	54.62	101.93	65.69	0.53	0.94	5.81E-04
	CVTSmoothing	27.78	54.22	97.8	66.28	0.62	0.93	1.18E-03
	GETMeSmoothing	**33.76**	51.07	**93.51**	69.61	**0.65**	0.90	2.14E-03
	OptimSmoothing	32.39	56.41	106.07	64.14	0.57	**0.96**	9.01E-03
	NN-Smoothing	28.28	53.72	112.79	66.69	0.51	0.93	4.74E-04
	GMSNet	28.23	55.76	112.27	64.71	0.52	0.95	6.47E-04

CVT, Centroidal Voronoi tessellation.
The best results are highlighted in **bold**, while the second-best results are annotated with a gray background.

Unedited

sonably distributed node degrees, our approach produces very smooth meshes, as shown in Fig. 7g and 7h. Moreover, our algorithm ensures robustness for highly distorted meshes, as demonstrated in Fig. 7e and 7f. As shown in Table 2, our proposed algorithm outperforms most heuristic mesh-smoothing methods, and the mesh element quality metrics closely approximate the results obtained using optimization-based algorithms. Since we adopted MetricLoss as the objective function for model optimization, our model has achieved the best or near-best performance in terms of the mean *1/Aspect ratio* compared to the other methods. When the mesh topology is reasonable, GETMeSmoothing exhibits a significant advantage in improving the minimum angle of the mesh. This is due to its use of a min-heap to determine the order of smoothing for mesh elements, prioritizing the processing of the worst elements each time. However, in cases where the worst elements cannot be effectively addressed, its performance is not as strong as node-based methods, as observed in square and circle meshes. Although GMSNet does not make any assumptions about the order of smoothing nodes, it also performs well on highly distorted meshes.

For all test cases, our model generally outper-



(a) Square mesh before smoothing **(b)** Square mesh smoothed by NN-Smoothing **(c)** Square mesh smoothed by GMSNet

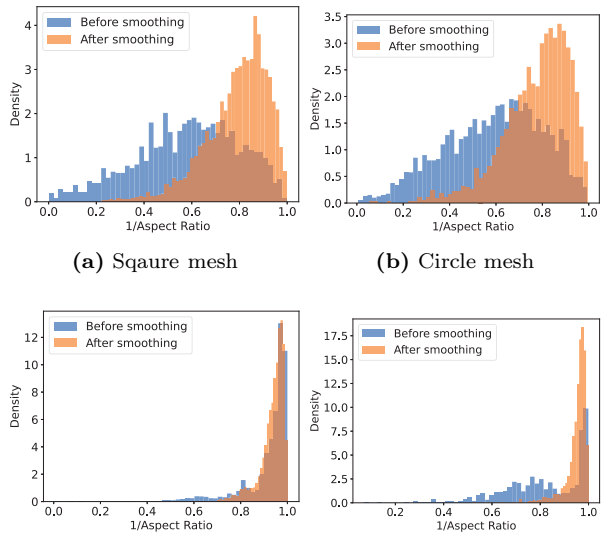
(d) Pipe mesh before smoothing **(e)** Pipe mesh smoothed by NN-Smoothing **(f)** Pipe mesh smoothed by GMSNet

Unedited

Fig. 8 Comparison between GMSNet and NN-Smoothing. The same color scheme as Fig. 8 has been applied here. The meshes before smoothing are shown in (a) and (d). The meshes after smoothing with NNSmoothing and GMSNet are shown in (b), (e), and (c), (f), respectively. Both intelligent smoothing methods significantly enhance mesh quality, yielding substantial improvements.

forms the NN-Smoothing model, despite the fact that we trained only one model to smooth nodes of different degrees, whereas NN-Smoothing requires training seven models. Our model has only 5% of the parameters compared to the NN-Smoothing model. The comparison of the smoothing performance of the two models is illustrated in Fig. 8. It can be observed that the smoothing performances of both models are nearly identical. For the mesh with a poorer topology, the square mesh, both models significantly improve the orthogonality of the mesh elements. Additionally, GMSNet provides a slightly better distribution of mesh density. From an efficiency standpoint, our proposed method demonstrates a remarkable average speedup of 13.56 times over optimization-based approaches, as evidenced by a direct comparison of the average time required to smooth each node using both algorithms. Compared to the simple MLP in NN-Smoothing, our model incorporates additional network modules, such as node encoders, graph convolutional layers, regularization layers, and more. These modules inevitably introduce additional computational overhead compared to NN-Smoothing. However, we believe that by adjusting hyperparameters (Liaw et al., 2018), optimizing network modules, and performing serialization operations (Lopes, 2023), our model should be able to achieve higher performance. We leave this for future work in the engineering of this method.

