

# **Rule-based Machine Learning Algorithms for Smart Automatic Quadrilateral Mesh Generation System**

**Jie Pan**

**A Thesis**

**in**

**The Concordia Institute**

**for**

**Information Systems Engineering**

**Presented in Partial Fulfillment of the Requirements**

**for the Degree of**

**Doctor of Philosophy (Information and Systems Engineering) at**

**Concordia University**

**Montréal, Québec, Canada**

**November 2021**

**© Jie Pan, 2021**

CONCORDIA UNIVERSITY  
School of Graduate Studies

This is to certify that the thesis prepared

By: **Jie Pan**  
Entitled: **Rule-based Machine Learning Algorithms for Smart Automatic  
Quadrilateral Mesh Generation System**

and submitted in partial fulfillment of the requirements for the degree of

**Doctor of Philosophy (Information and Systems Engineering)**

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

\_\_\_\_\_  
*Dr. Kudret Demirli* Chair

\_\_\_\_\_  
*Dr. Sofiane Achiche* External Examiner

\_\_\_\_\_  
*Dr. Tsz Ho Kwok* External to Program

\_\_\_\_\_  
*Dr. Chun Wang* Examiner

\_\_\_\_\_  
*Dr. Jiayuan Yu* Examiner

\_\_\_\_\_  
*Dr. Yong Zeng* Thesis Supervisor

\_\_\_\_\_  
*Dr. Jingwei Huang* Thesis Co-supervisor

Approved by \_\_\_\_\_  
Dr. Mohammed Mannan, Graduate Program Director

11/22/2021 \_\_\_\_\_  
Dr. Mourad Debbabi, Dean  
Gina Cody School of Engineering and Computer Science

# Abstract

## Rule-based Machine Learning Algorithms for Smart Automatic Quadrilateral Mesh Generation System

**Jie Pan, Ph.D.**

**Concordia University, 2021**

Mesh generation, as one of six basic research directions identified in NASA Vision 2030, is an important area in computational geometry and plays a fundamental role in numerical simulations in the area of finite element analysis (FEA) and computational fluid dynamics (CFD). With the rapid progress of high-performance computing hardware, mesh generation methods are required to handle geometric domains with more complex shapes and higher resolution in reliable and fast fashions. Yet, existing mesh generation methods suffer from high computational complexity, low mesh quality in complex geometries, and speed limitations, and have continued to be the bottleneck in those simulation tasks.

This thesis addresses the quadrilateral mesh generation problem from three aspects, element extraction, sequential decision making, and data generation, and their combinations. First, a self-learning system, FreeMesh-S, for finite element extraction system is investigated. Element extraction is a major mesh generation method for its capabilities to generate high-quality meshes around the domain boundary and can be formulated into a sequential decision making process. Three kinds of primitive element extraction rules are conceptually identified. FreeMesh-S, then learns the rules by 1) sampling the element generation rules by a reinforcement learning (RL) algorithm, 2) extracting high quality samples, and 3) training the final rules by a feedforward neural network (FNN). The comprehensive experiments demonstrate the effectiveness of the self-learned meshing rules by FreeMesh-S.

Second, an RL-based computational framework for automatic mesh generation is proposed to improve algorithm automation further. A state-of-the-art RL algorithm, soft actor-critic (SAC), is used to learn the mesh generator's policy from trials. It achieves a fully automatic mesh generation without human intervention and any extra clean-up operations, which are typically needed in current commercial software. The reward function is carefully designed to balance the contradiction between the instant element quality and the remaining boundary quality, in order to achieve an overall high quality mesh. The experiments have shown the competitive performance with two representative meshing methods with respect to generalizability, robustness, and effectiveness. The potentials of mesh generation as a benchmark problem for RL are also identified.

Last, a quality function-based data generation method for the meshing algorithm is devised to increase learning efficiency and algorithm performance. For any data-driven algorithms, high quality and balanced data are essential and deterministic to the performance. This method samples the input-output of the three rules according to their feature spaces; selects high quality samples by a quality function that evaluates if the output is an appropriate solution to the input; and trains an FNN model to simulate the mapping relation via the obtained data. The experiments show that the learning time is greatly reduced while the model has competitive performance comparing with other meshing methods.

To conclude, this thesis combines artificial intelligence techniques, rule-based system, neural networks, and RL, to automate the quadrilateral mesh generation while significantly reducing the time and expertise needed during the creation of high quality mesh generation algorithm. All the techniques can be directly generalized to 3D mesh generation.

# Acknowledgments

The way towards success is always full of tears and joy. The most important thing is to keep curiosity and a heart for reaching excellence. It has been much more difficult and meaningful than I thought from the first day of my Ph.D. journey. It has transformed me from confident to humble, from a passive thinker to an active thinker, and from being solution-driven to being purpose-driven and design-driven.

I would like to express my sincere gratitude to my supervisor, co-supervisor, co-authors, and colleagues for their professional assistance, guidance, experience, and support throughout this research. I would like to thank my co-supervisor, Prof. Jingwei Huang, for providing me your professional research knowledge and experience. Specially, I would like to express my thanks to my supervisor, Prof. Yong Zeng, for sharing your understanding of doing excellent research, and for supporting and encouraging me during difficult times.

I would like to thank Concordia University for providing me the wonderful research opportunity and NSERC Discovery Grant for the financial support.

I am grateful for all my friends from the Design Lab. They are generous in providing support, encouragement, and professional suggestions in my research and life. My awesome friend, Gaby, thanks for motivating me to be a better myself, and thanks for your always mental support.

I am honored to have my amazing family, thank my parents for the unrequited investment and support. Their endless support is the driving force keeping me moving forward. My little brother,

Yuhang, thanks for taking care of our parents when I'm absent. I would like to express my thanks to my grandparents, for always caring and thinking about me. Because of all your love and support, I could be able to finish my research and enjoy my life.

# Contribution of Authors

Chapter 4 of this thesis was published in *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*. Chapter 5 of this thesis has been submitted to the journal of *IEEE Transactions on Neural Networks and Learning Systems*. Chapter 6 of this thesis is prepared to submit to the journal of *IEEE Transactions on Knowledge and Data Engineering*. The author of this thesis was responsible for the development, testing, and application of the methods discussed in this research, along with preparation of manuscripts submitted to peer-reviewed journals. Dr. Yong Zeng, Professor, Concordia Institute for Information Systems Engineering, Concordia University, supervised this thesis and provided valuable guidance and advice on various aspects of the research.

The following individuals provided advice and assistance in all aspects of this thesis, and contributed to the review and editing of each manuscript: Dr. Jingwei Huang, Associate Professor, Department of Engineering Management and Systems Engineering, Old Dominion University; Dr. Yunli Wang, Research Officer, National Research Council Canada; Dr. Gengdong Cheng, Professor, Department of Engineering Mechanics, Dalian University of Technology.

## List of publications related to the thesis:

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

Pan, J., Huang, J., Cheng, G., & Zeng, Y. (2021). Reinforcement learning for automatic quadrilateral

mesh generation: a soft actor-critic approach. *IEEE Transactions on Neural Networks and Learning Systems*. (Under review).

Pan, J., Huang, J., Cheng, G., & Zeng, Y. (2021). Sampling balanced high quality data to train an automatic mesh generator for its optimal performance. *IEEE Transactions on Knowledge and Data Engineering*. (Under review).



# Contents

<b>List of Figures</b>	<b>xiii</b>
<b>List of Tables</b>	<b>xviii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	3
1.2 Objective . . . . .	4
1.3 Outline . . . . .	5
<b>2 Literature Review</b>	<b>6</b>
2.1 Mesh generation . . . . .	7
2.1.1 Conventional methods . . . . .	7
2.1.2 Machine learning-based methods . . . . .	10
2.2 Reinforcement learning . . . . .	14
2.3 Imbalanced learning . . . . .	16
2.3.1 Data-level approaches . . . . .	16
2.3.2 Algorithm-level approaches . . . . .	18
2.4 Research challenges and opportunities . . . . .	19
<b>3 Contributions</b>	<b>22</b>
<b>4 A self-learning finite element extraction system based on reinforcement learning</b>	<b>26</b>

4.1	Introduction . . . . .	27
4.1.1	Motivation: why is it necessary to use the element extraction method? . . . .	28
4.1.2	The challenge of the element extraction method . . . . .	30
4.1.3	Contribution . . . . .	30
4.2	Smart designing of the smart element extraction system . . . . .	31
4.2.1	How is the element extraction method smart? . . . . .	32
4.2.2	How can the element extraction system be smartly evolving and designed? . . . .	35
4.3	A self-learning system for element extraction . . . . .	37
4.3.1	Element extraction as a reinforcement learning problem . . . . .	38
4.3.2	A2C RL network for element extraction based mesh generation . . . . .	42
4.3.3	FNN as policy approximator for fast learning and meshing . . . . .	51
4.3.4	Summary . . . . .	56
4.4	Experiments . . . . .	57
4.4.1	Experiment settings . . . . .	57
4.4.2	Experiment 1: training effectiveness and efficiency . . . . .	59
4.4.3	Experiment 2: comparisons of the proposed method with existing methods . . . .	65
4.4.4	Summary . . . . .	72
4.5	Discussion . . . . .	73
4.5.1	Domain knowledge dependency . . . . .	73
4.5.2	The relation between smart design and smart system . . . . .	74
4.5.3	Limitations . . . . .	75
4.6	Conclusions . . . . .	76
<b>5</b>	<b>Reinforcement learning for automatic quadrilateral mesh generation: a soft actor-critic approach</b> . . . . .	<b>83</b>
5.1	Introduction . . . . .	84
5.1.1	Mesh generation challenges . . . . .	84
5.1.2	Related work . . . . .	86
5.1.3	Contributions . . . . .	89

5.2	Problem formulation and fundamentals . . . . .	90
5.2.1	Problem formulation . . . . .	90
5.2.2	Reinforcement learning . . . . .	93
5.3	RL based mesh generation . . . . .	95
5.3.1	Action formulation . . . . .	95
5.3.2	State representation . . . . .	96
5.3.3	Reward function . . . . .	99
5.3.4	Meshing scheme via SAC . . . . .	101
5.4	Experimental results . . . . .	104
5.4.1	Implementation details . . . . .	104
5.4.2	Evaluation . . . . .	107
5.5	Discussion . . . . .	113
5.6	Conclusion . . . . .	115
<b>6</b>	<b>Sampling balanced high quality data to train an automatic mesh generator for its optimal performance</b>	<b>116</b>
6.1	Introduction . . . . .	117
6.2	Problem formulation and fundamentals . . . . .	119
6.2.1	Mesh generation . . . . .	119
6.2.2	Data generation . . . . .	122
6.3	Quality function-based data generation for mesh generation . . . . .	126
6.3.1	Data generation procedure . . . . .	126
6.3.2	Quality function for performance measurement . . . . .	128
6.3.3	Sample balancing . . . . .	129
6.3.4	FNN training . . . . .	130
6.4	Experiment results . . . . .	130
6.4.1	Implementation details . . . . .	130
6.4.2	Evaluation . . . . .	132
6.5	Discussion . . . . .	137

6.6 Conclusion . . . . .	137
<b>7 Conclusions and future works</b>	<b>139</b>
7.1 Conclusions . . . . .	139
7.2 Future works . . . . .	140
<b>Bibliography</b>	<b>142</b>

# List of Figures

Figure 4.1	Quad mesh generation process (Yao, Yan, Chen, & Zeng, 2005) . . . . .	32
Figure 4.2	Three primitive generation rules (Zeng & Cheng, 1993) . . . . .	33
Figure 4.3	Changes of intermediate boundary shapes: all boundaries are represented in red lines from (2) to (9). . . . .	33
Figure 4.4	Challenges in selecting a proper rule to extract an element for an edge $V_1V_2$ . . . . .	34
Figure 4.5	Trade-off between qualities of an extracted element and the remaining boundary. Subfigures (a), (b), and (c) correspond to the situation in Figure 4.4 (b); Subfigures (d) and (e) correspond to the situation in Figure 4.4 (f). Points A, A1, and A2 form better elements than B, B1, and B2. . . . .	35
Figure 4.6	Element extraction architecture . . . . .	38
Figure 4.7	Architecture of Reinforcement Learning (Sutton & Barto, 2018) . . . . .	39
Figure 4.8	Element extraction process as Reinforcement Learning . . . . .	39
Figure 4.9	Partial boundary . . . . .	42
Figure 4.10	A2C agent architecture for element extraction . . . . .	43
Figure 4.11	A2C network structure. In general, the actor will simulate three primitive rules, as shown in Figure 4.2 (a), (b), and (c). In the current implementation, the extraction rule in Figure 4.2 (a) is not considered and it will be implemented in the next version of the system. . . . .	46
Figure 4.12	Coordinate transformation . . . . .	48

Figure 4.13 Invalid situations of the element. $P_0$ is the reference point and $P$ is the newly generated point. . . . .	49
Figure 4.14 Different quality of elements . . . . .	49
Figure 4.15 New boundary angles . . . . .	50
Figure 4.16 FNN agent architecture . . . . .	52
Figure 4.17 FNN module structure . . . . .	53
Figure 4.18 Three types of outputs . . . . .	53
Figure 4.19 Example of experience extraction of one element. In (a), each element has a tuple $\langle elementid; elementquality \rangle$ inside, where element id refers to the generation order of the element; and element quality is calculated by $\eta_i^e$ ; (b)-(d) show three types of patterns collected for training samples: (b) type 0; (c) type 1; (d) type 2. . . . .	55
Figure 4.20 Proposed self-learning system FreeMesh-S: architecture . . . . .	57
Figure 4.21 experimental domains . . . . .	58
Figure 4.22 Model training process . . . . .	60
Figure 4.23 Partial results of sampling . . . . .	61
Figure 4.24 The first meshing result of the FNN model . . . . .	62
Figure 4.25 Temporal training cost of A2C network . . . . .	63
Figure 4.26 All the meshing results of FreeMesh-S . . . . .	67
Figure 4.27 Meshing results of GB method . . . . .	68
Figure 4.28 Meshing results of GD method . . . . .	69
Figure 4.29 Meshing results of CP method . . . . .	70
Figure 4.30 Number of domains that each method achieves the best performance in each metrics . . . . .	70
Figure 4.31 Multi-connected domain to single-connected domain. (1) Multi-connected domain; (2) Single-connected domain. . . . .	76
Figure 5.1 Meshing problem. (a) The initial geometry is defined by the boundary $B$ consisting of a set of vertices $V$ ; (b) The final mesh is the result of discretization into a set of quadrilateral elements $Q$ . . . . .	91

Figure 5.2	A sequence of actions took by the mesh generator 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 cutting off the element and serves as the meshing boundary in the next time step $t_{i+1}$ . This process continues until the updated boundary becomes an element. . . . .	92
Figure 5.3	The RL-based computational framework for automatic mesh generation. The environment represents the meshing boundary. The agent is the mesh generator that could implement various RL techniques. . . . .	93
Figure 5.4	Action spaces for each type. Subfigures (a)-(c) correspond to three types of actions, respectively. The blue area is the area with the reference vertex $V_1$ as the origin and a radius $r$ to choose the candidate vertices, such as, $V_3$ in type 1 and $V_3$ and $V_4$ in type 2. . . . .	97
Figure 5.5	Partial observation of the meshing boundary. For example, the partial boundary, where $L_r = 4, n = 2, g = 3$ , is represented as the state. First, two vertices on the left and right sides of the reference vertex $V_0$ are selected, respectively. Second, the angle $\angle V_{l,1}V_0V_{r,1}$ is evenly split into three angles $\theta_1, \theta_2$ , and $\theta_3$ ; three fan-shaped areas are hence formed with these angles and a radius $L_r$ . Then, the closest vertex in each area is selected. . . . .	98
Figure 5.6	Element quality with different shapes. The quality value ranges from 0 to 1. The element with the best quality 1 is a square. . . . .	100
Figure 5.7	The quality of the updated boundary. $Q_1$ is the newly generated element. Once it is removed, it forms two angles $\theta_1$ and $\theta_2$ with existing boundary segments. The quality is jointly measured by these two angles, and the closest Euclidean distance $d_{min}$ of the newly added vertex $V_3$ is to the existing segments. $d_1$ and $d_2$ are the Euclidean lengths of segments $V_3V_4$ and $V_2V_3$ . . . . .	101
Figure 5.8	NN structures comparison. S1-4 represent four different neural network structures in Hidden layers, including [256, 256], [128, 128, 128], [64, 128, 64, 32, 16], and [64, 128, 256, 128, 64], respectively. . . . .	105

Figure 5.9	RL state comparison. The observation range is formed by $L_r-n-g$ , which represents the radius of the fan shape in the state, the number of neighbouring vertices on the left and right side of the reference vertex, and the number of vertices in the fan-shaped area, respectively. This range determines how far and how much information the agent will be observed. . . . .	106
Figure 5.10	Different mesh densities controlled by reward function. Three kinds of densities (a)-(c) are controlled by parameters in the density term, $v = 1.5, v = 1$ , and $v = 0.5$ , respectively. Subfigure (d) is the comparison results of the number of generated elements by three kinds of densities over 10 episodes. . . . .	108
Figure 5.11	Meshing performance comparison results over 8 kinds of quality indices. BQ represents the Blossom-Quad method; F-RL represents the FreeMesh-RL method. L, H indicate if the lower value or higher value is preferred, respectively. . . . .	112
Figure 6.1	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 cutting off the element and serves as the meshing boundary in the next time step $t_{i+1}$ . This process continues until the updated boundary becomes a quadrilateral element. . . . .	121
Figure 6.2	An example of the input with $L_r = 4, n = 2, g = 3$ (Pan, Huang, Cheng, & Zeng, 2021). . . . .	125
Figure 6.3	Quality function-based data generation procedure for mesh generation. The mesh generation algorithm consists of a set of input-output pairs. . . . .	127
Figure 6.4	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 ( $\eta^e$ in Equation 39), boundary quality ( $\eta^b$ ), quality ( $\eta$ ), angle (i.e., $\angle V_{l,1}V_0V_{r,1}$ in the input), and averaged segment length. . . . .	131
Figure 6.5	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. . . . .	132



Figure 6.6 Comparison of 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  $\tau \geq 0.7$ . Type 0, 1, and 2 correspond to the three basic rules in the output. Only type 1 needs generating 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 vertex. . . . . 133

Figure 6.7 Comparison of angle distribution of samples. The angle ranges from 0 to  $\pi$ . Type 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 quality threshold  $\tau \geq 0.7$ . . . . . 134

# List of Tables

Table 4.1	Examples of extracted samples for element 15: the reference point $P_0(x_0, y_0)$ and output point $P_i(x_i, y_i)$ are the same across these samples, respectively . . . . .	54
Table 4.2	Examples of extracted samples in Table 4.1 after coordinate transformation following Eq. 4.3.2 . . . . .	56
Table 4.3	Description of 10 testing domains . . . . .	58
Table 4.4	Mesh quality metrics . . . . .	58
Table 4.5	Self-evolving process of FNN model. . . . .	61
Table 4.6	Self-evolving process of FNN model. . . . .	78
Table 4.7	Different element quality threshold comparison: L, H indicates if the lower value or higher value is preferred, respectively. The hidden layer of this FNN model is [32, 64, 128, 64, 32, 16]. Each metric value is the average value of the three domains (D1, D2, and D3). . . . .	78
Table 4.8	Comparison of FNN network structures: L, H indicate if the lower value or higher value is preferred, respectively. Each metric value is the average value over the three domains (D1, D2, and D3). . . . .	79
Table 4.9	Meshing speed (elements per second) of A2C and FNN for all 10 domains. Avg. indicates the average speed; STD indicates the standard deviation. . . . .	79

Table 4.10	Mesh quality metrics of four methods on 10 domainsL, H indicate if the lower value or higher value is preferred, respectively. The value in bold means the best among other approaches in a domain. GB refers to Gmsh-Blossom; GD indicates Gmsh-DelQuad; CP is CUBIT-Pave; and SS means the proposed self-learning system, FreeMesh-S. . . . .	80
Table 4.11	Comparison of geometry knowledge required to develop and implement the algorithms and system . . . . .	81
Table 4.12	Averaged mesh quality metrics of four methods on 10 domains: L, H indicate if the lower value or higher value is preferred, respectively. The value in bold means the best among other approaches in a domain. GB refers to Gmsh-Blossom; GD indicates Gmsh-DelQuad; CP is CUBIT-Pave; and SS means the proposed self-learning system, FreeMesh-S. . . . .	82
Table 5.1	Training hyperparameters . . . . .	105
Table 5.2	Meshing same shape with different density by FreeMesh-RL. . . . .	109
Table 5.3	Scalability evaluation for three domains . . . . .	109
Table 5.4	Meshing results comparison . . . . .	110
Table 5.5	Averaged mesh quality metrics over three domains . . . . .	111
Table 6.1	Meshing results comparison . . . . .	136
Table 6.2	Averaged mesh quality metrics over the three domains . . . . .	136

# Chapter 1

## Introduction

Tremendous achievements have been made with the advancement of machine/deep learning techniques in many aspects of modern industry: from pattern recognition of image and speech ([Q. Zhang, Yang, Chen, & Li, 2018](#)) to autonomous driving ([Y. Wu, Liao, Liu, Li, & Lu, 2021](#)), and to the protein structure prediction ([Jumper et al., 2021](#)). Machine learning algorithms empower systems with the ability to automatically learn and improve from experience without being explicitly programmed and without human intervention ([LeCun, Bengio, & Hinton, 2015](#)).

Mesh generation is one of the critical fields of modern industry and serves as the fundamental for numerical simulations in finite element analysis (FEA), computational fluid dynamics (CFD), or graphic model rendering [Gordon and Hall \(1973\)](#); [Roca and Loseille \(2019\)](#). For example, solving partial differential equations (PDE) using finite element methods requires target geometric regions or objects to be discretized into polygonal or polyhedral meshes first; topology optimization utilizes finite element analysis to study the behaviors of geometric objects ([K. Zhang, Cheng, & Xu, 2019](#)). It is also identified as one of the six basic research directions in NASA's Vision 2030 CFD study ([Slotnick et al., 2014](#)).

The purpose of mesh generation is to discretize complex geometries into a finite set of (geometrically simple and bounded) elements, such as triangles or quadrilaterals (in 2D geometries) or

tetrahedral or hexahedral (in 3D geometries). Although triangular and tetrahedral meshes have the availability of fast and robust generators, quadrilateral and hexahedral meshes are particularly favored by many applications for their ability to model structure behaviors with less computational power and higher accuracy (Docampo-Sanchez & Haines, 2019). However, quadrilateral and especially hexahedral are harder problems, and the existing generation methods are constantly the significant bottleneck for the simulations of FEA/CFD, due to the computational complexity, low mesh quality in complex geometries, and the dependence of heuristic knowledge in algorithm development (Rushdi, Mitchell, Mahmoud, Bajaj, & Ebeida, 2017; Slotnick et al., 2014; Zhao, Chen, Zheng, Huang, & Zheng, 2015). There is an actual demand for an automatic mesh generator that would work on arbitrarily complex geometries while maintaining high mesh quality.

There are mainly two types of methods: structured and unstructured mesh generators. All elements in structured meshes have the same valence in their vertices and are arranged in regular patterns (Thompson, Soni, & Weatherill, 1998). Most real-world engineering problems have complex geometric boundaries; structured meshes have poor mesh quality near the domain boundaries; hence, unstructured meshes are preferred, which can adapt the mesh structure to complex boundary shapes (Bommes et al., 2013; Garimella, Shashkov, & Knupp, 2004; Owen, 1998; Remacle et al., 2012).

Conventional methods for unstructured quadrilateral mesh could be classified into two categories: indirect and direct methods (Shewchuk, 2012). Indirect methods start with a triangular mesh and then transform the triangular elements into quadrilateral elements by various strategies, including optimization (Brewer, Diachin, Knupp, Leurent, & Melander, 2003), refinement and coarsening (Garimella et al., 2004), simplification (Daniels, Silva, Shepherd, & Cohen, 2008), perfect matching (Remacle et al., 2012). These methods, however, suffer from a large number of irregular vertices, which is undesired in numerical simulations.

Direct methods generate quadrilateral elements directly. These methods include 1) advancing front

technique, which recursively generates elements from the domain boundary and updates its boundary inwardly by cutting the generated elements until the whole domain is filled with quadrilateral elements (Blacker & Stephenson, 1991; White & Kinney, 1997; Zeng & Cheng, 1993; Zhu, Zienkiewicz, Hinton, & Wu, 1991); 2) modifying the quadtree background grid to conform to the domain boundaries (Atalay, Ramaswami, & Xu, 2008; Baehmann, Wittchen, Shephard, Grice, & Yerry, 1987; Liang, Ebeida, & Zhang, 2010; Liang & Zhang, 2012); 3) packing techniques, including square packing (Shimada, Liao, & Itoh, 1998) and circle packing (Bern & Eppstein, 2000); and 4) template-based mapping methods (Gengdong & Hua, 1996). However, the generated quadrilateral meshes are usually not complete (e.g., containing triangular elements), having flat or inverted quadrilateral elements, and too much irregular arrangement. Therefore, a large amount of cleanup operations are implemented to improve the mesh quality. The strategies range from pre-processing, such as dividing complex geometries into small regular regions, to generating regular mesh (C. Liu, Yu, Chen, & Li, 2017), to post-processing, such as reducing the singularity (Verma & Suresh, 2017), performing iterative topological changes (e.g., splitting, swapping, and collapsing elements) (Docampo-Sanchez & Haimes, 2019), and mesh adaptation (Verma & Suresh, 2018).

Therefore, many extra efforts are needed to improve meshing algorithms, including using heuristic cleanup operations to reduce the flat or inverted elements by re-adjusting location and connectivity of element vertices; using global and local remeshing techniques to reconstruct generated mesh in terms of topological and geometric features (Verma & Suresh, 2017); and pre-processing the geometry to ease the element generation process (C. Liu et al., 2017). Although the mesh quality has improved, the extra treatments make the meshing algorithms suffer from high complexity and speed limitations, and incur expensive trial-and-error tuning for researchers.

## 1.1 Motivation

The aforementioned work names only a few recent advancements towards the automation of mesh generation. Although those extra treatments could achieve the high-quality, they bring in additional computational expense besides already complicated meshing algorithms and decrease the automation to some extent. Moreover, algorithm development is usually heuristic and requires human

designers to search for knowledge in a time-consuming manner. It is urgent and necessary to build an efficient computational framework for mesh generation to sidestep the complexities and computational burden of existing meshing algorithms and relieve human algorithm designers from an inefficient and incomplete search of heuristic knowledge. This framework should provide high-quality meshes for various complex geometries while maintaining robust automation, and avoiding extra treatments.

Many researchers combine mesh generation with artificial intelligence algorithms, including expert systems (Zeng & Cheng, 1993), and neural networks (NNs) (Papagiannopoulos, Clausen, & Avelan, 2021; Vinyals, Fortunato, & Jaitly, 2015; Yao et al., 2005; Z. Zhang, Wang, Jimack, & Wang, 2020), to overcome those difficulties in algorithm development. However, these methods are still not mature and cannot replace standard mesh generation algorithms in the industry. There are a few reasons: 1) some methods have complex NN structures and are hard to train a robust generator; 2) their problem formulation makes the generator barely adaptable to complex geometric domains; 3) their training datasets are often low quality and imbalanced, causing low learning efficiency and compromised model performance.

Among the existing unstructured mesh generation methods, element extraction methods provide the optimal mesh with high quality and have strong adaptability to complex geometry shapes. However, it is difficult, expensive, and time-consuming for human algorithm designers to develop high quality and robust element extraction rules, even for two-dimensional domains. Their applications have been largely limited. The rapid progress of deep learning techniques offers the potential for element extraction methods to have radical advances to overcome these obstacles.

## **1.2 Objective**

The objective of this thesis is to propose a highly cost-effective system that can automatically design a self-learning 2D element extraction mesh generation algorithm.

The objective can be achieved by three actions:

- (1) To design smart element extraction rules adaptable to complex geometries while maintaining high mesh quality via a design methodology.
- (2) To design a self-learning mechanism to automatically acquire robust and high-quality element extraction rules using machine learning algorithms.
- (3) To increase the sampling and learning efficiency of extraction rules by examining the correlations between the input and output of the rule, and hence directly generating high quality sample data.

### **1.3 Outline**

This thesis is organized as follows: In Chapter 2, we review state-of-the-art techniques for the three subjects we are interested in: (1) quadrilateral mesh generation, (2) reinforcement learning, and (3) imbalanced learning. In Chapter 3, we detail our contributions. Chapter 4 investigates a self-learning finite element extraction system. Chapter 5 proposes a reinforcement learning based computational framework for automatic mesh generation. Chapter 6 designs a quality function-based data generation method for the optimal performance of a mesh generation algorithm. Finally, conclusions as well as our perspectives are presented In Chapter 7.



## Chapter 2

# Literature Review

The main objective of this research is to design a smart mesh generation system to sidestep the complexities and computational burden of existing meshing algorithms, relieve human algorithm designers from an inefficient and incomplete search of heuristic knowledge, and provide high-quality meshes for various complex geometries while maintaining robust automation and avoiding any extra treatments. This objective can be achieved through the previously mentioned three actions:

- (1) We have designed three primitive element extract rules adaptable to arbitrary complex geometries based on a design methodology. These rules correspond to three basic geometry situations. Any geometry can be recursively represented by or divided into a sequence of basic situations.
- (2) We have designed a self-learning mechanism to automatically acquire robust and high-quality element extraction rules based on a reinforcement learning (RL) algorithm and a feedforward neural network (FNN). To reduce further human intervention, we have proposed a sole RL based learning method to obtain the rules.
- (3) We have also investigated the correlations between the input and output of the rule, and defined a quality function-based data generation method to increase the sampling and learning efficiency of extraction rules.

This chapter will review existing technologies for mesh generation, reinforcement learning, and imbalanced learning.

## 2.1 Mesh generation

### 2.1.1 Conventional methods

According to the connectivity of quadrilateral elements in meshes, there are structured and unstructured meshes. All elements in structured meshes have the same valence in their vertices and are arranged in regular patterns (Thompson et al., 1998). Since the geometries in real-world simulation problems have complex boundary shapes, structured meshes are not favored because of their poor mesh quality near the domain boundaries. In contrast, unstructured meshes have strong adaptability to complex boundary shapes and are frequently adopted (Bommes et al., 2013; Garimella et al., 2004; Owen, 1998; Remacle et al., 2012).

Existing approaches for unstructured mesh generation can be classified into two categories: indirect and direct methods (Shewchuk, 2012). Indirect methods for quadrilateral mesh generation often start with a triangular mesh and then transform triangular elements into quadrilateral elements (Garimella et al., 2004). The classical transformation strategies may include optimization (Brewer et al., 2003), refinement and coarsening (Garimella et al., 2004), or simplification (Daniels et al., 2008). Some indirect methods split the triangle mesh into three quadrilateral elements by adding a mid-point (Peters & Reif, 1997; Prautzsch & Chen, 2011). Owen, Staten, Canann, and Saigal (1999) proposed Q-Morph that created an all-quadrilateral mesh by following a sequence of systematic triangle transformations. Q-Morph, however, required heuristic operations such as topological cleanup and smoothing to guarantee the quality of the final quadrilateral mesh. Ebeida, Karamete, Mestreau, and Dey (2010) provided an indirect method, named Q-Tran, which is of provably good quality and smoothing post-processing free in generating quadrilateral elements.

Remacle et al. (2012) proposed a fast-indirect method, Blossom-Quad, to generate all-quadrilateral meshes by finding the best matching triangular pairs and combining them into quadrilaterals. The quality of the generated quadrilateral mesh is not stable and limited by the initial triangulation.

Some triangles are easy to remain on the boundaries and appear pairwise. [Remacle et al. \(2013\)](#) proposed an advanced method, DelQuad, to overcome the limitations of Blossom-Quad by generating triangular mesh suitable to be recombined into high-quality quadrilateral meshes. However, it is a challenge for all the triangles to combine into quadrilateral elements. The class of indirect methods often suffers from many irregular vertices, which is undesired in numerical simulations. Therefore, some methods provide post-processing to improve this irregular situation. For example, [Verma and Suresh \(2017\)](#) proposed a robust approach to increase the topological quality of generated quadrilateral meshes by reducing the singularity for many existing approaches; [Docampo-Sanchez and Haimes \(2019\)](#) described a technique to recover the regularity by performing iterative topological changes, such as splitting, swapping, and collapsing, on the quadrilateral mesh generated by Catmull–Clark subdivision ([Catmull & Clark, 1978](#)).

Direct approach for quadrilateral mesh generation is to construct quadrilateral elements without any intermediate triangular mesh. [Zhu et al. \(1991\)](#) built a mesh generator for quadrilateral elements based on the advancing front technique. The quadrilateral elements were generated sequentially or in parallel in each sub-boundary. [Blacker and Stephenson \(1991\)](#) proposed an approach named Pave, to generate quadrilateral meshes directly in an iterative way from the boundaries of the input domain towards the interior of the domain. [White and Kinney \(1997\)](#) improved the robustness of the original Paving algorithm by changing the generation method from row by row to element by element. The Paving algorithm is currently implemented as part of the CUBIT software ([Blacker et al., 2016](#)). The challenge to these methods is that their ruleset is not complete, and their usages involve too much heuristic knowledge. The meshing stability is thus hard to ensure, causing heuristic cleanup operations.

[Zeng and Cheng \(1993\)](#) proposed FreeMesh, a knowledge-based method, to recursively generate quadrilateral elements from the domain boundary until the whole domain was filled with quadrilateral elements. The mesh generation was considered as a design problem and could be resolved recursively into atomic design and a sub-design problem ([Zeng & Cheng, 1991](#); [Zeng & Yao, 2009](#)). The atomic design was to generate an element based on the boundary situation. The author analyzed

the geometry boundary characteristics and proposed three primitive boundary situations. Correspondingly, three primitive element generation rules (i.e., adding one, two, and three edges) for each situation were designed, which could theoretically guarantee the completion of any geometries. After cutting the generated element, the original boundary was updated and became a new sub-design problem. This process recursively repeated until the sub-design problem turned into an atomic design. Although the rules were limited to just a few, each rule still involved human heuristic knowledge.

