Artificial Intelligence for Engineering Design, Analysis and Manufacturing

www.cambridge.org/aie

Sampling balanced high-quality data to train an automatic mesh generator

Jie Pan[1], Jingwei Huang[2], Gengdong Cheng[3] and Yong Zeng[1]

[1]Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Canada; [2]Department of Engineering Management & Systems Engineering, Old Dominion University, Norfolk, VA, USA and [3]Department of Engineering Mechanics, Dalian University of Technology, Dalian, China

Research Article

Cite this article: Pan J, Huang J, Cheng G and Zeng Y (2025). Sampling balanced high-quality data to train an automatic mesh generator. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **39**, e30, 1–12
https://doi.org/10.1017/S089006042510019X

Received: 22 March 2024
Revised: 18 June 2025
Accepted: 30 August 2025

Keywords:
data generation; data synthesis; mesh generation; optimal performance; algorithm design

Corresponding author:
Yong Zeng;
Email: yong.zeng@concordia.ca

Abstract

In real-world scenarios, high-quality data are often scarce and imbalanced, yet it is essential for the optimal performance of data-driven algorithmic models. Data synthesis methods are commonly used to address this issue; however, they typically rely heavily on the original dataset, which limits their ability to significantly improve performance. This article presents a quality function-based method for directly generating high-quality data and applies it to a mesh generation algorithm to demonstrate its efficiency and effectiveness. The proposed approach samples input–output pairs of the algorithm based on their feature spaces, selects high-quality samples using a defined quality function that evaluates the suitability of outputs for their corresponding inputs, and trains a feedforward neural network to learn the mapping relationship using the selected data. Experimental results show that the learning cost is significantly reduced while maintaining competitive performance compared to two representative meshing algorithms.

Highlights

- A direct sampling method is proposed to generate balanced high-quality training data for mesh generation.
- This significantly reduces the learning cost of an automatic mesh generator.
- A full quadrilateral mesh is generated without human intervention and extra clean-ups.
- The meshing performance is competitive with representative commercial software in several aspects.

Introduction

High-quality data serve as the basis for optimal model performance in most machine learning algorithms. In real-world scenarios, the data distribution is often imbalanced, with minority classes occurring less frequently and posing challenges in data collection. Nevertheless, these minor classes often carry greater importance in determining problem resolution. The performance of many classification, regression, and semi-supervised models is significantly compromised when dealing with imbalanced data (He and Garcia, 2009; Yang et al., 2021). Over the past two decades, two types of methods, namely data-level and algorithm-level approaches (Krawczyk, 2016; Tanaka and Tanaka and Aranha, 2019), have been proposed to address this data imbalance issue. While these methods have seen continuous improvement, they remain confined to the scope of the given dataset and typically offer only incremental gains. As a result, models still struggle with generalizing to unseen scenarios, particularly those involving rare classes.

To overcome these limitations, this article introduces a data generation method based on a quality function, aimed at directly producing high-quality, diverse samples beyond those present in the original dataset. We apply this method in the context of mesh generation to demonstrate its ability to improve both efficiency and performance.

Mesh generation is a critical area in computational geometry and underpins numerical simulations in finite element analysis (FEA), computational fluid dynamics (CFD), and graphic model rendering (Gordon and Hall, 1973; Roca and Loseille, 2019). Its primary goal is to discretize complex geometries into a finite set of geometrically simple and bounded elements – such as triangles or quadrilaterals in two dimensions (2D), or tetrahedra or hexahedra in three dimensions (3D). However, existing mesh generation algorithms often struggle to consistently produce high-quality meshes, especially when dealing with complex geometries (Slotnick et al., 2014). These methods typically rely on heuristic rules and demand substantial manual effort in preprocessing (e.g., decomposing intricate domains) and postprocessing (e.g., correcting poor-quality elements). This process is time-consuming, often taking days or even weeks for highly complex 3D domains.

Moreover, traditional methods depend heavily on domain-specific knowledge and rigid rule-based formulations, making them difficult to generalize. Designing robust algorithms capable of handling arbitrary and complex geometries remains a major challenge. The increasing complexity of required heuristics also leads to higher computational costs and slower generation speeds. These limitations hinder their ability to meet the growing demands for accuracy and efficiency in large-scale numerical simulations using CFD and FEA. As emphasized in the National Aeronautics and Space Administration's CFD Vision 2030 study, current meshing technologies are a primary bottleneck in advancing CFD workflows (Slotnick et al., 2014).

To address these challenges, many researchers have begun integrating mesh generation with artificial intelligence (AI) techniques, including expert systems (Zeng and Cheng, 1993) and neural networks (NNs; Yao et al., 2005; Vinyals et al., 2015; Zhang et al., 2020; Papagiannopoulos et al., 2021). While these approaches show promise, AI-based methods are not yet mature enough to fully replace standard mesh generation algorithms. Several limitations remain: (1) the complex architectures of NNs make it difficult to ensure robust and generalizable mesh generators; (2) rigid problem formulations often reduce adaptability when applied to diverse and irregular geometric domains; and (3) training datasets frequently suffer from low-quality and imbalanced distributions, which hinder learning effectiveness and reduce model reliability.

The first two issues can be mitigated by reformulating the mesh generation task as a sequential decision-making process (Pan et al., 2021, 2023). In this setup, mesh elements are constructed iteratively from the geometry's boundary until the entire domain is filled. Since each iteration requires only a partial boundary, this formulation enables the use of simple NN structures and offers strong adaptability to complex geometries. While reinforcement learning (RL) can, in principle, learn generation rules through trial and error using partial boundaries, it often struggles to explore rare or extreme boundary configurations, such as sharp corners or narrow regions, due to their low occurrence in training data. As a result, RL-based models require extensive training time to generalize across a wide variety of geometric shapes. In this study, we address the challenge of low-quality and imbalanced training datasets. We propose a quality function-based data generation method that directly synthesizes high-quality training samples containing diverse boundary situations. The enhanced dataset enables the training of a simple yet effective feedforward NN (FNN), resulting in a robust and efficient mesh generator, FreeMesh-DG.

The main contributions of this study are summarized as follows:

1. A quality function-based method is proposed to address the imbalanced data problem in mesh generation, significantly improving model performance.
2. The resulting mesh generator can fully mesh geometric domains without human intervention or additional post-processing operations.
3. The model achieves the best performance in two key quantitative metrics – singularity and taper – when compared with two representative meshing algorithms.

The remainder of the article is organized as follows. Section title "*Problem formulation and fundamentals*" reviews related work on imbalanced learning and mesh generation. Section title "*Quality function-based data generation for mesh generation*" details the implementation of the proposed quality function-based data generation method for mesh generation. Section title "*Experiment results*" evaluates the performance of the proposed approach and compares it with other state-of-the-art meshing algorithms. Section title "*Discussion*" discusses the key improvements introduced by this method and its relevance to both the mesh generation and machine learning communities. Finally, Section title "*Conclusion*" concludes the article and outlines potential future research directions.

Problem formulation and fundamentals

This section discusses the existing data generation methods for imbalanced learning and explains the formulation of the mesh generation problem and techniques.

Data generation

In real-world environments, data are often non-uniformly distributed due to rare occurrence and difficulty of collecting high-quality samples. The quantity and quality of data fundamentally determine the performance of trained data-driven machine learning models. A large dataset is crucial for establishing complex decision boundaries in classification problems and for avoiding overfitting – particularly in deep learning methods that involve numerous parameters (LeCun et al., 2015). Imbalanced datasets tend to bias the predictions of algorithms toward majority classes (He and Garcia, 2009), even though minority classes are often more important and informative. To address this issue, data augmentation and synthesis techniques are employed to increase the dataset size by either slightly modifying existing samples or generating new, artificial data derived from the original dataset (He and Garcia, 2009; Krawczyk, 2016).

Conventional methods for handling imbalanced datasets primarily focus on modifying existing datasets to achieve a more balanced distribution. Two widely used strategies include: (1) removing examples from the majority class (undersampling) and (2) generating new examples for the minority class (oversampling). The basic form of random undersampling selects and removes samples from the majority class without replacement. To address the potential information loss associated with this approach, Liu et al. (2008) proposed two informed undersampling techniques. Similarly, Yen and Lee (2009) introduced a cluster-based undersampling method to mitigate the disjunct problem.

On the oversampling side, random oversampling replicates existing minority class samples to increase their representation. However, this method often leads to overfitting. To overcome this, synthetic sampling techniques have been developed. A well-known method is Synthetic Minority Oversampling Technique (SMOTE), proposed by Chawla et al. (2002), which generates synthetic examples based on feature-space similarities between minority class samples. Building upon SMOTE's success, several enhancements have been introduced, including borderline-SMOTE (Han et al., 2005), safe-level-SMOTE (Bunkhumpornpat et al., 2009), and adaptive synthetic sampling (He et al., 2008).