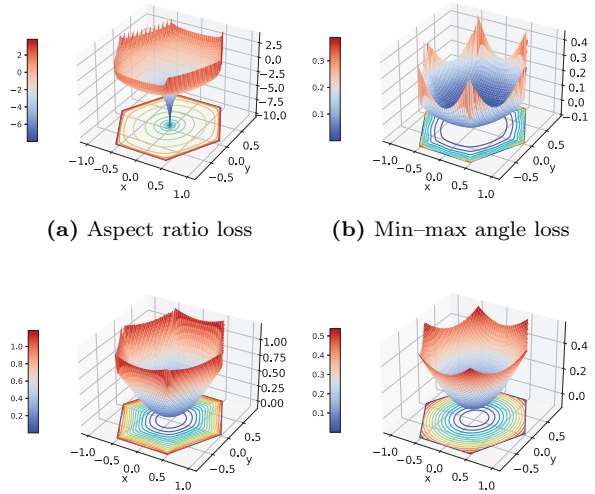
We also validate the effectiveness of our pro-



(a) Sqaure mesh (b) Circle mesh

(c) Airfoil mesh (d) Pipe mesh

Fig. 9 The quality distribution of mesh elements before and after GMSNet smoothing. In each subplot, we show the distribution of mesh element quality before and after smoothing. GMSNet significantly increased the density of high-quality mesh elements while reducing the density of low-quality mesh elements.



(a) Aspect ratio loss (b) Min–max angle loss

(c) Cosine loss (d) The proposed loss

Fig. 10 Plots of different loss functions. The range and smoothness of the loss function influence the model's convergence. In (a), the z-coordinate is given as $log(z)$. From (a) and (b), it can be seen that the loss functions based on aspect ratio and min–max angle have a large range of values or drastic changes in values, which are unfavorable for optimization. In contrast, the cosine loss and the proposed loss in (c) and (d) exhibit smooth changes, which are beneficial for model training.

posed algorithm by examining the distribution of the mesh element quality. The quality distributions of the mesh elements before and after smoothing are shown in Fig. 9. It can be observed that the algorithm significantly increases the proportion of high-quality mesh elements and improves the overall mesh quality. In summary, the experimental results demonstrate the effectiveness and efficiency of our proposed model for mesh-smoothing tasks.

4.3 The influence of different loss functions

In Section 3.2.3, we designed a loss function for model training. In this section, we discuss the effectiveness of MetricLoss and how to choose the appropriate loss function. We compare the performance of commonly used mesh quality metrics as loss functions during model training, including:

- Min–max angle loss:
  $\mathcal{L}(\boldsymbol{x}, \mathbf{S}(\boldsymbol{x})) = \frac{1}{|\mathbf{S}(\boldsymbol{x})|}\sum_{i=1}^{|\mathbf{S}(\boldsymbol{x})|} \min\max(\theta_{ij})$, for $j = 1, 2, 3$, where θ_{ij} is the angle in triangle T_i.

- Aspect ratio loss:
  $\mathcal{L}(\boldsymbol{x}, \mathbf{S}(\boldsymbol{x})) = \frac{1}{|\mathbf{S}(\boldsymbol{x})|}\sum_{i=1}^{|\mathbf{S}(\boldsymbol{x})|} \frac{m_i^2+n_i^2+l_i^2}{4\sqrt{3}S_i}$, where the symbols are defined in Section 3.2.3.