Some methods proposed a quadrilateral mesh generator by modifying the quadtree background grid to conform to the domain boundaries. They often started with a uniform Cartesian background grid or a quadtree structure generated based on the local feature sizes. Quadrilaterals that conform to the domain boundaries, are then fitted into that grid or quadtree leaves. [Baehmann et al. \(1987\)](#) was among the first methods to modify a balanced quadtree to generate quadrilateral meshes. [Liang et al. \(2010\)](#) proposed a quadtree-based method to create guaranteed-quality and adaptable meshes. They specifically designed four categories of templates to adjust the boundary edge to improve the angle along the boundary thus. [Liang and Zhang \(2012\)](#) improved their previous work in preserving better angle quality and boundary conforming by adding a sharp feature layer. Grid-based algorithms were robust but often generated poor quality elements at the boundary.

There are also domain decomposition methods. The geometry domains are divided into simpler and regular convex or mappable regions, and the meshes could be generated for each sub-region via template-based, mapping, or geometric algorithms. Various techniques can achieve decomposition. [Tam and Armstrong \(1991\)](#) introduced a medial axis decomposition approach. The medial axis could be considered a series of lines and curves derived from the midpoints of maximal circles in the area. [Joe \(1995\)](#) decomposed the domain into convex polygons using geometric decomposition algorithms. [Quadros, Ramaswami, Prinz, and Gurumoorthy \(2004\)](#) proposed an algorithm to divide a complex domain into a sub-domain using a medial axis and utilized an advancing front method for adaptive meshes. [Gengdong and Hua \(1996\)](#) proposed a template-based mapping method. There are some methods using square packing ([Shimada et al., 1998](#)) and circle packing ([Bern & Eppstein, 2000](#)) to generate all quadrilaterals. Circle packing can only bound the maximum angle to  $120^\circ$ .

[Atalay et al. \(2008\)](#) utilized a quadtree to construct quadrilateral mesh with a guaranteed minimum angle bound of  $18.43^\circ$ . Generally, these methods could produce high internal quality, but they are not robust and may require heuristic cleanup operations, especially if the domain has non-manifold boundaries.

In general, among the direct unstructured mesh generation methods, the element extraction method provides the highest quality and is preferred for applications that need high quality boundary meshes ([Docampo-Sanchez & Haimes, 2019](#); [Park, Noh, Jang, & Kang, 2007](#)). However, all the methods cannot guarantee the mesh quality; the generated quadrilateral meshes are usually not complete (e.g., containing triangular elements), have flat or inverted quadrilateral elements, and too much irregular arrangement. Therefore, a large number of cleanup operations are implemented to improve the mesh quality. The strategies range from pre-processing, such as dividing complex geometries into small regular regions, to generating regular mesh ([C. Liu et al., 2017](#)), to post-processing, such as reducing the singularity ([Verma & Suresh, 2017](#)), performing iterative topological changes (e.g., splitting, swapping, and collapsing elements) ([Docampo-Sanchez & Haimes, 2019](#)), and mesh adaptation ([Verma & Suresh, 2018](#)). [Rushdi et al. \(2017\)](#) aimed to solve the angle and post-processing problem and developed an all-quadrilateral algorithm to achieve better quality angles, which falls in the range  $[45^\circ, 135^\circ]$ , without any cleanup operations.

### **2.1.2 Machine learning-based methods**

Machine learning (ML) techniques have been approved successfully to solve complex and time-consuming problems in modern industry. Many researchers have started to combine mesh generation with artificial intelligence techniques to reduce the dependence on heuristic knowledge. Existing work can be classified into three kinds: 1) mesh optimization, 2) mesh reconstruction, and 3) mesh generation. First, mesh optimization refers to optimizing the mesh quality using ML techniques. [Jadid and Fairbairn \(1994\)](#) utilized an NN to remesh a square shaped structure using triangular elements. [Chedid and Najjar \(1996\)](#) proposed an artificial neural network (ANN) based method to predict the mesh density distribution based on the geometric features to reduce the computation time of mesh refinements. The mesh density at a given vertex of a mesh referred to the

number of the vertices vicinity of that vertex.

Some methods are designed to reduce the computational cost in numerical simulation by refining meshes. [Capuano and Rimoli \(2019\)](#) proposed a smart finite element method to reduce the computational cost performing numerical iteration. [Z. Zhang et al. \(2020\)](#) proposed a MeshingNet method that combined deep neural networks (NNs) with an external mesh generator to refine meshes with a better element distribution that could accurately solve Partial Differential Equations (PDE) with FEM. Traditionally, to generate such a mesh needed a posteriori estimation, which was computationally expensive. MeshingNet could directly learn the a posteriori error from the initial mesh from an existing generator, and predict a new mesh density for refinement. It could also be used to adjust the mesh with a desired number of elements. [J. Yang et al. \(2021\)](#) trained a mesh refinement policy to dynamically adjust the mesh resolution for a better trade-off between simulation accuracy and computation cost via RL. Their main idea was to reallocate the computational budget to regions where a higher resolution was needed because uniform meshes were computationally inefficient. [L. Zhang et al. \(2021\)](#) proposed a numerical discretization scheme, the hierarchical deep learning neural network for finite element method, to solve the PDE. The shape function of the element was estimated by a deep neural network (DNN). By optimizing the nodal information, the interpolant accuracy can thus be improved.

Second, mesh reconstruction is to reconstruct meshes from images or other sources. Neural Mesh Flow (NMF) ([Gupta, 2020](#)) was a 3D mesh reconstruction method by deforming a template mesh into a target mesh using several Neural Ordinary Differential Equation (NODE, a deep neural network model ([Chen, Rubanova, Bettencourt, & Duvenaud, 2018](#))) blocks. NMF had its strength in the mesh property of manifoldness, which was beneficial for graphic rendering and 3D printing. Pixel2mesh ([N. Wang et al., 2018](#)) was a deep learning architecture to produce 3D triangular mesh from a single RGB image. The generation process was a series of deformation from an ellipsoid with perceptual image features extracted by a convolutional neural network (CNN) to a target mesh model represented by a Graph Convolutional Network (GCN ([Defferrard, Bresson, & Vandergheynst, 2016](#))). [Wen, Zhang, Li, and Fu \(2019\)](#) extended the Pixel2mesh for better shape quality from multi-view images. MeshCNN ([Hanocka et al., 2019](#)) utilized convolutional neural

network (CNN) architecture with convolution, pooling, and unpooling layers to conduct triangular mesh simplification while retaining the topological features of the domain. The simplification was executed by collapsing less informative edges using the pooling and unpooling operations, and would then facilitate the data processing and decrease computational cost. These recent advances show that mesh related works have gained significant attention in computational graphics and computer vision.

Last, mesh generation is to generate a mesh with ML algorithms. [Dolšak \(2002\)](#) proposed a rule-based expert system for guiding mesh design, such as the choice of element types and the specification of mesh density. [Manevitz, Yousef, and Givoli \(1997\)](#) combined an expert system with a self-organizing neural network ([Kohonen, 2012](#)) to generate mesh elements (triangular or quadrilateral). The expert system could determine the size of the mesh and appropriate densities in different regions of a domain based on geometric and physical considerations, while the self-organizing neural network realized the mesh by assigning the coordinates to nodes. To overcome the drawbacks of SOM in tackling inaccurate mesh in the domain border and mesh construction for non-convex domains, [Nechaeva \(2006\)](#) proposed an adaptive mesh generation algorithm based on self-organizing maps (SOM) (an unsupervised neural networks-based method), which adapts a given uniform mesh onto a target physical domain through mapping. However, many poorly shaped elements were still generated, especially along the domain boundary.

Some works focus on triangular mesh generation using neural networks (NNs). Pointer networks ([Vinyals et al., 2015](#)), was a new neural architecture that aimed to solve combinatorial problems by NNs and could be used to generate triangular meshes by outputting a set of triplets of integers (each forms a triangle) that correspond to the order of input points. The input contains both the points on the boundary and internal area of the geometry domain. As it is not designed for meshing problems, the final mesh is not robust and partially covered with triangular elements with even intersecting edges. [Papagiannopoulos et al. \(2021\)](#) proposed a triangular mesh generation method with three NNs. Given the training data derived from Constrained Delaunay Triangulation algorithm ([Chew, 1989](#)), three NNs could predict the number of candidate inner vertices (to form a triangular element), coordinates of those vertices, and their connection relations with existing segments on

the boundary, respectively. Because of the fixed input size and complex network architectures, the method cannot adapt to arbitrary and complex geometry domains. Additionally, the resulting meshes are constrained by the quality and sample diversity of the selected training data, especially when it comes to geometries with complex boundary shapes. They also cannot be applied to mesh generation or element extraction of quadrilaterals.

Zeng and Cheng (1993) proposed a rule-based expert system method, FreeMesh, to recursively extract quadrilateral elements following a domain boundary until the remaining boundary becomes a quadrilateral element, based on the recursive logic of design (Zeng & Cheng, 1991). Since the usage of rules requires much heuristic knowledge, Yao et al. (2005) improved the FreeMesh approach by introducing an ANN to learn the element extraction rules from a set of pre-selected samples for good quality quadrilateral meshes. The improved method eased the acquisition and application of these rules and could handle more complex boundary domains. The trained model by ANN served as the mapping function to transform the input (a specific boundary situation) to the output (rule type and a newly generated vertex). The element extraction process remained the same with FREEMESH. However, the training data was difficult to prepare and cover all the possible boundary situations.

To conclude, although many artificial intelligence (AI) techniques have started to be explored and applied to the field of mesh generation, a robust and computational mesh generation method that can replace the conventional mesh generation algorithms is still missing. With the rapid progressing of High Performance Computing (HPC), mesh generation algorithms will intuitively encounter tasks requiring higher resolution and fast and reliable processing. Conventional methods will be problematic by then. It remains open to exploring with great potential for a novel meshing algorithm that exploits both emerging HPC capabilities and machine learning/deep learning algorithms. In general, supervised NN-based methods rely on labelled training data from existing conventional methods or handmade data, which cannot guarantee the overall quality and adapt to unseen domain boundaries; some of them also have complex network structures, which increases the training difficulty; and some are barely to apply to other complex domains because of their fixed input size and generation strategies.



## 2.2 Reinforcement learning

Reinforcement learning (RL) is simultaneously a problem, a class of solution methods that work well on the problem, and the field that researches this problem and its solution methods (Kaelbling, Littman, & Moore, 1996; Sutton & Barto, 2018). The problem usually refers to the optimal control or the sequential decision making problem, mathematically known as the Markov decision process (MDP) problem. The hypothesis behind RL is that all the goals can be represented by the maximization of the expected cumulative reward. The technique enables an agent to learn from the interactions with its environment by trial-and-error via reward feedback from its actions and experiences. Eventually, a policy will be learned by maximizing the accumulated reward and to guide the agent to select actions under each environmental situation appropriately. According to different learning fashions, existing model-free RL methods can be categorized into direct (policy-based) and indirect (value function-based) methods.

Value function-based methods estimate the expected accumulated reward starting from a given state or performing a given action in a given state for the agent. The optimal policy is implicitly derived by choosing the action with the optimal value. Deep Q-Network (DQN) was a recent breakthrough in estimating value function via a deep neural network (Mnih et al., 2013), which stabilized the training of action value function using experience replay and target network and generalized a framework for end-to-end RL tasks using the same algorithm. Many successors are continuously improving DQN from various aspects, e.g., asynchronous advantage actor-critic (A3C) (Mnih et al., 2016), dueling network to better estimate action value function (Z. Wang et al., 2016), and combating sparse reward issues by Hindsight Experience Replay (HER) (Andrychowicz et al., 2017). Since the element extraction method of quadrilateral mesh is intuitively an MDP, the mesh generation problem can be cast as an RL problem. For the mesh generation problem, the action is continuous and requires choosing appropriate vertices to form an element from an appropriate area. Correspondingly, the meshing boundary can be considered as the environment; it is altered and updated into various shapes after each action. Hence, to discretize both state and action spaces is not ideal, which may distort the feedback regarding the impact of the agent's actions on the environment and adversely hinders exploring feasible action space, causing a sub-optimal policy.

Policy-based methods directly estimate a policy to map a state to an action or a distribution over actions. The policy in modern methods is usually modelled by a parameterized function using NNs. Its estimation is often combined with the value function estimation to increase stability and reduce variance. According to the sampling methods, there are on-policy methods and off-policy methods. On-policy methods are sample inefficient because they only use the data once, with estimations based on the trajectories from the current policy. Asynchronous advantage actor-critic (A3C) (Mnih et al., 2016) integrated policy gradient with the online critic. Proximal Policy Optimization (PPO) (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017) was a representative method that utilized the trust region to monotonically improve the policy, with simpler implementation, better generalization, and better sample efficiency.

Off-policy is much more sample efficient; they can learn from any trajectories from any policies. Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) concurrently learned the policy and the Q-function. The Q-function was learned by experience replay and then it was used to learn the policy. Although the sampling efficiency of DDPG was improved, it was very sensitive to hyperparameter tuning. To achieve faster learning efficiency and better stability in hyperparameters tuning, soft actor-critic (SAC) (Haarnoja, Zhou, Abbeel, & Levine, 2018; Haarnoja, Zhou, Hartikainen, et al., 2018) was proposed by simultaneously maximizing the accumulated return and the policy entropy. Since the state and action spaces of the mesh generation problem are continuous, off-policy methods are most appropriate to learn the meshing policy in fast and stable fashions.

Although many concepts, algorithms, and issues in RL are studied, their theoretical understanding is still missing (Li, 2018; Osband et al., 2019). Some common challenges for RL problems are credit assignment, sparse reward, sample efficiency, and exploration and exploitation. Specifically, there is an urgent research challenge for the applicability of RL in real-world problems: the systematic and comparative study of deep RL algorithms. Many RL algorithms are hyperparameters sensitive, and their performance varies significantly across different hyperparameter settings and different problem environments. Their theoretical analysis is still in its infancy. The principled benchmarks are needed to help develop an understanding of the strengths and weaknesses of RL algorithms (Osband et al., 2019). Other vital challenges may include algorithm generalizability, limited samples in real-life

problems, partial observable tasks, reward function design, and optimal state representation (Dulac-Arnold, Mankowitz, & Hester, 2019). For instance, Wei, Wang, Liu, and Polycarpou (2020) had to train different models for various elevator settings in optimal elevator control because of the fixed representation of the state.

## 2.3 Imbalanced learning

High quality and balanced data are vital for the performance of data-driven algorithms. However, in real life tasks, the data is unequally distributed; high quality data is rare to collect. The quantity and quality of the data decide the performance of data-driven machine learning algorithms. A large amount of data is essential to establish a complex decision boundary for classification problems and avoid overfitting problems, especially in deep-learning methods entailing many parameters (LeCun et al., 2015). Imbalanced data can quickly compel the prediction of algorithms towards majority classes (He & Garcia, 2009). However, the minor classes are often more important and valuable. Existing methods can be categorized into data-level and algorithm-level methods.

### 2.3.1 Data-level approaches

Data augmentation/synthesis are the techniques to enlarge data size by adding slightly modified samples or newly generated artificial data from the existing dataset (He & Garcia, 2009; Krawczyk, 2016). Conventional methods concentrate on modifying the existing dataset to have a balanced distribution. Two common strategies are 1) removing examples from majority groups (undersampling) and 2) increasing new objects for minority groups (oversampling). The random undersampling method selects samples from the majority groups without replacement and removes them from the dataset. X.-Y. Liu, Wu, and Zhou (2008) proposed two informed undersampling methods to solve the deficiency of information loss in the traditional random undersampling. Yen and Lee (2009) devised the under-sampling method based on clustering (SBC) to avoid the disjunct problem.

The random oversampling method is a basic method, and it randomly selects minority examples and replicates them to the dataset. Since this method can easily cause an overfitting problem, synthetic sampling methods are thus developed. Chawla, Bowyer, Hall, and Kegelmeyer (2002) developed

a method called SMOTE, to create artificial data based on the features space similarities between minority examples. As SMOTE was very successful in various applications, there were a few improvements to the SMOTE, including borderline-SMOTE (Han, Wang, & Mao, 2005), safe-level-SMOTE (Bunkhumpornpat, Sinapiromsaran, & Lursinsap, 2009), and Adaptive Synthetic Sampling (ADS-SYN) (He, Bai, Garcia, & Li, 2008). Jo and Japkowicz (2004) proposed cluster-based over-sampling (CBO) to resolve the small disjunct problem caused by within-class imbalance. Some researchers are using the structure information underlying the data to generate synthetic samples with better quality. Xie, Jiang, Ye, and Li (2015) tried to learn the cluster structure of the training samples and generated samples for minority groups based on the density. C.-L. Liu and Hsieh (2019) proposed a model-based synthetic sampling (MBS) to increase the data diversity by capturing the relationship between data features via regression models. Markov chain models are also utilized to synthesize time-varying stochastic data, such as wind speed (Shamshad, Bawadi, Hussin, Majid, & Sanusi, 2005) and vehicle velocity in driving cycles (Lee & Filipi, 2010), by constructing transition matrices. W. Yang and Nam (2022) proposed a covariance matrix-based method that used a) the correlations between features obtained from the original dataset and b) random noise.

Many neural network-based data synthesis models are constantly proposed. Habibie, Holden, Schwarz, Yearsley, and Komura (2017) built a generative model for human motion samples based on variational autoencoder (VAE). Y. Yang, Zha, Chen, Wang, and Katabi (2021) utilized deep neural networks to represent the feature space of data samples. Generative adversarial network (GAN) has been widely used for data synthesis due to its flexibility and efficiency in high dimensional datasets (Tanaka & Aranha, 2019). Z. Wang, Wang, and Wang (2018) combined GAN with autoencoders and verified the synthetic data for the vibration signal of the gearbox. Xuan et al. (2018) explored the integration of convolutional neural networks (CNN) and GAN on the generation of pearl images. Despite its wide usage, the data generated by GAN based methods can be low quality if 1) the training of generator is not stable, 2) the size of the original dataset is small, and 3) the dimension and nonlinearity of the dataset is high (Tanaka & Aranha, 2019; W. Yang & Nam, 2022).

### 2.3.2 Algorithm-level approaches

Instead of creating balanced data distributions through different sampling strategies, algorithm-based methods directly modify the existing learners to alleviate the bias towards majority groups and adapt them to the training dataset having skewed distributions, finally enhancing the algorithm's performance (He & Garcia, 2009). These methods require a thorough understanding of the modified learning algorithm and an exact identification of the causes for its failures in the imbalanced training data (Krawczyk, 2016). The most dominant branch is cost-sensitive approaches (Zhou & Liu, 2010). They adjust the cost matrices that incorporate a varying cost for each considered group of examples for a given learner by assigning a higher cost for minority groups. In some application domains, cost-sensitive techniques are more optimal than sampling methods (McCarthy, Zabar, & Weiss, 2005; Zhou & Liu, 2005).

Some methods adjust the misclassification cost to the dataset as a form of dataspace weighting. Sun, Kamel, Wong, and Wang (2007) proposed three methods, AdaC1, Adac2, and Adac3, which introduce the cost items that denote the uneven identification importance between classes into the learning framework of AdaBoost. The learning can be biased intentionally towards classes with higher identification importance. Fan, Stolfo, Zhang, and Chan (1999) devised a method called AdaCost, which used a cost-adjustment function to increase the weights of costly misclassifications aggressively and conservatively decrease the weights of high-cost examples. However, it is challenging to set actual values in the cost matrix and its associated cost items; they are usually not defined by domain experts.

The cost-sensitive methods can be combined with decision trees, including applying cost-sensitive adjustments to 1) the decision threshold, 2) the node split criteria, and 3) pruning schemes (Drummond & Holte, 2000; He & Garcia, 2009). Similarly, it can be introduced into neural networks by applying cost-sensitive modifications to 1) probabilistic estimate, 2) output prediction, 3) learning rate, and 4) loss function (He & Garcia, 2009; Kukar, Kononenko, et al., 1998). Khan, Hayat, Benamoun, Sohel, and Togneri (2017) proposed a cost-sensitive deep neural network to automatically

learn feature representations for both the majority and minority groups by introducing the class sensitive cost into the loss function. There are some other methods. [G. Wu and Chang \(2005\)](#) proposed a method called Kernel-boundary alignment (KBA) to consider the imbalance condition during the training state by involving not only the distance between all data points and vectors but also the class distribution of the support vectors. [Huang, Li, Loy, and Tang \(2016\)](#) defined a euclidean embedding function that considered the class- and cluster-level margins to learn the discriminative representation for imbalanced image classification problems. [Dong, Gong, and Zhu \(2018\)](#) introduced a class rectification loss function to guide the model incrementally exploring the minority class decision margins, for multi-label classification problems in deep learning.

## 2.4 Research challenges and opportunities

The main research challenges can be summarized as follows:

- (1) Conventional mesh generation methods heavily rely on heuristic knowledge and post processing operations to improve mesh quality, causing algorithm complexity and low meshing speed. Element extraction methods generally provide high quality among the existing direct unstructured methods ([Liang & Zhang, 2012](#)) and are especially preferred for applications that need high quality boundary meshes ([Docampo-Sanchez & Haimes, 2019](#); [Park et al., 2007](#)). [Zeng and Cheng \(1993\)](#) proposed the least number of element extraction rules but were sufficient to generate meshes for any complex geometries. However, the rules that ensure robust and high mesh quality are difficult, expensive, and time-consuming for human algorithm designers to develop, even for two-dimensional domains.
- (2) Machine learning-based methods are still in their infancy and are not able to replace standard mesh generators in the industry. There are a few reasons: a) some of the NNs structures are too complex and difficult to train robust generators; b) their inputs are fixed and cannot adapt to complex geometries; c) the imbalanced training datasets compromise the learning efficiency and algorithm performance.

The element extraction rules can be smartly designed and evolving using machine learning algorithms. RL is a well-known self-taught learning paradigm for solving sequential decision making problems and has been successfully demonstrated in many application domains. By formulating the mesh generation as a Markov decision process (MDP) problem, the meshing policy (i.e., element extraction rules) can be learned by the interactions between the agent and the environment (i.e., the geometries to mesh) without the need for human intervention. Compared with most existing benchmark problems and complex applications of RL (Machado et al., 2018; Osband et al., 2019), the mesh generation problem also poses new challenges for learning an optimal policy: 1) the geometries have diverse shapes and sizes, which requires the policy to be adaptable and general; 2) the step-wise reward design for obtaining the mesh is difficult since the objective is not intuitive; 3) there is a trade-off between the current element quality and the remaining environment because the generated element will shape the remaining boundary of the geometry for future meshing.

The mesh generation problem can be further used to understand many RL topics, such as state representation and reward specification. Usually, the performance of newly invented RL algorithms is evaluated over many benchmark problems (Mnih et al., 2013). However, existing benchmark problems may have some limitations: 1) the internal dynamic mechanisms of different problems may impact the performance and hardly provide accurate evaluation for RL methods, especially complex problems (Berner et al., 2019; Vinyals et al., 2017); 2) the state representations are different across different problems, which may hinder the algorithm analysis; 3) some of the problems are simple, causing a simple reward function (i.e., game scores (Mnih et al., 2013)), which cannot study the reward specification in-depth. The challenges of the mesh generation problem enable it to be a potential benchmark problem for understanding these issues. The diverse geometries can test the scalability and generalizability of the learned policy; the range of agent's observation can be easily adjusted and used to study the partial MDP problems; the reward design can be used to study exploration-exploitation and credit assignment problems.

The imbalanced and low quality data always hinder the learning efficiency and algorithm performance. High quality data and minor examples are less frequent to occur. Even with self-learning

algorithms (e.g., RL algorithms), the high quality data is still hard to collect. Data augmentation/synthesis is the technique to improve the algorithm performance by adding more valuable training data, creating data variability, preventing data scarcity, and reducing overfitting. It is also beneficial for increasing the generalization ability of the models and reducing the costs of collecting and labelling data. However, the performance improvement is limited because: 1) the data characteristics are hard to discover and represent, which often depends on the researcher's domain-specific knowledge; 2) the representative features of unseen situations are usually intrinsically missing from the existing dataset. Most simple transformations to the existing data will not fundamentally improve the algorithm performance and make the algorithm general to unseen situations. There is still a long way to devising a data generation method that captures the nature of the data, especially those with high dimensions and nonlinearity.



## Chapter 3

# Contributions

This thesis focuses on the element extraction method for quadrilateral mesh generation. Element extraction is one of the unstructured mesh generation methods and is known for its good mesh quality around the domain boundary. However, it is easy to produce bad elements in the middle of the domain which prevents it from the main-stream method. Its challenge lies in establishing a small number of robust and effective rules applicable to various boundary shapes. This thesis aims to use machine learning algorithms to train robust and effective rules for automatically generating high quality meshes.

The first contribution is the one to one correspondence between element extraction algorithm and reinforcement learning. The novelty is that the mesh generation problem is formulated into a sequential decision problem that enables it to be solved by RL techniques. In the early stage, the primitive rules to form an element in the element extraction process are manually designed (Blacker & Stephenson, 1991; Zeng & Cheng, 1993), which relies on the inefficient searching of heuristic knowledge. The mesh quality is hardly guaranteed and needs extra treatments to improve. It also increased the computational burden and slowed down the meshing speed. An artificial neural network (ANN) based method tried to replace the human design of rules by machine learning (Yao et al., 2005). However, its training data came from existing methods or hand-crafted, which cannot cover enough situations for the model to adapt. Only limited domains have been successfully meshed. To

automatically generate samples for model learning, therefore, is urgent and essential.

We find that reinforcement learning (RL) is such a method that has a self-taught paradigm to let the agent learn from the interaction with the environment through state, action, and reward. To achieve the learning automation, RL is introduced into the mesh generation problem (in Chapter 4). The element extraction process can be summarized as two main components: mesh generator and domain boundary. Mesh generator extracts an element from the current boundary state; the domain boundary evaluates the element quality and updates itself by cutting off the generated element; the mesh generator receives the newly updated state again and starts the next round of element extraction. Intuitively, the mesh generator corresponds to the agent; the domain boundary corresponds to the environment. The state is represented by the partial observation of the boundary and consists of a number of vertices. The action is defined as the primitive rules to produce the element. The reward function refers to the mesh quality for the current element and future elements. The solution to the meshing problem is to find the meshing policy  $\pi$  that maximizes the value of the initial state (i.e., the accumulated rewards in future steps).

The second contribution is the integration of a rule-based system, reinforcement learning, and FNN into one system. The novelty is that a general framework for smart mesh generation system by rule-based machine learning algorithms is proposed. The rule-based system has a number of advantages: 1) the complexity of the problem is significantly reduced by designing a few primitive rules; 2) the primitive rules can provide more generalizability; and 3) it enables the system capable of making adaptive decisions corresponding to various environmental situations. The limitation of a sole rule-based system is that the adaptive quality is low, which cannot guarantee the overall quality. Therefore, the rules should self-evolve to achieve overall high mesh quality for diverse boundary shapes. The FNN model can fast simulate the rules but requires high quality data to achieve robust performance. Reinforcement learning is hence introduced to generate high quality learning samples.

Based on this framework, we have proposed three methods: a self-learning mesh generation system (see Chapter 4); a soft actor-critic (SAC) based method (see Chapter 5); and a quality function-based

method (see Chapter 6). The first method utilized a reinforcement learning algorithm, advantage actor-critic (A2C), to generate samples by trial-and-error learning. Then an experience extraction module was applied to extract high quality samples that were fed to train an FNN model. The model was served as the final mesh generator to complete mesh.

During the development, we have found that the automation can be further improved by directly learning the rules using the RL algorithm without experience extraction and FNN training. Therefore, the second method, SAC-based meshing method, is proposed. The reward function that measures the action quality under each state is improved to guarantee the overall mesh quality and the mesh completion in finite steps. Through this process, we realize that the input and output of the rules (i.e., state and action) can be directly measured by the adapted reward function (i.e., the quality function). The third method is hence proposed to use the quality function as the criteria to directly generate balanced high quality data for an FNN model training. It will significantly reduce the time required for RL to explore all kinds of boundary situations in finding appropriate actions.

The last contribution is to propose the mesh quality function that simultaneously takes into account the quality of the existing mesh and future mesh. The quality function is used as the critical component (i.e., reward function) in RL and the criteria to generate high quality data to train the ML-based mesh generator. The novelty of the mesh quality function is that it creatively resolved the challenges of the element extraction method. We have determined three types of primitive rules in the element generation, including adding 1, 2, and 3 edges, respectively. The challenges lie in two questions (see Chapter 4): 1) which rule should be selected among the three primitive rules under a specific boundary situation? 2) How to determine relationships between the coordinates of the newly added points and the geometric conditions? To resolve them, three kinds of quality terms are considered, including element quality  $\eta_e$ , boundary quality  $\eta_b$ , and density factor  $\mu$ . The element quality measures the quality of the generated element by jointly considering the edge quality and angle quality of the element. The boundary quality function is measured by the newly formed angles between the boundary and the generated element and the distance between the newly inserted vertex and the surrounding boundary edges. The reason to include this quality is that the remaining boundary

influences the generation of future elements. The density factor is specifically used in RL algorithms to guarantee the meshing can be complete in finite steps. It controls the size of the generated element.

Since the objectives of the three proposed methods are different, the mesh quality function is also slightly changed from method to method. In the first self-learning system (see Chapter 4), the reward function is only formed by the element quality and boundary quality; in the SAC-based method (see Chapter 5), it considered all three terms. But there are some improvements about the element and boundary quality. The boundary quality additionally measures a distance between the newly inserted vertex and the boundary edges to avoid mesh collision. The last quality function-based method (see Chapter 6) only considers the changed element and boundary quality with different weights.

To evaluate the meshing performance of the proposed methods, we have selected eight kinds of quality indices. Compared with a number of commercial software, the proposed methods:

- (1) are comparable on all the selected quality indices;
- (2) have the best performance in having no triangle elements;
- (3) have slightly better performance in some other indices (i.e., taper, stretch, and singularity);
- (4) have suboptimal performance in some indices (i.e., element quality,  $|MinAngle-90|$ ,  $|MaxAngle-90|$ , scaled Jacobian) but not in a significant manner.

All those contributions together have brought forward a highly cost-effective system that can automatically design a self-learning 2D element extraction mesh generation algorithm, which can perform with a comparable quality to commercial systems.

## **Chapter 4**

# **A self-learning finite element extraction system based on reinforcement learning**

### **Abstract**

Automatic generation of high-quality meshes is the base of CAD/CAE systems. The element extraction is a major mesh generation method for its capabilities to generate high-quality meshes around the domain boundary and to control local mesh densities. However, its widespread applications have been inhibited by the difficulties in generating satisfactory meshes in the interior of a domain or even in generating a complete mesh. The primary challenge in the element extraction method is to define element extraction rules for achieving high-quality meshes in both the boundary and the interior of a geometric domain with complex shapes. This paper presents a self-learning element extraction system FreeMesh-S that can automatically acquire robust and high-quality element extraction rules. The FreeMesh-S is enabled by two central components: 1) three primitive structures of element extraction rules, which are constructed according to boundary patterns of any geometric boundary shapes; 2) a novel self-learning schema, which is used to automatically define and refine

the relationships between the parameters included in the element extraction rules, by combining an Advantage Actor-Critic (A2C) reinforcement learning network and a Feedforward Neural Network (FNN). The A2C network learns the mesh generation process through random mesh element extraction actions using element quality as a reward signal and produces high-quality elements over time. The FNN takes the mesh generated from the A2C as samples to train itself for the fast generation of high-quality elements. FreeMesh-S is demonstrated by its application to two-dimensional quad mesh generation. The meshing performance of FreeMesh-S is compared with three existing popular approaches on ten pre-defined domain boundaries. The experimental results show that even with much less domain knowledge required to develop the algorithm, FreeMesh-S outperforms those three approaches in important indices. FreeMesh-S significantly reduces the time and required expertise to develop high-quality mesh generation algorithms.

*Keywords:* Smart design, smart mesh generation, self-learning system, reinforcement learning, artificial neural networks, element extraction

## 4.1 Introduction

Automatic mesh generation is an important yet not well-solved problem essential to the numerical computation for CAD/CAE applications, including finite element analysis, computational fluid dynamics, and geometric modeling (Gordon & Hall, 1973; Roca & Loseille, 2019). For example, solving partial differential equations (PDE) using finite element methods requires target geometric regions or objects to be discretized into polygonal or polyhedral meshes first; topology optimization utilizes finite element analysis to study the behaviors of geometric objects (K. Zhang et al., 2019). The mesh quality greatly affects the accuracy, stability, and efficiency of those engineering applications. Quad meshes are particularly favored by many applications for their ability to model structure behaviors with less computational power and higher accuracy (Docampo-Sanchez & Haines, 2019). The quality of a finite element mesh is measured by metrics related to geometrical and topological properties of the mesh (Pébay et al., 2008; Verma & Suresh, 2017). Common metrics for a quad mesh include minimum and maximum angles, aspect ratio, stretch, taper, and singularity (Rushdi et al., 2017). In general, it is impossible for a mesh to achieve high scores on every metric because

of the complexities of the boundary shapes and the need for a minimum number of elements in a mesh.

#### **4.1.1 Motivation: why is it necessary to use the element extraction method?**

Conventionally, according to the connectivity of quadrilateral elements in meshes, there are structured and unstructured meshes. All elements in structured meshes have the same valence in their vertices and are arranged in regular patterns (Thompson et al., 1998). Most real-world engineering problems have complex geometric boundaries, structured meshes would have poor mesh quality near the domain boundaries; hence, unstructured meshes are preferred, which can adapt the mesh structure to complex boundary shapes (Bommes et al., 2013; Garimella et al., 2004; Owen, 1998; Remacle et al., 2012).

Existing approaches for unstructured mesh generation can be classified into two categories: indirect and direct methods (Shewchuk, 2012). Indirect methods for quad mesh generation usually start with a triangular mesh and then transform the triangular elements into quadrilateral elements (Garimella et al., 2004). The classical transformation strategies may include optimization (Brewer et al., 2003), refinement and coarsening (Garimella et al., 2004), or simplification (Daniels et al., 2008). Some indirect methods split the triangle mesh into three quadrilateral elements by adding a mid-point (Peters & Reif, 1997; Prautzsch & Chen, 2011). Owen et al. (1999) proposed Q-Morph that created an all-quadrilateral mesh by following a sequence of systematic triangle transformations. Q-Morph, however, required heuristic operations such as topological cleanup and smoothing to guarantee the quality of the final quad mesh. Ebeida et al. (2010) provided an indirect method, named Q-Tran, which is of provably-good quality and smoothing post-processing free in generating quadrilateral elements. Remacle et al. (2012) proposed a fast-indirect method, Blossom-Quad, to generate all-quadrilateral meshes by finding the best matching triangular pairs and combining them into quadrilaterals. The quality of the generated quad mesh is not stable and limited by the initial triangulation. Remacle et al. (2013) proposed an advanced method, DelQuad, to overcome the limitation of Blossom-Quad by generating triangular mesh suitable to be recombined into high-quality quad meshes. However, it is a challenge for all the triangles to form quadrilateral elements. The

class of indirect methods often suffers from a large number of irregular vertices, which is undesired in numerical simulations. Therefore, some methods provide post-processing to improve this irregular situation. For example, [Verma and Suresh \(2017\)](#) proposed a robust approach to increase the topological quality of generated quad meshes by reducing the singularity for many existing approaches; [Docampo-Sanchez and Haimes \(2019\)](#) described a technique to recover the regularity by performing iterative topological changes, such as splitting, swapping, and collapsing, on the quad mesh generated by Catmull–Clark subdivision ([Catmull & Clark, 1978](#)).