To further address within-class imbalance, Jo and Japkowicz (2004) proposed cluster-based oversampling, aimed at resolving the small disjunct problem. Other researchers have explored leveraging structural information within the data to generate more representative synthetic samples. For instance, Xie et al. (2015) employed clustering techniques to model data density and generate samples accordingly. Liu and Hsieh (2019) developed a model-based synthetic sampling method that enhances data diversity by capturing inter-feature relationships through regression models.

https://doi.org/10.1017/S089006042510019X Published online by Cambridge University Press

Markov chain models have also been applied to synthesize time-varying stochastic data, such as wind speed (Shamshad et al., 2005) and vehicle velocity in driving cycles (Lee and Filipi, 2010), by constructing transition matrices. More recently, Yang and Nam (2022) proposed a covariance matrix-based method that incorporates random noise and feature correlations from the original dataset to generate synthetic data.

Numerous NN-based data synthesis models have been developed in recent years. Habibie et al. (2017) introduced a generative model for human motion data using a variational autoencoder. Yang et al. (2021) employed deep NNs to represent the feature space of data samples. Among various approaches, the generative adversarial network (GAN) has gained popularity due to its flexibility and efficiency in synthesizing data from high-dimensional datasets (Tanaka and Aranha, 2019). Wang et al. (2018) combined GANs with autoencoders to generate synthetic vibration signals for gearboxes, while Xuan et al. (2018) explored the integration of convolutional NNs with GANs for pearl image generation. Despite their widespread application, GAN-based methods may produce low-quality data due to several challenges, including (1) instability during the training of the generator, (2) small size of the original dataset, and (3) high dimensionality and nonlinearity of the data (Tanaka and Aranha, 2019; Yang and Nam, 2022).

Data augmentation improves model performance by increasing the amount of valuable training data, enhancing variability, reducing overfitting, and alleviating data scarcity. It also helps to boost the generalization ability of models and reduces the cost of data collection and annotation. However, the performance gains are often limited. First, it is difficult to identify and represent the intrinsic characteristics of the data, which typically requires domain-specific knowledge. Second, representative features of unseen scenarios are often missing from the existing dataset. Simple transformations of available data rarely lead to fundamental improvements in model generalizability. Consequently, there remains a significant challenge in designing data generation methods that can truly capture the underlying nature of complex, high-dimensional, and nonlinear datasets.

Mesh generation

With the rapid advancement of high-performance computing hardware, mesh generation methods are increasingly expected to handle geometric domains of greater complexity and resolution, while maintaining reliability and efficiency. In response, many machine learning-based algorithms have been developed to generate high-quality meshes. For example, Nechaeva (2006) proposed an adaptive mesh generation algorithm based on self-organizing maps (SOM), an unsupervised NN method. This algorithm adapts a given uniform mesh to a target physical domain via mapping, aiming to address the limitations of SOM in handling inaccurate meshes near domain boundaries and in constructing meshes for nonconvex domains.

Vinyals et al. (2015) introduced a novel neural architecture known as Pointer Networks to tackle combinatorial problems using NNs. While not originally intended for mesh generation, the model could generate triangular meshes by outputting triplets of integers (each forming a triangle) that indicate the connectivity of input points. However, its application to meshing problems lacked robustness and completeness – the resulting mesh was often only partially covered with triangular elements and contained intersecting edges. Papagiannopoulos et al. (2021) proposed a triangular mesh generation method employing three separate NNs. These networks respectively predicted the number of candidate inner vertices, their coordinates, and their connectivity to existing boundary segments. Nevertheless, this approach struggled to generalize to arbitrary and complex geometric domains due to its fixed input size and complex architecture. Additionally, the meshing performance was heavily constrained by the quality and diversity of the training data used to construct the generator.

The mesh generation problem can be formulated as a sequential decision-making process (Pan et al., 2021). In this formulation, the geometric domain is discretized into quadrilateral elements, as shown in Figure 1. At each time step, an element Q_i (in red) is generated from the existing boundary (in blue), B_i, which consists of piecewise linear segments denoted by a sequence of vertices $[V_1, V_2, \ldots, V_{N_i}]$. After generating each element, the boundary is updated by removing the corresponding segment, and the updated boundary is then used to generate the next element. This iterative process continues until the remaining boundary vertices form the final mesh element. The completed mesh must satisfy specific geometrical and topological criteria to ensure quality and correctness (Zeng and Cheng, 1993; Zeng and Yao, 2009).

A key challenge in the sequential mesh generation framework lies in determining how to construct an element based on the partial boundary (Pan et al., 2021). Yao et al. (2005) addressed this by formally defining a reference vertex and its neighboring vertices as the input, with the corresponding rule type and vertex coordinates as the output. They manually created training data consisting of specific input–output pairs and used an artificial NN (ANN) to approximate the mapping relationship. Building on this

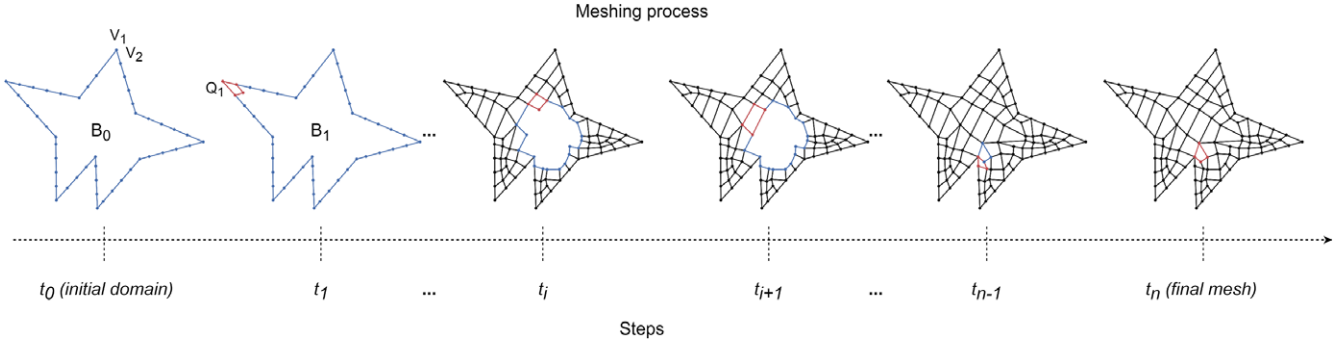


Figure 1. A sequence of decisions to complete the mesh. At each time step t_i, an element (in red) is extracted from the current boundary (in blue). The boundary is then updated by removing the element and serves as the meshing boundary in the next time step t_{i+1}. This process continues until the updated boundary becomes the final element.

https://doi.org/10.1017/S089006042510019X Published online by Cambridge University Press

foundation, Pan et al. (2021, 2023) refined the input definition by incorporating more contextual information around the reference vertex, such as the vertices located in the fan-shaped region formed by the reference point and its adjacent vertices. They also explored the use of FNNs and RLs to automatically generate high-quality training samples and enhance the accuracy of the learned mapping. While RL-based approaches have successfully learned effective meshing policies that adapt to complex geometries, they still face notable limitations: (1) the training process is time-consuming, and (2) the geometry used for training must be manually selected. These limitations stem from RL's reliance on extensive trial-and-error exploration to cover the wide range of geometric variations encountered in practice.

This article proposes a data generation method that provides comprehensive data, capturing all possible boundary scenarios, to train an efficient mesh generator to resolve this challenge. The method enables the training of an efficient mesh generator without relying on computationally intensive trial-and-error strategies. As a result, it significantly reduces the training burden and provides a more direct and reliable pathway for producing high-quality meshes.

Meshing problem formulation for data generation

This article proposes a method to directly generate high-quality, balanced training data for the meshing algorithm. The overall procedure is outlined as follows:

1. Formulate the control problem as a set of state–action (i.e., input–output) pairs;
2. Define the feature spaces for each dimension of the state and action;
3. Sample state–action pairs across their respective feature spaces;
4. Design a quality function to evaluate each state–action pair, typically using a simulation environment of the problem;
5. Specify quality criteria to filter and retain only qualified samples;
6. Evaluate and rebalance the collected samples to ensure a representative and uniformly distributed dataset.

The mesh generation problem has been formulated as a sequential decision-making process that consists of a set of state-action (i.e., input–output) pairs. The input is formally represented as follows,

$$x_t = \left\{ V_{l,n}, \ldots, V_{l,1}, V_{r,1}, \ldots, V_{r,n}, V_1, \ldots, V_g \right\}. \tag{1}$$

where $V_{l,i}$ and $V_{r,i}$ denote the i-th vertex at the left and right side of the reference vertex V_0 along the boundary; g neighboring points, $V_1, \ldots, V_g$ were the closest vertices to the reference vertex located in the corresponding fan-shaped area $\theta_1, \ldots, \theta_g$ with radius L_r, where $\theta_1 = \theta_2 = \ldots = \theta_g$. An example of the input is shown in Figure 2.