- Cosine loss:
  $\mathcal{L}(\boldsymbol{x}, \mathbf{S}(\boldsymbol{x})) = \frac{1}{3|\mathbf{S}(\boldsymbol{x})|}\sum_{i=1}^{|\mathbf{S}(\boldsymbol{x})|} (\cos(\theta_{ij}) - \frac{1}{2})^2$

- MetricLoss:
  $\mathcal{L}(\boldsymbol{x}, \mathbf{S}(\boldsymbol{x})) = \frac{1}{|\mathbf{S}(\boldsymbol{x})|}\sum_{i=1}^{|\mathbf{S}(\boldsymbol{x})|} (1 - \frac{4\sqrt{3}S_i}{m_i^2+n_i^2+l_i^2})$

The plots of the aforementioned loss functions with respect to the variations in the central node of the *StarPolygon* are shown in Fig. 10. It can be observed that the loss function we employed exhibits smoother transformations as the input changes, which facilitates the optimization process. We can also see that the aspect ratio loss and min–max angle loss are not suitable to be employed as the loss functions. When encountering highly distorted elements, the former leads to numerical values approaching positive infinity, causing a sharp increase in loss and inducing training instability. The latter, on the other hand, results in abrupt changes in loss when facing highly distorted elements, while the loss function remains relatively flat when the central node is situated in the neighborhood of the optimal position. This circumstance also hinders the optimization process. Although the original angle-based loss function is difficult to train, the loss function

Unedited

Table 3 Comparison of cosine loss and proposed loss on Pipe and Square meshes

Mesh	Algorithm	Min. Angle		Max. Angle		$\frac{1}{\text{Aspect ratio}}$	
		Min	Mean	Max	Mean	Min	Mean
Pipe	Cosine loss	26.77	55.71	110.43	64.67	0.54	0.95
	MetricLoss	26.37	55.56	107.73	64.82	0.56	0.95
Square	Cosine loss	11.04	43.76	146.87	79.61	0.25	0.79
	MetricLoss	13.53	44.06	150.46	79.65	0.22	0.79

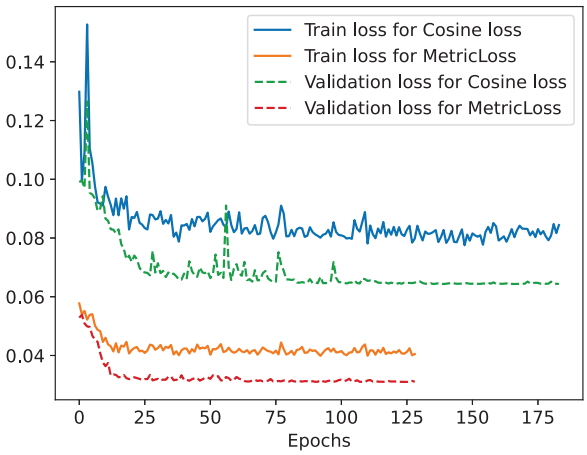


Fig. 11 Comparison of different loss functions. The loss functions based on min–max angle and aspect ratio encountered training failures during the initial epochs. In contrast, both MetricLoss and the transformed Cosine loss function yielded convergent results.

transformed using the cosine function is more stable, as shown in Fig. 10c. We trained the model using different loss functions, employing the same model configuration and training configuration. The train loss and validation loss of models with different loss functions are depicted in Fig. 11. It can be observed that MetricLoss and cosine loss function are suitable for training the model, as the model converges rapidly after several iterations. Moreover, MetricLoss is remarkably stable during the training process. The experiments between the two models based on different loss functions are shown in Table 3. It can be seen that the models show similar performance.

When training with the other two loss functions, the model either fails to converge or exhibits severe oscillations. This suggests that loss functions that use mesh quality metrics with a large range of numerical values or those that exhibit rapid changes in response to highly distorted elements can be detrimental to the model's training process. In summary, various loss functions can be used for mesh-smoothing model training. When constructing the loss function through mesh quality metrics, it is necessary to consider the behavior of the function when encountering distorted elements, and to avoid selecting loss functions with a larger range of numerical values or abrupt changes. Inappropriate mesh quality metrics can also be transformed and employed for model training, as demonstrated by the min–max angle loss and cosine loss.

5 Conclusion

In the present paper, we propose a GNN model, GMSNet, for intelligent mesh smoothing. The proposed model takes the neighbors of mesh nodes as input and learns to directly output the smoothed positions of the mesh nodes, avoiding the computational overhead associated with the optimization-based smoothing. A fault-tolerance mechanism, shift truncation, is also applied to prevent negative volume elements. With a lightweight design, GMSNet can be applied to mesh nodes with varying degrees, and is not affected by the data input order. We also introduce an novel loss function, MetricLoss, based on mesh quality metrics, eliminating the need for high-quality mesh generation cost to train the model. Experiments on two-dimensional triangle meshes demonstrate that our proposed model achieves outstanding smoothing performance with an average acceleration of 13.56 times compared to optimization-based smoothing. The results also show that GMSNet achieves superior performance than the NN-Smoothing model with just 5% model parameters. We also illustrate that MetricLoss achieves fast and stable model training through experiments.