The direct method for quadrilateral mesh generation is to construct quadrilateral elements without any intermediate triangular mesh. [Zhu et al. \(1991\)](#) built a mesh generator for quadrilateral elements based on the advancing front technique. Some methods proposed a quad mesh generator by modifying the quadtree background grid to conform to the domain boundaries ([Baehmann et al., 1987](#); [Liang et al., 2010](#); [Liang & Zhang, 2012](#)). [Zeng and Cheng \(1993\)](#) proposed FREEMESH, a knowledge-based method, to recursively generate quadrilateral elements from the domain boundary until the whole domain was filled with quadrilateral elements. [Blacker and Stephenson \(1991\)](#) proposed an approach named Paving, to generate quad meshes directly in an iterative way from the boundaries of the input domain towards the interior of the domain. [White and Kinney \(1997\)](#) improved the original Paving algorithm by changing the generation method from a row by row to an element by element. The Paving algorithm is currently implemented as part of the CUBIT software ([Blacker et al., 2016](#)). The challenge for those methods is to reduce the flat or inverted elements. Most of these methods require heuristic post-cleanup operations to improve mesh quality. There are some methods using square packing ([Shimada et al., 1998](#)) and circle packing ([Bern & Eppstein, 2000](#)) to generate all quadrilaterals. Circle packing can only bound the maximum angle to  $120^\circ$ . [Atalay et al. \(2008\)](#) utilized a quadtree to construct quadrilateral mesh with a guaranteed minimum angle bound of  $18.43^\circ$ . [Rushdi et al. \(2017\)](#) aimed to solve the angle and post-processing problem and developed an all-quadrilateral algorithm to achieve better quality angles, which falls in the range  $[45^\circ, 135^\circ]$ , without any cleanup operations, including pillowing, swapping, or smoothing. Among the direct unstructured mesh generation methods, the element extraction is preferred for applications that need high quality boundary meshes ([Docampo-Sanchez & Haimes, 2019](#); [Park et al., 2007](#)).



### 4.1.2 The challenge of the element extraction method

Element extraction methods extract elements one by one, starting from the boundary of a domain and updating its boundary inwardly by cutting off the generated element. An updated boundary is also called a front (Owen, 1998). The basic procedure is 1) choosing a vertex (called reference point in this paper) from the front; 2) constructing an element around the reference point; 3) removing the generated element, and; 4) updating the front. During the procedure, a few crucial questions must be answered: 1) how to choose the reference point in step 1? and 2) how to decide the other three points to form an element in step 2? The common challenge is that it is difficult to assure element quality as bad-quality elements occur when the front collides with itself (Shewchuk, 2012; Suresh & Verma, 2019). To answer properly the two mentioned questions will significantly affect whether this challenge can be solved and even whether the meshing task can be completed at all.

It is extremely difficult, expensive, and time-consuming for human algorithm designers to develop high quality and robust element extraction rules for even two-dimensional domains. Despite its capability to generate high-quality meshes along a domain boundary, applications of the element extraction method have been largely limited. Therefore, researchers and developers have had to turn to alternative solutions that are easier to develop by compromising the qualities of the overall mesh (Sarrate Ramos, Ruiz-Gironés, & Roca Navarro, 2014).

### 4.1.3 Contribution

This paper proposes a self-learning system FreeMesh-S to generate quadrilateral elements by automatically acquiring robust and high-quality element extraction rules. The system takes two steps in obtaining the extraction rules. In the first step, three primitive extraction rules are proposed according to the boundary patterns, which follows a design methodology—Environment-Based Design (Zeng, 2004, 2015; Zeng & Yao, 2009). Then, a novel self-learning schema is formed to automatically define and refine the relationships between the parameters included in the element extraction rules by combining reinforcement learning and artificial neural networks. By applying the Advantage Actor-Critic (A2C) reinforcement learning networks, the system can generate high-quality

quadrilateral elements while maintaining the quality of the remaining geometry for the continuous generation of good quality elements. By taking the good quality elements generated from the A2C method as samples, a feedforward neural network (FNN) is trained for the fast generation of high-quality meshes. This proposed system answers the two previously mentioned questions automatically and relieves human algorithm designers from an inefficient and incomplete search of heuristic knowledge.

Comparing with three existing meshing approaches, the main contribution of this proposed system includes: (1) to construct element extraction rules by replacing the conventionally inefficient and incomplete search of heuristic knowledge with an automatic self-learning schema; (2) to achieve high performance in all the quality measurement indices with the highest score in the metrics of Taper and Scaled Jacobian; (3) to eliminate the need for in-depth knowledge in computational geometry, which makes it algorithm developer-friendly; and (4) to provide insights about smart designing of smart system.

The rest of this paper is organized as follows. Section 4.2 discusses the smartness of the proposed element extraction system and the smart designing embedded in the system. Section 4.3 presents the proposed system by using quad mesh generation as an example. Section 4.4 compares the performances of the proposed system with the other three widely adopted methods. Section 4.5 discusses the practical and theoretical benefits of this research. Finally, Section 4.6 concludes this paper.

## **4.2 Smart designing of the smart element extraction system**

To address the challenge identified in Section 4.1, we adopt a concept of smart designing of smart systems. A smart system has two important properties: 1) making adaptive decisions corresponding to various environmental situations, and; 2) self-evolving to improve the system performance (Akhras, 2000). For a smart element extraction system, its extraction rules should be adaptive to various domain boundaries and can self-evolve to achieve the overall mesh quality. This section introduces the design process of the smart element extraction system and links the system with two machine learning methods: reinforcement learning and artificial neural network, for self-evolving

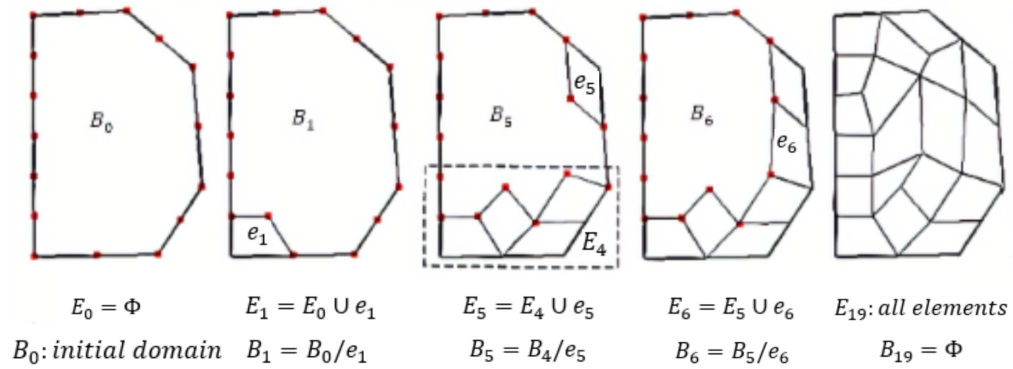


Figure 4.1: Quad mesh generation process (Yao et al., 2005)

the element extraction rules.

#### 4.2.1 How is the element extraction method smart?

##### What is the element extraction method?

Mesh generation is considered as a design problem and can be resolved recursively into the atomic design and a sub-design problem (Zeng & Cheng, 1991; Zeng & Yao, 2009). The quad mesh design should satisfy the following conditions: (1) each element is a quadrilateral; (2) the inner corner of each element should be between  $45^\circ$  and  $135^\circ$ ; (3) the aspect ratio (the ratio of opposite edges) and taper ratio (the ratio of neighboring edges) of each quadrilateral should be within a predefined range; (4) the transition from a dense mesh to a coarse mesh should be smooth (Zeng & Cheng, 1991; Zeng & Yao, 2009). Zeng and Cheng (1993) proposed a knowledge-based method, FREEMESH, to recursively extract quadrilateral elements following a domain boundary until the remaining boundary becomes a quadrilateral element, based on the recursive logic of design (Zeng & Cheng, 1991). In FREEMESH, the atomic design is to generate an element in a specified boundary region while the corresponding sub-design problem is to generate a good quality mesh over the domain by cutting the generated element from the original domain (see Figure 4.1).

In extracting elements one by one, three types of element extraction rules (i.e., adding one, two, and three edges) are defined to construct a good quality element (see Figure 4.2). The challenge is for these rules to generate a good quality element around the current domain boundary while guaranteeing that these design rules are still applicable to the remaining domain for the subsequent

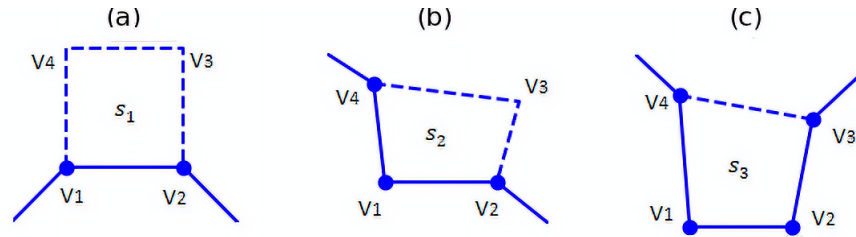


Figure 4.2: Three primitive generation rules (Zeng & Cheng, 1993)

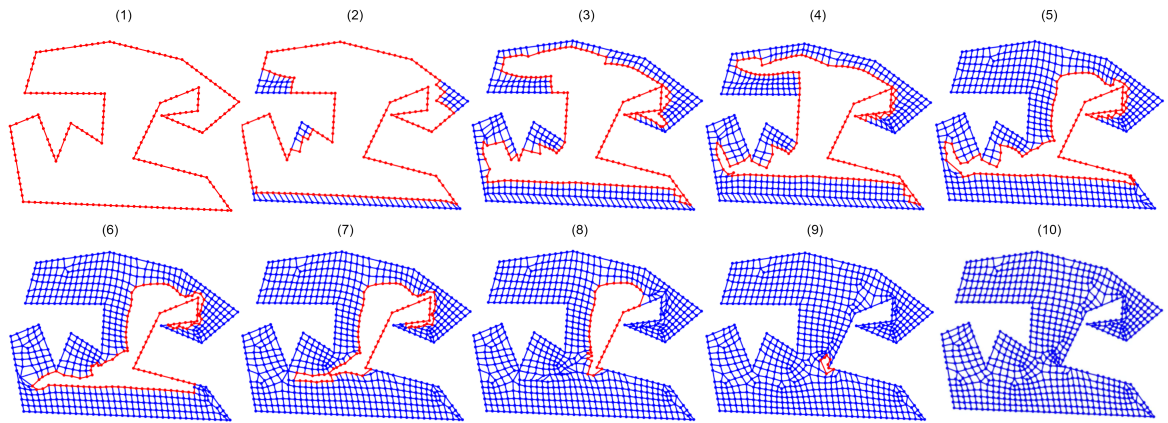


Figure 4.3: Changes of intermediate boundary shapes: all boundaries are represented in red lines from (2) to (9).

construction of good elements.

### How is the element extraction method smart?

A smart system can make adaptive decisions corresponding to various environmental situations. For an element extraction method, the environment is domain boundaries, which keep changing throughout the entire mesh generation process. Even starting with a simple domain, the initial boundary can evolve into many complex intermediate boundary shapes. Following the process specified in Figure 4.1, Figure 4.3 shows an example element extraction process, where the initial boundary is shown in Figure complex intermediate boundary shapes. Following the process specified in Figure 4.1, Figure 4.3 (1) and the final mesh is shown in Figure 4.3 (10). Obviously, the intermediate boundaries could include a variety of situations, which are not predictable.

The smartness of element extraction systems lies in how the variety of complex boundary shapes are processed by the three element extraction rules shown in Figure 4.2. First, each edge in any

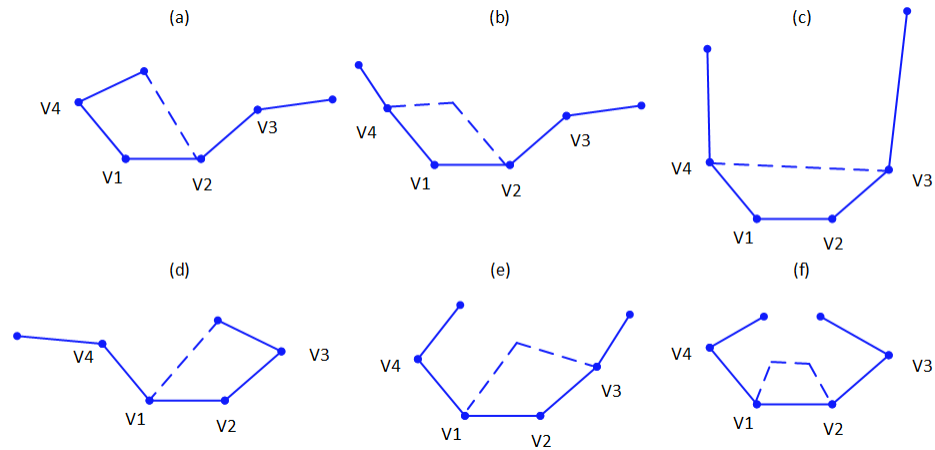


Figure 4.4: Challenges in selecting a proper rule to extract an element for an edge  $V_1V_2$ .

boundary shape can be processed by one of those three extraction rules. The three rules define three kinds of patterns for a selected boundary edge  $V_1V_2$  in a domain, as specified in solid lines in Figure 4.2 (a)-(c). The three rules are sufficient to process any environmental situations (boundary shapes). Secondly, among all the boundary edges, the system would be able to smartly pick up one edge and a corresponding rule to generate an element. In Figure 4.4,  $V_1$ ,  $V_2$ ,  $V_3$  and  $V_4$  are all the same for different environment situations while different rules are preferred if more conditions are considered from the environment. For instance, the boundary edge  $V_1V_2$  is selected as the target edge in all the subfigures (i.e., (a)-(f)); the extraction rule in Figure 4.2 (c) is applied in situations in Figure 4.4 (a), (c) and (d); the extraction rule in Figure 4.2 (b) is applied in situations Figure 4.4 (b) and (e); and the last extraction rule in Figure 4.2 (a) is applied in the situation Figure 4.4 (f). This example demonstrates that it is challenging to select a proper rule even just considering two more connected boundary points.

Thirdly, when rules in Figure 4.2(a) or (b) are selected for a boundary edge, the positioning of the newly added point(s) will be smartly determined according to its local surrounding environment situations so that a good element can be generated while the remaining boundary does not create an impossible situation for the three rules to process. For instance, in Figure 4.5, it shows the challenges in positioning the new point in extracting a new element for two base situations in Figure 4.4 (b) and (f). The red lines represent different remaining boundary shapes to form a complete domain boundary. Although point B, B1, and B2 can form a better element in each situation, point A,

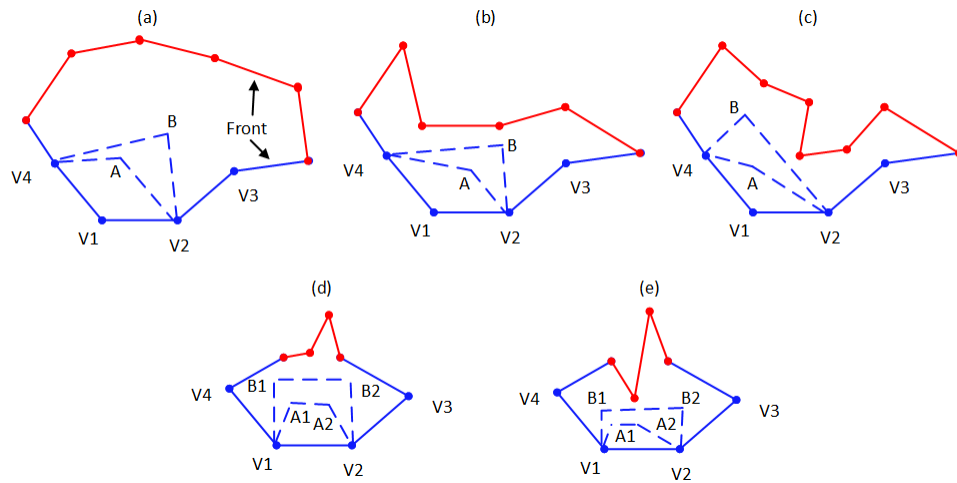


Figure 4.5: Trade-off between qualities of an extracted element and the remaining boundary. Subfigures (a), (b), and (c) correspond to the situation in Figure 4.4 (b); Subfigures (d) and (e) correspond to the situation in Figure 4.4 (f). Points A, A1, and A2 form better elements than B, B1, and B2.

A1, and A2 are the optimal options due to the trade-off of quality of the element quality and the remaining boundary. The coordinates of newly added point(s) are adapted to various remaining boundary shapes.

In summary, the smart element extraction process is enabled by a knowledge-based reasoning process, where the three basic rules shown in Figure 4.2 can be used to recursively process complex 2D domain boundaries. The method smartly extracts elements one by one adaptive to the situations implied in the current domain boundary.

## 4.2.2 How can the element extraction system be smartly evolving and designed?

As Figure 4.4 and Figure 4.5 show, two major challenges exist to enable the smartness of an element extraction system, which are as follows:

- (1) How to decide which of the three rules in Figure 4.2 should be applied for a selected edge ( $V_1V_2$ ) from all kinds of geometric boundaries?
- (2) For the first and second types of rules in Figure 4.2, how to determine relationships between the coordinates of the newly added points ( $V_3$  and  $V_4$ ; or  $V_3$ ) and the geometric conditions surrounding the selected edge ( $V_1V_2$ )?

Conventionally, the questions above are answered manually by algorithm developers through a trial-and-error process. Obviously, it is extremely difficult, very expensive, and time-consuming for human algorithm designers to develop such robust element extraction rules.

The element extraction rules can be smartly designed and evolving by using machine learning algorithms. Many researchers have been combining machine learning algorithms with finite element methods, such as the optimization of mesh density by artificial neural networks (Chedid & Najjar, 1996; Z. Zhang et al., 2020), the placement of nodes of existing quad elements by a self-organizing neural network (Manevitz et al., 1997; Nechaeva, 2006), mesh design by an expert system (Dolšak, 2002), remeshing structural elements by neural networks (Jadid & Fairbairn, 1994), the relationship simulation between quad element state and forces (Capuano Rimoli 2019), triangular mesh simplification (Hanocka et al., 2019), and the mesh optimization to solve partial differential equation (L. Zhang et al., 2021). However, they are not focusing on the mesh generation and the element extraction directly. There are a few works focusing on triangular mesh generation using neural networks (NNs). Pointer networks, (Vinyals et al., 2015), are able to transform a sequence of points into a sequence of triangular mesh elements (three points). But it heavily relies the initial distribution of the points and cannot complete the meshing due to intersecting connections or low triangular element coverage. Papagiannopoulos et al. (2021) proposed a triangular mesh generation method with NNs using supervised learning. They trained three NNs based on the datasets of meshed domains by the Constrained Delaunay Triangulation algorithm (Chew, 1989), to predict the number of candidate inner vertices (to form a triangular element), coordinates of those vertices, and their connection relations with existing segments on the boundary, respectively. The limitations of their method lie in that the overall mesh quality is doomed by the training data, and it cannot be applied to arbitrary and complex domain boundaries because of the fixing number of boundary vertices in the model input. They also cannot be applied to mesh generation or element extraction of quadrilaterals.

According to the nature of the problem, there are two machine learning algorithms that can be exploited to enable the automatic evolution of element extraction rules. The first is Feedforward Neural Networks (FNNs) whereas the second is Reinforcement Learning (RL). The FNN can learn from given samples on how to decide which rule should be applied and how to position the new

point(s) for a new element when necessary. Yao et al. (2005) improved the FREEMESH approach by introducing an artificial neural network (ANN) to learn the element extraction rules from a set of pre-selected samples of good quality quad meshes. The improved method eases the acquisition and application of these rules and could handle more complex domain boundaries. The trained model by ANN serves as the mapping function to transform the input (a specific boundary region) to output (rule type and a point), which is used to form an element following the FREEMESH construction process. This kind of system provides a smart designing mechanism that can automatically generate smart rules to extract elements from any geometric domain.

The RL, with its abilities for trial-and-error learning, models element extraction as a sequential decision-making process. No samples are needed for the RL except that the mesh generation requirements need to be formulated into a reward feedback function. The RL will produce smart element extraction rules for generating high-quality quad meshes through self-learning and evolving. The system proposed in this paper combines these two different learning techniques: reinforcement learning and supervised learning (i.e., Feedforward Neural Networks).

### **4.3 A self-learning system for element extraction**

This section introduces a self-learning element extraction system, FreeMesh-S, which automatically generates mesh elements by self-learned extraction rules from a combination of an RL algorithm and an FNN. Quadrilateral element generation in single-connected 2D domains is used as an example to demonstrate this system. The subsequent subsection introduces how to formulate the element extraction into a reinforcement learning problem. Section 4.3.2 introduces an RL algorithm, A2C, to learn to extract mesh elements by trial-and-error. Section 4.3.3 elaborates how to utilize the FNN to accelerate the learning of extraction rules by supervised learning. Finally, Section 4.3.4 summarizes a general self-learning framework for element extraction.



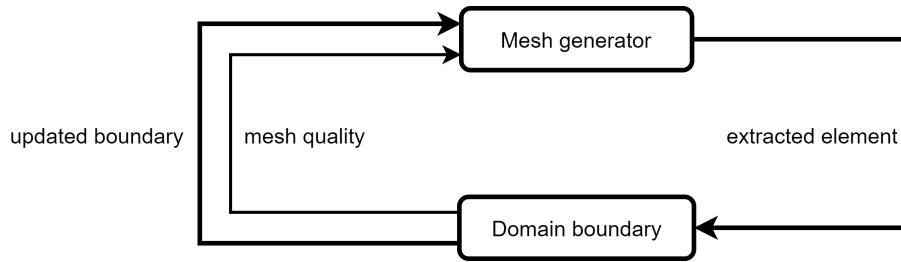


Figure 4.6: Element extraction architecture

### 4.3.1 Element extraction as a reinforcement learning problem

#### Architecture similarity between element extraction and reinforcement learning

The element extraction process in Figure 4.1 can be generalized into architecture as shown in Figure 6. The mesh generator applies the extraction rules to a domain boundary and generates an element; the domain boundary will be updated by removing the extracted element which is scored for its quality. This recursive process continues until the domain is filled with mesh elements. The extraction rules, however, embedded in the mesh generator should be specified in advance. By formulating this process into an RL problem, these extraction rules can be automatically learned by the machine itself (i.e., the mesh generator).

Reinforcement learning (RL) is a technique that enables an agent to learn from the interactions with its environment by trial-and-error via reward feedback from its actions and experiences (Kaelbling et al., 1996; Sutton & Barto, 2018). The hypothesis behind RL is that all the goals can be represented by the maximization of the expected cumulative reward. As is shown in Figure 4.7, the agent, at each time step  $t$ , observes a state  $S_t$  from the environment, and conducts an action  $A_t$  applied to the environment. The environment responds to the action and transits into a new state  $S_{t+1}$ . It then reveals the new state and provides reward  $R_t$  to the agent. This process forms an iteration and repeats until a given condition is satisfied (i.e., the RL problem is solved).

It can be obviously seen from Figure 4.6 and Figure 4.7 that element extraction and reinforcement learning bear a similar framework of problem-solving. As illustrated in Figure 4.8, the mesh generation process shown in Figure 4.6 has exactly the same architecture as the RL process does, and the mesh generation process can be seen as an instance of an RL process. In the context of element extraction, the agent is a mesh generator; the environment is a domain boundary to mesh; the state

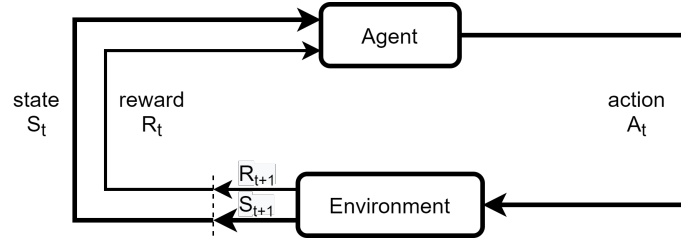


Figure 4.7: Architecture of Reinforcement Learning (Sutton & Barto, 2018)

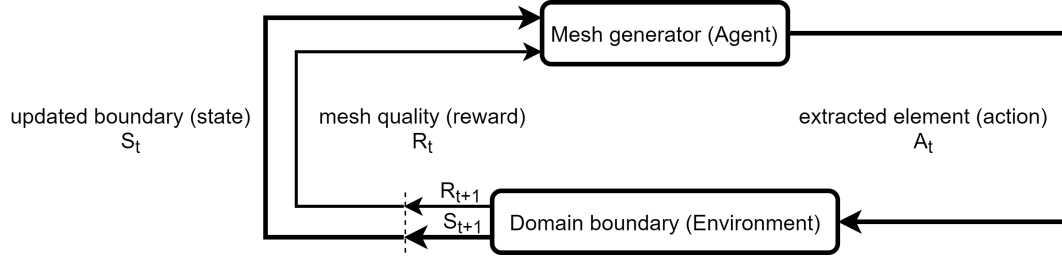


Figure 4.8: Element extraction process as Reinforcement Learning

of the environment consists of a set of focused vertices and edges in the boundary; an action to take by the agent is to extract an element from an edge in the state; the reward is the combination of qualities of both the current extracted element and the remaining boundary; the selection of a specific action is guided by a policy of the agent, and; the agent's goal is to maximize the aggregated rewards - the total quality of all extracted elements and their corresponding boundaries.

### Formulation of the element extraction problem in the reinforcement learning framework

In RL, how the learning agent behave (to select an action at each state) is described as the agent's policy. Mathematically, a policy is represented as a probability distribution over all possible actions at each state, denoted as  $\pi(a|s)$ . The value of a state is the expected future rewards starting at that state, which determines how beneficial for the agent to enter that state. A value function is designed to estimate the value for each state. The value function  $V^\pi(s)$  of a state  $S_t$  under a policy is formally denoted as,  $V^\pi(s) = E[G_t|S_t = s, \cdot]$ , for any  $s \in S$ , where  $S$  is a set of environment states; and  $G_t$  is a discounted sum of the sequence of rewards achieved over time, which is calculated by,

$$G_t = \sum_{k=t+1}^T \gamma^{k-t-1} R_k \quad (1)$$

where  $0 \leq \gamma \leq 1$ , is a discount rate; and  $T$  is a final time step. The value function of an action at a state under a policy can be expressed as  $Q^\pi(s, a) = E[G_t | S_t = s, A_t = a, \pi]$ . These value functions can be estimated from experience with different methods. Policy and value function are essential for almost every RL algorithms.

Mathematically, the RL is a sequential decision-making process, which can be formalized as a Markov Decision Process (MDP) consisting of a set of environment states  $S$ , a set of possible actions  $A(s)$  for a given state  $s$ , a set of rewards  $R$ , and a transition probability  $P(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$  where  $s', s \in S, a \in A(s), r \in R$ ;  $s'$  is the new state at  $t + 1$  and  $r$  is the reward after action  $a$  at state  $s$ . The transition probability indicates that the current state is only dependent on the preceding state and action, so called Markov property, and thereby defines the dynamics of the MDP. This requires that the state must contain information about the past interactions that could distinguish the future and the present. The MDP and agent then produce a sequence  $[S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2, \dots]$ . In general, the MDP framework can be applied to various problems in many ways because of its flexibility. Once a goal is represented as choices of agents, the basis for choice from the environment, and a reward measurement of the goal, it can be formalized as an RL problem. Existing reinforcement learning algorithms can be categorized into three types: policy-based, value-based, and model-based methods (Kaelbling et al., 1996; Sutton & Barto, 2018), which learn policies, value functions, and models (Grondman, Busoniu, Lopes, & Babuska, 2012). Deep reinforcement learning (DRL) intends to make behavioral decisions through learning from the experience of interacting with the environment and from the evaluative feedback (Littman, 2015).

Mathematically, as RL does, the element extraction process can also be represented as an MDP, which consists of a set of boundary environment states  $S$ , a set of possible actions  $A(s)$  in state  $s$  to form an element for each boundary state, a set of rewards  $R$ , and a state transition probability  $P(s', r | s, a)$ . The extraction process will produce a sequence  $[S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2, \dots]$ , where the notations are explained as follows:

- $t$ : denotes the index associated with the time step of element extraction;
- $S_t$ : the partial boundary of the domain (at time  $t$ ) from which the  $t$ th element will be extracted;
- $A_t$ : the action to produce the  $t$ th extracted element, which consists of four vertices of the extracted

element.

This process shows the natural alignment of the element extraction method with reinforcement learning.

Correspondingly, the three rules, which are used to extract elements in Figure 4.2, can be viewed as three actions: (a) to add two new vertices and three edges; (b) to add one new vertex and two edges; (c) to add just one edge. The determination of which rule to apply and especially where to position a candidate vertex to form a high-quality element is highly challenging because the position of a new vertex has substantial impacts on the quality of meshing in the future steps. These actions address the two questions raised in Section 4.2.2.

In this work, we focus on using RL to select the optimal position of a candidate vertex, and the action is defined as to locate a vertex position. In this way, the action space could be a two-dimensional continuous domain. The policy that needs to be learned with RL is the element extraction rule, which maps a state to an action (the position of the new vertex). The full state of the environment at time  $t$  would consist of all vertices of the boundary at the time. However, not all vertices are equally useful for the extraction of a new element. Zeng and Cheng (1993) identified a small set of vertices on the current boundary, which are highly relevant to the construction of a new element. We select this partial boundary to represent the state, which makes computing much more efficient. This selected partial boundary denoted as  $S_t$ , is composed of four parts (1) a reference point,  $P_0$ , which is a vertex selected from the current boundary and will be used as the relative origin for the new element to be constructed and extracted; (2)  $n$  neighboring vertices in the right side, where  $n=2$  in this paper; (3)  $n$  neighbor vertices in the left side, where  $n=2$ ; (4) three neighboring points  $P_1$ ,  $P_2$ , and  $P_3$  in the fan-shaped area  $_{1,2}$  and  $_3$  with radius  $d$ , as shown in Figure 4.9. This partial boundary is denoted as:

$$S_t = \{P_{l2}, P_{l1}, P_0, P_{r1}, P_{r2}, P_1, P_2, P_3\} \quad (2)$$

which is still a high dimensional continuous domain, but its dimension is very significantly reduced from the full boundary. We treat this partial boundary as the state partially observed by the agent (Kaelbling et al., 1996).

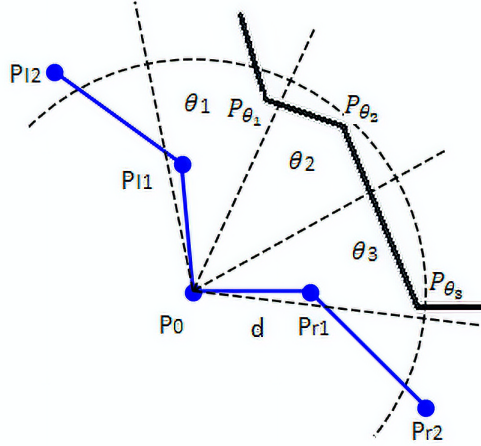


Figure 4.9: Partial boundary

Regarding reward, we define it as the combination of qualities of both the extracted element and the remaining boundary because their trade-off is essential to the overall quality of the final mesh. The reward details will be defined in Section 4.3.2.

Regarding the agent’s policy and the value function of a state (or action under a state), as discussed earlier, the position of a new vertex for a new element has substantial impacts on the quality of meshing in the future steps; it is highly challenging to find an optimized policy to locate the vertex position and to find a reasonable estimation of the value of that position. Both an optimal policy and the value function are unknown to the agent but can be approximated with a parameterized function by learning from experience. For this reason, the advantage actor-critic method is used in this paper.

### 4.3.2 A2C RL network for element extraction based mesh generation

The overall architecture of the advantage actor-critic (A2C) is shown in Figure 4.10. In the architecture, the “actor” mimics the policy, and the “critic” mimics state value functions. We use an A2C network-based agent to interact with the environment and gradually generate high-quality elements by trial-and-error learning. The following subsections will introduce the basics of the A2C method and explain how to generate mesh elements with the A2C method.

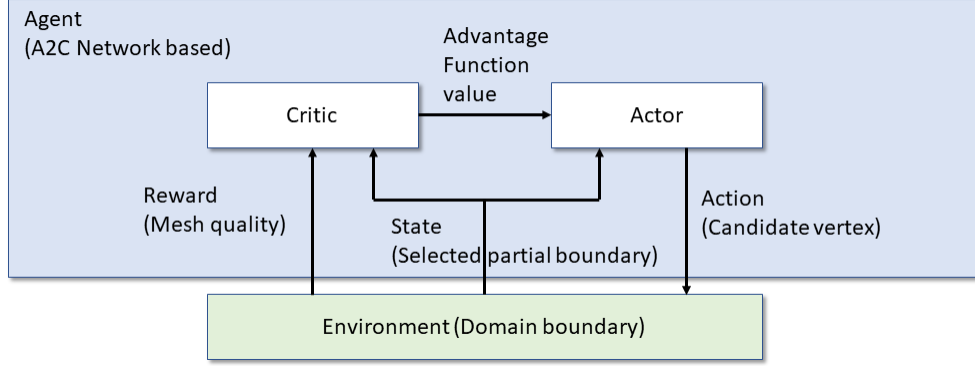


Figure 4.10: A2C agent architecture for element extraction

## A2C

The RL problem is to learn how to select an action at each state over time to maximize the accumulated reward  $G_t$ , which is a discounted sum of the sequence of rewards achieved over time, as shown in Eq.1. Actor-critic is a subclass of policy-gradient methods, which learns approximations to both policy and value functions (Grondman et al., 2012; Sutton & Barto, 2018). Policy-gradient methods directly optimize an agent’s actions without consulting a value function. Policy-gradient methods are via learning a parameterized policy  $\pi_\theta$ , where  $\theta$  is the policy’s parameter vector. The probability that an action  $a$  has been taken in state  $s$  at time  $t$  with policy parameter  $\theta$  is represented as  $\pi(a|s, \theta) = Pr\{A_t = a|S_t = s, \theta_t = \theta\}$ . The general idea is to increase the probability of actions being taken in each state when they have high optimality. The performance of the policy can be denoted as  $J(\theta) = V^{\pi_\theta}(s)$ , which is calculated by

$$V^{\pi_\theta}(s) = \sum_{a \in A} \pi_\theta(a|s) \sum_{s' \in S} Pr(s'|s, a) \{r + \gamma V^{\pi_\theta}(s')\} \quad (3)$$

when every episode starts in state  $s$ .  $r$  is the reward after action  $a$  at state  $s$ . According to policy gradient theorem, the gradient of this performance is denoted as,

$$\nabla_\theta J(\theta) \propto \sum_s d(s) \sum_a Q^\pi(s, a) \nabla_\theta \pi(a|s, \theta) \quad (4)$$

where  $d(s)$  is a state distribution under policy  $\pi$ ,  $Q^\pi(s, a)$  is state-action value function under policy  $\pi$ . The policy is improved by the gradient-ascent algorithm as

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta) \quad (5)$$

where  $\alpha$  is a step size.