The output is formally defined as

$$y_t = [\text{type}_t, V_t], \tag{2}$$

where $\text{type}_t \in \{0,1,2,3\}$, which correspond to the four basic rules, respectively; V_t are the coordinates of the newly added vertex, as shown in Figure 3. To form an element based on a partial boundary, type 1 involves adding two extra edges; type 0 and 2 involve adding one extra edge, where the specific rule is decided by distance from the reference point (Pan et al., 2023). Type 3 refers to the addition of three edges to complete a square. However, type 3 does not need to be learned by the model as it can be hard-coded as a deterministic operation. In this study, we only consider three rule types, $\text{type}_t \in \{0,1,2\}$.

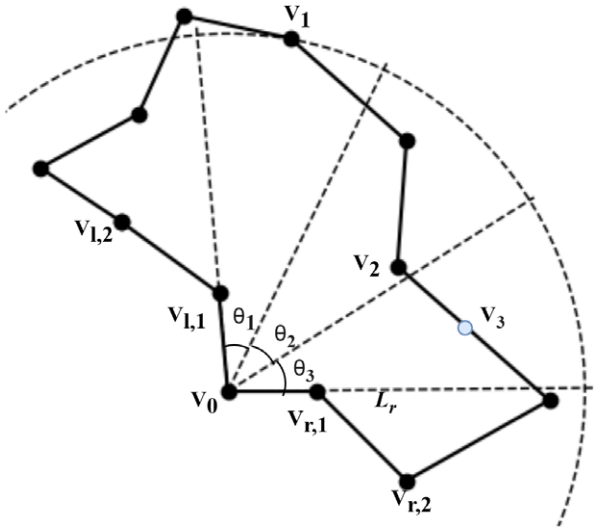


Figure 2. An example of the input with $L_r = 4, n = 2, g = 3$ (Pan et al., 2023).

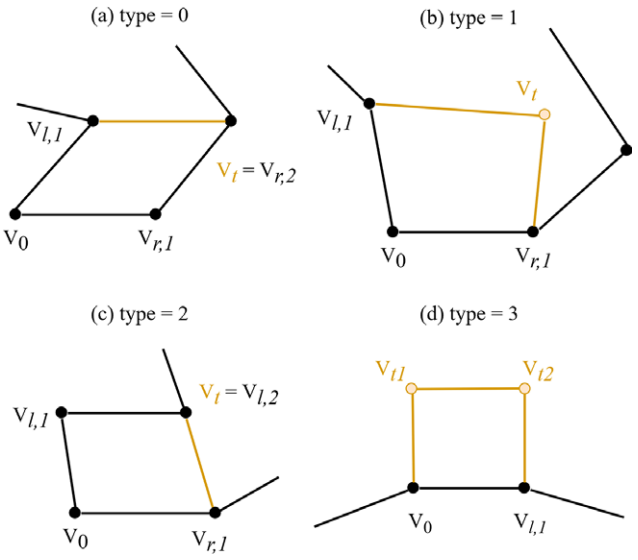


Figure 3. Four basic rule types to form a quadrilateral element. V_0 is the reference vertex. The newly generated vertices V_t and edges are marked in yellow. Type 1 involves adding two extra edges; type 0 and 2 involve adding one extra edge; and type 3 involves adding three edges.

With the input and output formulations established, the feature space for each dimension of the data can be readily determined. Various boundary situations can then be represented by uniformly sampling from these feature spaces. A quality function is employed to evaluate all possible actions for each situation, allowing the selection of appropriate actions based on predefined criteria.

Quality function-based data generation for mesh generation

This section presents the quality function-based data generation method for the mesh generation algorithm. An overview of the overall data generation procedure is first provided, followed by a detailed explanation of the key steps: defining the quality function,

https://doi.org/10.1017/S089006042510019X Published online by Cambridge University Press

balancing the sampled data, and training an FNN to develop the final mesh generation algorithm.

Data generation procedure

The overall data generation procedure consists of three main steps: (1) sampling input–output pairs based on the defined feature space of each dimension; (2) applying a quality threshold to filter out low-quality samples; and (3) balancing the remaining samples according to their types. The resulting dataset is then used to train an FNN model to learn the mapping between input and output. An overview of the data generation procedure for the mesh generation algorithm is shown in Figure 4.

The mesh generation problem has been formulated as a sequential decision-making task in the previous section. At each time step i, a partial boundary environment is taken as the input x_i (see Equation 1), and the corresponding output y_i consists of the rule type and coordinates of the newly generated vertex (see Equation 2). This process repeats iteratively until the last element is formed. As decisions are made, the partial boundary is dynamically updated, resulting in various situations. The meshing problem can be effectively addressed by identifying the optimal decision for each situation. The core idea of the proposed data generation method is to produce a sufficient number of high-quality solutions across diverse boundary situations. These solutions form the dataset $D = \{x_i, y_i\}_{i=1}^{N}$ to train an optimal decision policy.

The lower and upper bounds of each dimension in the input X is denoted as $X_{L,U} = ([x_{l,1}, x_{u,1}], \ldots, [x_{l,n}, x_{u,n}])$, where $x_{l,j}$ and $x_{u,j}$ represents the lower and upper bounds for jth input of X; n is the total number of the input dimensions. Similarly, the bounds for each dimension in the output Y are denoted as $Y_{L,U} = ([y_{l,1}, y_{u,1}], \ldots, [y_{l,n}, y_{u,n}])$, where n is the number of output dimensions. Using these bounds, a large number of input–output pairs D can be sampled from a uniform distribution to construct the dataset. The detailed sampling procedure is presented in Algorithm 1.

Algorithm 1 Sampling input–output pairs.
Require: input and output value bounds $X_{L,U}$ and $Y_{L,U}$, sample number N;
1: $D \leftarrow []$
2: **for** $k \leftarrow 1$ to N **do**
3: $D \leftarrow D \cup \{(x_k, y_k) | x_k \sim U(X_{L,U}), y_k \sim U(Y_{L,U})\}$ ▷ $U(*)$ is the uniform distribution.
4: **end for**

Quality function for performance measurement

To effectively guide learning in a sequential decision-making problem, the quality of each solution step must be both well-defined and measurable. It is not only the final result that matters, but also how each decision contributes toward achieving the overall objective. A quality function is thus introduced to evaluate the performance of each step in the decision sequence. After generating a large set of input–output samples, the quality function is applied to assess them.

To align with this objective, the quality function evaluates each sample through a two-stage process:

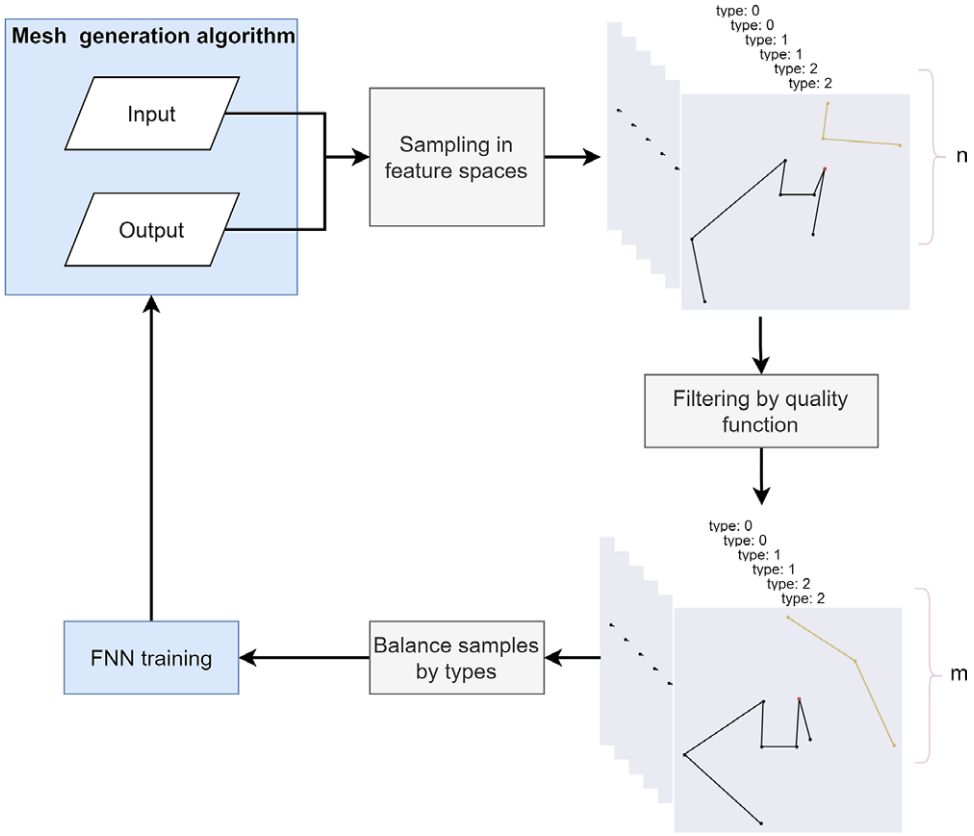


Figure 4. Quality function-based data generation procedure for mesh generation. The mesh generation algorithm consists of a set of input–output pairs. The points and edges in black are from the right and left sides of the reference vertex, and the ones in yellow are three neighboring points from the corresponding fan-shaped area.

https://doi.org/10.1017/S089006042510019X Published online by Cambridge University Press

1. Validity check: It verifies the structural integrity of the input and output. If the input vertices form self-intersecting segments or if the output leads to intersections with the existing boundary, the quality is set to 0.
2. Geometric quality evaluation: If the input and output are valid, the function then evaluates the quality of the newly formed quadrilateral element and the updated boundary.