Future work includes applying our proposed method to other types of mesh elements and extending it to surface and volume meshes. Introducing edge flipping and mesh density modification into the mesh-smoothing process to achieve superior mesh-

Unedited

smoothing effects is also an approach worth exploring. Improving the model's efficiency to enable its application on large-scale meshes needs careful consideration.

Contributors

Zhichao Wang and Xinhai Chen designed the research. Zhichao Wang processed the data and drafted the paper. Xinhai Chen, Junjun Yan, and Jie Liu helped organize the paper. Zhichao Wang and Xinhai Chen revised and finalized the paper.

Conflict of interest

All the authors declare that they have no conflict of interest.

Data availability

The data used in this study are available from the corresponding author upon reasonable request.

References

Baker TJ, 2005. Mesh generation: art or science? *Prog Aerosp Sci*, 41(1):29-63. https://doi.org/10.1016/j.paerosci.2005.02.002

Bohn J, Feischl M, 2021. Recurrent neural networks as optimal mesh refinement strategies. *Comput Math Appl*, 97:61-76. https://doi.org/10.1016/j.camwa.2021.05.018

Bridgeman J, Jefferson B, Parsons SA, 2010. The development and application of CFD models for water treatment flocculators. *Adv Eng Softw*, 41(1):99-109. https://doi.org/10.1016/j.advengsoft.2008.12.007

Cai TL, Luo SJ, Xu KYL, et al., 2021. GraphNorm: a principled approach to accelerating graph neural network training. Proc 38th Int Conf on Machine Learning, p.1204-1215.

Canann SA, Tristano JR, Staten ML, 1998. An approach to combined laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes. Proc 7th Int Meshing Roundtable, p.479-494.

Chen L, 2004. Mesh smoothing schemes based on optimal Delaunay triangulations. Proc 13th Int Meshing Roundtable, p.109-120.

Chen XH, Liu J, Pang YF, et al., 2020. Developing a new mesh quality evaluation method based on convolutional neural network. *Eng Appl Comput Fluid Mech*, 14(1):391-400. https://doi.org/10.1080/19942060.2020.1720820

Chen XH, Liu J, Gong CY, et al., 2021. MVE-Net: an automatic 3-D structured mesh validity evaluation framework using deep neural networks. *Comput Aided Des*, 141:103104. https://doi.org/10.1016/j.cad.2021.103104

Chen XH, Li TJ, Wan Q, et al., 2022. MGNet: a novel differential mesh generation method based on unsupervised neural networks. *Eng Comput*, 38(5):4409-4421. https://doi.org/10.1007/s00366-022-01632-7

Chowdhary KR, 2020. Natural language processing. Springer, New Delhi, p.603-649. https://doi.org/10.1007/978-81-322-3972-7_19

Damjanović D, Kozak D, Živić M, et al., 2011. CFD analysis of concept car in order to improve aerodynamics. Int Scientific and Expert Conf TEAM, 1(2):108-115.

Daroya R, Atienza R, Cajote R, 2020. REIN: flexible mesh generation from point clouds. Proc IEEE/CVF Conf on Computer Vision and Pattern Recognition Workshops, p.1444-1453. https://doi.org/10.1109/CVPRW50498.2020.00184

Du Q, Gunzburger M, 2002. Grid generation and optimization based on centroidal voronoi tessellations. *Appl Math Comput*, 133(2-3):591-607. https://doi.org/10.1016/S0096-3003(01)00260-0

Du Q, Wang DS, 2003. Tetrahedral mesh generation and optimization based on centroidal voronoi tessellations. *Int J Numer Methods Eng*, 56(9):1355-1373. https://doi.org/10.1002/nme.616

Du Q, Faber V, Gunzburger M, 1999. Centroidal voronoi tessellations: applications and algorithms. *SIAM Rev*, 41(4):637-676. https://doi.org/10.1137/S0036144599352836

Durand R, Pantoja-Rosero BG, Oliveira V, 2019. A general mesh smoothing method for finite elements. *Finite Elem Anal Des*, 158:17-30. https://doi.org/10.1016/j.finel.2019.01.010