We can estimate the performance gradient by Monte Carlo sampling, which is proportional to the actual gradient. The gradient can then be represented as,

$$\nabla_\theta \mathcal{J}(\theta) \propto E\left[\sum_a Q^\pi(S_t, a) \nabla_\theta \pi(a|S_t, \theta)\right] = E[G_t \nabla_\theta \ln \pi(A_t|S_t, \theta)] \quad (6)$$

where  $S_t$  and  $A_t$  are sampled state and action at time  $t$ ,  $G_t$  is the return after  $t$ , and;  $E[\cdot]$  is the expected value of the expression with random variables  $S_t$  and  $A_t$  in the Monte Carlo sampling. The gradient-ascent algorithm could be,

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla_\theta \ln \pi(A_t|S_t, \theta) \quad (7)$$

Because the variance of return is high in Monte Carlo policy gradient, the learned value function can reduce the gradient variance and provide an informative direction for policy optimization. The actor represents the learned policy whereas the critic is the learned value function. The critic estimates the state-action value function as an approximator with parameters  $w$ , i.e.  $\hat{Q}^{\pi_\theta}(s, a; w) \approx Q^{\pi_\theta}(s, a)$ , where  $Q^{\pi_\theta}(s, a)$  is the state-action value function using policy  $\pi_\theta$ . Correspondingly,  $V^{\pi_\theta}(s)$  can be estimated as  $\hat{V}^{\pi_\theta}(s; w) = E_{a \sim \pi}[\hat{Q}^{\pi_\theta}(s, a; w)]$ . By introducing an advantage function to the critic,

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s) \quad (8)$$

In this way, the policy optimization will be more directional. It leads to the advantage actor-critic (A2C) method. The gradient is changed to the following:

$$\nabla_\theta \mathcal{J}(\theta) \propto E\left[\sum_a A^{\pi_\theta}(s, a) \nabla_\theta \pi(a|S_t, \theta)\right] \quad (9)$$

The actor updates the policy parameters by

$$\theta_{t+1} = \theta_t + \alpha A^{\pi_\theta}(S_t, A_t) \nabla_\theta \ln \pi(A_t | S_t, \theta) \quad (10)$$

The critic updates its weights by

$$w_{t+1} = w_t + \beta A^{\pi_\theta}(S_t, A_t) \nabla_w \hat{V}^{\pi_\theta}(s; w) \quad (11)$$

By the definition of the value function  $V^{\pi_\theta}(s)$ , advantage function  $A^{\pi_\theta}(S_t, A_t)$  can be estimated as

The critic updates its weights by

$$\delta_t = R_{t+1} + \gamma \hat{V}^{\pi_\theta}(S_{t+1}; w) - \hat{V}^{\pi_\theta}(S_t; w) \quad (12)$$

which is also called temporal difference (TD) error for state value at time t. For a more detailed description of the A2C model please refer to (Sutton & Barto, 2018). For the specific description of our A2C RL model for mesh please refer to the next subsection. Advantage Actor-Critic (A2C) network is a class of A2C model, in which an artificial neural network is used as an approximator for the policy or for the value function. We use this approach for mesh generation.

### **A2C RL network for element extraction**

In the proposed self-learning element extraction system, the A2C network is responsible for the preliminary sampling of the element extraction rules (a set of three rules in Figure 4.2). Its network structure is shown in Figure 4.11. The actor and critic are the two heads of the network and share the same input and one hidden layer. The input is the partial boundary  $S_t$ . The value head of the critic is the estimation of the state-value function. The policy head of the actor will output two variables (i.e., two means) to form two normal distributions in which both the variances are set to 1, to sample the coordinates, x and y, of the candidate point respectively. The training process of the A2C network is illustrated in Algorithm 1. For a given 2D domain environment, the A2C network updates the parameters of the actor and the critic during each episode using the temporal difference (TD) method (i.e., Eq. 11). Each episode terminates when the domain is full of quadrilateral elements or exceeds



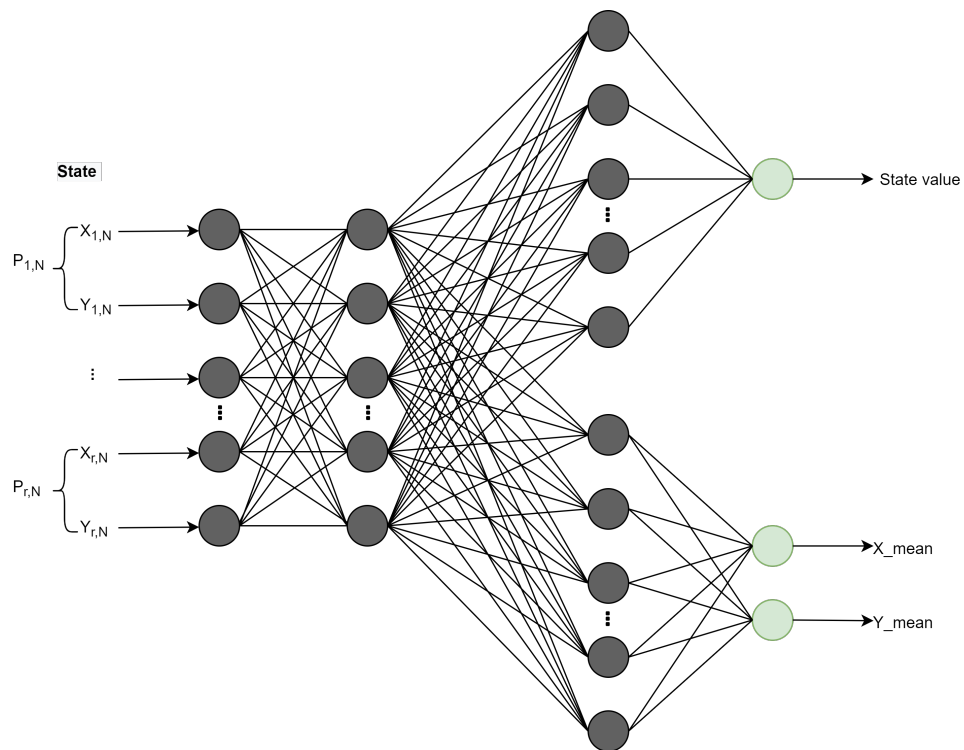


Figure 4.11: A2C network structure. In general, the actor will simulate three primitive rules, as shown in Figure 4.2 (a), (b), and (c). In the current implementation, the extraction rule in Figure 4.2 (a) is not considered and it will be implemented in the next version of the system.

the maximum step. The episode number  $M$  indicates the maximum iteration. The maximum step in each episode is defined by  $T$ . Symbol  $\lambda$  is the discount factor during reward accumulation.

---

**Algorithm 1** A2C reinforcement learning System

---

**Input:** Episode number  $M$ ; each episode's max step number  $T$ ; discount factor  $\lambda$ ; and step size  $\alpha$  and  $\beta$

**Output:** Constructed A2C

- 1: Initialize the network policy parameter vector with random weights
  - 2: **for** episode  $i = 1, 2, \dots, M$  **do**
  - 3:   get an initial state  $S_1^i$  ( $i$  is omitted hereafter) of the environment at episode  $i$
  - 4:   **for**  $t = 1, 2, \dots, T$  **do**
  - 5:     sample action  $A_t \sim \pi(\cdot|S_t, \theta)$
  - 6:     get a next state  $S_{t+1}$  after the environment is updated with its boundary for action  $A_t$
  - 7:     get the reward  $R_{t+1}$  by calculating the quality of the formed element and remaining boundary
  - 8:     calculate TD error  $\delta_t = R_{t+1} + \gamma \hat{V}^{\pi_\theta}(S_{t+1}; w) - \hat{V}^{\pi_\theta}(S_t; w)$
  - 9:     update the critic:  $w_{t+1} = w_t + \beta A^{\pi_\theta}(S_t, A_t) \nabla_w \hat{V}^{\pi_\theta}(s; w)$
  - 10:    update the actor policy:  $\theta_{t+1} = \theta_t + \alpha A^{\pi_\theta}(S_t, A_t) \nabla_\theta \ln \pi(A_t|S_t, \theta)$
  - 11:    **end for**
  - 12: **end for**
- 

**State representation** The state is the actor's observation of the environment, which is defined as a partial boundary  $S_t$ , expressed as Eq.2. To determine which part of the boundary as the state, there are two fundamental steps: 1) to identify the reference point  $P_0$  from the existing boundary; and 2) to find the remaining points according to Eq.2. A new element will be extracted around this formed partial boundary. First, the reference point is calculated by iterating all the points on the boundary, computing the angle of each point formed by its left and right connected points, and selecting the point having the least angle as the  $P_0$ . Then, the other points in  $S_t$  will be determined by Eq.2. The absolute coordinates of all points selected for representing a state will be transformed into a new coordinate system that assumes the reference point  $P_0$  as the origin, and the vector from the reference point to the first neighboring point along the counter-clock direction as the unit vector, along the positive direction of the x-axis (Yao et al., 2005). The transformation (i.e., rotation, scaling, and transit) from the original coordinate system  $Oxy$  to the new one  $O'x'y'$  is shown in

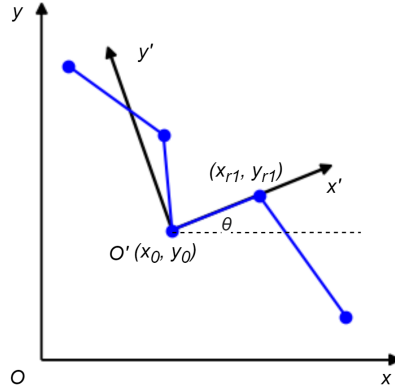


Figure 4.12: Coordinate transformation

Figure 4.12, and can be represented as:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{d} & 0 & 0 \\ 0 & \frac{1}{d} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$d = \sqrt{(x_0 - x_{r1})^2 + (y_0 - y_{r1})^2} \quad (13)$$

**Reward function** The actor will receive reward feedback from the environment to indicate how good the performed action is at the current state  $S_t$ , which the environment in RL refers to the mesh domain boundary. The reward function is defined as follows:

- (1) if the point (action) is outside the domain, the reward is set to be -0.1;
- (2) if the generated element has straddled segments with itself or other elements, as shown in Figure 4.13, the reward is set to be -0.1;
- (3) if the element has no situation with 1) and 2), the reward is the combination of current  $i$ th element quality  $\eta_i^e$  and the quality of the remaining boundary  $\eta_i^b$ , which is calculated by  $\sqrt{\eta_i^e * \eta_i^b}$ . The element quality  $\eta_i^e$  is measured by both its edge quality and angle quality, and calculated as follows, which are adapted from (Zeng & Yao, 2009),

$$\eta_i^e = \sqrt{q^e * q^a} \quad (14)$$

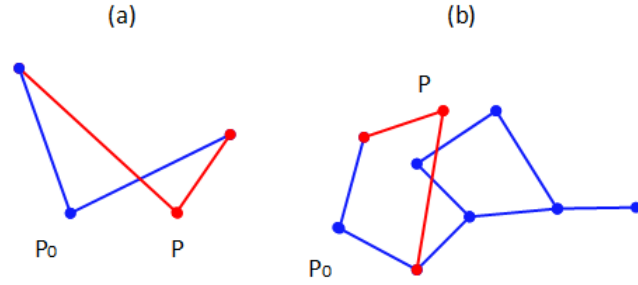


Figure 4.13: Invalid situations of the element.  $P_0$  is the reference point and  $P$  is the newly generated point.

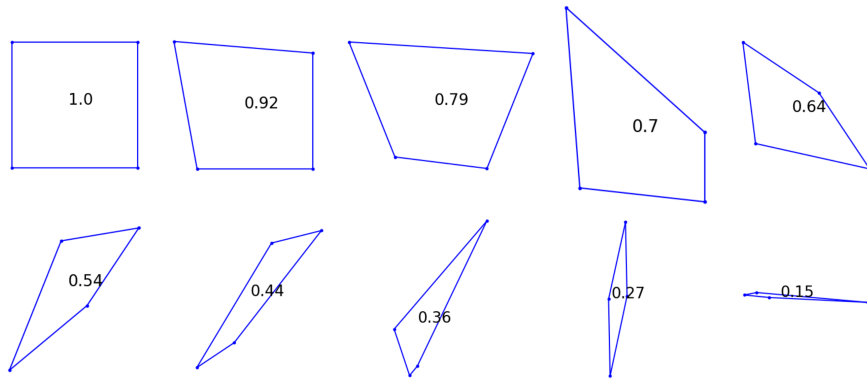


Figure 4.14: Different quality of elements

$$q^e = \sqrt[4]{\prod_{j=1}^4 \left( \frac{l_j}{\sqrt{A_i}} \right)^{\text{sign}(\sqrt{A_i} - l_j)}} \quad (15)$$

$$q^a = \sqrt[4]{\prod_{j=1}^4 \left( 1 - \frac{|a_j - 90|}{90} \right)} \quad (16)$$

where  $q^e$  refers to the quality of edges of this element;  $l_j$  is the length of the  $j$ th edge of the element;  $A_i$  is the area of the  $i$ th element;  $q^a$  refers to the quality of the angles of the element; and  $a_j$  is the degree of  $j$ th inner angle of the element. The quality  $\eta_i^e$  will range from 0 to 1, which is better if greater. Examples of various element qualities are shown in Figure 4.14.

The quality of remaining boundary,  $\eta_i^b$ , is calculated as follows,

$$\eta_i^b = \sqrt{\prod_{k=1}^2 \left( \frac{\text{Min}(\theta_k, 60)}{60} \right)} \quad (17)$$

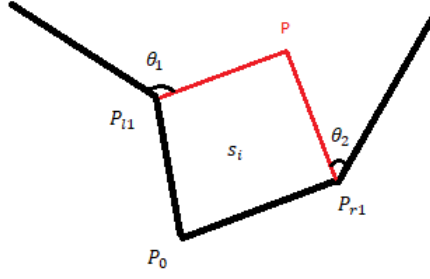


Figure 4.15: New boundary angles

where  $\theta_k$  refers to the degree of the  $k$ th generated angle, as shown in Figure 4.15. The quality  $\eta_i^b$  also ranges from 0 to 1, which the higher value is the better.

Finally, we have the reward function as

$$R_t = \begin{cases} -0.1, & \text{if the generated point is outside the domain or} \\ & \text{the formed elements has straddled segments;} \\ \sqrt{\eta_i^e * \eta_i^b}, & \text{otherwise.} \end{cases} \quad (18)$$

To have a negative reward prevents the actor from performing inappropriate actions and generating invalid elements. Once the actor generates a valid element, it will immediately receive a reward value to tell how good that action is. This is a stepwise reward signal, which smoothly guides the actor to achieve the final goal. The reason why to consider both the quality of the current element and the remaining boundary is that the actor needs to learn the trade-off between them to achieve an overall good mesh quality and complete the meshing task.

#### Actor representation

In a fully observable MDP, the agent observes the true state of environment at each time  $t$  and performs an action according to a parameterized policy  $\pi_\theta$ . The actor constantly chooses actions and produces the best one for a given environmental state  $S_t$  overtime. Typically for type (b) action shown in Figure 4.2, the selected action  $A_t$  (line 5 of Algorithm 1) is to sample a point  $P_t$  within a specified area with the reference point  $P_0$  as the center. The selection is sampled from the parameterized policy,  $A_t = P_t \sim \pi_\theta(s_t)$ . Finally, a new element is formed by four points,  $P_t$ ,  $P_{l1}$ ,  $P_0$ , and  $P_{r1}$ , after the environment receives the point  $P_i$ .

The actor will also update (line 10 of Algorithm 1) the policy distribution at each time step with respect to the parameter vector  $\pi$  by using the sampled policy gradient:

$$\nabla_{\theta} \mathcal{J}(\pi_{\theta}) = \sum_{s,a} [\nabla_{\theta} \ln \pi_{\theta}(a|s) A^{\pi_{\theta}}(s,a)] \quad (19)$$

in which the direction  $A^{\pi_{\theta}}(s,a)$  is suggested by the critic (Section 4.3.2).

#### Critic representation

The critic is used to observe the states and rewards from the environment and estimates the value function  $\hat{V}^{\pi_{\theta}}(s; w)$  accounting for both immediate and future reward to be received by the following policy  $\pi_{\theta}$ . The value function will determine whether the position of the selected point can achieve the best performance in both the quality of the extracted element and the remaining boundary in the long run. To reduce the variance of state value estimation and produce a positive direction for optimization, the advantage function is adopted, as shown in Eq.8. Since the temporal-difference (TD) error, defined in Eq.12, can be used as the unbiased estimation of the advantage function, it will guide the updating of parameter vector  $\theta$  in the actor. The updating of parameters of the critic in Figure 4.10 is shown in the line 9 of Algorithm 1.

### 4.3.3 FNN as policy approximator for fast learning and meshing

A2C network can acquire new knowledge with a slow trial-and-error process; hence, it is practically infeasible to do element extraction directly using RL. To address this challenge, A2C is used to generate good meshes to extract successful (state, action) pairs as samples for training a multilayer FNN. The FNN can be seen as an RL policy approximator. The architecture of the integrated approach is shown in Figure 4.16.

The architecture in Figure 4.16 provides a policy-only approach without consulting value functions. This entire process simulates the human learning process, which acquires successful samples from trial-and-error, extracts experience from those samples, and enhances the decision-making ability by the extracted experience. It is a novel combination of two methods to form a human-like learning schema to solve the mesh generation problem. The learning process consists of experience extraction and FNN learning. Experience extraction is a prerequisite for the FNN component. When

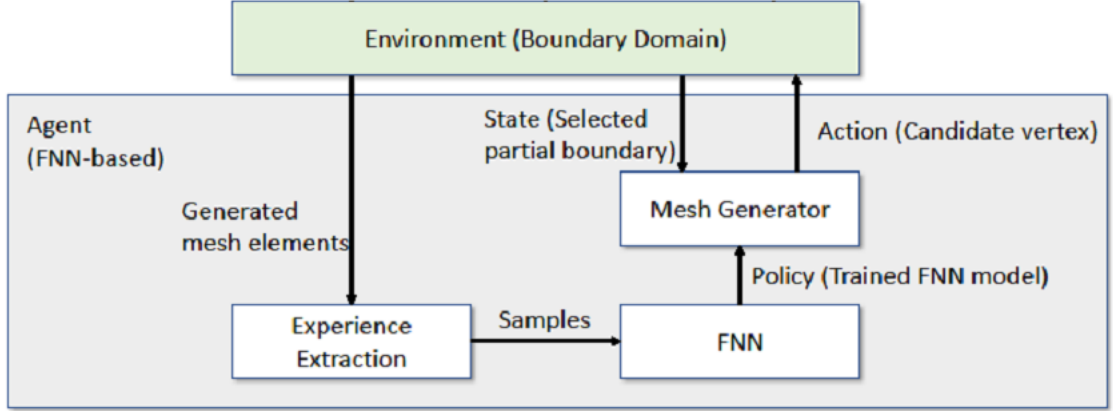


Figure 4.16: FNN agent architecture

the learning finishes, the trained model applies to various domain boundaries for generating mesh elements.

### FNN learning

The FNN module can learn from the acquired data from the experience extraction module to 1) decide which extraction rule should be applied and 2) determine the position of the newly generated point for a new element when necessary. Its network structure is shown in Figure 4.17 and mathematically described as follows (Yao et al., 2005):

$$[type_i, P_i] = f([P_{(l,N)}, \dots, P_{l2}, P_{l1}, P_0, P_{r1}, \dots, P_{(r,N)}, P_{\theta_1}, P_{\theta_2}, P_{\theta_3}]) \quad (20)$$

where  $type_i$  is the output type, as shown in Figure 4.18, which equals to one of the three values (i.e., 0, 1, and 2);  $P_i$  is the output point, consisting of  $(x_i, y_i)$ ; the input points are the state; and  $N$  equals 2 in this paper. The objective function used in FNN is denoted in the following formula:

$$MSE = \frac{1}{n} \sum_i^n [(x_i - \tilde{x}_i)^2 + (y_i - \tilde{y}_i)^2 + (type_i - \widetilde{type}_i)^2] \quad (21)$$

where  $\tilde{x}_i, \tilde{y}_i, \widetilde{type}_i$  are the FNN model output for  $i$ th input and MSE is the sum of the mean squared errors of these three variables.

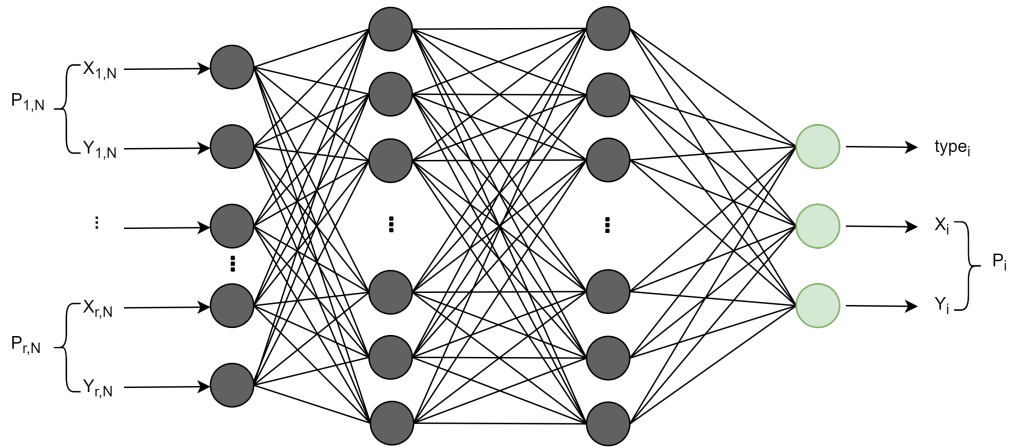


Figure 4.17: FNN module structure

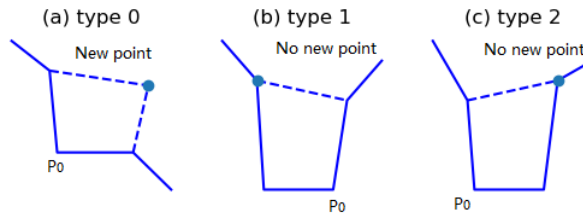


Figure 4.18: Three types of outputs

Through supervised learning, these element extraction rules are quickly acquired. The trained model can easily distinguish which rule should be applied under a specific environment state (input) and can generate a point (output) to form a high-quality element correspondingly. In this way, the experience is transferred from the slow trial-and-error A2C process to the fast one step FNN. This process also simulates the human decision process as specified by (Kahneman, 2011).

### Experience extraction

The experience extraction (EE) module intends to extract samples from existing elements derived from the A2C model for the FNN model training. A sample here is an instance of extraction rules, which is represented by the mappings from a state (input) to the output type and point, as shown in Figure 4.17. For the meshing result generated in each episode by the A2C, EE will check if they are qualified to be extracted as samples. Two criteria are hence used to determine the quality of samples: 1) element quality  $\eta_i^e$  is used to filter out unqualified elements, and; 2) the number of total qualified elements in a meshing result of A2C should exceed a threshold  $\eta$ .



An example of the sample extraction process is shown in Figure 4.19. A trajectory stops when there are no valid elements generated within the maximum step T. A trajectory from the A2C module is illustrated in Figure 4.19 (a), which meets the second criterion (e.g.,  $\eta = 10$ ) as previously mentioned. Each element has a property defined by a tuple  $(\text{element id}; \text{element quality})_i$ , where Element id is defined as the order of the element extracted. Many mesh elements can be used for EE based on the first criterion (i.e.,  $\eta_i^e > 0.7$ ), such as elements 0, 1, 3, 7, 14, 15, 19 and 23 while some other elements will be excluded, such as 5, 12, 13, and 17. For example, the element 15 with element quality 0.91 in Figure 4.19 (a) is used to show the experience extraction process. Three types of extracted samples are shown in Figure 4.19 (b)-(d). Eight points are selected as the input for each rule type, where points  $P_{l2}$ ,  $P_{l1}$ ,  $P_0$ ,  $P_{r1}$ , and  $P_{r2}$  (in red), are points in the boundary from left to right with  $P_0$  as the reference point, and points  $P_{\theta_1}$ ,  $P_{\theta_2}$ , and  $P_{\theta_3}$  (in blue) are collected from the fan-shaped area  $\theta_1$ ,  $\theta_2$  and  $\theta_3$  with radius  $d=3$  with the unit length being  $\|P_0P_{r1}\|$ , as shown in Figure 4.9; and the output point is chosen as  $P_i$  (in black). It should be noted that the point  $P_{r2}$  is  $P_i$  in Figure 4.19 (c) and the point  $P_{l2}$  is  $P_i$  in Figure 4.19 (d).

The input-output pair for each training sample will be arranged according to Eq. 20. Ten training samples with the same reference point and output point are shown in Table 4.1. Those input-output pairs will be normalized and transformed from the original coordinate system  $Oxy$  to the new one  $O'x'y'$  (see Figure 4.12). The transformation results are shown in Table ???. Consequently, the coordinates of all the sampling points are constructed into the same scale and their relationships remain unchanged.

Table 4.1: Examples of extracted samples for element 15: the reference point  $P_0(x_0, y_0)$  and output point  $P_i(x_i, y_i)$  are the same across these samples, respectively

input															output			
$x_{l2}$	$y_{l2}$	$x_{l1}$	$y_{l1}$	$x_0$	$y_0$	$x_{r1}$	$y_{r1}$	$x_{r2}$	$y_{r2}$	$x_{\theta_1}$	$y_{\theta_1}$	$x_{\theta_2}$	$y_{\theta_2}$	$x_{\theta_3}$	$y_{\theta_3}$	$type_i$	$x_i$	$y_i$
10	2	8.87	0.55	10.09	0.13	9.39	1.51	8.27	2.35	6.72	3.25	6.72	1.05	5.57	0.94	0	8.02	0.93
10	2	8.87	0.55	10.09	0.13	9.39	1.51	8.27	2.35	6.72	3.25	6.72	1.05	5.57	0.94	0	8.02	0.93
10	2	8.87	0.55	10.09	0.13	9.39	1.51	8.27	2.35	6.95	3.55	5.65	1.49	5.57	0.94	0	8.02	0.93
8.02	0.93	8.87	0.55	10.09	0.13	9.39	1.51	10	2	6.94	2.14	7.38	1.19	5.57	0.94	1	8.02	0.93
8.02	0.93	8.87	0.55	10.09	0.13	9.39	1.51	10	2	8	4	7.12	0.56	5.57	0.94	1	8.02	0.93
8.02	0.93	8.87	0.55	10.09	0.13	9.39	1.51	10	2	6.72	3.25	6.72	1.05	5.57	0.94	1	8.02	0.93
10	2	8.87	0.55	10.09	0.13	9.39	1.51	8.02	0.93	6.94	2.14	7.38	1.19	5.57	0.94	2	8.02	0.93
10	2	8.87	0.55	10.09	0.13	9.39	1.51	8.02	0.93	8	4	7.12	0.56	5.57	0.94	2	8.02	0.93
10	2	8.87	0.55	10.09	0.13	9.39	1.51	8.02	0.93	6.72	3.25	6.72	1.05	5.57	0.94	2	8.02	0.93
10	2	8.87	0.55	10.09	0.13	9.39	1.51	8.02	0.93	6.95	3.55	5.65	1.49	5.57	0.94	2	8.02	0.93

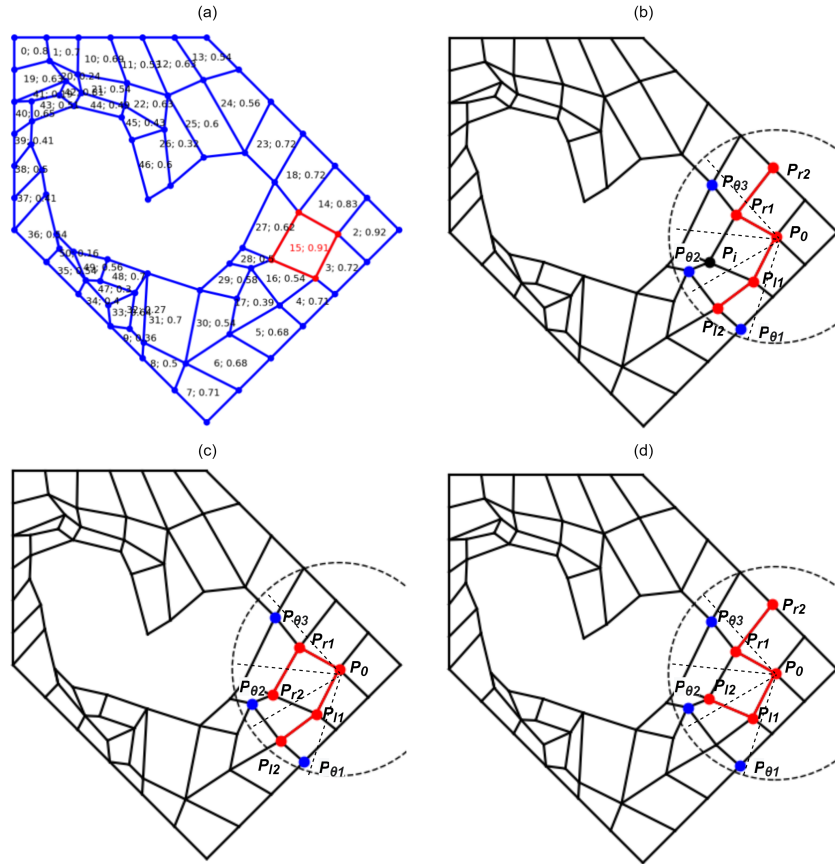


Figure 4.19: Example of experience extraction of one element. In (a), each element has a tuple  $\langle \text{elementid}; \text{elementquality} \rangle$  inside, where element id refers to the generation order of the element; and element quality is calculated by  $\eta_i^e$ ; (b)-(d) show three types of patterns collected for training samples: (b) type 0; (c) type 1; (d) type 2.

Table 4.2: Examples of extracted samples in Table 4.1 after coordinate transformation following Eq. 4.3.2

input														output				
$x_{l2}$	$y_{l2}$	$x_{l1}$	$y_{l1}$	$x_0$	$y_0$	$x_{r1}$	$y_{r1}$	$x_{r2}$	$y_{r2}$	$x_{\theta1}$	$y_{\theta1}$	$x_{\theta2}$	$y_{\theta2}$	$x'_{\theta3}$	$y_{\theta3}$	$type_i$	$x_i$	$y_i$
1.2	0.67	0.04	0.9	0	0	1	0	1.81	0.4	2.84	0.07	1.12	1.59	0.71	2.92	0	1.07	0.96
1.2	0.67	0.04	0.9	0	0	1	0	1.81	0.4	2.78	1.03	1.52	1.67	0.71	2.92	0	1.07	0.96
1.2	0.67	0.04	0.9	0	0	1	0	1.81	0.4	2.89	0.81	2.08	2.16	0.71	2.92	0	1.07	0.96
1.07	0.96	0.04	0.9	0	0	1	0	1.1	0.49	2.08	1.23	1.4	1.25	0.71	2.92	1	1.07	0.96
1.08	0.96	0.04	0.9	0	0	1	0	1.1	0.49	2.84	0.07	1.12	1.59	0.71	2.92	1	1.07	0.96
1.09	0.96	0.04	0.9	0	0	1	0	1.1	0.49	2.78	1.03	1.52	1.67	0.71	2.92	1	1.07	0.96
1.2	0.67	0.04	0.9	0	0	1	0	1.07	0.96	2.08	1.23	1.4	1.25	0.71	2.92	2	1.07	0.96
1.2	0.67	0.04	0.9	0	0	1	0	1.07	0.96	2.84	0.07	1.12	1.59	0.71	2.92	2	1.07	0.96
1.2	0.67	0.04	0.9	0	0	1	0	1.07	0.96	2.78	1.03	1.52	1.67	0.71	2.92	2	1.07	0.96
1.2	0.67	0.04	0.9	0	0	1	0	1.07	0.96	2.89	0.81	2.08	2.16	0.71	2.92	2	1.07	0.96

The extraction process will be repeated for all the qualified elements in an existing mesh to gather training samples. In this way, a large number of training data can be easily extracted with a single mesh. Furthermore, the training data contains sufficiently diversified situations that may appear in meshing any complex geometric domain.

#### 4.3.4 Summary

By using quadrilateral mesh generation as an example, the self-learning element extraction system, FreeMesh-S, demonstrates how the extraction rules are acquired by the combination of an A2C and an FNN. The general architecture is illustrated in Figure 4.20. Through considering the mesh generation as a design problem, three atomic design rules (i.e., three extraction rules) are proposed to recursively extract quadrilateral elements by (Zeng & Cheng, 1993), which are complete and sufficient to mesh various complex boundary shapes. Given these defined rules, the agent of RL (i.e., A2C here) learns their transition relationships through continuously self-evolving, known as the policy. The FNN serves as a policy-only approach to extract samples from those high-quality elements from the trial-and-error, which enables the agent to choose one of the three extraction rules to apply and to position the coordinates of newly added points.