As a result, the quality function $\eta(x_i, y_i)$ is defined as:

$$\eta(x_i,y_i) = \begin{cases} 0, & \text{invalid input and output;} \\ (1-\alpha)\eta_i^e + \alpha\eta_i^b, & \text{otherwise;} \end{cases} \tag{3}$$

where η_i^e denotes the quality of the generated element, evaluated based on the uniformity of its edges and internal angles; η_i^b represents the quality of the updated adjacent boundary, assessed by two factors: (1) the smoothness of the angle transitions between the new element and the remaining boundary, and (2) the minimum distance from the newly added vertex to its surrounding edges, which helps avoid overly sharp or narrow situations. The parameter α is a weight balancing the two components; it is experimentally set to 0.618 to prioritize boundary quality, as it has a stronger impact on ensuring overall mesh quality in the long run than the quality of individual elements.

The element quality η_t^e is measured by its edges and internal angles, and is calculated as follows:

$$\begin{aligned} \eta_t^e &= \sqrt{q^{\text{edge}} q^{\text{angle}}}, \\ q^{\text{edge}} &= \frac{\sqrt{2}\min_{j\in\{0,1,2,3\}}\{l_j\}}{D_{max}}, \\ q^{\text{angle}} &= \frac{\min_{j\in\{0,1,2,3\}}\{\text{angle}_j\}}{\max_{j\in\{0,1,2,3\}}\{\text{angle}_j\}}, \end{aligned} \tag{4}$$

where q^{edge} refers to the quality of edges of this element; l_j is the length of the jth edge of the element; D_{max} is the length of the longest diagonal of the tth element; q^{angle} refers to the quality of the angles of the element; and angle_j is the degree of the jth inner angle of the element. The quality η_t^e ranges from 0 to 1.

The quality of the updated adjacent boundary η_t^b is illustrated in Figure 5 and denoted as follows:

$$\begin{aligned} \eta_t^b &= \sqrt{\frac{\min_{k\in\{1,2\}}\{\min(\varsigma_k, M_{\text{angle}})\}}{M_{angle}} q^{\text{dist}}}, \\ q^{\text{dist}} &= \begin{cases} \dfrac{d_{\min}}{(d_1+d_2)/2}, & \text{if } d_{\min} < (d_1+d_2)/2; \\ 1, & \text{otherwise.} \end{cases} \end{aligned} \tag{5}$$

where ς_k refers to the degrees of the kth generated angle; d_{min} is the distance of V_t to its closest edge if type = 1; d_1 is the distance between the newly generated point and $V_{l,1}$; d_2 is the distance between the newly generated point and $V_{r,1}$; and M_{angle} is a threshold to judge if the newly generated angles are becoming sharp. The q^{dist} equals to 1 if type $\in \{0,2\}$. The quality η_t^b ranges from 0 to 1, with a larger value representing better quality. When the formed new angles are less than M_{angle} (e.g., $< 60°$), the boundary quality decreases. It penalizes the generation of sharp angles that harm the overall mesh quality and meshing completeness.

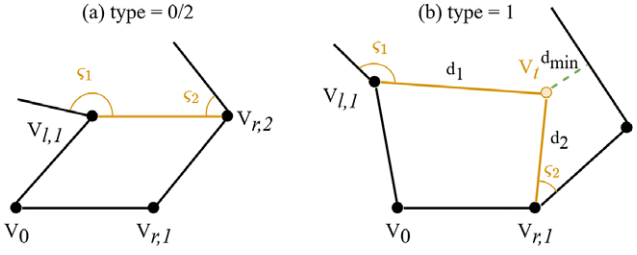


Figure 5. Updated adjacent boundary quality for different solution types. V_0 is the reference vertex.

These two quality measures are chosen to capture the trade-off between the quality of the current element and the impact on the remaining adjacent boundary. A detailed explanation of the element quality η_i^e and boundary quality η_i^b can be found in our previous work (Pan et al., 2023). Since each generated element incrementally reshapes the boundary, prioritizing perfect element quality in the current step can lead to unfavorable or even invalid boundary configurations in later steps. This trade-off is essential to ensure both high overall mesh quality and the completeness of the final mesh.

A large number of high-quality samples can be obtained by applying a quality threshold τ to the evaluation results. The detailed selection process is outlined in Algorithm 2. The resulting dataset D_τ is designed to: (1) comprehensively cover the diverse input scenarios that may arise during mesh generation, and (2) include all corresponding high-quality outputs for each scenario. This forms a critical foundation for training a robust and generalizable mesh generator capable of adapting to a wide range of geometric domains while ensuring consistently high mesh quality.

Algorithm 2 Filtering samples.
Require: Sample dataset D, quality thresholdτ.
1: $D_\tau \leftarrow []$
2: **for** each sample in D **do**
3: $x, y \leftarrow$ sample
4: **if** $\eta(x,y) \geq \tau$ **then**
5: $D_\tau \leftarrow D_\tau \cup (x,y)$
6: end if
7: **end for**

Sample balancing

Since the first dimension of the output corresponds to the rule type, $y_{0,i} = \text{type}_i$, which takes on discrete values, $\text{type}_i \in \{0,1,2\}$. To maintain a balanced dataset $D_b = \{x_i, y_i\}_{i=1}^{M}$, the total number of samples for each type is specified as $M/3, M/3$ and $M/3$, respectively, which ensures that the sample numbers are uniformly distributed.

FNN model

The final balanced dataset D_b is used to train an FNN model. FNNs, also known as multilayer perceptrons, are classical ANNs, where information moves in one direction from input nodes, through hidden layers, to the output nodes. Despite their simple structure, FNNs possess powerful general function approximation capabilities. Based on Kolmogorov–Arnold representation theorem, several theoretical studies (Hecht-Nielsen, 1987, 1992; Cybenko, 1989; Hornik et al., 1989) show that FNNs are universal approximators, such as an FNN with just one hidden layer of a sufficient number of nodes using arbitrary squashing functions (e.g., sigmoid function) can approximate any continuous function with certain properties (e.g., Borel measurable) to any degree of accuracy.

https://doi.org/10.1017/S089006042510019X Published online by Cambridge University Press

In the context of modern deep learning, deep architectures often utilize convolutional layers for feature extraction, while fully connected layers (i.e., FNNs) serve to map extracted features to target classes or output spaces (Krizhevsky et al., 2012; LeCun et al., 2015). These fully connected layers typically employ the Rectified Linear Unit (ReLU) activation function to mitigate the vanishing gradient problem. Recent studies, such as Lu et al. (2017), have shown that FNNs with ReLU activations and bounded width can also serve as universal approximators when given sufficient depth. Given the power of FNNs as universal approximators, in mesh generation, FNN is an ideal model for mapping a meshing state, represented by the input x_t (Equation 1), into a meshing action, represented as output y_t (Equation 2), in a sequential decision-making framework.

In this study, the network structure follows the same structure as proposed in the work (Pan et al., 2021). A combined loss function, cross-entropy loss for the action type classification and mean square error loss for the vertex coordinate prediction, is used to evaluate the FNN model prediction error. The trained FNN meshing model is named FreeMesh-DG, and its structure is shown in Algorithm 3. The algorithm incrementally constructs a quadrilateral mesh by repeatedly selecting a reference point along the boundary and applying one of three predefined rules to extract a new element. The process begins with an initial boundary consisting of n vertices and continues until only four vertices remain, which can be directly connected to form the final element.