Escobar JM, Montenegro R, Montero G, et al., 2005. Smoothing and local refinement techniques for improving tetrahedral mesh quality. *Comput Struct*, 83(28-30):2423-2430. https://doi.org/10.1016/j.compstruc.2005.03.022

Fidkowski KJ, Chen GD, 2021. Metric-based, goal-oriented mesh adaptation using machine learning. *J Comput Phys*, 426:109957. https://doi.org/10.1016/j.jcp.2020.109957

Field DA, 1988. Laplacian smoothing and Delaunay triangulations. *Commun Appl Numer Methods*, 4(6):709-712. https://doi.org/10.1002/cnm.1630040603

Freitag LA, Knupp PM, 2002. Tetrahedral mesh improvement via optimization of the element condition number. *Int J Numer Methods Eng*, 53(6):1377-1391. https://doi.org/10.1002/nme.341

Freitag LA, Ollivier-Gooch C, 1997. Tetrahedral mesh improvement using swapping and smoothing. *Int J Numer Methods Eng*, 40(21):3979-4002. https://doi.org/10.1002/(SICI)1097-0207(19971115)40:21<3979::AID-NME251>3.0.CO;2-9

Guo YF, Hai YQ, 2021. Adaptive surface mesh remeshing based on a sphere packing method and a node insertion/deletion method. *Appl Math Model*, 98:1-13. https://doi.org/10.1016/j.apm.2021.05.003

Guo YF, Wang L, Zhao K, et al., 2020. An angle-based smoothing method for triangular and tetrahedral meshes. Proc 15th Chinese Conf, p.224-234. https://doi.org/10.1007/978-981-33-6033-4_17

Guo YF, Wang CR, Ma Z, et al., 2022. A new mesh smoothing method based on a neural network. *Comput Mech*, 69(2):425-438. https://doi.org/10.1007/s00466-021-02097-z

Unedited

Hai YQ, Guo YF, Cheng SY, et al., 2021. Regular position-oriented method for mesh smoothing. *Acta Mech Solida Sin*, 34(3):437-448.
https://doi.org/10.1007/s10338-020-00201-z

Han X, Gao H, Pffaf T, et al., 2022. Predicting physics in mesh-reduced space with temporal attention. Proc Int Conf on Learning Representations, 39.

Herrmann LR, 1976. Laplacian-isoparametric grid generation scheme. *J Eng Mech Div*, 102(5):749-756.
https://doi.org/10.1061/JMCEA3.0002158

Kingma DP, Ba J, 2015. Adam: a method for stochastic optimization.
https://doi.org/10.48550/arXiv.1412.6980

Kipf TN, Welling M, 2016. Semi-supervised classification with graph convolutional networks.
https://doi.org/10.48550/arXiv.1609.02907

Klingner BM, Shewchuk JR, 2008. Aggressive tetrahedral mesh improvement. Proc 16th Int Meshing Roundtable, Berlin, Germany, p.3-23.
https://doi.org/10.1007/978-3-540-75103-8_1

Knupp PM, 2001. Algebraic mesh quality metrics. *SIAM J Sci Comput*, 23(1):193-218.
https://doi.org/10.1137/S1064827500371499

LeCun Y, Bengio Y, Hinton G, 2015. Deep learning. *Nature*, 521(7553):436-444.
https://doi.org/10.1038/nature14539

Lee DT, Schachter BJ, 1980. Two algorithms for constructing a Delaunay triangulation. *Int J Comput Inf Sci*, 9(3):219-242.
https://doi.org/10.1007/BF00977785

Li GH, MÃijller M, Thabet A, et al., 2019. DeepGCNs: can GCNs go as deep as CNNs? Proc IEEE/CVF Int Conf on Computer Vision, p.9266-9275.
https://doi.org/10.1109/ICCV.2019.00936

Liaw R, Liang E, Nishihara R, et al., 2018. Tune: a research platform for distributed model selection and training.
https://doi.org/10.48550/arXiv.1807.05118

Lino M, Cantwell C, Bharath AA, et al., 2021. Simulating continuum mechanics with multi-scale graph neural networks.
https://doi.org/10.48550/arXiv.2106.04900