The generalized architecture is not only limited to quad mesh generation, but any problem that can be formulated into atomic design problems and sequential decision-making problems.

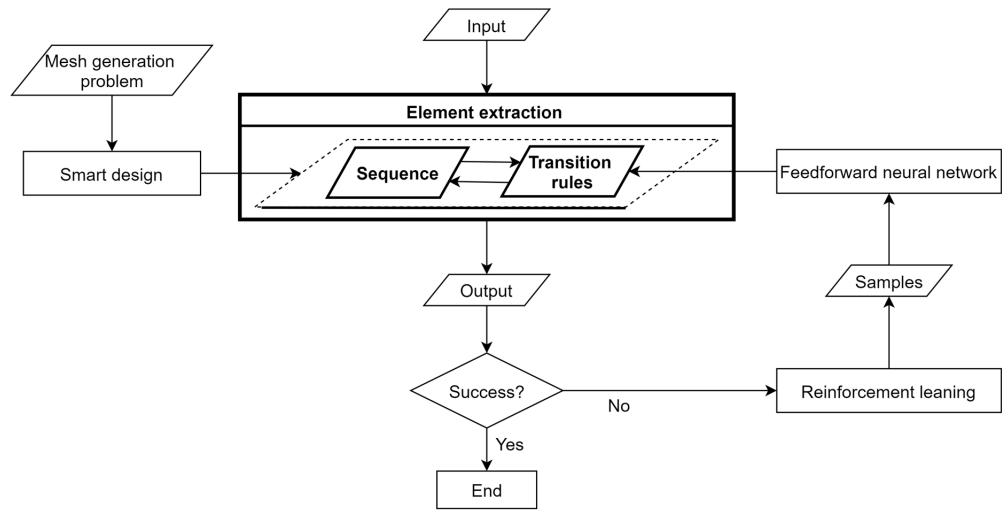


Figure 4.20: Proposed self-learning system FreeMesh-S: architecture

## 4.4 Experiments

To comprehensively evaluate the performance of the learned element extraction rules by the proposed self-learning system, FreeMesh-S, two experiments were conducted. Experiment 1 tests the performance of the training and the impact of the training parameters on the quality of the generated mesh. Experiment 2 compares the quality of meshes by FreeMesh-S against three widely adopted meshing approaches over 10 predefined 2D domain boundaries. The following subsections will discuss the experiment details and results.

### 4.4.1 Experiment settings

#### Experimental domain boundaries

To provide comprehensive and various challenging situations for meshing, testing domain boundaries are chosen based on whether a boundary includes sharp angles, bottleneck regions, unevenly distributed segments, and holes. Hence, 10 domain boundaries are selected to test the performance of the proposed system (see Figure 4.21 and Table 4.3).

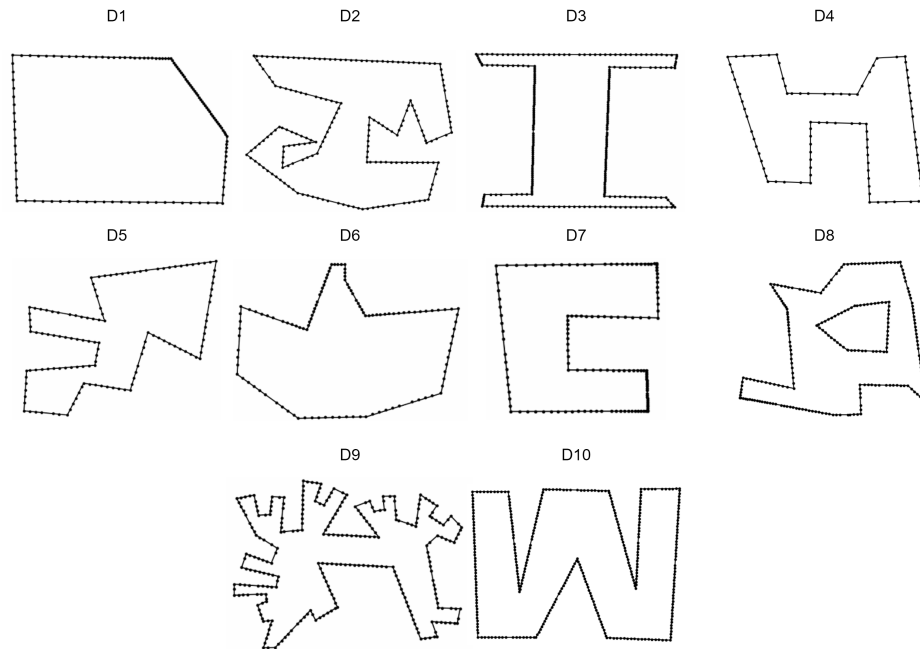


Figure 4.21: experimental domains

Table 4.3: Description of 10 testing domains

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
<b>Sharp angles</b>		✓	✓			✓		✓		✓
<b>Bottleneck region</b>		✓		✓	✓			✓	✓	
<b>Unevenly distributed segments</b>	✓		✓				✓	✓	✓	✓
<b>Hole</b>								✓		
<b>Vertex numbers</b>	120	196	272	92	128	286	140	269	110	358
<b>Perimeter</b>	17.2	40.9	19.5	24.8	26.6	46.9	19	26.5	24.9	40.1
<b>Unit boundary vertex numbers</b>	6.9	4.8	13.9	3.7	4.8	6.1	7.4	10.2	4.4	8.9

### Mesh performance evaluation metrics

The meshing performance is measured by 7 mesh quality metrics, including 6 geometric metrics, and 1 topological metric, as shown in Table 4.

Table 4.4: Mesh quality metrics

Topological Metric	Singularity
Geometric Metrics	Element quality,  MinAngle-90 ,  MaxAngle-90 , Scaled Jacobian, Stretch, and Taper

Seven quality metrics are:

- Singularity: the number of irregular nodes in the interior of a mesh. A node is considered

irregular if it does not have four incident edges (Verma & Suresh, 2017);

- Element quality  $\eta^e$ : an index defined by Equation 14.
- $|\text{MinAngle}-90|$ : the absolute differences between the smallest internal angle of an element and  $90^\circ$ ; The smaller differences imply a better element;
- $|\text{MaxAngle}-90|$ : the absolute differences between the largest internal angle of an element and  $90^\circ$ ; The smaller differences imply a better element;
- Scaled Jacobian: the minimum Jacobian (Knupp, 2000) at each corner of an element divided by the lengths of the two edge vectors, which varies from minus one to plus one, where a higher value implies a better element; the negative value, typically, means the element is inverted;
- Stretch: an index referring to the ratio between the shortest element edge length and the longest diagonal length; and
- Taper: the maximum absolute difference between the value 1 and the ratio of the areas of two triangles separated by a diagonal within a quadrilateral element, where the smaller value represents that the element becomes closer to a triangular shape.

All the metrics are averaged for all elements in the mesh, respectively. Some indices (e.g., scaled Jacobian, stretch, and taper) are calculated using Verdict software (Pébay et al., 2008).

#### **4.4.2 Experiment 1: training effectiveness and efficiency**

The training of the proposed self-learning meshing system consists of two parts: A2C model training and FNN model training. For the A2C network, the actor and critic share the same hidden layer (128 nodes). Since the environment state consists of 8 2D points, the input layer has 16 nodes. For the action is the coordinates of the candidate point, which is continuous in a 2D space, the policy is represented by two normal distributions to estimate x, y coordinates, respectively. Two mean values of the normal distributions are the outputs of the actor, and two variance values are set to 1. The final x, y coordinates are sampled from these two distributions according to the obtained mean and variance pairs. To accelerate the learning speed, the sampled x, y coordinates are clipped into  $[-1.5, 1.5]$ . The learning rate of 0.0001 is set for the network and the reward discount rate is set as 0.99.

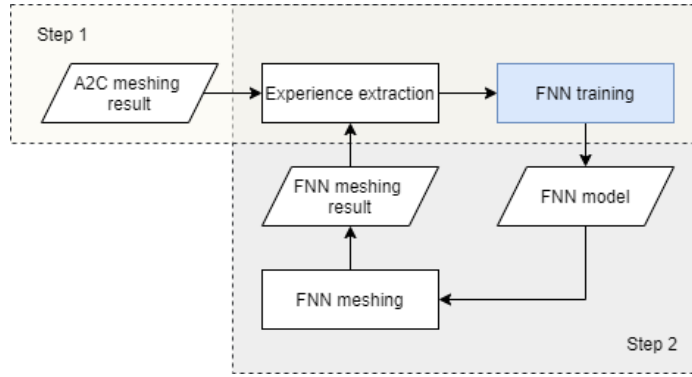


Figure 4.22: Model training process

### Self-sampling and evolving of A2C and FNN training

There are two steps to complete the training of the FNN network, as is shown in Figure 4.22. In Step 1, the A2C module will generate a mesh, which may cover only a partial domain to mesh. The generated mesh is then used to produce initial training samples for FNN. In Step 2, the FNN will be trained, applied, and further evolved.

In this presented research, an empty domain, such as that shown in Figure 4.23 (a), is used to generate elements without any prior knowledge about the parameters in the extraction rules in Figure 4.2. The A2C network can generate high-quality elements in the domain after some rounds of trial and error. Those high-quality elements in each episode of training can be extracted as training samples for the FNN network, which can be reused for the boundary of different shapes. Figure 4.23 (d)-(f) are episodes that include sufficient good samples to train the FNN. An episode can be selected as a source for element sampling if two criteria are met:  $\eta_i^e > 0.7$  and  $\eta > 20$ , as introduced in Section 4.3.3.

Table 4.5 shows the total number of elements (#tE), number of valid elements (#vE), number of extracted samples (#eS), and the number of samples per valid element (#S2E) for each episode. The samples are extracted using the experience extraction (EE) module introduced in Section 4.3.3. It can be seen from Table 4.5 that the episode in Figure 4.23 (d) has 37 valid elements, which allows the extraction of 10,369 different samples (see the extraction process in Figure 4.19), where the number of samples per valid element has reached 280. The similar information can be observed in Figure 4.23 (e) and Figure 4.23 (f). This significant number of leverage over an element brings

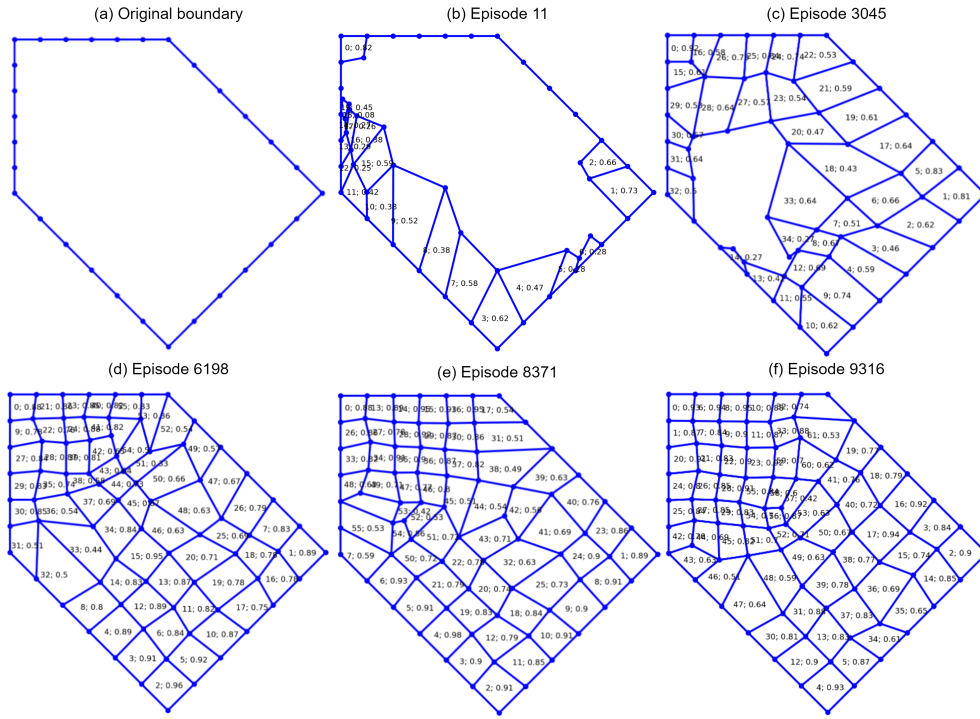


Figure 4.23: Partial results of sampling

the EE module’s huge efficiency and effectiveness in collecting samples. This effectiveness and efficiency in data collection ensure that the FNN model can be sufficiently trained.

Table 4.5: Self-evolving process of FNN model.

Episode	#tE	#vE	#eS	#S2E
Figure 4.23 (b)	20	2	54	27
Figure 4.23 (c)	35	6	1,174	195
Figure 4.23 (d)	56	37	10,369	280
Figure 4.23 (e)	56	41	12,230	298
Figure 4.23 (f)	62	45	14,906	331

#tE: total number of elements;  
 #vE: number of valid elements with quality  $\eta^e > 0.7$ ;  
 #eS: number of extracted samples;  
 #S2E: the number of samples per valid element.

Then trained FNN model can be applied to another domain, such as domain D2, to do the meshing even though the extracted samples are from a different domain (see Figure 4.23 (a)). The meshing result of this FNN model is shown in Figure 4.24.

The meshing, however, is not complete in this domain. In step 2 of the training process as illustrated



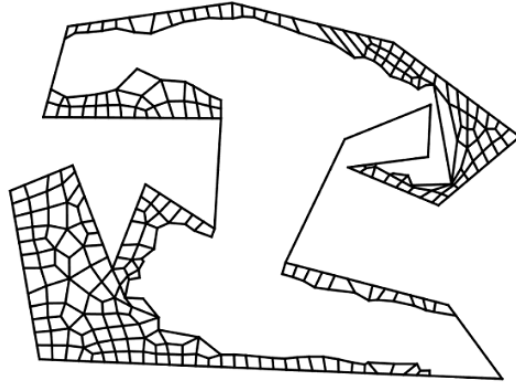


Figure 4.24: The first meshing result of the FNN model

in Figure 4.22, therefore, there is a self-evolving process of the FNN model by learning from the meshing results generated by itself. While repeating step 2, the element extraction rules simulated by the model are gradually evolved and adapted to the characteristics of the boundary shape. For example, the training process of the subsequent 4 rounds of step 2 is shown in Table 4.6. In each round, the same FNN model will be trained using the extracted samples from the previous meshing result (i.e., sample source) and then the trained model is applied to generate a new mesh. Obviously, the meshing performance is getting improved and adapted to the boundary shape continuously while a significantly high number of valid samples can be extracted.

### Efficiency of A2C training

The time cost for training the A2C network on the domain (see Figure 4.23 (a)) is shown in Figure 4.25. It shows the temporal changes in the average number of elements per continuous 100 episodes during the training process. The red line accounts for the total number of elements whose quality meets  $\eta^e > 0$ . The black, green, and blue lines refer to  $\eta^e > 0.3$ ,  $\eta^e > 0.5$ , and  $\eta^e > 0.7$ , respectively. All the lines show an increasing trend over time, which means that the learning process is successful. The black and red lines are very close, which indicates that almost every element has a quality bigger than 0.3. The number of elements with quality  $\eta^e > 0.7$  accounts for around 30% of total number of elements when the training is converged. At the end of the training period, the domain has an average element number of about 75, and the average number of elements with  $\eta^e > 0.7$  is around 22.

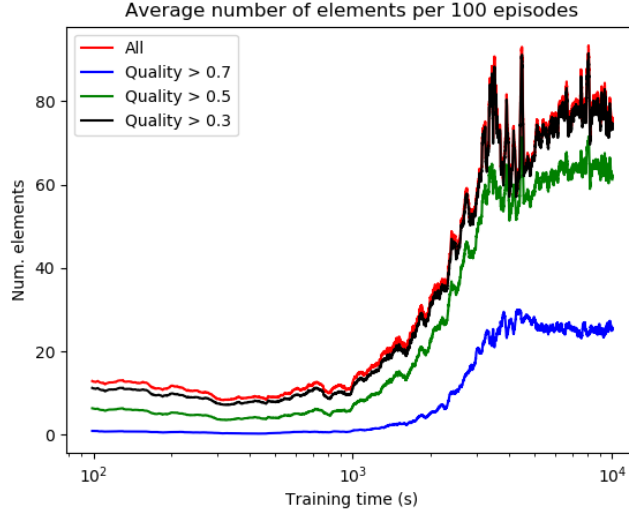


Figure 4.25: Temporal training cost of A2C network

### Impact of training parameters on mesh metrics

#### Element quality thresholds comparison

In the module of experience extraction, the quality threshold is used to control the selection of elements and determines whose experience is valuable for subsequent learning. To compare the different performances of the trained FNN model against the quality of selected samples, three different element selection thresholds ( $\eta^e > 0.5$ ,  $\eta^e > 0.7$ , and  $\eta^e > 0.8$ ) are tested on the three experimental domains (D1, D2, and D3). The performance is measured by the 7 quality metrics.

The comparison results are shown in Table 4.7. Obviously, taking the quality threshold  $\eta^e > 0.7$  achieves best performance in 4 metrics (i.e., Singularity, —MaxAngle - 90—, Scaled Jacobian, and Taper); taking the threshold  $\eta^e > 0.8$  has best results in Element quality, —MinAngle - 90—, and Scaled Jacobian; and the last threshold has the best performance only in Stretch. It is understandable that the FNN model has better performance in most of the metrics when the experience of elements with higher quality is sampled. However, the FNN model with a higher element quality threshold may not produce sufficient training data because the number of extracted samples will decrease when the threshold is raised. The other metrics, such as Stretch, is not sensitive to whether the elements with higher quality are chosen. Therefore, this paper chooses quality  $\eta^e > 0.7$  as the ideal and balanced threshold for experience extraction, which thereby trains FNN models.

### FNN structures comparison

To compare different performances of FNN models against the network structures, 4 different hidden layer structures ([64, 64], [256, 256], [64, 128, 64, 32, 16], and [32, 64, 128, 64, 32, 16]) are tested on three experimental domains. Four FNN models are trained using the 4 different structures according to the process introduced at the beginning of this section. The model performance is measured by the 7 quality metrics. The comparison results are shown in Table 4.8, where each metric value is the average of values for three respective domains. Obviously, the models have better performance in slender structures than the structures of [64, 64] and [256, 256]; and especially, the model having the structure of [64, 128, 64, 32, 16] achieves the best performance in the 6 metrics. This paper, therefore, chooses this structure as the optimal FNN network structure and uses this trained FNN model to compare the meshing performance with other meshing approaches.

### Comparison of meshing speed for A2C and FNN

The mesh generation speed is important to finite element analysis applications. In this paper, an extra indicator, elements per second, is adopted to measure the performance differences between the A2C and FNN models, both of which were developed by the same team under the similar resources. Other methods are not compared since they are developed by different teams of various software development capabilities and with different programming languages, which will have a great impact on the performance of the system. These two models are tested on the 10 domains given in Figure 4.21. The comparison result is shown in Table 4.9. The average speed for A2C over 10 domains is 19.71 elements per second while that of FNN equals to 28.73. The FNN model excels A2C by almost 9 elements per second. The performance difference is that FNN can directly predict the coordinate of the candidate point to form an element and the A2C model still needs two sampling operations from two normal distributions to get the x and y coordinates. In addition, the A2C model may not be adaptable to all of the domains and could require some trial-and-error to generate a valid element, which causes a big drop in the speed. For example, the meshing speed of A2C is only 3.82 elements per second in domain D7. The standard deviations of A2C and FNN are high and both the meshing speeds increase when the domain has fewer vertex numbers. The probable reason for this trend is because the proposed method will update the boundary and find the

next reference point every time after an element was extracted and hence the iteration number will be correspondingly greater for more boundary vertices.

#### 4.4.3 Experiment 2: comparisons of the proposed method with existing methods

To examine the performance of the proposed FreeMesh-S system, the effectiveness comparison experiment is conducted with the other three approaches.

##### Experimental comparison approaches

The meshing performance of the self-learning system will be compared with three popular quad meshing approaches, Blossom-Quad (Remacle et al., 2012), Delquad (Remacle et al., 2013), and Paving algorithm (Blacker & Stephenson, 1991; White & Kinney, 1997). Blossom-Quad and DelQuad are two indirect algorithms to generate quad meshes in 2D domains. The former takes advantage of the blossom algorithm to find the perfect matching of a pair of triangles that are generated by the Delaunay triangulation algorithm and then to combine matched pairs into quadrilateral elements. The latter one improves the triangulation process of Blossom-Quad by building triangular elements that are more suitable to recombine into quadrilaterals. Paving and its redesigned version are direct methods to generate quadrilaterals from domain boundaries. They generate layered quadrilateral elements from the boundary toward the interior until only six boundary nodes are left, which will be tackled following predefined patterns.

In the following, two widely used software systems, Gmsh (Geuzaine & Remacle, 2009) (implements the Blossom-Quad and Delquad algorithms) and CUBIT (Blacker et al., 2016) (implements the Paving approach), are used as quad element generator to compare meshing performance with the proposed self-learning system FreeMesh-S. Gmsh and CUBIT are both very popular meshing software and have been developed by researchers for many years since 1991. CUBIT is developed at Sandia National Laboratories and even has a commercial product (Csimsoft). The comparisons will be made only on the quality indices while the computational time will not be considered since it depends on how the systems are designed and implemented and which programming language is used. The computational complexity of FreeMesh-S and Paving is  $O(n^2)$ , which is calculated using their respective procedures, whereas that of Blossom-Quad and DelQuad is reported to be  $O(n^2)$

(Johnen, 2016).

### **Model applicability validation**

To validate that the trained FNN model can be applied to arbitrary domain boundaries without any additional training, the 10 experimental domains are tested. The meshing results of FreeMesh-S are shown in Figure 4.26. All the domains are discretized into high-quality quadrilateral elements successfully, and the transition of elements is smooth, which demonstrates the excellent applicability to various boundary shapes without additional A2C and FNN training.

### **Comparing results and analysis**

All the meshing results of four methods, Gmsh-Blossom (GB), Gmsh-DelQuad (GD), and CUBIT-Pave (CP), and the proposed self-learning system FreeMesh-S (SS), on the 10 experimental domains, are shown in Figure 4.27, Figure 4.28, Figure 4.29, and Figure 4.21, respectively. All the domains have been successfully discretized into mesh elements, in which the methods CP and SS have more regular quadrilateral elements than GB and GD; and GD performs better than GB. The increase of regularity of meshes can reduce the computational consumption and improve the accuracy of the analysis results. GD and CP methods, however, cannot mesh all the domains into quadrilaterals. There are triangles in certain experimental domains, which require extra cleanup operations to eliminate them.

To quantitatively measure the different performance of the 4 methods, the statistics of the 7 metrics are illustrated in Table 4.12. Each value in the table is the average over 10 domains (see detailed statistics in Table 4.10). Clearly, the proposed SS method achieves the best performance in 2 quality metrics (Scaled Jacobian and Taper); CP method achieves the best performance in 5 quality metrics (Singularity, Element quality, Min angle, Max angle, and Scaled Jacobian); and GB is the best method from the aspect of Stretch. Moreover, the number of domains that each method achieves the best in each metric are shown in Figure 4.30. For each metric, hence, there are the following results.

- Singularity: Even though the CP method obtains the best averaged performance against other

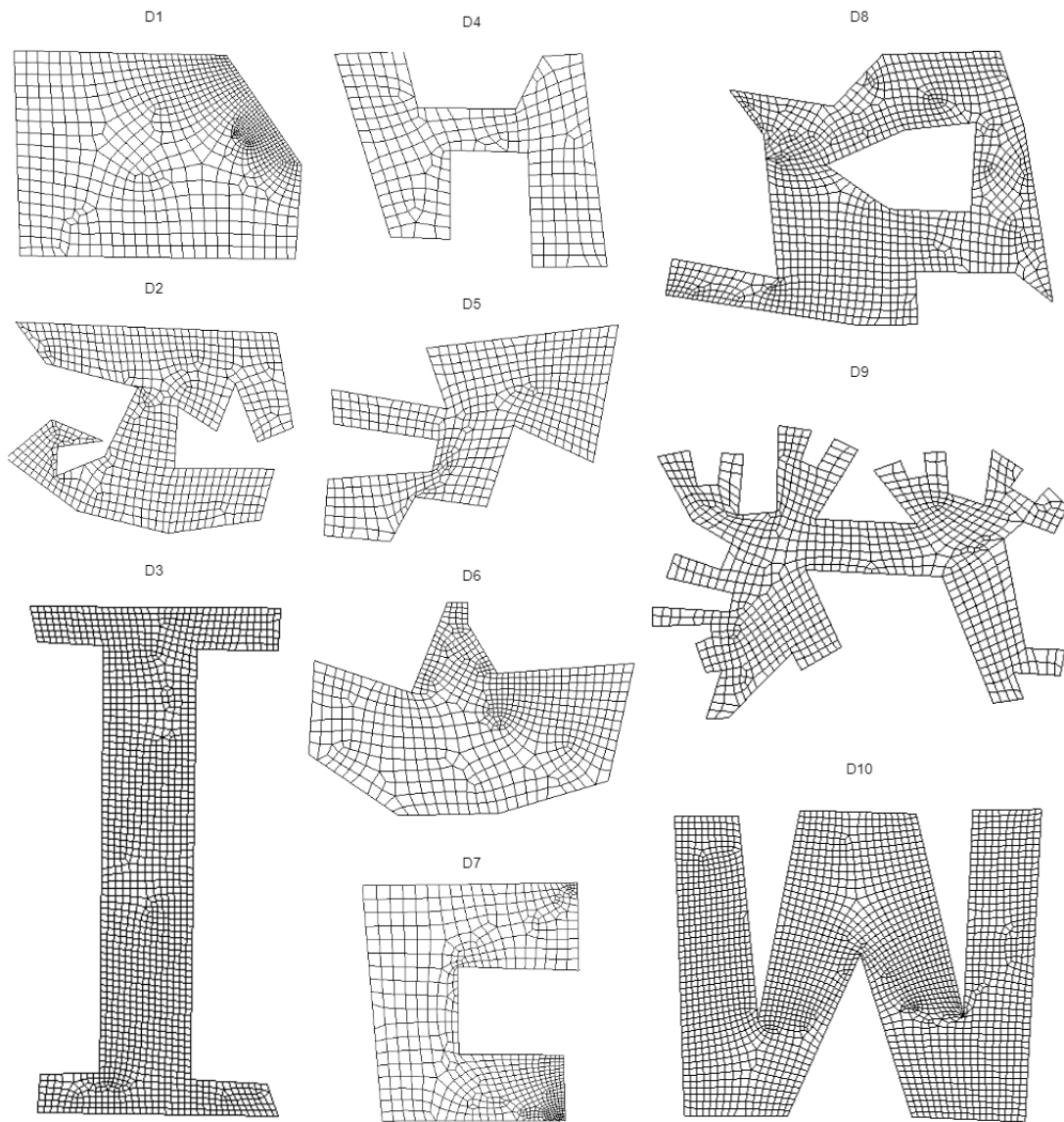


Figure 4.26: All the meshing results of FreeMesh-S

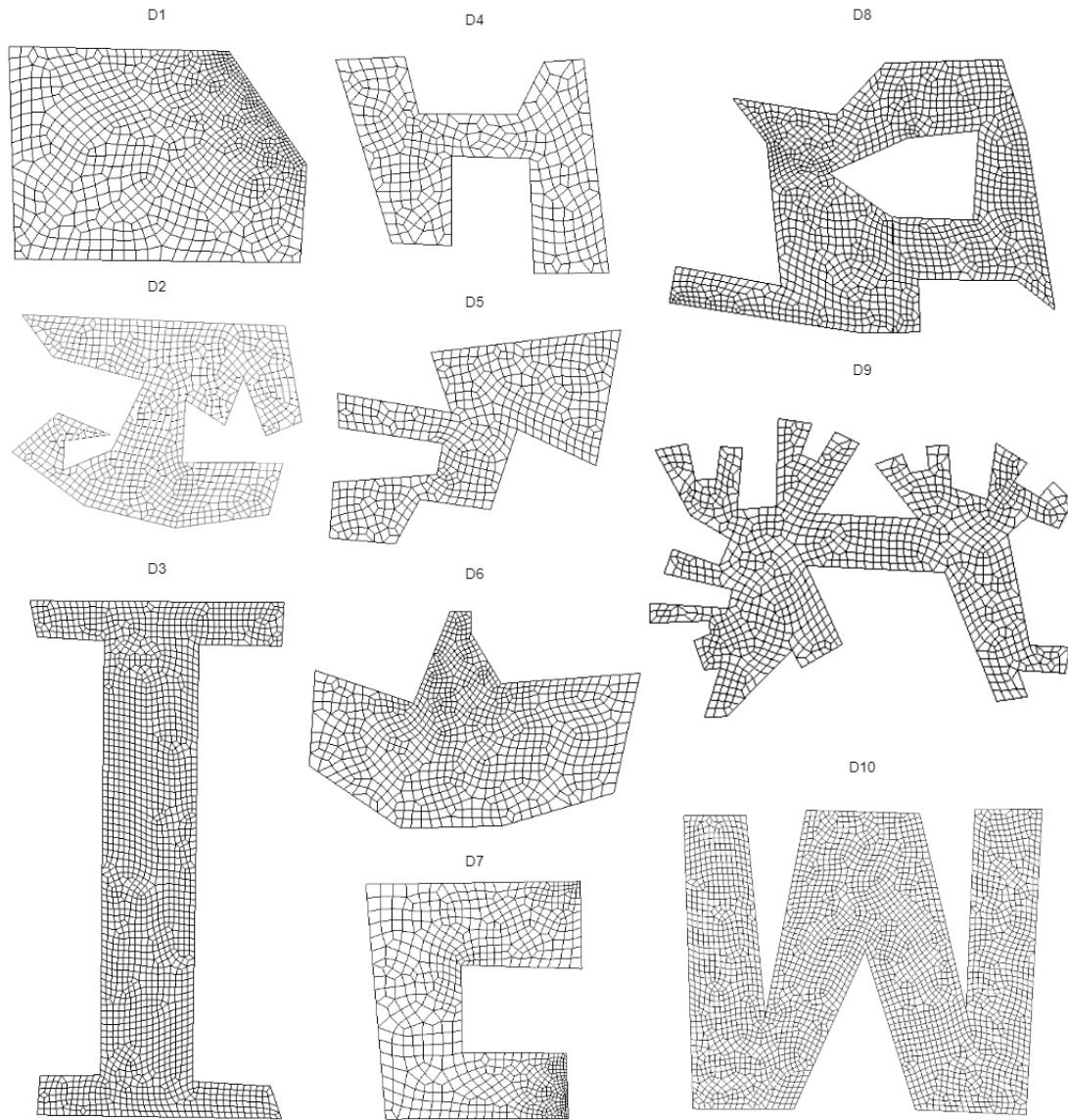


Figure 4.27: Meshing results of GB method

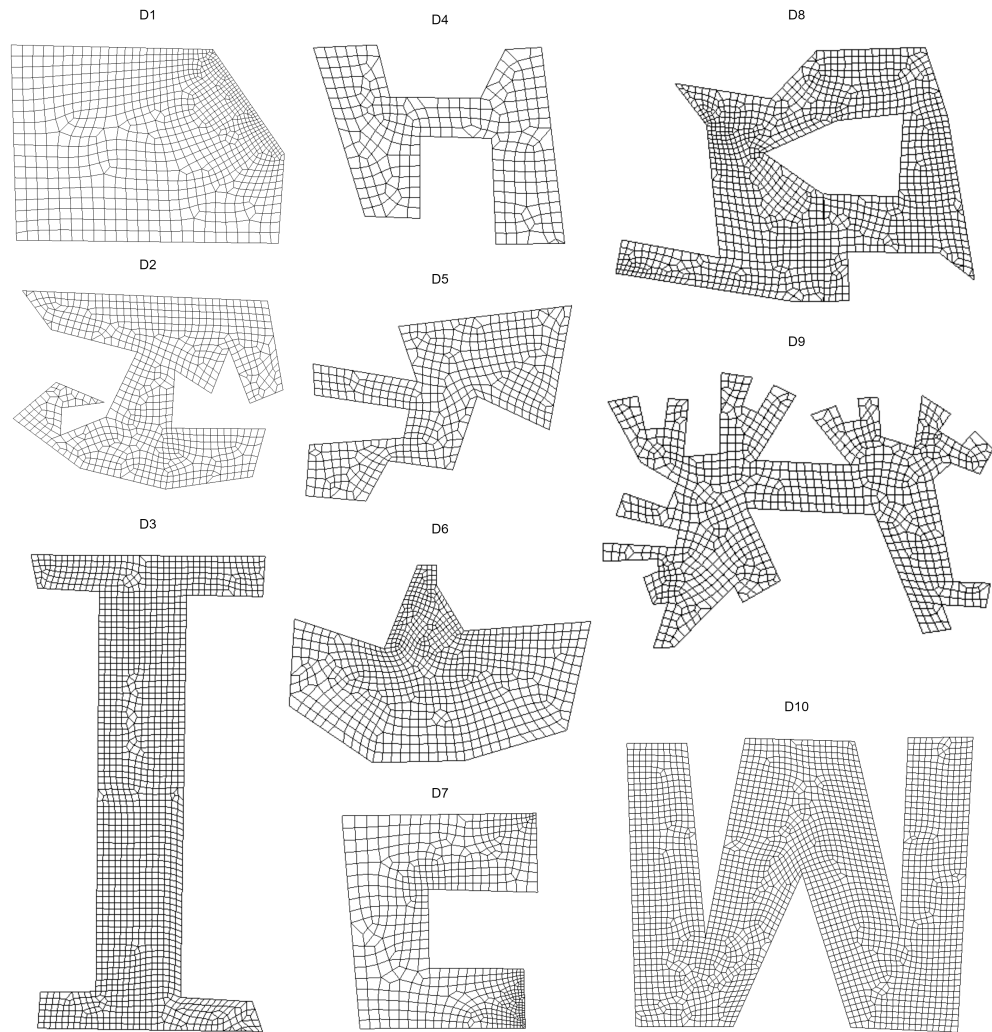


Figure 4.28: Meshing results of GD method



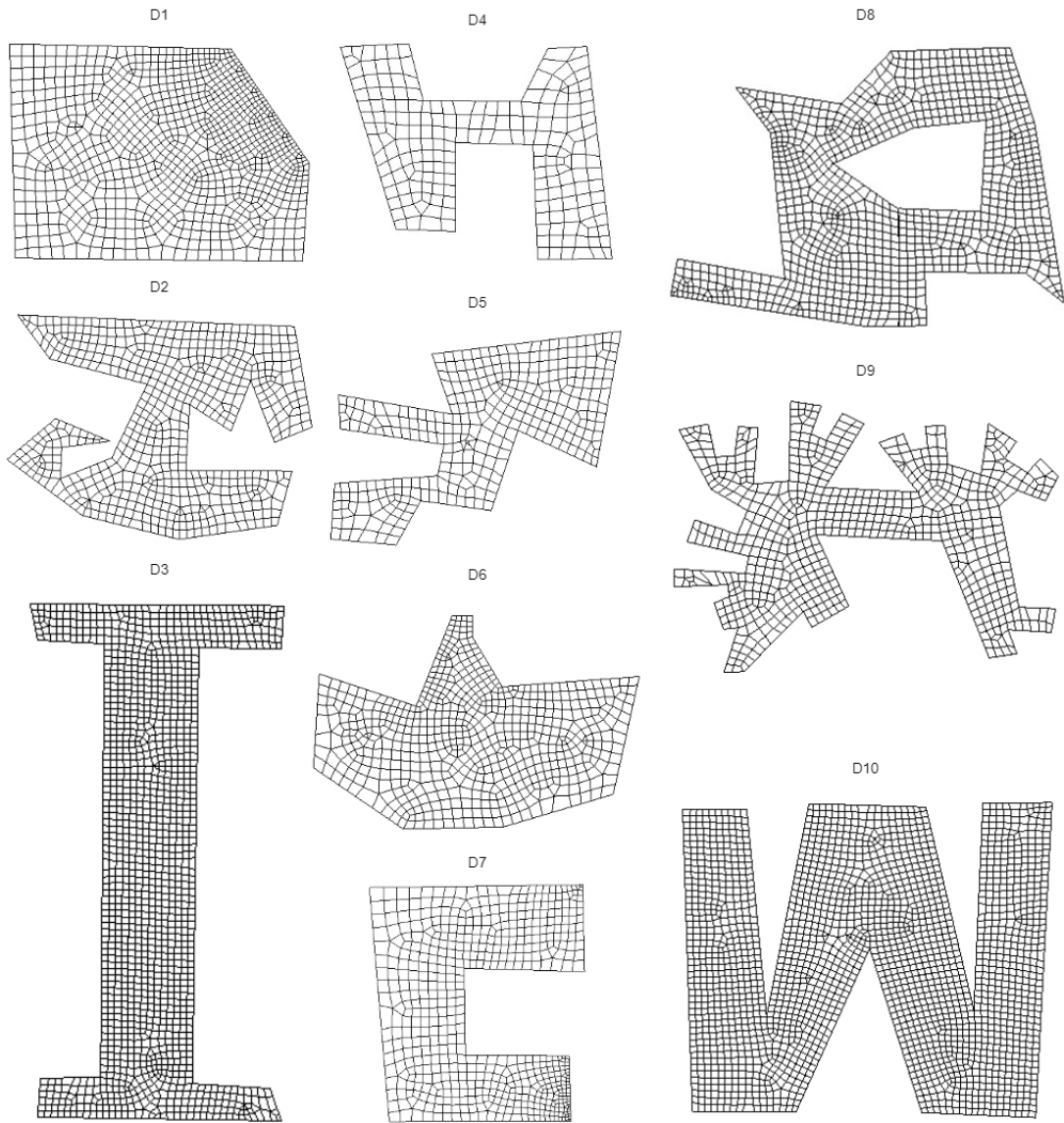


Figure 4.29: Meshing results of CP method

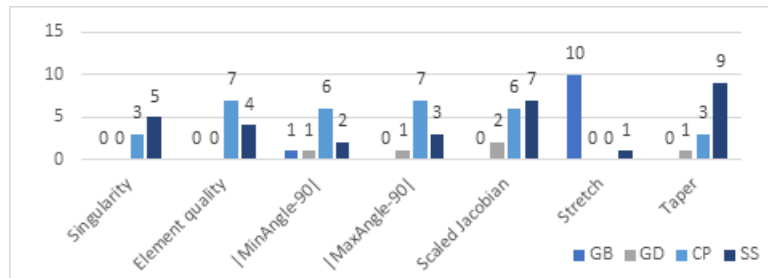


Figure 4.30: Number of domains that each method achieves the best performance in each metrics

methods, the self-learning system FreeMesh-S (SS) has very close results and excels in 2 domains when counting domains with best performances. GB and GD methods are less likely to handle the irregular vertices during meshing because of their dependence on the triangulation and cannot achieve good performance in all the domains. GD method, however, is better than GB because it conducts a pre-processing on the triangulation to make the triangular elements more suitable to be merged into quadrilaterals. Singularity impacts the numerical stability in CFD applications, wrinkles in subdivision surfaces, and breakdown of structured patterns on manifolds (Suresh & Verma, 2019).