Algorithm 3 FreeMesh-DG algorithm structure.
Require: Number of boundary vertices$n$.

```
m ← n                                   ▷ Initialize the control variable
while m > 4 do                          ▷ The last four points can form an element automatically
    Find the reference vertex from the boundary
    if rule 3 applies then
        Extract 1 element around the reference vertex by adding 2 points
        m ← m + 2
    else if rule 1 applies then
        Extract 1 element around the reference vertex by adding 1 point
    else                                ▷ Apply rule 0 or 2
        Extract 1 element around the reference vertex
        m ← m − 2
    end if
    Update the boundary
end while
```

The time complexity of the FreeMesh-DG algorithm is analyzed based on the number of boundary vertices n. The core operations, that is, applying meshing type 3, finding reference vertices, and updating boundaries, are all constant-time operations. Rule type 0/2 must be applied at a baseline complexity of $O(n)$. Rule type 1, which may be applied multiple times between type 0/2 operations, contributes an additional variable cost. In the worst case, this results in a cumulative complexity of $O(n^2)$, making the overall time complexity of the algorithm $O(n^2)$.

Experiment results

In this section, we demonstrate the effectiveness of the proposed method to train a meshing model, FreeMesh-DG, through a series of experiments, including data diversity verification and meshing performance comparison. We begin by outlining the implementation details, followed by a comprehensive evaluation of the experimental results.

Implementation details

We set $n = 3$ and $g = 3$ in the input formulation (in Equation 1). Each input is composed of vertices represented in a 2D polar coordinate system (r, θ), with the lower and upper bounds for each axis defined as $[0,1]$ and $[0,\pi]$, respectively. The output space comprises three dimensions, with bounds $[0,1],[0,1],[0,\pi]$, corresponding to rule parameters and vertex coordinates. The network structure of the FNN has five hidden layers as $[64,128,64,32,16]$, where its optimal setting is evaluated in Pan et al. (2021). The learning rate is set as 1e-3. All the experiments are conducted on a computer with an i7–8700 CPU and an Nvidia GTX 1080 Ti GPU with 32 GB of random access memory. A total of one million steps are used to train the policy network.

Quality threshold analysis

After generating a large amount of input–output samples, a quality threshold τ must be determined to filter high-quality solutions. Five metrics are selected to characterize the dataset: element quality (η^e in Equation 3), boundary quality (η^b), quality (η), angle (i.e., $\angle V_{l,1} V_0 V_{r,1}$ in the input), and averaged segment length. Four candidate thresholds, $\tau \in \{0.6,0.7,0.75,0.8\}$, are evaluated across these metrics. The comparison results are shown in Figure 6. As the threshold increases, the mean values of the three quality metrics also increase, indicating improved solution quality; however, the corresponding value ranges narrow, suggesting reduced sample diversity. The average segment length shows a slight decrease with minimal change in range. The mean angle values remain around 90°, while their ranges become more concentrated. The general principle for selecting an appropriate quality threshold is to ensure high quality while preserving sufficient diversity, such as the angle distribution. Therefore, we choose $\tau = 0.7$ as a balanced threshold to construct the final dataset for model training.

Sample size analysis

This section analyzes the sufficient dataset size required to achieve optimal model performance. While a larger dataset can help establish a clearer decision boundary for the target problem, excessive data may not yield significant performance gains and can instead introduce unnecessary computational overhead. To investigate this trade-off, four sample sizes are evaluated: $M \in \{5e3, 1e4, 4e4, 1e5\}$. The element quality ($\eta^e$ in Equation 3) is used as the primary performance metric, and the results are shown in Figure 7. Among the four configurations, the smallest dataset ($M = 5e3$) yields the lowest mean element quality, indicating suboptimal performance. For the other three dataset sizes, increasing the sample size results in only limited improvements in mean quality. However, a notable reduction in outliers is observed, suggesting enhanced stability with larger datasets. Therefore, a sample size $M = 4e4$ is adopted in this study because of a favorable balance between model performance and computational efficiency.

Evaluation

Data diversity verification

Many NN-based methods use existing mesh generators as the data sources for training. For instance, Papagiannopoulos et al. (2021) utilized the output data of Gmsh, a mesh generator that had

https://doi.org/10.1017/S089006042510019X Published online by Cambridge University Press

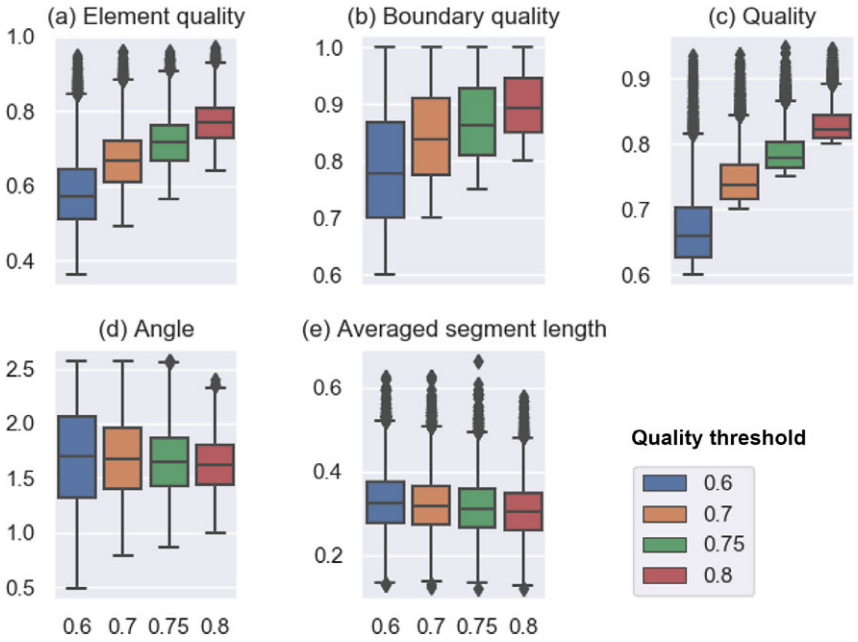


Figure 6. Comparison of datasets with four levels of quality thresholds, $\tau \in \{0.6,0.7,0.75,0.8\}$. Five kinds of metrics are used to represent the characteristics of the dataset, including element quality (η^e in Equation 3), boundary quality (η^b), quality (η), angle (i.e., $\angle V_{l,1}V_0V_{r,1}$ in the input), and the averaged segment length.

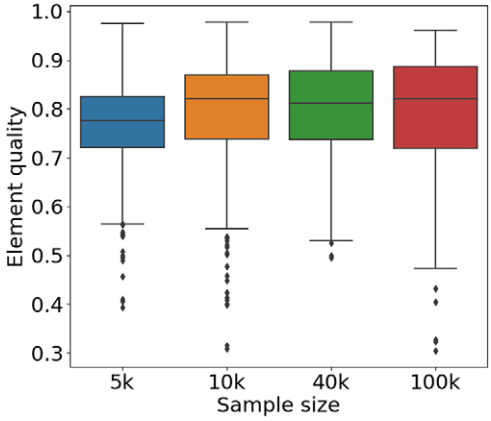


Figure 7. Comparison of element quality with four levels of sample size, $M \in \{5e3,1e4,4e4,1e5\}$. Element quality is used to measure the meshing results with different levels of sample size.

implemented the Blossom-Quad algorithm (Remacle et al., 2012), to train their NNs. However, the diversity and balance of the obtained data are thus limited, which could compromise the final model performance. To assess the impact of these limitations, we extract training samples from a domain meshed using Gmsh and compare their diversity and distribution with those obtained using our proposed method. The detailed extraction process is described in our previous work (Pan et al., 2021).

The comparison results of vertex distributions obtained by the two methods are shown in Figure 8. Since all input–output pairs are represented in a polar coordinate system centered at the reference vertex, the data exhibit circular patterns. It is evident that the value ranges of neighboring vertices (in blue), vertices in the fan-shaped area (in yellow), and the newly generated vertices (in red) are narrower and unbalanced across three output types from the samples by Gmsh [in Figure 8(a)] compared to those generated by FreeMesh-DG [in Figure 8(b)]. While the proposed method yields uniformly distributed vertices in the fan-shaped area, the Gmsh-based samples show missing values in specific regions for neighboring vertices, indicating a lack of coverage for certain meshing scenarios. Additionally, the newly generated vertices in the Gmsh samples are confined to a smaller region near the reference point, which may hinder the model's generalizability, especially in handling highly irregular boundary geometries.

Additionally, we examine the angle distribution in the samples produced by the two methods, as shown in Figure 9. The results indicate that the samples generated by FreeMesh-DG [in Figure 9(b)] have a uniform distribution for angles across three action types, with values ranging from 0.5 to 2.5 radians. In contrast, the samples extracted from Gmsh [in Figure 9(a)] follow a normal distribution across three types, with a narrower range confined to (1, 2.25) radians. This comparison clearly demonstrates that FreeMesh-DG achieves broader and more balanced angle coverage along the boundary, thereby offering more diverse and representative training data to enhance model generalizability and performance.