Liu Y, Wang WP, LÃľvy B, et al., 2009. On centroidal voronoi tessellationâĂŤenergy smoothness and fast computation. *ACM Trans Graphics*, 28(4):101.
https://doi.org/10.1145/1559755.1559758

Lloyd S, 1982. Least squares quantization in PCM. *IEEE Trans Inf Theory*, 28(2):129-137.
https://doi.org/10.1109/TIT.1982.1056489

Lopes NP, 2023. Torchy: a tracing JIT compiler for PyTorch. Proc 32nd ACM SIGPLAN Int Conf on Compiler Construction, p.98-109.
https://doi.org/10.1145/3578360.3580266

Moukalled F, Mangani L, Darwish M, 2016. The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab®. Springer, Cham, Germany, p.110-128.
https://doi.org/10.1007/978-3-319-16874-6

Pak M, Kim S, 2017. A review of deep learning in image recognition. Proc 4th Int Conf on Computer Applications and Information Processing Technology, p.1-3.
https://doi.org/10.1109/CAIPT.2017.8320684

Pan W, Lu XQ, Gong YH, et al., 2020. HLO: half-kernel laplacian operator for surface smoothing. *Comput Aided Des*, 121:102807.
https://doi.org/10.1016/j.cad.2019.102807

Papagiannopoulos A, Clausen P, Avellan F, 2021. How to teach neural networks to mesh: application on 2-D simplicial contours. *Neural Netw*, 136:152-179.
https://doi.org/10.1016/j.neunet.2020.12.019

Parthasarathy VN, Kodiyalam S, 1991. A constrained optimization approach to finite element mesh smoothing. *Finite Elem Anal Des*, 9(4):309-320.
https://doi.org/10.1016/0168-874X(91)90004-I

PaszyÅĎski M, Grzeszczuk R, Pardo D, et al., 2021. Deep learning driven self-adaptive Hp finite element method. Proc 21st Int Conf on Computational Science, p.114-121.
https://doi.org/10.1007/978-3-030-77961-0_11

Peng WH, Yuan ZL, Wang JC, 2022. Attention-enhanced neural network models for turbulence simulation. *Phys Fluids*, 34(2):025111.
https://doi.org/10.1063/5.0079302

Pfaff T, Fortunato M, Sanchez-Gonzalez A, et al., 2021. Learning mesh-based simulation with graph networks.
https://doi.org/10.48550/arXiv.2010.03409

Prasad T, 2018. A comparative study of mesh smoothing methods with flipping in 2D and 3D. MS Thesis, The State University of New Jersey, New Jersey, USA.

Reddy DR, 1976. Speech recognition by machine: a review. *Proc IEEE*, 64(4):501-531.
https://doi.org/10.1109/PROC.1976.10158

Samstag RW, Ducoste JJ, Griborio A, et al., 2016. CFD for wastewater treatment: an overview. *Water Sci Technol*, 74(3):549-563.
https://doi.org/10.2166/wst.2016.249

Sharp N, Crane K, 2020. A laplacian for nonmanifold triangle meshes. *Comput Graphics Forum*, 39(5):69-80.
https://doi.org/10.1111/cgf.14069

Song WB, Zhang MR, Wallwork JG, et al., 2022. M2N: mesh movement networks for pde solvers. Proc 36th Conf on Neural Information Processing Systems, p.7199-7210.

Spalart PR, Venkatakrishnan V, 2016. On the role and challenges of CFD in the aerospace industry. *Aeronautical J*, 120(1223):209-232.
https://doi.org/10.1017/aer.2015.10

Ulyanov D, Vedaldi A, Lempitsky V, 2017. Instance normalization: the missing ingredient for fast stylization.
https://doi.org/10.48550/arXiv.1607.08022

Vartziotis D, Papadrakakis M, 2013. Improved GETMe by adaptive mesh smoothing. *Comput Assisted Methods Eng Sci*, 20(1):55-71.

Vartziotis D, Wipper J, 2019. The GETMe Mesh Smoothing Framework: A Geometric way to Quality Finite Element Meshes. Taylor & Francis Group, Boca Raton, USA.