- Element quality: FreeMesh-S (SS) resembles with GD method and outperforms GB in this metric while the CP method is the best one and excels in 7 domains. Nevertheless, SS is still comparable with other methods in 4 domains. This measure is to identify how good a single element is, which can be calculated by the Equation 14.
- $|\text{Min/Max Angle} - 90|$ : CP is better than the other 3 methods. FreeMesh-S (SS) has a similar performance with the GD method and outperforms GB. These two metrics are to show the difference between the minimal or maximum internal corners of an element and the degree of 90, because a square is considered as the perfect element in quad meshing, i.e., both of them are  $90^\circ$ .
- Scaled Jacobian: CP and SS have better results than the other two methods while SS is slightly better than CP by excelling in 1 domain with the best performance. GB is the worst one. Scaled Jacobian relates to the interpolation error of finite element solution, for which the value of the best shape is 1.
- Stretch: GB is the best one among the 4 methods and defeats all the methods in 10 domains. FreeMesh-S (SS) has a very similar performance with GD while CP is slightly worse than the two methods by 0.01. Stretch is calculated by the ratio between the shortest edge and longest diagonal length, which indicates the degree of deformation.
- Taper: SS outperforms all the other methods and excels in 9 domains. Taper represents the ratio of the areas of the two triangles separated by a diagonal within a quadrilateral element, which indicates the balanced shape of an element.

In general, the proposed system FreeMesh-S (SS) achieves high performance in all the metrics,

where all the values are within acceptable ranges as specified by the Verdict (Pébay et al., 2008), and outperforms the other three approaches in the metric of Scaled Jacobian and Taper. CP approach has the best performance in metrics of Singularity, Element quality,  $|\text{MinAngle} - 90|$ ,  $|\text{MaxAngle} - 90|$ , and Scaled Jacobian. Even though almost losing all the competitions with other methods, the GB method has the best performance in the aspect of Stretch. For the GD method, it exceeds the GB method in all the other metrics except the Stretch but is still poorer than SS and CP. GD is a quad-dominant method, which cannot discretize the whole domain into quadrilateral elements. Without extra cleanup operations (i.e., element deleting and insertion), CP and GD methods are difficult to generate a full-quad mesh. The proposed FreeMesh-S (SS) directly generates full-quadrilateral elements by the simulated extraction rules from A2C and FNN modules, which does not require any heuristic operations to handle exceptional triangle elements and inverted or flat quadrilateral elements.

#### 4.4.4 Summary

The meshing performance of the proposed self-learning system is thoroughly evaluated by comparing it with the other three popular meshing approaches over 10 complex domain boundaries. As been indicated in the introduction section, there is no single method that could perform the best in every measurement metric. This paper chooses 7 commonly used indices to quantify meshing performance. The proposed FreeMesh-S achieves high in all of them and outperforms other methods in the metrics of Scaled Jacobian and Taper. FreeMesh-S has also a similar performance in Singularity with the best method – CP, which is suitable for applications needing regular meshes. Moreover, by analyzing the computational efficiency of 4 methods, all of them have similar time complexity.

In the model training stage, the A2C shows that the agent can gradually generate elements with very good quality via trial-and-error learning, which is knowledge free and needs no human intervention. The experience extraction module successfully extracts samples (i.e., input-output pairs) for the FNN training, which can turn several hundred elements into several hundred thousand samples. That is essential for the FNN model to get sufficient data and converge fast. The success of the combination of A2C and FNN demonstrates that the proposed self-learning schema is efficient to

obtain the extraction rules and sheds light on its applicability to other computational geometric problems.

## 4.5 Discussion

The proposed self-learning system, FreeMesh-S, has achieved high performance in meshing various complex domain boundaries when comparing with other meshing approaches, as illustrated in the previous experiment section. Furthermore, there are a few important implications of the proposed system from the perspectives of practical and theoretical system design.

### 4.5.1 Domain knowledge dependency

To develop an approach or a system of mesh generation, it requires researchers or developers to be equipped with equivalent knowledge for the selected method. The lifecycle of mesh generation can be divided into four phases: pre-processing, element generation, quality measurement, and post-processing. The primary geometry knowledge required during the different development phases is shown in Table 4.11, respectively. For each phase, there are the following results:

- Pre-processing: Since the GB and GD are indirect methods to produce mesh elements, they need to generate triangulations in handling domains first. They need knowledge (e.g., Delaunay triangulation) to generate those triangles. GD intends to find ways to optimize the triangles in order to recombine them into high-quality quadrilateral elements. It requires, therefore, at least 10 kinds of knowledge in the pre-processing phase; instead, the GB only needs Delaunay triangulation. CP and SS are direct methods for quad meshing, which do not need any pre-processing operations.
- Element generation: GD and GB have a similar procedure to generate elements. The required knowledge for them is the same, including 8 main categories. CP method requires less knowledge and needs 4 categories. SS needs the least knowledge to generate elements, which is also easy for understanding.
- Quality measurement: All the approaches require 2 classes of knowledge. However, the required categories in SS, aspect, and taper ratio, are the simplest ones.

- Post-processing: GB and GD both need operations, including swapping the edges and removing the duplicated vertex in elements. GD also needs to smoothen the mesh using Lp Central Voronoi Tessellation (LPCVT) algorithm (Lévy & Liu, 2010). Moreover, GD is a quadrilateral-dominant method, which means that triangles may exist. CP method also requires geometrical operations, such as deleting and inserting elements and smoothing. SS does not involve any extra operations, and only needs smoothing, which is cleanup free.

Generally, the domain knowledge required for SS is far less and easy to understand than others while maintaining the promising meshing results. GD method has the most complicated geometry-related concepts to learn. GB is slightly less complicated than GD in the pre-processing stage. Since CP and SS are direct meshing methods, they do not need any pre-processing knowledge. However, because CP relies on heuristic post-processing, it needs more knowledge in the development of those cleanup operations. Therefore, SS is a development-friendly method for especially novice developers or researchers. This important benefit makes it less knowledge-dependent in the method development and relieves humans from defining the input-output relations implied in the element extraction rules.

Designing a geometric algorithm is usually extremely difficult and time-consuming because the learning cost of geometric knowledge is high especially for novel researchers. The self-learning and self-evolving FreeMesh-S system takes two steps (i.e., defining the primitive rules and the self-learning process) in developing the algorithm, which smartly balances the human efforts and machine intelligence in the solution development.

#### **4.5.2 The relation between smart design and smart system**

In the experiment, it was found that the derived element extraction rules are applicable to 2D domains with almost any shape. The final obtained model is capable of meshing domains with arbitrary boundaries without any additional training to fit their geometric specialty. Many existing meshing algorithms have difficulty in achieving high-quality elements in domains with sharp angles or in restricted domains, such as one side of the geometric boundary (Rushdi et al., 2017). Even within a more regular geometric domain, they usually create flat or inverted elements that require heavy post-processing operations. It is challenging to guarantee that a method works on domains

with arbitrary shapes while maintaining high-quality. One of the most important properties of a smart system is that it can make adaptive decisions to maintain the overall performance of the system. In the proposed FreeMesh-S system, the element extraction rules can be adaptive to various mesh boundaries. Three types of atomic rules were conceptually designed for the meshing problem through human intelligence, which is supported by a design methodology (Zeng & Cheng, 1991; Zeng & Yao, 2009). This primitive property of the solution provides the foundation for the smart applicability to arbitrary domains.

However, to manually obtain these rules are time-consuming and even cannot meet the specified quality requirements. For example, human designers spend a lot of time to observe various kinds of errors and come up with many heuristic operations such as element splitting, swapping, and collapsing to refine the overall mesh quality (Blacker & Stephenson, 1991; Rushdi et al., 2017; White & Kinney, 1997). Therefore, these rules should be smartly designed to be obtained automatically. It is also another important property of a smart system to self-evolve from trial-and-error. The proposed FreeMesh-S can automatically improve the performance of the extraction rules by integrating trial-and-error learning and supervised learning. The formed self-learning schema by A2C and FNN can quantitatively and automatically construct the input-output relations of the atomic rules without any human intervention.

In summary, the smart system can make adaptive decisions according to the changes in the external environment and can be self-evolved from experiences. The smart design will greatly balance human efforts and machine intelligence and let the system equip with these two properties. The FreeMesh-S has shed light on how to smartly design such a smart system and can be extended to other fields where the problem can be cast as recursive or atomic design problems.

### **4.5.3 Limitations**

There are still a few limitations for the proposed self-learning system to resolve in the future. One challenge is that a domain with a very sharp angle is difficult to mesh. For example, a domain has a corner with a  $1^\circ$  angle. It is, however, a common problem for almost every meshing method (Shewchuk, 2012). The problem can be solved by cutting off the sharp corners or adding heuristic rules to form a quadrilateral element first before applying the standard meshing method to the



Figure 4.31: Multi-connected domain to single-connected domain. (1) Multi-connected domain; (2) Single-connected domain.

remaining boundary. The FreeMesh-S is also limited to mesh in single-connected domains. For a multi-connected domain (e.g., domain D9), the comprised solution taken by FreeMesh-S is to insert a cutting line to transform it into a single-connected domain, as shown in Figure 4.31. If there is more than one hole inside the domain, the same operation can be applied to each one of them.

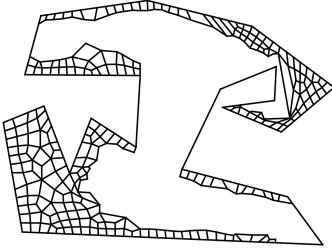
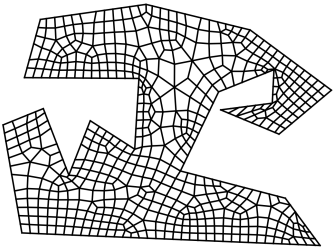
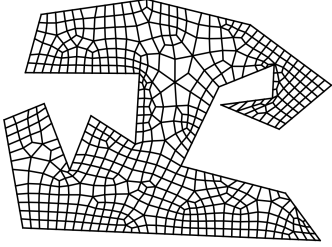
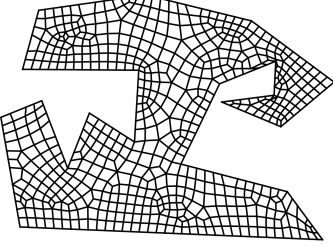
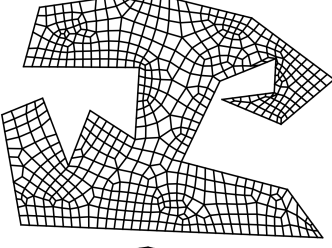
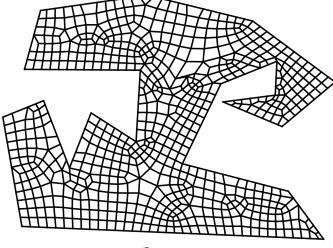
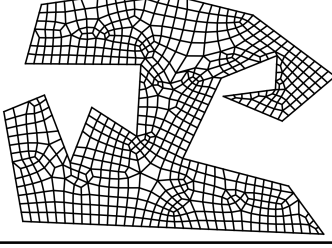
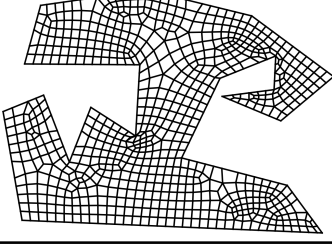
## 4.6 Conclusions

This paper aims to solve a challenging problem – learning from experience about the element extraction rules for achieving high-quality meshes in both the boundary and interior of complex geometric domains. Conventionally, to guarantee the overall mesh quality, element extraction methods rely heavily on heuristic operations, which is extremely difficult, expensive, and time-consuming for human algorithm designers. In this paper, a self-learning system FreeMesh-S is proposed to generate quadrilateral elements by automatically acquiring robust and high-quality element extraction rules. Firstly, according to the recursive logic, the extraction rules are categorized into three types (i.e., adding 1, 2, and 3 edges, respectively). Then, a novel learning schema formed by A2C and FNN is used to construct the input-output relations quantitatively and automatically. A2C simulates human trial-and-error learning to generate effective samples for training the three element extraction rules using FNN. The experiment results demonstrate that derived element extraction rules can be adaptive to various boundary shapes and achieve high performance in all quality metrics, especially the highest in the metrics of Scaled Jacobian and Taper, in comparison with the other three widely used meshing methods while requiring the minimal and simplest geometric knowledge.

Furthermore, this paper, for the first time, formulates the mesh generation problem as a Markov Decision Process (i.e., sequential decision making) problem. It closely bridges machine learning techniques with mesh generation and provides a smart way to balance the human efforts and machine intelligence in algorithm development, which could shed light on how to smartly design a smart system.



Table 4.6: Self-evolving process of FNN model.

	Sample source	#tE	#vE	#eS	Meshing result
Round 1		241	124	35,324	
Round 2		558	419	279,770	
Round 3		605	477	295,039	
Round 4		678	530	345,158	

The hidden layer of this FNN model is [256, 256];

#tE: total number of elements;

#vE: number of valid elements with quality  $\eta^e > 0.7$ ;

#eS: number of extracted samples.

Table 4.7: Different element quality threshold comparison: L, H indicates if the lower value or higher value is preferred, respectively. The hidden layer of this FNN model is [32, 64, 128, 64, 32, 16]. Each metric value is the average value of the three domains (D1, D2, and D3).

	Quality $\eta^e > 0.5$	Quality $\eta^e > 0.7$	Quality $\eta^e > 0.8$
<b>Singularity (L)</b>	65.7	<b>57.3</b>	62
<b>Element quality (H)</b>	0.78	0.82	<b>0.84</b>
<b> MinAngle - 90  (L)</b>	15.3	11.6	<b>11.3</b>
<b> MaxAngle - 90  (L)</b>	16.6	<b>12</b>	12.1
<b>Scaled Jacobian (H)</b>	0.93	<b>0.95</b>	<b>0.95</b>
<b>Stretch (L)</b>	<b>0.82</b>	0.84	0.85
<b>Taper (L)</b>	0.08	<b>0.06</b>	0.07

Table 4.8: Comparison of FNN network structures: L, H indicate if the lower value or higher value is preferred, respectively. Each metric value is the average value over the three domains (D1, D2, and D3).

	Hidden layers [64, 64]	Hidden layers [256, 256]	Hidden layers [64, 128, 64, 32, 16]	Hidden layers [32, 64, 128, 64, 32, 16]
<b>Singularity (L)</b>	72.6	84.7	<b>52.3</b>	57.3
<b>Element quality (H)</b>	0.84	0.84	<b>0.87</b>	0.82
<b> MinAngle - 90  (L)</b>	11.5	11.2	<b>9.</b>	11.6
<b> MaxAngle - 90  (L)</b>	12.6	12.1	<b>10</b>	12
<b>Scaled Jacobian (H)</b>	0.95	0.95	<b>0.97</b>	0.95
<b>Stretch (L)</b>	0.85	0.86	0.89	<b>0.84</b>
<b>Taper (L)</b>	0.08	0.09	<b>0.06</b>	<b>0.06</b>

Table 4.9: Meshing speed (elements per second) of A2C and FNN for all 10 domains. Avg. indicates the average speed; STD indicates the standard deviation.

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	Avg.	STD
<b>A2C</b>	31.84	16.73	11.71	38.36	28.43	13.25	3.82	12.41	33.95	6.63	19.71	11.69
<b>FNN</b>	35.66	20.78	17.62	55.16	40.03	18.32	26.94	16.17	42.1	14.48	28.73	13.08

Table 4.10: Mesh quality metrics of four methods on 10 domainsL, H indicate if the lower value or higher value is preferred, respectively. The value in bold means the best among other approaches in a domain. GB refers to Gmsh-Blossom; GD indicates Gmsh-DelQuad; CP is CUBIT-Pave; and SS means the proposed self-learning system, FreeMesh-S.

	Singularity (L)			Element quality (H)			MinAngle - 90  (L)			MaxAngle - 90  (L)			Scaled Jacobian (H)			Stretch (L)			Taper (L)									
	GB	GD	CP	SS	GB	GD	CP	SS	GB	GD	CP	SS	GB	GD	CP	SS	GB	GD	CP	SS	GB	GD	CP	SS				
D1	217	81	87	47	0.79	0.87	0.87	0.88	15.89	9.70	9.5	8.6	18.1	10.6	11.3	9.5	0.93	0.97	0.96	0.97	0.81	0.88	0.88	0.9	0.14	0.07	0.09	0.07
D2	188	136	62	82	0.78	0.81	0.84	0.82	16.55	14.28	11.7	12.8	18.3	16.0	13.4	13.8	0.92	0.93	0.94	0.94	0.81	0.84	0.86	0.84	0.14	0.11	0.1	0.09
D3	213	86	42	28	0.81	0.90	0.92	0.91	7.15	5.4	5.7	6.4	7.68	5.7	6.1	6.56	0.97	0.99	0.98	0.99	0.92	0.93	0.93	0.92	0.04	0.03	0.03	0.02
D4	97	54	22	21	0.75	0.81	0.82	0.83	19.8	13.9	12.2	13	21.9	15.7	13.8	13.8	0.90	0.93	0.95	0.95	0.78	0.83	0.84	0.84	0.17	0.13	0.11	0.09
D5	123	85	35	39	0.77	0.80	0.82	0.82	17.2	14.8	12.1	13.6	19	16.3	13.8	13.9	0.92	0.93	0.94	0.95	0.80	0.82	0.84	0.84	0.15	0.12	0.1	0.07
D6	251	202	101	81	0.76	0.79	0.81	0.78	18.8	16.5	13.1	15.5	20	18	15	16.2	0.91	0.92	0.94	0.93	0.79	0.81	0.83	0.80	0.16	0.14	0.11	0.09
D7	163	84	46	90	0.78	0.85	0.86	0.8	16.2	11.6	9.3	13.3	18.8	13.2	10.6	14.3	0.092	0.95	0.97	0.94	0.80	0.85	0.87	0.81	0.15	0.1	0.08	0.1
D8	178	101	79	61	0.79	0.83	0.84	0.85	16	12.8	11.7	10.6	17.8	14.1	13.8	11.7	0.93	0.95	0.95	0.96	0.81	0.86	0.86	0.86	0.14	0.093	0.11	0.08
D9	273	179	81	110	0.77	0.83	0.85	0.82	17.4	12.9	11.6	13.6	19	14.3	12.2	14.4	0.92	0.94	0.95	0.94	0.8	0.85	0.87	0.84	0.13	0.1	0.09	0.09
D10	449	201	57	69	0.8	0.87	0.92	0.89	4.6	9.7	5.6	6.9	16.2	10.6	6.1	7.3	0.94	0.96	0.98	0.98	0.83	0.9	0.93	0.9	0.11	0.05	0.03	0.03
Avg.	215.2	120.9	61.2	62.8	0.78	0.84	0.86	0.84	14.96	12.16	10.3	11.4	17.7	13.5	11.6	12.2	0.84	0.95	0.96	0.96	0.82	0.86	0.87	0.86	0.13	0.09	0.08	0.07

Table 4.11: Comparison of geometry knowledge required to develop and implement the algorithms and system

	<b>Gmsh-Blossom</b>	<b>Gmsh-DelQuad</b>	<b>CUBIT-Pave</b>	<b>Self-learning system</b>
Pre-processing	Delaunay triangulation	Surface; Parametrization/ reparametrization; Surface curvature; Laplace equation; Dirichlet boundary conditions; L norm; frontal Delaunay; Delaunay triangulation		
Element generation	Mesh size field; blossom algorithm; cross-field; perfect matching of graph; Tutte's theorem; cubic graph; quad-vertex-merge optimization; doublet collapse optimization	Mesh size field; blossom algorithm; cross-field; perfect matching of graph; Tutte's theorem; cubic graph; quad-vertex-merge optimization; doublet collapse optimization	Boundary shape; isoparametric smooth; Laplacian smoothing; segment intersection	Segment intersection; boundary shape; Laplacian smoothing
Quality measurement	Adimensional length; mesh size field	Adimensional length; mesh size field	Oddy ratio; distortion metric	Aspect ratio; taper ratio
Post-processing	Edge swap; vertex duplication	LPCVT smoothing, edge swap; vertex duplication	Element deletion; element insertion, Laplacian smoothing	Laplacian smoothing

Table 4.12: Averaged mesh quality metrics of four methods on 10 domains: L, H indicate if the lower value or higher value is preferred, respectively. The value in bold means the best among other approaches in a domain. GB refers to Gmsh-Blossom; GD indicates Gmsh-DeQuad; CP is CUBIT-Pave; and SS means the proposed self-learning system, FreeMesh-S.

	<b>GB</b>	<b>GD</b>	<b>CP</b>	<b>SS</b>
<b>Singularity (L)</b>	215.2	120.9	<b>61.2</b>	62.8
<b>Element quality (H)</b>	0.78	0.84	<b>0.86</b>	0.84
<b> MinAngle - 90  (L)</b>	14.96	12.16	<b>10.3</b>	11.4
<b> MaxAngle - 90  (L)</b>	17.7	13.5	<b>11.6</b>	12.2
<b>Scaled Jacobian (H)</b>	0.84	0.95	<b>0.96</b>	<b>0.96</b>
<b>Stretch (L)</b>	<b>0.82</b>	0.86	0.87	0.86
<b>Taper (L)</b>	0.13	0.09	0.08	<b>0.07</b>

## Chapter 5

# Reinforcement learning for automatic quadrilateral mesh generation: a soft actor-critic approach

### Abstract

This paper proposes, implements, and evaluates a Reinforcement Learning (RL) based computational framework for automatic mesh generation. Mesh generation, as one of six basic research directions identified in NASA Vision 2030, is an important area in computational geometry and plays a fundamental role in numerical simulations in the area of finite element analysis (FEA) and computational fluid dynamics (CFD). Existing mesh generation methods suffer from high computational complexity, low mesh quality in complex geometries, and speed limitations. By formulating the mesh generation as a Markov decision process (MDP) problem, we are able to use soft actor-critic, a state-of-the-art RL algorithm, to learn the meshing agent's policy from trials automatically, and achieve a fully automatic mesh generation system without human intervention and any extra clean-up operations, which are typically needed in current commercial software. In our experiments and comparison with a number of representative commercial software, our system demonstrates promising performance with respect to generalizability, robustness, and effectiveness.

*Keywords:* Reinforcement learning (RL), mesh generation, soft actor-critic (SAC), neural networks (NNs), computational geometry.

## 5.1 Introduction

Reinforcement learning (RL) has been flourished in many fields, such as games (Silver et al., 2016), healthcare (Gottesman et al., 2019), natural language processing (W. Y. Wang, Li, & He, 2018), and NP-hard problems (Mazyavkina, Sviridov, Ivanov, & Burnaev, 2021) (to name a few). However, it is rarely applied to the area of computational geometry, especially in the field of mesh generation. Mesh generation is the fundamental step in conducting numerical simulations in the fields of finite element analysis (FEA), computational fluid dynamics (CFD), or graphic model rendering (Gordon & Hall, 1973; Roca & Loseille, 2019). Mesh generation techniques have been identified as one of the six basic research directions in NASA’s Vision 2030 CFD study (Slotnick et al., 2014). It discretizes complex geometric domains into a finite set of (geometrically simple and bounded) elements, such as triangles or quadrilaterals (in 2D geometries) or tetrahedra or hexahedron (in 3D geometries). Since the reliable automation and high quality of mesh representation matter significantly to the numerical simulation results, mesh generation has continued to be significant bottlenecks in those fields due to algorithm complexities, inadequate error estimation capabilities, and complex geometries (Slotnick et al., 2014). The rapid progress of deep learning and RL techniques offers the potential for mesh generation to have radical advances to overcome those challenges.

### 5.1.1 Mesh generation challenges

In most real-world engineering problems, the geometries to mesh have complex shapes and sizes; hence unstructured meshes are preferred because of the robustness and efficiency in adaption (Bommes et al., 2013; Garimella et al., 2004; Owen, 1998). Quadrilateral elements of unstructured mesh can achieve more accurate numeral simulation results comparing with triangular mesh (Verma & Suresh, 2017), which will be used in this paper. Conventional methods for quadrilateral mesh generation can be classified into two categories: indirect and direct methods (Shewchuk, 2012). Indirect methods start with a triangular mesh and then transform the triangular elements into quadrilateral

elements by various strategies, including optimization (Brewer et al., 2003), refinement and coarsening (Garimella et al., 2004), simplification (Daniels et al., 2008), perfect matching (Remacle et al., 2012). These methods, however, suffer from a large number of irregular vertices, which is undesired in numerical simulations. Direct methods generate quadrilateral elements directly. These methods include 1) advancing front technique, which recursively generates elements from the domain boundary and updates its boundary inwardly by cutting the generated elements until the whole domain is filled with quadrilateral elements (Blacker & Stephenson, 1991; White & Kinney, 1997; Zeng & Cheng, 1993; Zhu et al., 1991); 2) modifying the quadtree background grid to conform to the domain boundaries (Atalay et al., 2008; Baehmann et al., 1987; Liang et al., 2010; Liang & Zhang, 2012); 3) packing techniques, including square packing (Shimada et al., 1998) and circle packing (Bern & Eppstein, 2000); and template-based mapping methods (Gengdong & Hua, 1996). However, the generated quadrilateral meshes are usually not complete (e.g., containing triangular elements), having flat or inverted quadrilateral elements, and too much irregular arrangement. Therefore, a large amount of cleanup operations are implemented to improve the mesh quality. The strategies range from pre-processing, such as dividing complex geometries into small regular regions to facilitate generating regular mesh (C. Liu et al., 2017), to post-processing, such as reducing the singularity (Verma & Suresh, 2017), performing iterative topological changes (e.g., splitting, swapping, and collapsing elements) (Docampo-Sanchez & Haimes, 2019), and mesh adaptation (Verma & Suresh, 2018).

Therefore, many efforts have been made to improve meshing algorithms, including using heuristic clean-up operations to reduce the flat or inverted elements by re-adjusting location and connectivity of element vertices; using global and local remeshing techniques to reconstruct generated mesh in terms of topological and geometric features (Verma & Suresh, 2017); and pre-processing the geometry to ease the element generation process (C. Liu et al., 2017). Although the mesh quality has improved, the extra treatments make the meshing algorithms suffering from high complexity and speed limitations, and incur expensive trial-and-error tuning for researchers.

The aforementioned work names only a few recent advancements towards the automation of mesh generation. Although those extra treatments could achieve the high-quality, they bring in additional



computational expense besides already complicated meshing algorithms and decrease the automation to some extent. Moreover, algorithm development is usually heuristic and requires human designers to search for knowledge in a time-consuming manner. It is urgent and necessary to build an efficient computational framework for mesh generation to sidestep the complexities and computational burden of existing meshing algorithms and relieve human algorithm designers from an inefficient and incomplete search of heuristic knowledge. This framework should provide high-quality meshes for various complex geometries while maintaining robust automation, and avoiding any extra treatments.

### 5.1.2 Related work

Machine learning techniques have been approved successfully to solve complex and time-consuming problems in various industrial areas. Researchers have started to combine mesh generation with artificial intelligence. Existing work can be classified into three kinds: 1) mesh optimization. [Z. Zhang et al. \(2020\)](#) combined deep neural networks (NNs) with an external mesh generator to refine meshes with a better element distribution that can accurately solve Partial Differential Equations (PDE). [J. Yang et al. \(2021\)](#) trained a mesh refinement policy to dynamically adjust the mesh resolution for a better trade-off between simulation accuracy and computation cost via RL. 2) mesh reconstruction. Neural Mesh Flow (NMF) ([Gupta, 2020](#)) was a 3D mesh reconstruction method by deforming a template mesh into a target mesh using several Neural Ordinary Differential Equation (NODE, a deep neural network model ([Chen et al., 2018](#))) blocks. NMF had its strength in the mesh property of manifoldness, which was beneficial for graphic rendering and 3D printing. Pixel2mesh ([N. Wang et al., 2018](#)) was a deep learning architecture to produce 3D triangular mesh from a single RGB image. The generation process was a series of deformation from an ellipsoid with image perceptual features extracted by a convolutional neural network (CNN) to a target mesh model represented by a Graph Convolutional Network (GCN ([Defferrard et al., 2016](#))). [Wen et al. \(2019\)](#) extended the Pixel2mesh for better shape quality from multi-view images. MeshCNN ([Hanocka et al., 2019](#)) was a NN architecture for simplifying meshes by specialized convolution and pooling operations that collapse edges. These recent advances show that mesh related works have gained significant attention in computational graphics and computer vision. 3) mesh generation. [Nechaeva \(2006\)](#)

proposed an adaptive mesh generation algorithm based on self-organizing maps (SOM) (an unsupervised neural networks-based method), which adapts a given uniform mesh onto a target physical domain through mapping. The proposed method intended to overcome the drawbacks of SOM in tackling inaccurate mesh in the domain border and mesh construction for non-convex domains. Pointer networks (Vinyals et al., 2015), was a new neural architecture that aimed to solve combinatorial problems by NNs and could be used to generate triangular meshes by outputting a set of triplets of integers (each forms a triangle) that correspond to the order of input points. The input contains both the points on the boundary and internal area of the geometry domain. As it is not designed for meshing problems, the final mesh is not robust and partially covered with triangular elements with even intersecting edges. Papagiannopoulos et al. (2021) proposed a triangular mesh generation method with three NNs. Given the training data derived from Constrained Delaunay Triangulation algorithm (Chew, 1989), three NNs could predict the number of candidate inner vertices (to form a triangular element), coordinates of those vertices, and their connection relations with existing segments on the boundary, respectively. Because of the fixed input size and complex network architectures, the method cannot adapt to arbitrary and complex geometry domains. Additionally, the resulting meshes are constrained by the quality and sample diversity of the selected training data, especially when it comes to geometries with complex boundary shapes.