Consequently, the proposed method produces more diverse and balanced data compared to existing mesh generators. This advantage arises primarily because: (1) problematic or rare meshing scenarios seldom occur in datasets generated by existing methods, and (2) those methods often lack the capability to handle such cases effectively. In contrast, FreeMesh-DG is designed to directly generate data for a wide range of scenarios by uniformly sampling the feature space while ensuring high mesh quality.

Performance comparison

To evaluate the meshing performance, we compare the mesh quality generated by FreeMesh-DG against two widely adopted meshing methods across five predefined 2D domain boundaries (i.e., domains D0–D4). These domains are designed to encompass a variety of geometric features, such as sharp angles, bottleneck regions, uneven edge distributions, and internal holes, to ensure

https://doi.org/10.1017/S089006042510019X Published online by Cambridge University Press

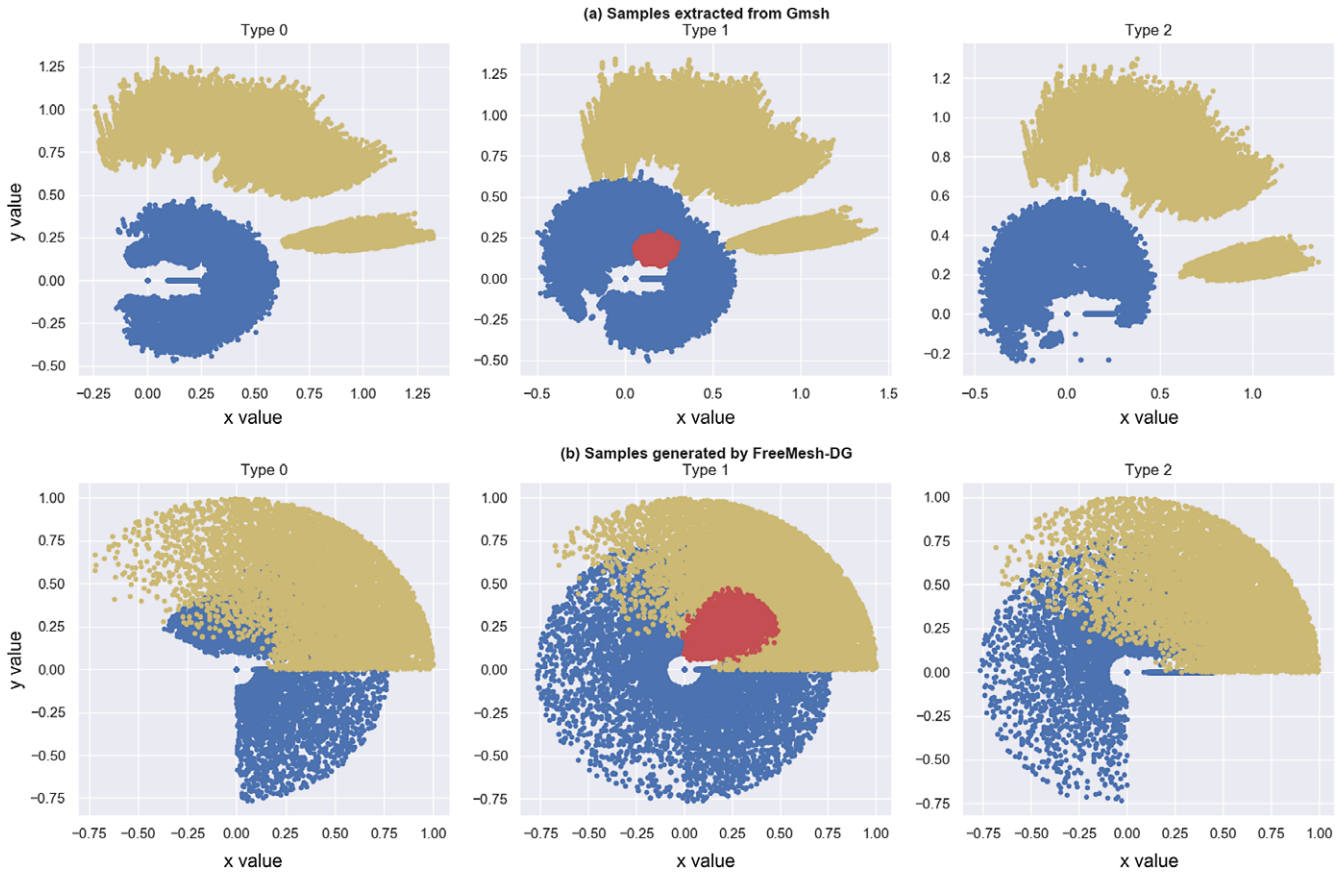


Figure 8. Comparison of the vertex distribution of samples. The first row [i.e., Subfigure (a)] is the distribution of all the vertices in the input–output samples extracted from Gmsh. The second row [i.e., Subfigure (b)] is the vertex distribution of samples generated by FreeMesh-DG with quality threshold $\tau \geq 0.7$. Types 0, 1, and 2 correspond to the three basic rules in the output. Only type 1 needs to generate a new vertex (in red) to form an element. Blue vertices represent the neighboring vertices around the reference Vertex; yellow vertices represent the vertices in the fan-shaped area; all of them are included in the input. The *x* and *y* axes are the coordinate axes of the vertex.

diverse and comprehensive testing. The two benchmark approaches used for comparison are Blossom-Quad (Remacle et al., 2012) and Pave (Blacker and Stephenson, 1991; White and Kinney, 1997).

The meshing results are summarized in Table 1. While all three methods successfully generated meshes for the tested domains, the results from Blossom-Quad and Pave contain triangular elements (highlighted in yellow), which indicate areas requiring additional refinement. Specifically, Blossom-Quad struggles with domains featuring sharp boundary angles, whereas Pave encounters difficulties in interior regions where two advancing boundaries converge. These limitations often necessitate extra cleanup operations to remove triangles or poor-quality elements. In contrast, FreeMesh-DG consistently produces meshes composed entirely of quadrilateral elements, eliminating the need for post-processing and ensuring cleaner, higher-quality outputs.

Table 2 presents the quantitative evaluation of meshing performance across the three methods using eight commonly adopted quality metrics: singularity, element quality (η^e in Equation 3), absolute deviation of minimum and maximum angles from 90° (|MinAngle − 90| and |MaxAngle − 90|), scaled Jacobian, stretch, taper, and the number of triangles (triangles; Pan et al., 2021). The reported values are averaged over the five test domains. FreeMesh-DG demonstrates superior performance in singularity, taper, and triangle count. A lower singularity value suggests enhanced mesh regularity, which is beneficial for improving numerical simulation accuracy. Likewise, a lower taper value indicates elements that are closer to ideal square shapes, and the complete absence of triangles eliminates the need for additional correction steps, thereby enhancing efficiency. On the other hand, the Pave method outperforms the others in terms of element quality, angle deviations, stretch, and scaled Jacobian, where a higher scaled Jacobian indicates more regular quadrilateral elements. Blossom-Quad yields suboptimal results across most metrics and only surpasses Pave in producing fewer triangles, a common challenge for indirect methods due to their reliance on initial triangulation (Remacle et al., 2013).

Discussion

The trained meshing model, FreeMesh-DG, has demonstrated competitive performance when compared to the other two meshing algorithms, achieving optimal results in three key quality indices: singularity, taper, and triangle count. Moreover, FreeMesh-DG offers further advantages, as outlined below.

To ensure a fair and comprehensive comparison, we evaluated the meshing performance using eight widely adopted quality metrics. Although the Pave method slightly outperformed FreeMesh-DG in five of these metrics, the differences are relatively minor. In contrast, FreeMesh-DG demonstrated clearly superior performance in key aspects, particularly in singularity count and triangle ratio. A lower singularity count reflects a better topological structure, which is critical for improving numerical stability and

https://doi.org/10.1017/S089006042510019X Published online by Cambridge University Press