Vartziotis D, Athanasiadis T, Goudas I, et al., 2008. Mesh smoothing using the Geometric Element Transformation Method. *Comput Methods Appl Mech Eng*, 197(45-48):3760-3767.
https://doi.org/10.1016/j.cma.2008.02.028

Vollmer J, Mencl R, MÃijller H, 1999. Improved laplacian smoothing of noisy surface meshes. *Comput Graphics Forum*, 18(3):131-138.
https://doi.org/10.1111/1467-8659.00334

Unedited

Wallwork JG, Lu JY, Zhang MR, et al., 2022. E2N: error estimation networks for goal-oriented mesh adaptation. https://doi.org/10.48550/arXiv.2207.11233

Wang ZH, Chen XH, Li TJ, et al., 2022. Evaluating mesh quality with graph neural networks. *Eng Comput*, 38(5):4663-4673. https://doi.org/10.1007/s00366-022-01720-8

Wu TF, Liu XJ, An W, et al., 2022. A mesh optimization method using machine learning technique and variational mesh adaptation. *Chin J Aeronaut*, 35(3):27-41. https://doi.org/10.1016/j.cja.2021.05.018

Wu ZH, Pan SR, Chen FW, et al., 2021. A comprehensive survey on graph neural networks. *IEEE Trans Neural Netw Learn Syst*, 32(1):4-24. https://doi.org/10.1109/TNNLS.2020.2978386

Xu KJ, Gao XF, Chen GN, 2018. Hexahedral mesh quality improvement via edge-angle optimization. *Comput Graphics*, 70:17-27. https://doi.org/10.1016/j.cag.2017.07.002

Zhang YJ, Bajaj C, Xu GL, 2009. Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. *Commun Numer Methods Eng*, 25(1):1-18. https://doi.org/10.1002/cnm.1067

Zhang ZY, Wang YX, Jimack PK, et al., 2020. MeshingNet: a new mesh generation method based on deep learning. Proc 20th Int Conf on Computational Science, p.186-198. https://doi.org/10.1007/978-3-030-50420-5_14

Zhang ZY, Jimack PK, Wang H, 2021. MeshingNet3D: efficient generation of adapted tetrahedral meshes for computational mechanics. *Adv Eng Softw*, 157-158:103021. https://doi.org/10.1016/j.advengsoft.2021.103021

Zhou T, Shimada K, 2000. An angle-based approach to two-dimensional mesh smoothing. Proc 7th Int Meshing Roundtable, p.373-384.

Unedited

A Appendix

A.1 Examples of the mesh dataset

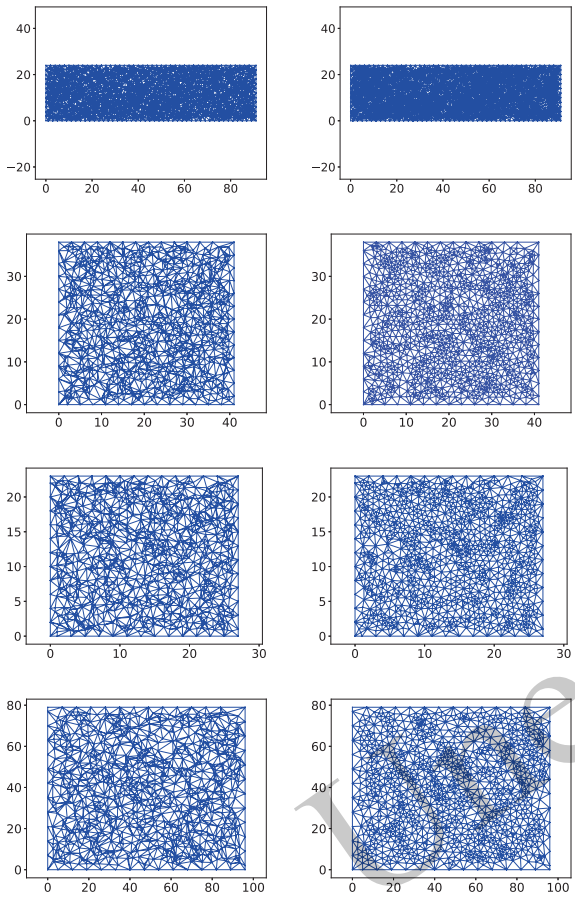


Fig. 12 Examples of meshes in the dataset. The left column represents the model input for GMSNet and NN-Smoothing training, while the column on the right represents the labeled mesh data to train the NN-Smoothing model.

Here, we present four samples from the datasets used to train the NN-Smoothing and GMSNet models. As depicted in Fig. 12, the meshes are only different in mesh size and element density.

Unedited