Zeng and Cheng (1993) proposed a knowledge-based method, FreeMesh, to recursively extract quadrilateral elements following a domain boundary until the remaining boundary becomes a quadrilateral element, based on the recursive logic of design (Zeng & Cheng, 1991). Yao et al. (2005) improved the FreeMesh approach by introducing an artificial neural network (ANN) to learn the element extraction rules from a set of pre-selected samples of good quality quad meshes. Pan, Huang, Wang, Cheng, and Zeng (2021) built a self-learning automatic quadrilateral mesh generation system by combining a feedforward neural network (FNN) and an RL algorithm (Advantage Actor-Critic, A2C). The RL is used to provide high-quality samples to train an FNN model that serves as the final mesh generator to mesh various geometry domains. The resulting meshes outperform three conventional methods in two quality metrics. The shortcoming is that the training procedure is not fully automatic and requires extract operations from human designers, such as carefully designing the strategies to select samples for the FNN training in order to balance samples for applying rules

in various situations.

Although many artificial intelligence (AI) techniques have started to be explored and applied to the field of mesh generation, a robust computational framework for mesh generation that can replace the standard mesh generation algorithms is still missing. With the rapid progressing of High Performance Computing (HPC), mesh generation algorithms will intuitively encounter tasks requiring higher resolution simulations, fast, and reliable processing, which could make conventional methods problematic. A new meshing algorithm that exploits both emerging HPC capabilities and machine learning/deep learning algorithms remains open exploration with great potential. In general, supervised NN-based methods rely on training data from existing conventional methods or handmade data, which cannot guarantee the overall quality and adapt to other domain boundaries; they have complex network structures, which increases the training difficulty; they are hard to apply to other domains with complex boundaries because of the fixed input size and generation strategies.

RL is a well-known self-taught learning paradigm for solving sequential decision making problems and has been successfully demonstrated in many fields. By formulating the mesh generation as a Markov decision process (MDP) (Pan, Huang, Wang, et al., 2021), the meshing policy can be learned by the interactions between the agent and the environment without the need for supervised labels. Comparing with most existing benchmark problems and complex applications of RL (Machado et al., 2018; Osband et al., 2019), the mesh generation problem poses new challenges for learning an optimal policy: 1) the environments (i.e., the geometries to mesh) have diverse shapes and sizes, which requires the policy to be adaptable and general; 2) the credit assignment for each time step to achieve the final mesh is difficult since the objective is not intuitive; 3) there is a trade-off between the immediate reward achieved by the current action and the future rewards that depend on the state of the environment caused by the current action, because the shape and size of the current generated element will shape the remaining boundary of the geometry to mesh. In the previous work (Pan, Huang, Wang, et al., 2021), these challenges are not addressed because the A2C's purpose is to generate samples for an FNN model rather than complete the mesh. Therefore, this article redesigns the state representation, action formulation, and reward function to address those challenges. The new reward function could address the issues, including a) the trade-off between the current element quality and an overall mesh high quality and b) the completion of the mesh within

finite steps. A novel RL technique, the soft actor-critic (SAC) method (Haarnoja, Zhou, Abbeel, & Levine, 2018; Haarnoja, Zhou, Hartikainen, et al., 2018), is used to solve the learning of the meshing policy. SAC is one of the state-of-the-art off-policy learning algorithms and has faster learning efficiency and better stability in hyperparameters tuning comparing with the A2C method.

The mesh generation problem also has the potential to study many RL topics, such as state representation and reward specification. Usually, the performance of newly invented or improved RL algorithms is evaluated crossing many benchmark problems (Mnih et al., 2013). However, these benchmark problems have some limitations: 1) the internal dynamic mechanisms of different problems may impact the performance and hardly provide accurate evaluation for RL methods, especially complex problems (Berner et al., 2019; Vinyals et al., 2017); 2) the state representations are different across different problems, which may hinder the algorithm analysis; 3) some of the problems are simple, and the reward function is relatively simple (i.e., game scores (Mnih et al., 2013)), which cannot study the reward specification in-depth. The challenges of the mesh generation problem enable it to be a potential benchmark problem for understanding many RL issues. The diverse environments can be used to test the scalability and generality of the learned policy; the range of agent’s observation can be easily adjusted and used to study the partial MDP; the reward designing can be used for testing exploration and credit assignment problems.

### 5.1.3 Contributions

In this paper, we propose, implement, and evaluate an RL based fully automatic mesh generation system. The main contributions of this paper are summarized as follows.

- (1) The proposed RL-based automatic mesh generation system achieves a level of fully automatic mesh generation without human intervention and any extra clean-up operations, which are typically needed in current commercial software.
- (2) In our experiments and comparison with a number of representative commercial software, our system demonstrates competitive performance in several aspects, including mesh regularity, high quality in various geometry domains, and the necessity of handling not expected elements (i.e., triangles).

- (3) Our experiments demonstrate an interesting approach to achieving generalizability and scalability comparing with many RL applications by using partial observation of the environment.
- (4) This RL application in mesh generation could contribute itself to RL as an excellent benchmark problem, which has the following features: diverse meshing environments that control the problem difficulties, flexible state representation by altering the observation, and no fixed representation of the meshing objective that provides freedom for exploring the reward specification.

The rest of the paper is organized as follows. Section 6.2 presents the formulation of the mesh generation problem into an MDP problem and fundamental components. Section 6.3 introduces the detailed implementation of SAC for mesh generation. Section 6.4 evaluates the performance of the proposed method and comparing with other state-of-the-art meshing approaches. Section 6.5 discusses the important improvements of the proposed method to mesh generation and RL communities. Section 6.6 concludes this article and indicates some future directions.

## 5.2 Problem formulation and fundamentals

In this section, the formulation of mesh generation as an MDP problem and related RL techniques are introduced.

### 5.2.1 Problem formulation

The problem of non-simplicial mesh generation (i.e. quadrilaterals) in single-connected 2D domains is studied. The purpose of mesh generation is to discretize a geometric domain (see Figure 6.1 (a)) into quadrilateral elements (see Figure 6.1 (b)). The boundary of the domain,  $B$ , is composed of piecewise linear segments and is then represented as a sequence of vertices  $[V_1, V_2, \dots, V_N]$ . The final mesh,  $\Omega$ , is composed of a set of quadrilateral elements  $[Q_1, Q_2, \dots, Q_M]$ . Consequently, a mesh should satisfy: 1) each element is a quadrilateral; 2) the inner corner of each element should be between  $45^\circ$  and  $135^\circ$ ; 3) the aspect ratio (the ratio of opposite edges) and taper ratio (the ratio of neighbouring edges) of each quadrilateral should be within a predefined range; 4) the transition between a dense mesh and a coarse mesh should be smooth (Zeng & Cheng, 1993; Zeng & Yao,

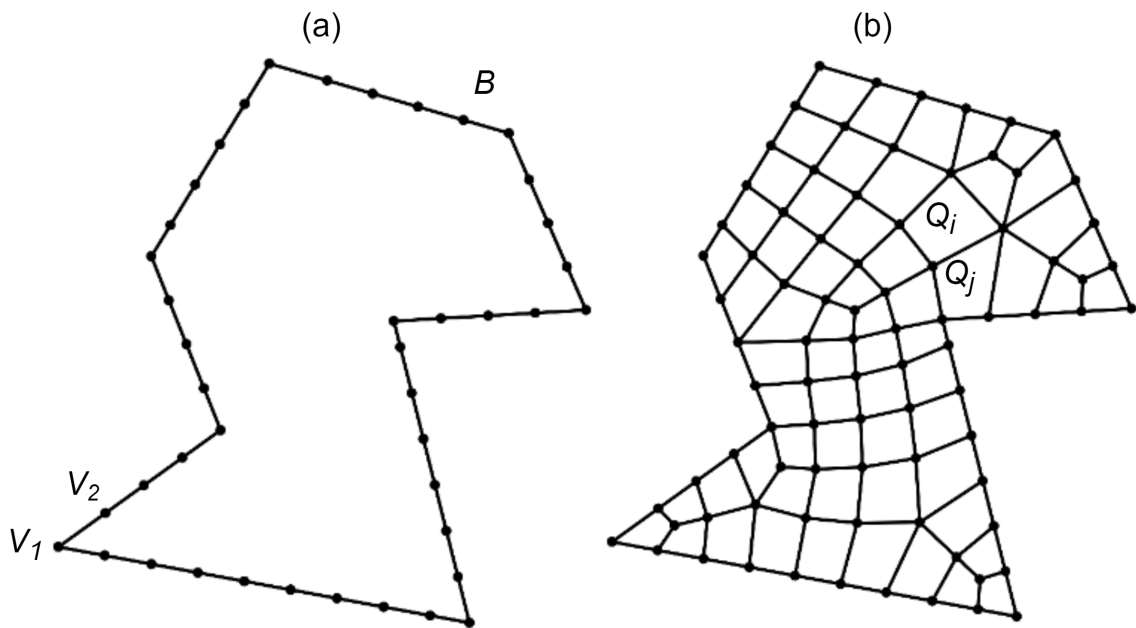


Figure 5.1: Meshing problem. (a) The initial geometry is defined by the boundary  $B$  consisting of a set of vertices  $V$ ; (b) The final mesh is the result of discretization into a set of quadrilateral elements  $Q$ .

2009).

In our previous work, the meshing problem is preliminarily formulated as an RL problem (Pan, Huang, Wang, et al., 2021). The mesh generation problem can be seen as a control problem with a sequence of steps to generate mesh elements. As illustrated in Figure 5.2, quadrilateral elements can be generated one by one, starting from the boundary of the domain and updating its boundary inwardly by removing the generated element, until there are only four vertices left in the updated boundary, which forms the last element. This procedure can be discretized into four steps: 1) choosing a vertex (called reference vertex in this paper) from the boundary; 2) constructing an element around the reference vertex; 3) removing the generated element, and; 4) updating the boundary. The meshing boundaries are continuously shaped by the generated elements. In each iteration, the meshing problem is evolved and depends on the previous solution (i.e., the mesh element) of the previous problem, which forms a sequential decision making process, also called an MDP. The quality of future elements hinges on the shape and size of the previously generated elements. Usually, when the updated boundary is approaching itself, the quality of the element will start to drop, or even elements meeting the requirements can not be constructed.

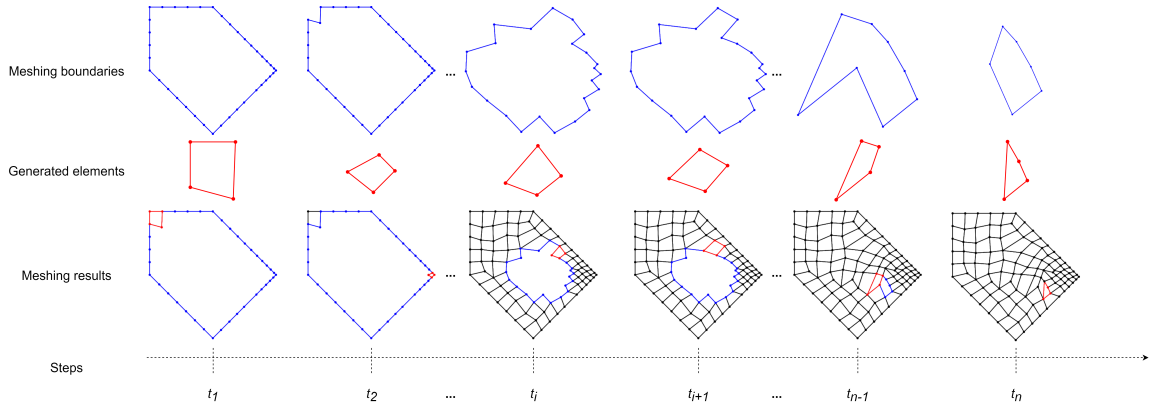


Figure 5.2: A sequence of actions took by the mesh generator to complete the mesh. At each time step  $t_i$ , an element (in red) is extracted from the current boundary (in blue). The boundary is then updated by cutting off the element and serves as the meshing boundary in the next time step  $t_{i+1}$ . This process continues until the updated boundary becomes an element.

In this formal framework, the meshing boundary is considered as the environment, in which the agent generates quadrilateral elements using a set of actions. The goal of the agent is to complete the mesh given any geometric objects, which satisfies the requirement mentioned above. The overall architecture is shown in Figure 5.3. The agent, at each time step  $t$ , observes a state  $S_t$  from the environment, and conducts an action  $A_t$  applied to the environment. The environment responds to the action and transits into a new state  $S_{t+1}$ . It then reveals the new state and provides a reward  $R_t$  to the agent. This process forms an iteration and repeats until a given condition is satisfied (i.e., the RL problem is solved).

The main problem in the previous method is that the procedure to obtain the final mesh agent is complex and involves three phases: RL-based sampling, experience extraction, and FNN training. It is not a fully automatic method and requires extract operations from human designers, such as carefully designing the strategies to select samples because the imbalanced samples will hinder the final performance.

To overcome this problem and reduce the extra operations, an automatic mesh generation method based solely on RL is proposed in this paper. The meshing policy is self-learned from the interactions between the RL agent and the environment through trial-and-error. The learned policy could adapt to any geometric domain because a partial boundary is adopted as the state representation

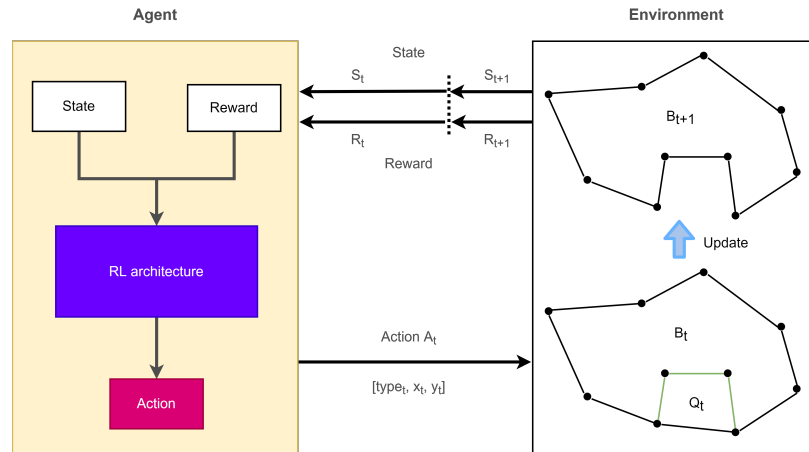


Figure 5.3: The RL-based computational framework for automatic mesh generation. The environment represents the meshing boundary. The agent is the mesh generator that could implement various RL techniques.

instead of the whole boundary. Comparing with most existing methods, this method is computationally efficient and does not need any extra operations after meshing a geometry. The mesh quality requirement is embedded in the reward function, which measures how well each element generated at each step is contributed to the final mesh. Additionally, the reward function can be customized or controlled by the down-streaming applications to meet their specific quality requirements, such as sparse or dense density. By controlling the reward function, it is a straightforward pathway to acquire needed mesh and avoids many time-consuming and computationally complex post-processing operations (e.g., clean-up and remeshing).

## 5.2.2 Reinforcement learning

In many DRL applications, the policies have a fixed size of the input state and are applicable to environments that are similar to the training environment. For instance, [Wei et al. \(2020\)](#) trained different models for elevator settings.

RL is a learning paradigm that has been used to address a sequential decision making problem, mathematically known as an MDP problem. The technique is to enable an agent to learn from the interactions with its environment by trial-and-error via reward feedback from its actions and experiences ([Kaelbling et al., 1996](#); [Sutton & Barto, 2018](#)). Eventually, a policy will be learned by the agent for guiding appropriately selecting actions under each environmental situation over time



to maximize the accumulated reward. According to different learning objectives, the policy could be learned in a direct (policy-based) or indirect (value-function-based) fashion.

Value-function-based methods estimate the expected reward for the agent to start from a state or to perform a given action in a given state. The optimal policy can be implicitly derived from the optimal value function. Deep Q-Network (DQN) is a recent breakthrough in estimating value function via a deep neural network (Mnih et al., 2013), which stabilizes the training of action value function using experience replay and target network and generalizes a framework for end-to-end RL tasks using the same algorithm. Many successors are continuously improving DQN from various aspects, e.g., asynchronous advantage actor-critic (A3C) (Mnih et al., 2016), dueling network to better estimate action value function (Z. Wang et al., 2016), and combating sparse reward issues by Hindsight Experience Replay (HER) (Andrychowicz et al., 2017). For the mesh generation problem, the action is continuous and requires choosing appropriate points to form an element from an area. Correspondingly, the meshing environment will be altered and evolved into various shapes after each action. Hence, to discretize both state and action spaces may distort the feedback regarding the impact of the agent's actions on the environment and adversely hinders exploring feasible action space, causing a sub-optimal policy.

Policy-based methods directly estimate a policy to guide the agent to choose actions under various environmental states. The policy is usually modelled by a parameterized function. The modern policy-based methods have several divisions, including deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) using policy gradient, A3C (Mnih et al., 2016) combining actor and critic together, Proximal Policy Optimization (PPO) (Schulman et al., 2017) via trust region, and SAC (Haarnoja, Zhou, Abbeel, & Levine, 2018; Haarnoja, Zhou, Hartikainen, et al., 2018) with off-policy learning.

RL methods have been gradually applied to many applications in various areas and achieved promising performance (Li, 2018). Since the mesh generation problem can be cast as an MDP problem, it is intriguing to solve the problem by RL. Formally, the MDP has a set of states  $S$ , a set of actions  $A(s)$  for a given state  $s$ , and a reward function  $R$ . Their formulation is altered and improved from the previous method to align the new objective, completing the mesh rather than providing training samples. The detailed definition will be explained in the next section.

Since the stability, robustness, and efficiency of meshing algorithms are critical for down-streaming applications, a customized RL algorithm, SAC is used in this article. SAC is one of the state-of-the-art policy-based algorithms and has advantages in better sample efficiency and stable convergence. The hypothesis behind RL is that all the goals can be represented by the maximization of the expected cumulative reward. In this article, a well designed reward function will guide the policy learning to accomplish the meshing task. The RL-based computational framework for mesh generation offers a direct solution to acquire the mesh satisfying the quality requirement; it provides a pathway to customize the ideal mesh for down-streaming applications by adjusting the reward function; and it is a fully automatic method for various complex geometry domains without extra clean-up treatments.

### 5.3 RL based mesh generation

In this section, a self-learning computational framework for automatic mesh generation via RL is proposed. Quadrilateral element generation in single-connected 2D domains is used as an example to demonstrate this method. The action formulation and state representation of the problem is described first. Then, the reward function is defined. Finally, the detailed implementation of SAC is explained.

#### 5.3.1 Action formulation

In this formal framework, the agent will take actions to generate an element at each time step according to the current boundary. The search for the best action includes 1) how to choose the reference vertex? and 2) how to decide the other three vertices to construct an element?

For the first question, the general principle to select a vertex is the narrow region first, which is to avoid creating hard boundary situations for the subsequent actions. The vertex will be the reference vertex  $V_i^*$  when it forms the least angle with its surrounding vertices, as denoted by

$$V_i^* = \arg \min_{V_i} \frac{1}{n_{rv}} \sum_{j=1}^{n_{rv}} \angle V_{l,j} V_i V_{r,j} \quad (22)$$

where  $V_{l,j}$  and  $V_{r,j}$  denote the  $j$ -th vertices at the left and right side of the reference vertex  $V_i$  along the boundary, respectively;  $n_{rv}$  represents how many surrounding vertices should be included, which  $n_{rv} = 2$  is used in this article;  $V_i$  is the  $i$ -th vertex on the boundary. Since this strategy is consistent across all the actions, the reference vertex selection is automatically implemented by the environment rather than by the agent.

For the second question, once the reference vertex is determined, there are three kinds of basic situations to form an element: adding zero, one, or two vertices (Zeng & Cheng, 1993). Therefore, as shown in Figure 5.4, the action is formally defined as  $[type, V_1, V_2]$ , where  $type \in \{0, 1, 2\}$ , which are corresponding to the three situations respectively;  $V_1$  and  $V_2$  are the coordinates of the newly added vertices. The coordinates space for the vertices are constrained to a fan-shaped area (in light blue) with radius  $r$ , which is calculated as follows:

$$r = \alpha * L, \quad (23)$$

$$L = \frac{1}{2n} \sum_{i=0}^n |V_{l,i}V_{l,i+1}| + |V_{r,i}V_{r,i+1}|, 0 < n < N/2.$$

where  $V_{l,i}$  and  $V_{r,i}$  denote the  $i$ -th vertex at the left and right side of the reference vertex along the boundary;  $V_{l,0} = V_{r,0}$  is the reference vertex;  $|V_*V_*|$  is the Euclidean distance between two vertices. We set  $\alpha = 1.5$  and  $n = 2$  in all our experiments. Usually, type 2 is only needed on special occasions, such as circular domains, and is operated at a constant level. Hence, to reduce the learning complexity, the action  $[type, V_1]$  is finally used in the experiments.

### 5.3.2 State representation

The state is the observation of the agent to the environment. The environment in mesh generation is the meshing boundary (see Figure 5.2). The full state of the environment at time  $t$  would consist of all the vertices of the boundary. However, not all the vertices are meaningful and necessary to represent the environment. To adapt to all kinds of geometries, the inclusion of all vertices is also not a good option. Therefore, a partial observation of the boundary environment is considered, which includes the following components: a reference vertex determining where the agent should generate

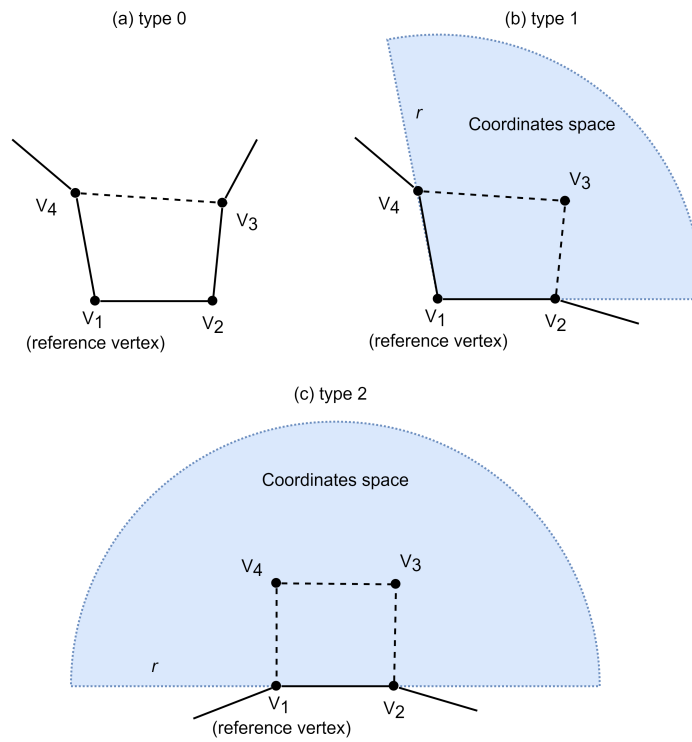


Figure 5.4: Action spaces for each type. Subfigures (a)-(c) correspond to three types of actions, respectively. The blue area is the area with the reference vertex  $V_1$  as the origin and a radius  $r$  to choose the candidate vertices, such as,  $V_3$  in type 1 and  $V_3$  and  $V_4$  in type 2.

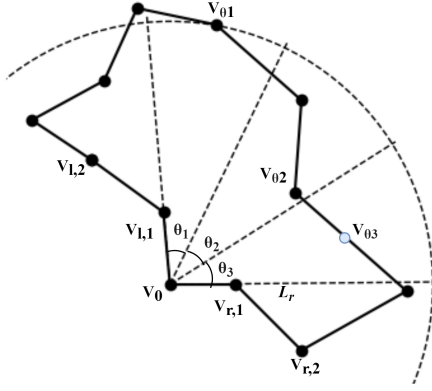


Figure 5.5: Partial observation of the meshing boundary. For example, the partial boundary, where  $L_r = 4, n = 2, g = 3$ , is represented as the state. First, two vertices on the left and right sides of the reference vertex  $V_0$  are selected, respectively. Second, the angle  $\angle V_{l,1}V_0V_{r,1}$  is evenly split into three angles  $\theta_1, \theta_2$ , and  $\theta_3$ ; three fan-shaped areas are hence formed with these angles and a radius  $L_r$ . Then, the closest vertex in each area is selected.

an element around, and its surrounding vertices providing a local environmental situation.

The state is denoted as  $s_t$ , and composed of five components (1) a reference vertex,  $V_0$ , which is used as the relative origin to generate the new element with a action  $a_t$ , and is calculated by iterating all the vertices on the boundary, computing the angle of each vertex formed by its left and right connected vertices, and selecting the vertex having the least angle; (2)  $n$  neighboring vertices on the right side of the reference vertex; (3)  $n$  neighboring vertices on the left side; (4)  $g$  neighboring points  $V_{\theta_1}, \dots, V_{\theta_g}$  in the fan-shaped area  $\theta_1, \dots, \theta_g$  with radius  $L_r$ , as shown in Figure 6.2 ( $g = 3$ ), where  $\theta_1 = \theta_2 = \theta_3$ ;  $L_r = \beta * L$ . If there are no vertices in the fan-shaped area (e.g.,  $\theta_3$ ), the furthest bisector vertex in the area or the intersection vertex between the bisector and the boundary edge is selected, such as  $V_{\theta_3}$  in Figure 6.2; (5)  $\rho_t$ , the area ratio between the updated domain and the original domain.

This partial boundary is arranged as follows:

$$S_t = \{V_{l,n}, \dots, V_{l,1}, V_{r,1}, \dots, V_{r,n}, V_{\theta_1}, \dots, V_{\theta_g}, \rho_t\}. \quad (24)$$

All the vertices are represented by a polar coordinates system with  $V_0$  as the origin and  $\overrightarrow{V_0V_{r,1}}$  as the reference direction. Since the x and y coordinates of the vertex  $V_0$  are both 0, it can be removed from the state representation. This representation only keeps the relative information among all the

vertices.

### 5.3.3 Reward function

The reward function represents the objective of mesh generation, which is to achieve overall high mesh quality and completeness with finite elements. This is a stepwise reward signal that measures the performance of each action. There are three main cases in total: 1) if the action forms an invalid element or has intersections with the boundary edges, the reward is set to -0.1; 2) if the action forms the last element, the reward is set to 10; 3) if it generates a valid element, the reward is the joint measurement of element quality, the quality of the remaining boundary, and the density factor. Finally, the reward function is formally defined as follows:

$$r_t(s_t, a_t) = \begin{cases} -0.1, & \text{invalid element;} \\ 10, & \text{the element is the last element;} \\ m_t, & \text{otherwise.} \end{cases} \quad (25)$$

The measurement  $m_t$  is calculated by the following equation:

$$m_t = \eta_t^e + \eta_t^b + \mu_t. \quad (26)$$

The element quality  $\eta_t^e$  is measured by its edges and internal angles situations, and is calculated as follows,

$$\begin{aligned} \eta_t^e &= \sqrt{q^{edge} q^{angle}}, \\ q^{edge} &= \frac{\sqrt{2} \min_{j \in \{0,1,2,3\}} \{l_j\}}{D_{max}}, \\ q^{angle} &= \frac{\min_{j \in \{0,1,2,3\}} \{angle_j\}}{\max_{j \in \{0,1,2,3\}} \{angle_j\}}, \end{aligned} \quad (27)$$

where  $q^{edge}$  refers to the quality of edges of this element;  $l_j$  is the length of the  $j$ th edge of the element;  $D_{max}$  is the length of the longest diagonal of the  $t$ th element;  $q^{angle}$  refers to the quality of the angles of the element; and  $angle_j$  is the degree of  $j$ th inner angle of the element. The quality  $\eta_t^e$

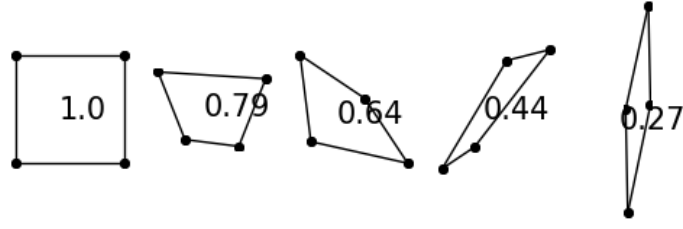


Figure 5.6: Element quality with different shapes. The quality value ranges from 0 to 1. The element with the best quality 1 is a square.

will range from 0 to 1, which is better if greater. Examples of various element qualities are shown in Figure 5.6.

The quality of the remaining boundary is measured by both the quality of the angles formed between the newly generated elements and the boundary, and the shortest distance of the generated vertex to its surrounding edges, and denoted as follows,

$$\eta_t^b = \sqrt{\frac{\min_{k \in \{0,1\}} \{ \min(\theta_k, M_{angle}) \}}{M_{angle}} q^{dist} - 1}, \quad (28)$$

$$q^{dist} = \begin{cases} \frac{d_{min}}{(d_1+d_2)/2}, & \text{if } d_{min} < (d_1 + d_2)/2; \\ 1, & \text{otherwise.} \end{cases}$$

where  $\theta_k$  refers to the degree of the  $k$ th generated angle;  $d_{min}$  is the distance of  $V_3$  to its closet edges. The details are shown in Figure 5.7. The quality  $\eta_t^b$  ranges from -1 to 0, which the larger value is the better quality. It serves as a penalty term to decrease the reward if the quality of the remaining boundary is getting worse. We set  $M_{angle} = 60$ . When the formed new angles are less than  $M_{angle}$ , the quality will decrease, which prevents the generation of sharp angles that are harmful to the overall mesh quality and may even fail the meshing process.

These two qualities together represent the trade-off between the generated element and the remaining boundary, which is critical to guarantee the overall mesh quality and completeness of the final mesh. The last term is a density factor  $\mu$ , which controls the mesh density and assure the meshing process will be completed in finite steps, and is calculated as follows,

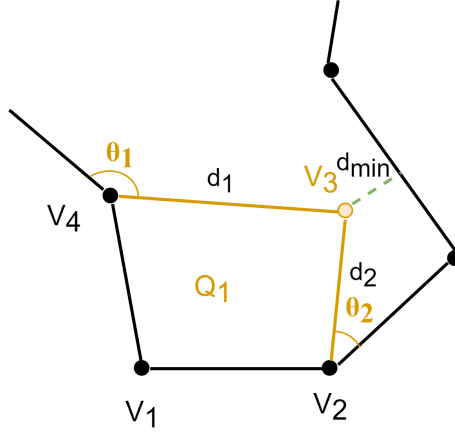


Figure 5.7: The quality of the updated boundary.  $Q_1$  is the newly generated element. Once it is removed, it forms two angles  $\theta_1$  and  $\theta_2$  with existing boundary segments. The quality is jointly measured by these two angles, and the closest Euclidean distance  $d_{min}$  of the newly added vertex  $V_3$  is to the existing segments.  $d_1$  and  $d_2$  are the Euclidean lengths of segments  $V_3V_4$  and  $V_2V_3$ .

$$\mu_t = \begin{cases} -1, & \text{if } \mathcal{A}_t < \mathcal{A}_{min}; \\ \frac{\mathcal{A}_t - \mathcal{A}_{min}}{\mathcal{A}_{max} - \mathcal{A}_{min}}, & \text{if } \mathcal{A}_{min} \leq \mathcal{A}_t < \mathcal{A}_{max} - 1; \\ 0, & \text{otherwise.} \end{cases} \quad (29)$$

where  $\mathcal{A}_t$  is the area of the element at time  $t$ ;  $\mathcal{A}_{min}$  is the estimated minimum area of the element that is tolerable to the meshing domain, and calculated by  $\mathcal{A}_{min} = ve_{min}^2$ ;  $\mathcal{A}_{max}$  is the estimated maximum area of the element, and calculated by  $\mathcal{A}_{max} = v \left( \frac{e_{max} - e_{min}}{\kappa} + e_{min} \right)^2$ ;  $e_{max}$  and  $e_{min}$  is the length of the longest and shortest edges in the boundary respectively;  $\kappa$  controls when to start the penalty and  $\kappa = 4$  in our experiment;  $v$  is a weight and ranges from  $[0, +\infty)$ , which the smaller value means the denser density, and vice versa. We set  $v = 1$  in our experiments for the medium density.

### 5.3.4 Meshing scheme via SAC

The computational framework of mesh generation is based on the SAC approach. SAC is one of the state-of-the-art RL algorithms for continuous action control problems (Haarnoja, Zhou, Abbeel, & Levine, 2018; Haarnoja, Zhou, Hartikainen, et al., 2018). To overcome the sample complexity and hyperparameter sensitivity, it adds an entropy item in addition to reward in the objective function,



and maximizes the reward return while maximizing the randomness of the policy.

The objective function of the policy is correspondingly denoted as

$$J(\pi) = \sum_{t=1}^T \mathbb{E}_{(s_t, a_t) \sim \rho_{\pi_\theta}} [r(s_t, a_t) + \alpha \mathcal{H}(\pi_\theta(\cdot | s_t))], \quad (30)$$

where  $\mathcal{H}(\cdot)$  is the entropy measure (Ziebart, 2010);  $\rho_{\pi_\theta}$  is the state-action marginal distribution of policy  $\pi$  parameterized by  $\theta$ ;  $\alpha$  indicates the significance of the entropy item, known as temperature parameter. With considering the entropy maximization, it allows the learned policy acts as randomly as possible while guaranteeing task completion, which gains the trade-off of the exploration-exploitation and thus accelerates the learning. This randomness is especially important for partially observable environment.

For learning an optimal maximum entropy policy, SAC drives from the policy iteration method in the maximum entropy framework. During the policy evaluation step, the soft Q-value can be computed iteratively and defined as

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho_\pi(s)} [V(s_{t+1})], \quad (31)$$

where

$$V(s_t) = \mathbb{E}_{a_t \sim \pi} [Q(s_t, a_t) - \alpha \log \pi(a_t | s_t)], \quad (32)$$

where  $\rho_\pi(s)$  and  $\rho_\pi(s, a)$  are the state and the state-action marginals of the trajectory distribution induced by a policy  $\pi(a_t | s_t)$ . The soft Q-function  $Q_\theta(s_t, a_t)$  is parameterized by a neural network with parameters  $\theta$ . The soft state value function  $V(s_t)$  is implicitly parameterized by the soft Q-function parameters. To update the soft Q-function, its gradient is estimated by the formula,

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(s_t, a_t) (Q_\theta(s_t, a_t) - (r(s_t, a_t) + \gamma (Q_{\bar{\theta}}(s_{t+1}, a_{t+1}) - \alpha \log(\pi_\phi(a_{t+1} | s_{t+1}))))), \quad (33)$$

where  $\pi_\phi$  is the current policy parameterized by a neural network with parameter  $\phi$ ;  $\bar{\theta}$  is obtained as an exponential moving average of the soft Q-function network weights.