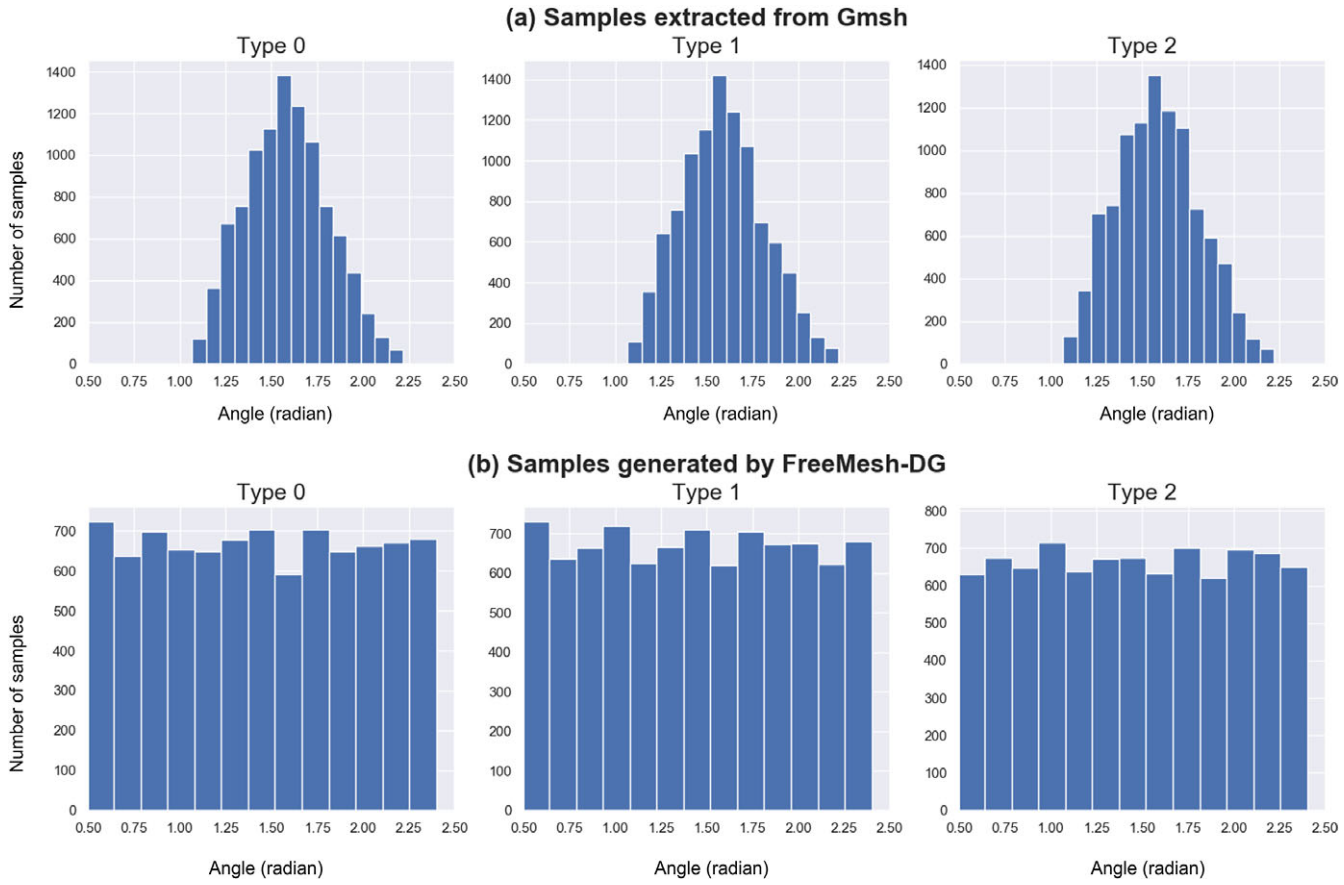
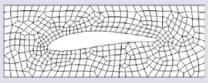
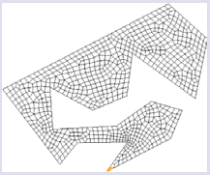
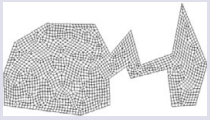
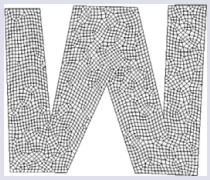
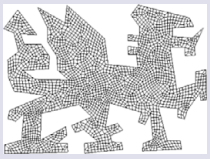
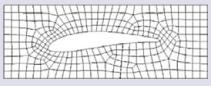
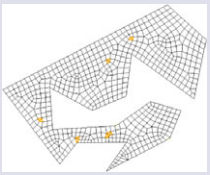
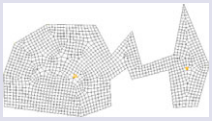
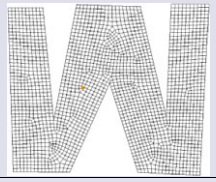
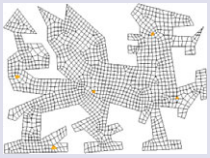
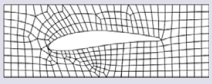
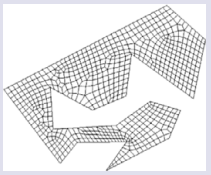
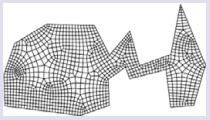
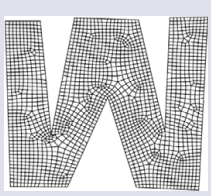
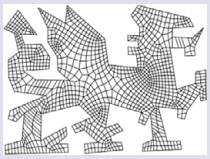


Figure 9. Comparison of the angle distribution of samples. The angle ranges from 0.5 to 2.5 radians. Types 0, 1, and 2 correspond to the three basic rules in the output. Subfigure (a) is the angle distribution of samples from Gmsh. Subfigure (b) is the vertex distribution of samples generated by FreeMesh-DG with a quality threshold $\tau \geq 0.7$.

Table 1. Meshing results comparison

Algorithms	Domain 0	Domain 1	Domain 2	Domain 3	Domain 4
Blossom-Quad					
Pave					
FreeMesh-DG					

Note: The elements in yellow represent existing triangles in the meshes.

https://doi.org/10.1017/S089006042510019X Published online by Cambridge University Press

Table 2. Averaged mesh quality metrics over the five domains

Metrics	Blossom-Quad	Pave	FreeMesh-DG
Singularity (L)	352.67 ± 200.11	90.6 ± 59.36	**87.6** ± **36.79**
Element quality (H)	0.74 ± 0.11	**0.85** ± **0.12**	0.81 ± 0.13
|MinAngle − 90| (L)	16.86 ± 9.77	**10.33** ± **7.91**	12.53 ± 9.8
|MaxAngle − 90| (L)	24.06 ± 10.83	**11.49** ± **10.76**	13.45 ± 11.8
Scaled Jacobian (H)	0.90 ± 0.11	**0.96** ± **0.08**	0.95 ± 0.09
Stretch (H)	0.79 ± 0.07	**0.87** ± **0.08**	0.83 ± 0.1
Taper (L)	0.13 ± 0.12	0.09 ± 0.11	**0.09** ± **0.1**
Triangle (L)	0.67 ± 0.94	6 ± 5.29	**0** ± **0**

Note: L, H indicate if the lower value or higher value is preferred, respectively. The value in bold means the best among other approaches in that specific metric.

accuracy in simulation tasks. Moreover, FreeMesh-DG produced meshes composed entirely of quadrilateral elements, avoiding the introduction of triangles that typically require additional post-processing and may degrade performance in applications tailored for structured quadrilateral meshes.

One of the key benefits of FreeMesh-DG is its ability to meet the diverse quality requirements of downstream applications by customizing the quality function. Different applications have distinct simulation tasks that necessitate meshes with specific topological and geometric characteristics. In contrast, existing mesh generators are limited to producing meshes with predefined characteristics, often requiring additional post-processing techniques such as remeshing and mesh adaptation (Verma and Suresh, 2017). These additional techniques are not only costly and time-consuming but also slow down the meshing process and increase algorithm complexity. FreeMesh-DG provides an efficient solution by directly designing the desired mesh through the adjustment of the quality function. The generated data can be quickly used to train the target algorithm, which is highly valuable in practical applications.

Extending our method to 3D or surface meshing is a natural and promising direction. As an intermediate step, surface meshing on 3D geometries can bridge the gap between planar 2D meshing and full 3D volume meshing. To enable this extension, our framework can be adapted by incorporating surface parameterization or intrinsic coordinate systems, allowing the model to operate on curved surfaces while preserving geometric fidelity. The core mechanisms, including learning from high-quality examples, generating elements through sequential decisions, and promoting topological smoothness, can be directly applied to surface contexts with minimal structural changes. This would make it possible to produce structured quad meshes on complex 3D surfaces, maintaining key advantages such as low singularity count and full quadrilateral coverage.

The approach employed by FreeMesh-DG can be extended to a variety of real-life problems. These problems can be formulated as sequential decision-making processes, where a few primitive rules (i.e., state-action pairs) are defined. A quality function is then used to select high-quality samples, and a NN is subsequently trained to capture the mapping relationships between states and actions using the obtained data. This process can significantly reduce the manual effort required to develop efficient and robust algorithms.

However, a limitation of FreeMesh-DG lies in its performance when meshing domains with sharp angles or narrow regions. Currently, the data generation strategy uniformly samples the entire domain, which can lead to suboptimal meshing in these extreme scenarios. As sharp angles and narrow regions are less frequently represented in the uniformly distributed training data, the model tends to struggle with such cases. Future work will focus on improving the data generation process by concentrating sampling efforts in areas with complex boundary conditions, thus enhancing the model's robustness in these challenging scenarios.

Conclusion

To address the issue of limited high-quality data and imbalance, this study introduces a quality function-based data generation method for the automatic quadrilateral mesh generation algorithm, FreeMesh-DG. The mesh generation process is formulated as a sequential decision-making problem, consisting of a set of state-action (i.e., input–output) pairs. A large volume of data is uniformly sampled from the input and output feature spaces to ensure comprehensive coverage of all possible input scenarios and corresponding output solutions. Extensive experiments demonstrate that the algorithm trained on this generated data performs competitively with two widely used mesh generation methods, even surpassing them in three key quality indices.

This article highlights the potential of FreeMesh-DG for the mesh generation and algorithm design communities. Several avenues for future work are identified, including the exploration of quality functions tailored to meet the diverse needs of downstream applications and the extension of this approach to 3D mesh generation.

Acknowledgments. The support of the NSERC Discovery Grant (RGPIN-2019-07048) is gratefully acknowledged.

References

Blacker TD and Stephenson MB (1991) Paving: A new approach to automated quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering* **32**, 811–847.

Bunkhumpornpat C, **Sinapiromsaran K and Lursinsap C** (2009) Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem, In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 475–482.

Chawla NV, **Bowyer KW**, **Hall LO and Kegelmeyer WP** (2002) Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research* **16**, 321–357.

Cybenko G (1989) Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* **2**, 303–314.

Gordon WJ and Hall CA (1973) Construction of curvilinear co-ordinate systems and applications to mesh generation. *International Journal for Numerical Methods in Engineering* **7**, 461–477.

Habibie I, **Holden D**, **Schwarz J**, **Yearsley J and Komura T** (2017) A recurrent variational autoencoder for human motion synthesis, In *28th British Machine Vision Conference*, London: British Machine Vision Association.

Han H, **Wang WY and Mao BH** (2005) Borderline-smote: A new over-sampling method in imbalanced data sets learning, In *International Conference on Intelligent Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 878–887.

He H, **Bai Y**, **Garcia EA and Li S** (2008) Adasyn: Adaptive synthetic sampling approach for imbalanced learning, In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. Hong Kong, China: IEEE, pp. 1322–1328.

He H and Garcia EA (2009) Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering* **21**, 1263–1284.

Hecht-Nielsen R (1987) Kolmogorov's mapping neural network existence theorem, In *Proceedings of the International Conference on Neural Networks*. NY, USA: IEEE Press New York, pp. 11–14.

https://doi.org/10.1017/S089006042510019X Published online by Cambridge University Press

Hecht-Nielsen R (1992) Theory of the backpropagation neural network, In *Neural Networks for Perception*. Elsevier, pp. 65–93.

Hornik K, **Stinchcombe M and White H** (1989) Multilayer feedforward networks are universal approximators. *Neural Networks* **2**, 359–366.

Jo T and Japkowicz N (2004) Class imbalances versus small disjuncts. *ACM Sigkdd Explorations Newsletter* **6**, 40–49.

Krawczyk B (2016) Learning from imbalanced data: Open challenges and future directions. *Progress in Artificial Intelligence* **5**, 221–232.

Krizhevsky A, **Sutskever I and Hinton GE** (2012) Imagenet classification with deep convolutional neural networks, *Advances in Neural Information Processing Systems* **25**. Lake Tahoe: Curran Associates Inc., pp 1097–1105.

LeCun Y, **Bengio Y and Hinton G** (2015) Deep learning. *Nature* **521**, 436–444.

Lee TK and Filipi ZS (2010) Synthesis and validation of representative real-world driving cycles for plug-in hybrid vehicles, In *2010 IEEE Vehicle Power and Propulsion Conference*. Lille: IEEE, pp. 1–6.

Liu CL and Hsieh PY (2019) Model-based synthetic sampling for imbalanced data. *IEEE Transactions on Knowledge and Data Engineering* **32**, 1543–1556.

Liu XY, **Wu J and Zhou ZH** (2008) Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **39**, 539–550.

Lu Z, **Pu H**, **Wang F**, **Hu Z and Wang L** (2017) The expressive power of neural networks: A view from the width, *Advances in Neural Information Processing Systems* **30**, Long Beach: Curran Associates Inc., pp 6232–6240.

Nechaeva O (2006) Composite algorithm for adaptive mesh construction based on self-organizing maps, In *International Conference on Artificial Neural Networks*. Springer, pp. 445–454.

Pan J, **Huang J**, **Cheng G and Zeng Y** (2023) Reinforcement learning for automatic quadrilateral mesh generation: A soft actor–critic approach. *Neural Networks* **157**, 288–304.

Pan J, **Huang J**, **Wang Y**, **Cheng G and Zeng Y** (2021) A self-learning finite element extraction system based on reinforcement learning. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 1–29. https://doi.org/10.1017/S089006042100007X.

Papagiannopoulos A, **Clausen P and Avellan F** (2021) How to teach neural networks to mesh: Application on 2-d simplicial contours. *Neural Networks* **136**, 152–179.

Remacle JF, **Henrotte F**, **Carrier-Baudouin T**, **Béchet E**, **Marchandise E**, **Geuzaine C and Mouton T** (2013) A frontal delaunay quad mesh generator using the l norm. *International Journal for Numerical Methods in Engineering* **94**, 494–512.

Remacle JF, **Lambrechts J**, **Seny B**, **Marchandise E**, **Johnen A and Geuzainet C** (2012) Blossom-quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm. *International Journal for Numerical Methods in Engineering* **89**, 1102–1119.

Roca X and Loseille A (2019) *27th International Meshing Roundtable*, Vol. **127**. Cham: Springer.

Shamshad A, **Bawadi M**, **Hussin WW**, **Majid T and Sanusi S** (2005) First and second order markov chain models for synthetic generation of wind speed time series. *Energy* **30**, 693–708.

Slotnick J, **Khodadoust A**, **Alonso J**, **Darmofal D**, **Gropp W**, **Lurie E and Mavriplis D** (2014) *CFD vision 2030 study: a path to revolutionary computational aerosciences*, Technical Report, NASA.

Tanaka FHK and Aranha C (2019) Data augmentation using GANs. arXiv preprint arXiv:1904.09135.

Verma CS and Suresh K (2017) A robust combinatorial approach to reduce singularities in quadrilateral meshes. *Computer-Aided Design* **85**, 99–110.

Vinyals O, **Fortunato M and Jaitly N** (2015) Pointer networks. *Advances in Neural Information Processing Systems 2*, Montreal: MIT Press, pp. 2692–2700.

Wang Z, **Wang J and Wang Y** (2018) An intelligent diagnosis scheme based on generative adversarial learning deep neural networks and its application to planetary gearbox fault pattern recognition. *Neurocomputing* **310**, 213–222.

White DR and Kinney P (1997) Redesign of the paving algorithm: Robustness enhancements through element by element meshing, In *6th International Meshing Roundtable*. Sandia National Laboratories, pp. 830.

Xie Z, **Jiang L**, **Ye T and Li X** (2015) A synthetic minority oversampling method based on local densities in low-dimensional space for imbalanced learning, In *International Conference on Database Systems for Advanced Applications*. Cham: Springer, pp. 3–18.

Xuan Q, **Chen Z**, **Liu Y**, **Huang H**, **Bao G and Zhang D** (2018) Multiview generative adversarial network and its application in pearl classification. *IEEE Transactions on Industrial Electronics* **66**, 8244–8252.

Yang W and Nam W (2022) Data synthesis method preserving correlation of features. *Pattern Recognition* **122**, 108241.

Yang Y, **Zha K**, **Chen YC**, **Wang H and Katabi D** (2021) Delving into deep imbalanced regression. arXiv preprint arXiv:2102.09554.

Yao S, **Yan B**, **Chen B and Zeng Y** (2005) An ANN-based element extraction method for automatic mesh generation. *Expert Systems with Applications* **29**, 193–206.

Yen SJ and Lee YS (2009) Cluster-based under-sampling approaches for imbalanced data distributions. *Expert Systems with Applications* **36**, 5718–5727.

Zeng Y and Cheng G (1993) Knowledge-based free mesh generation of quadrilateral elements in two-dimensional domains. *Computer-Aided Civil and Infrastructure Engineering* **8**, 259–270.

Zeng Y and Yao S (2009) Understanding design activities through computer simulation. *Advanced Engineering Informatics* **23**, 294–308.

Zhang Z, **Wang Y**, **Jimack PK and Wang H** (2020) Meshingnet: A new mesh generation method based on deep learning, In *International Conference on Computational Science*. Cham: Springer, pp. 186–198.

https://doi.org/10.1017/S089006042510019X Published online by Cambridge University Press