In the soft policy improvement stage, the policy parameter can be updated by minimizing the expected KL-divergence,

$$J(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}}[\mathbb{E}_{a_t \sim \pi_\phi}[\alpha \log(\pi_\phi(a_t|s_t)) - Q_\theta(s_t, a_t)]]. \quad (34)$$

Its gradient can be approximated by

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \alpha \log(\pi_\phi(a_t|s_t)) + (\nabla_{a_t} \alpha \log(\pi_\phi(a_t|s_t)) - \nabla_{a_t} Q(s_t, a_t)) \nabla_\phi f_\phi(\epsilon_t; s_t), \quad (35)$$

where  $\epsilon_t$  is an input noise vector and can be sampled from some fixed distribution;  $f_\phi(\epsilon_t; s_t)$  is a reparameterized policy using a neural network transformation.

Finally, the temperature  $\alpha$  is updated by minimizing the objective

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi}[-\alpha \log \pi_\phi(a_t|s) - \alpha \bar{\mathcal{H}}]. \quad (36)$$

The details of the algorithm are shown as Algorithm 2.

---

**Algorithm 2** SAC for mesh generation

---

- 1: initialize network parameters  $\theta_1, \theta_2$  and  $\phi$
  - 2:  $\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$
  - 3: **for** each episode **do**
  - 4:   **for** each time step **do**
  - 5:     Select action  $a_t \sim \pi_\phi(\cdot|s_t)$
  - 6:     Observe reward  $r$  and new state  $s'$
  - 7:     Store  $(s_t, a_t, r_t, s_{t+1})$  in replay buffer  $\mathcal{D}$
  - 8:   **end for**
  - 9:   **for** each gradient step **do**
  - 10:     Sample  $m$  batches from replay buffer
  - 11:     Update soft Q-function  $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$  for  $i \in \{1, 2\}$
  - 12:     Update policy network  $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
  - 13:     Update target network  $\bar{\theta}_i \leftarrow \tau \theta_i + (1 - \tau) \bar{\theta}_i$  for  $i \in \{1, 2\}$
  - 14:     Adjust temperature  $\alpha \leftarrow \alpha - \lambda_{\nabla_\alpha} J(\alpha)$
  - 15:   **end for**
  - 16: **end for**
-

## 5.4 Experimental results

In this section, the effectiveness of the proposed method, FreeMesh-RL, will be demonstrated by a few types of experiments: scalability verification and meshing performance comparison. Since the scalability challenge is common for NN-based meshing methods and RL-based applications, we will evaluate the FreeMesh-RL method over different complex domains or the same domain with different scales on size. The second type of experiment is to compare the meshing performance of FreeMesh-RL with two start-of-the-art meshing approaches over three predefined geometry domains. To obtain optimal meshing performance, we also examine the impact of observation and reward function on the learning of the policy.

To the best of our knowledge, there is no baseline environment of mesh generation in the RL field. We build a simulation environment with Python and PyTorch, which will be cleaned up and opened for the community to test various RL algorithms in the future.

### 5.4.1 Implementation details

We have examined the learning performance of different neural network architectures. The comparison results are shown in Figure 5.8 (b). The network architecture with three hidden layers, [128, 128, 128] (i.e., S2), achieves better learning performance while has fewer parameters comparing with others, which is selected in this work to approximate the soft Q-function, policy, and target networks. The hyperparameters used for SAC are listed in the Table 5.1.

The training domain boundary could be any size and shape but has to meet a few criteria, including having a sharp angle and bottleneck region, to ensure the richness of the samples. The domain chose for training is shown in Figure 5.8 (a). The trained model will be used to mesh other unseen domain boundaries in the later experiments.

All the neural network models are trained on a computer with an i7-8700 CPU and an Nvidia GTX 1080 Ti GPU with 32 GB of RAM. A total of 1.2 million time steps are used for training the policy.

Table 5.1: Training hyperparameters

parameter	Description	Value
$N_{\mathcal{D}}$	Experience pool size	1e6
$m$	Minibatch size	256
$\gamma$	Discount factor	0.99
$\lambda_Q, \lambda_{\pi}$	Learning rate	3e-4
$\tau$	Soft update factor	5e-3

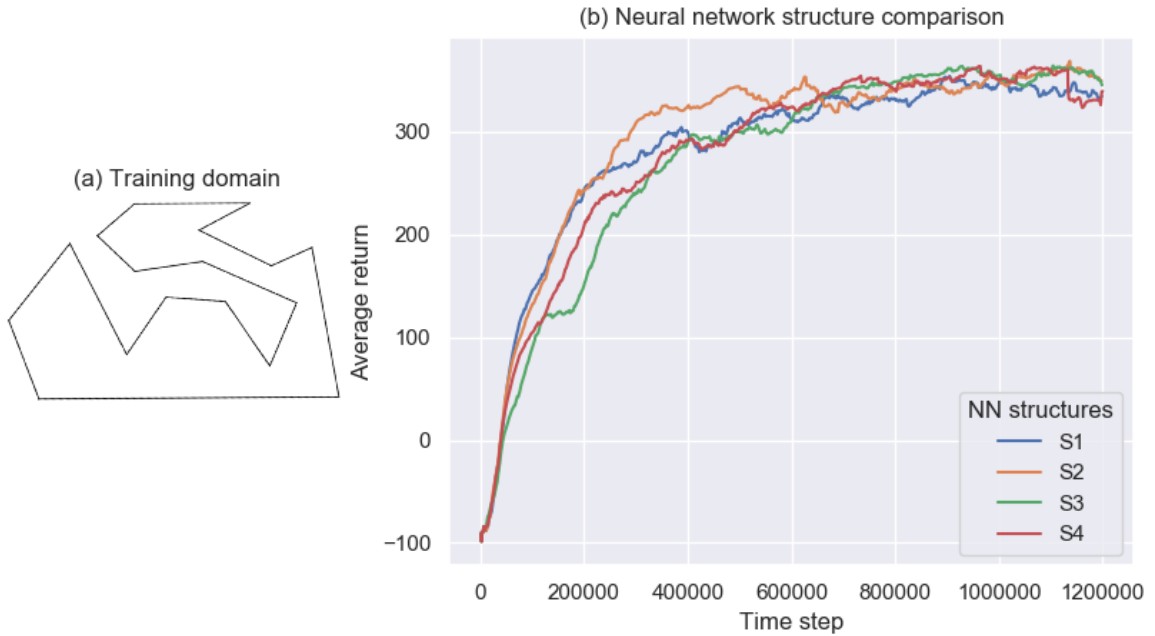


Figure 5.8: NN structures comparison. S1-4 represent four different neural network structures in Hidden layers, including [256, 256], [128, 128, 128], [64, 128, 64, 32, 16], and [64, 128, 256, 128, 64], respectively.

### Agent’s view of environmental state

The state is the agent’s observation from the environment, which will significantly influence the agent’s decision making. Since the partial observation is adopted in the proposed method, We need to decide how much is appropriate and effective to be observed by the agent to learn the meshing policy. The range of the observation is controlled by three factors,  $L_r$ ,  $n$  and  $g$ , in Equation 24. The factor  $L_r$  controls how far the agent can observe from its position while the other two factors determine how many vertices the agent perceives around the reference vertex. The leaning performance is compared with three kinds of settings,  $O1$  ( $L_r = 4, n = 2, g = 3$ ),  $O2$  ( $L_r =$

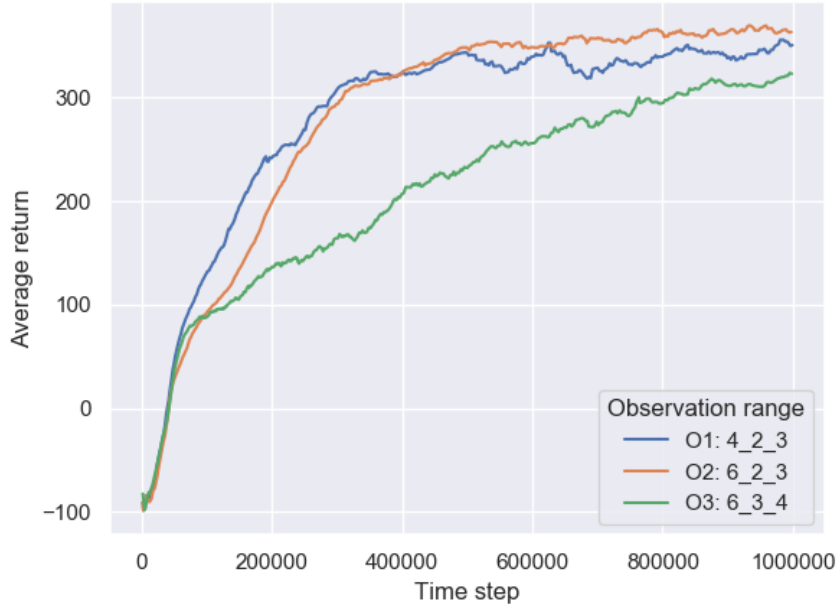


Figure 5.9: RL state comparison. The observation range is formed by  $L_r-n-g$ , which represents the radius of the fan shape in the state, the number of neighbouring vertices on the left and right side of the reference vertex, and the number of vertices in the fan-shaped area, respectively. This range determines how far and how much information the agent will be observed.

$6, n = 2, g = 3$ ), and  $O3$  ( $L_r = 6, n = 3, g = 4$ ). The comparison results are shown in Figure 5.9. By comparing  $O1$  and  $O2$ , it can be found that further observation contributes to more return. This is because the agent could adjust the position of the candidate vertex in advance to avoid a conflict with the remaining boundary. Meanwhile, since the chance of noticing vertices in the fan-shaped area will be increased, the more complex information will be included in the state, which the learning speed may become slower in the earlier stage. A similar phenomenon can be observed when comparing  $O2$  and  $O3$ . The more information the agent observes the more time to build the correlation is needed. In this paper, the observation  $O2$  is hence used as the state representation across all the experiments.

### Reward function

The reward function represents the objective of mesh generation, which mainly consists of overall high mesh quality and completeness with finite elements. There are three terms in the reward

function, element quality  $\eta_t^e$ , the quality of remaining boundary  $\eta_t^b$ , and density factor  $\mu_t$ , which guides the policy learning to fulfill the objective. The first two terms guarantee the mesh quality and the easiness of continuing meshing. The last term controls the meshing density by the factor  $v$  that can adjust the minimum tolerant element area, and assure the mesh termination within finite steps. Three kinds of density are compared, including sparse ( $v = 1.5$ ), medium ( $v = 1$ ), and dense ( $v = 0.5$ ) settings. The comparison results are shown in Figure 5.10 (a)-(c). We also examine the number of elements for each density after running 10 episodes, as shown in Figure 5.10 (d). The difference in the average number of elements between sparse and medium is about 20 elements while the difference between medium and dense density is around 50 in this testing domain boundary. The medium density is used across all the remaining experiments.

## 5.4.2 Evaluation

The effectiveness of the proposed FreeMesh-RL will be measured in two aspects: scalability and meshing performance.

### Scalability verification

To validate the scalability of the learned meshing policy, we have constructed three geometry domains with the same shape but different vertex densities (i.e., 6.8 : 9.9 : 20.1, as shown in Table 5.3) on the boundaries. Three domains are meshed by the same trained RL model, and the results are shown in Table 5.2. It can be seen that all the domains have been successfully meshed; the elements on the boundaries are denser than in the interior area, which is beneficial for reducing computational burden; the transitions between dense and coarse meshes are smooth. We have also examined the meshing speed over three domains, which achieves 237 elements per second on average. Consequently, the results show that the self-learned mesh policy achieves good scalability to different scales of the meshing problem and is not constrained to the scale of the training domain. Many existing NN-based meshing methods, must train different models for adapting various sizes of domains, which is not practical for large-scale problems. The reason for this adaptability is that the training domain will be continuously updated and evolved into various shapes that capture most boundary patterns (Pan, Huang, Wang, et al., 2021).

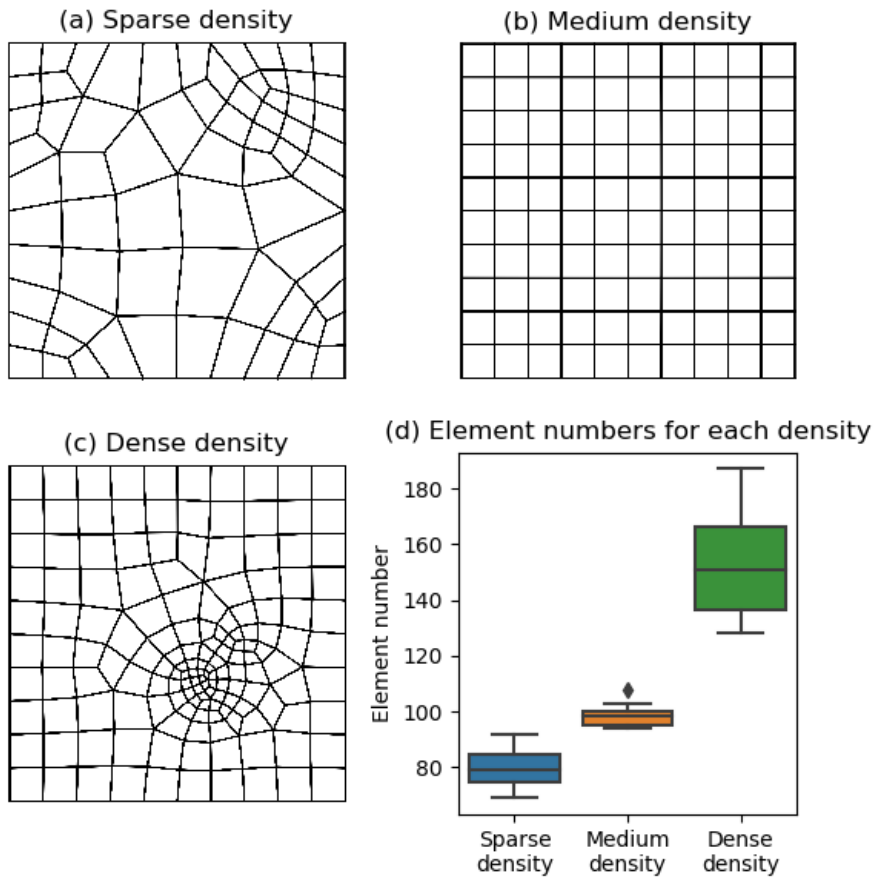
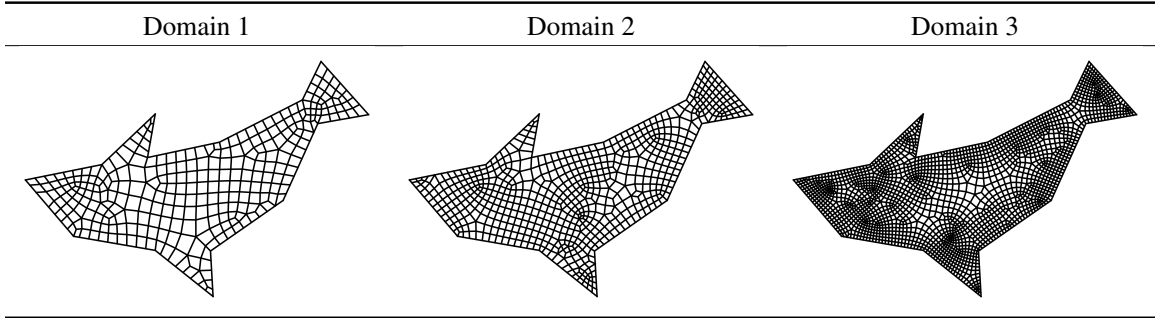


Figure 5.10: Different mesh densities controlled by reward function. Three kinds of densities (a)-(c) are controlled by parameters in the density term,  $v = 1.5$ ,  $v = 1$ , and  $v = 0.5$ , respectively. Subfigure (d) is the comparison results of the number of generated elements by three kinds of densities over 10 episodes.

Table 5.2: Meshing same shape with different density by FreeMesh-RL.



The shapes of Domain 1-3 remain the same. The difference is the vertex density on their boundaries. All the meshes are generated from one trained model by FreeMesh-RL.

Table 5.3: Scalability evaluation for three domains

	Domain 1	Domain 2	Domain 3
#vertices	102	150	304
Perimeter	15.1	15.1	15.1
#vertices per unit length	6.8	9.9	20.1
#elements	289	665	2157
Execution time (s)	0.9	2.6	15.3

#vertices - the number of vertices on the boundary  
#elements - the number of generated elements

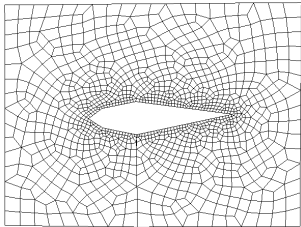
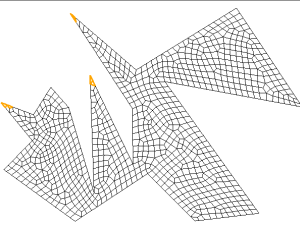
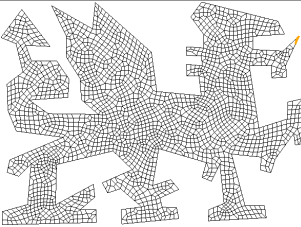
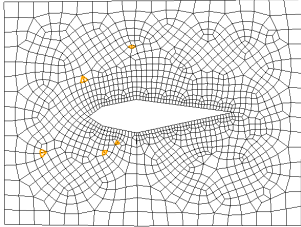
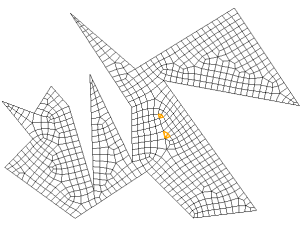
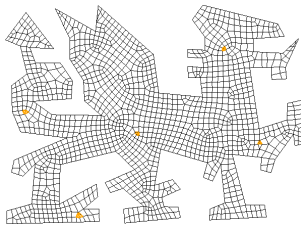
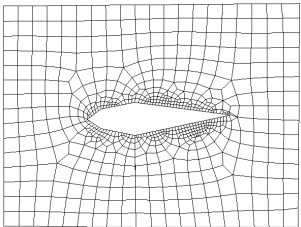
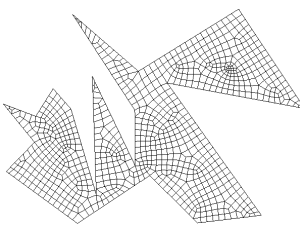
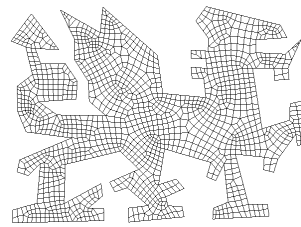
### Comparison to conventional methods

To evaluate the meshing performance, we compare the quality of meshes by FreeMesh-RL against two widely adopted meshing approaches over three predefined 2D domain boundaries (i.e., domains D4-6). These domains possess different features, including sharp angles, bottleneck regions, unevenly distributed edges, and holes, to increase the testing diversity. The two conventional meshing approaches are Blossom-Quad (Remacle et al., 2012) and Pave (Blacker & Stephenson, 1991; White & Kinney, 1997). Blossom-Quad is an indirect method, which generates quadrilateral elements by finding the perfect matching of a pair of triangles generated in advance. The method is implemented by an open source generator Gmsh (Geuzaine & Remacle, 2009). Pave is another state-of-the-art meshing method for directly generating quadrilateral elements, which is implemented by the CUBIT software (Blacker et al., 2016). Both of them are prevalent in the industry and haven't been replaced by any machine learning-based methods yet, which are appropriate to be used as comparison methods.



The meshing results are shown in Table 6.1. Although all the methods can complete the meshes for the three domains, there are some subtle differences. Only FreeMesh-RL generates fully quadrilateral mesh. The other two methods have difficulties in discretizing the domains into full quadrilaterals and generate triangles in domains (marked in yellow colour in domains). Specifically, Blossom-Quad has a problem in handling domains with sharp angles on the boundary while Pave has the issue in the interior of the domain. Extra operations (e.g., clean-ups) are usually required for these methods to eliminate triangles or bad quality elements, which slows down the meshing speed and decreases the automation to some extent. Contrarily, FreeMesh-RL doesn't need those operations and avoids explicit treatments. Another advantage of FreeMesh-RL is that the generated mesh can easily grade from very small to large elements over a short distance, as shown in Table 6.1 Domain 1, which is beneficial for reducing computational burden during simulations (Shewchuk, 2012).

Table 5.4: Meshing results comparison

Algorithms	Domain 4	Domain 5	Domain 6
Blossom-Quad			
Pave			
FreeMesh-RL			

The elements in yellow represent existing triangles in the meshes.

To quantitatively analyze the meshing results of three methods, we have selected eight common

quality metrics to measure the meshing performance, including singularity (i.e., the number of irregular nodes whose number of incident edges in the interior of a mesh are not equal to four), element quality ( $\eta^e$  in Equation 27),  $|MinAngle - 90|$ ,  $|MaxAngle - 90|$ , scaled Jacobian, stretch, taper, and the number of triangles (#triangles) (Knupp, Ernst, Thompson, Stimpson, & Pebay, 2006; Pan, Huang, Wang, et al., 2021). The measurement results are averaged over three domains and are shown in Table 6.2 and Figure 5.11. FreeMesh-RL has achieved the best performance in the indices of singularity, taper, scaled Jacobian, and triangle. The smaller singularity means better regularity, which can provide better accurate results for numerical simulations. The smaller taper and bigger Scaled Jacobian represent the mesh element has a more regular shape like a square. If there exist triangles, it indicates that the method is not full quadrilaterals and involves extra treatments to deal with them. The Pave method achieves the best performance in the other indices (i.e., element quality, min and max angles, and stretch). Blossom-Quad has the worse performance and is only slightly better than Pave as having fewer triangles. It is common for indirect methods (i.e., Blossom-Quad) to have a suboptimal performance because they heavily rely on triangulation in advance. Remacle et al. (2013) tried to optimize the triangulation. But the improvement is limited (Pan, Huang, Wang, et al., 2021). The computational complexities for the three methods are also compared, which is all  $O(n^2)$ .

Table 5.5: Averaged mesh quality metrics over three domains

Metrics	Blossom-Quad	Pave	FreeMesh-RL
Singularity (L)	388 ± 209.50	146.70 ± 51.50	<b>132 ± 50</b>
Element quality (H)	0.72 ± 0.12	<b>0.79 ± 0.12</b>	0.79 ± 0.13
$ MinAngle - 90 $ (L)	6.55 ± 6.91	<b>3.69 ± 4.60</b>	4.02 ± 5.10
$ MaxAngle - 90 $ (L)	22.16 ± 11.14	<b>15.69 ± 14.71</b>	15.73 ± 12.48
Scaled jacobian (H)	0.91 ± 0.08	0.94 ± 0.13	<b>0.94 ± 0.10</b>
Stretch (H)	0.79 ± 0.08	<b>0.84 ± 0.10</b>	0.83 ± 0.11
Taper (L)	0.15 ± 0.11	0.12 ± 0.14	<b>0.11 ± 0.11</b>
Triangle (L)	2.70 ± 2.50	8 ± 2.80	<b>0 ± 0</b>

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.

In general, FreeMesh-RL shows optimal performance on the four metrics against the other two methods, has competitive performance with Pave on the remaining four metrics, and does not require extra operations to handle triangles or bad-quality elements. This demonstrates the learned meshing policy by SAC is effective in generating high quality meshes. Meanwhile, all the meshes

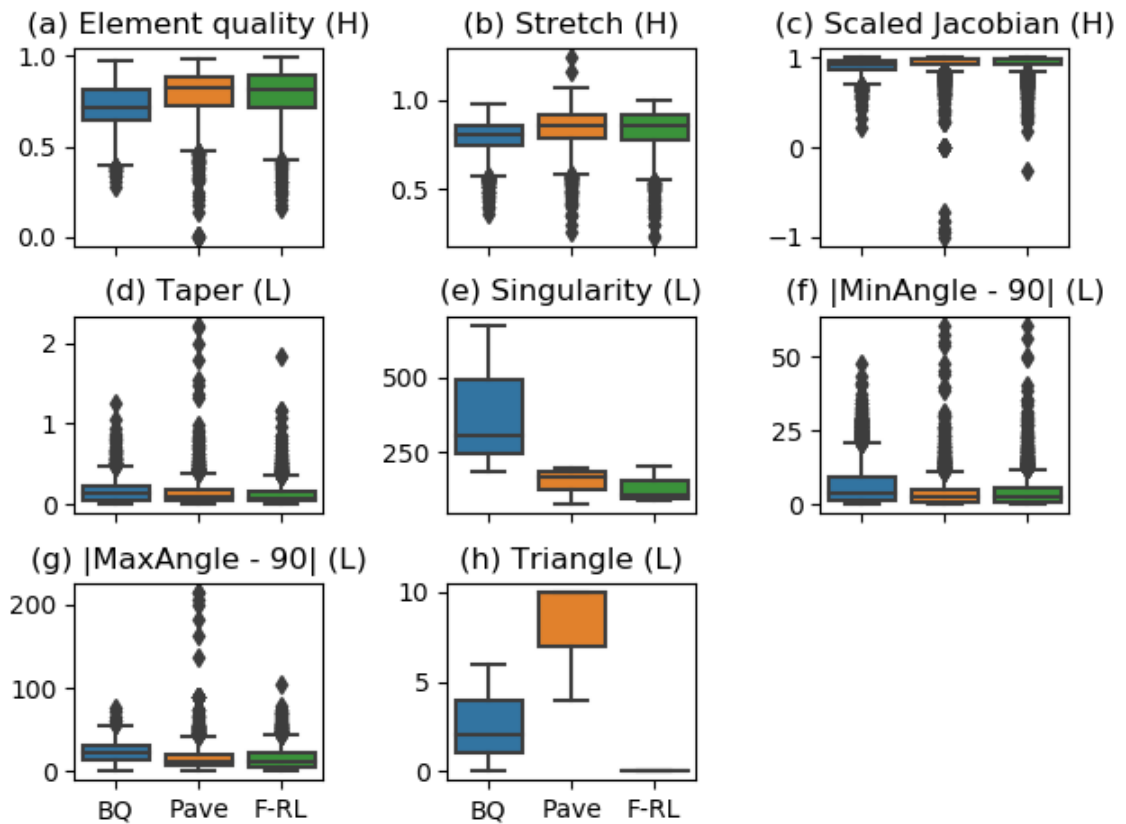


Figure 5.11: Meshing performance comparison results over 8 kinds of quality indices. BQ represents the Blossom-Quad method; F-RL represents the FreeMesh-RL method. L, H indicate if the lower value or higher value is preferred, respectively.

are generated by only one trained model without any additional training. The model is also adaptable to various geometry domains, which demonstrates the good generalizability and effectiveness of this computational framework.

## 5.5 Discussion

The proposed computational framework, FreeMesh-RL, has achieved several important improvements to both mesh generation and RL.

First, comparing with conventional or NN-based mesh generation methods, FreeMesh-RL does not require heuristic knowledge or any labelled data, by using RL in particular SAC algorithm. It relieves people from in-efficient searching for domain specific knowledge and time-consuming trial-and-error and reduces the algorithm complexity by eliminating extra clean-up operations. Those obstacles greatly hinder the advancement of meshing methods towards high processing speed and mesh quality in complex geometry domains ([Papagiannopoulos et al., 2021](#); [Rushdi et al., 2017](#); [Slotnick et al., 2014](#)).

Second, the meshing policy in FreeMesh-RL can be self-learned by a well-defined reward function. The reward function determines various characteristics of the mesh, including mesh quality and density. Many downstream applications in FEA/CFD, usually, require different mesh properties. In FreeMesh-RL, some of those requirements can be fulfilled by generating a customized mesh generator directly via adjusting the reward function rather than using extra expensive and time-consuming treatments, such as mesh adaption or remeshing. For instance, as shown in [Figure 5.10](#), the mesh density is controlled by a density factor in the reward function. Usually, more elements can provide accurate simulation results for applications but cost more computationally. In the graphic rendering area, the sparse mesh is preferred because of this reason. To adjust the mesh density is barely not applicable in any conventional mesh methods. This computational platform provides the easiest and cheapest fashion to design customized mesh generators for researchers.

Many RL topics are still not well understood ([Duan, Chen, Houthoof, Schulman, & Abbeel, 2016](#)). For instance, the designs of state representation and reward function are critical topics for the performance of RL algorithms and for understanding the mechanism of RL. The state representation will

influence the generalizability, effectiveness, and learning speed (Lesort, Díaz-Rodríguez, Goudou, & Filliat, 2018; Niv, 2019). To obtain the generalizability of the policy, a partial observation of the environment is hence utilized in FreeMesh-RL to eliminate the agent’s dependency on the global environment, which is the critical technique to achieve adaptability to other geometry domain environments. Many RL applications have to train different models for different environmental settings, which is not practical and expensive in real-life environments. As shown in the Figure 5.9, how far the observation and how much information (i.e., vertices on the domain boundary) should be included are controlled by three factors, respectively. The learning efficiency and overall return vary correspondingly. The low dimension state representation achieves better learning efficiency and provides easier interpretation and utilization for humans, thus improving the policy’s performance and generalizability. Some advanced topics for state representation may be researched in-depth by using this platform, such as partial observability, informative representations, state abstraction, and state representation learning (Dulac-Arnold et al., 2019; Lesort et al., 2018).

Finally, FreeMesh-RL has the potential to serve as an RL research platform, for example, to be used to study the reward function and examine the performance of RL algorithms. Many RL benchmark problems have very simple reward functions and environments, such as Atari games (Mnih et al., 2013), while some other famous problems are too complex, such as Dota2 (Berner et al., 2019) and StarCraft (Vinyals et al., 2017). The simple problems cannot fully reflect the performance of RL algorithms whereas the complex problems cannot be easily used to study topics, such as hyper-parameters tuning, generalizability, sample efficiency, and reward specification (Li, 2018). Mesh generation can be an excellent case to research RL. The difficulty of the mesh generation problem can be easily adjusted by altering the size and shape of input geometry domains. The reward function can also be customized to fulfill different objectives, which is not applicable in many benchmark problems. As stated by Silver, Singh, Precup, and Sutton (2021), reward function is associated with almost all the agent’s behaviours that it can learn. With mesh generation, FreeMesh-RL can be developed as a testbed for understanding those topics.

## 5.6 Conclusion

To overcome the difficulties of conventional methods in achieving the balance between high quality mesh and computational complexity, a novel RL-based method for automatic quadrilateral mesh generation, FreeMesh-RL, is presented in this article. Since the geometry domains are naturally different and complex, a partial observation of the environment is adopted as the state to eliminate the agent's dependency on the global environment and thus achieve adaptability to arbitrary meshing environments. The well designed reward function integrates the element quality, remaining boundary quality, and mesh density together, which allows the achievement of overall high mesh quality and the completeness of the meshing. Using extensive experiments on the various geometry domains, it was demonstrated that the proposed method can be adaptable to different environments and achieves competitive performance compared with representative commercial meshing software, while no complex extra operations are further needed. Consequently, FreeMesh-RL is a computationally efficient meshing framework that is fully automatic, can adapt to various complex geometries, and satisfies mesh quality requirements without extra treatments.

Furthermore, this paper has discussed the potential of the FreeMesh-RL to be a computational platform to understand many RL topics. Several future works can be conducted, such as the comparative analysis of deep RL algorithms using the FreeMesh-RL as the testbed, and the extension of this framework to the 3D mesh generation problem that is still challenging and unsolved.

## Chapter 6

# Sampling balanced high quality data to train an automatic mesh generator for its optimal performance

### Abstract

In real life situations, high quality data is scarce and imbalanced but essential for the optimal performance of data-driven algorithm models. Data synthesis methods are important techniques to address this problem. However, their main challenge lies in the dependence on the original dataset, which significantly limits the performance improvement. This article presents a quality function-based method to generate high quality data directly and uses a mesh generation algorithm as an example to demonstrate the efficiency and performance. This method samples the input-output pairs of the algorithm according to their feature spaces; selects high quality samples by the defined quality function that evaluates if the output is a suitable solution to the input; and trains an FNN model to simulate the mapping relation via the obtained data. The experiments show that the learning time of the algorithm is significantly reduced while its performance is competitive with two representative meshing algorithms.

*Keywords:* Data generation, data synthesis, mesh generation, optimal performance, algorithm design

## 6.1 Introduction

High quality data is the source for the optimal performance of most machine learning algorithms. In real world problems, the distribution of the examples is imbalanced since the minority scenarios are less frequently occurred and are even challenging to collect. These minor classes are more valuable and vital to determine whether a problem is finally solved or not. The performance of many classification, regression, and semi-supervised algorithms is significantly compromised with the imbalanced data (He & Garcia, 2009; Y. Yang et al., 2021). To tackle this imbalanced problem and increase the algorithm performance, two types of methods, data-level and algorithm-level, have been proposed and constantly improved in the last two decades (Krawczyk, 2016; Tanaka & Aranha, 2019). However, since the treatments are constrained on the given dataset, the algorithm performance is barely improved; the algorithm still cannot handle unseen situations, especially for minor ones. Instead of operating on the given dataset, this article proposes a data generation method through defining a quality function to retrieve high quality data that covers diverse problem situations and uses mesh generation as an application.

Mesh generation is one of the critical fields of computational geometry and serves as the fundamental for numerical simulations in finite element analysis (FEA), computational fluid dynamics (CFD), or graphic model rendering (Gordon & Hall, 1973; Roca & Loseille, 2019). It is also identified as one of the six basic research directions in NASA's Vision 2030 CFD study (Slotnick et al., 2014). The purpose of mesh generation is to discretize complex geometries into a finite set of (geometrically simple and bounded) elements, such as triangles or quadrilaterals (in 2D geometries) or tetrahedra or hexahedron (in 3D geometries). Due to the computational complexity, low mesh quality in complex geometries, and the dependence of heuristic knowledge in algorithm development, the existing mesh generation methods are constantly the significant bottleneck for the advancements of FEA/CFD (Slotnick et al., 2014). A robust and high performance algorithm of mesh generation is hence needed to overcome those challenges. Many researchers combine mesh generation with



artificial intelligence (AI) algorithms, including expert system (Zeng & Cheng, 1993), and neural networks (NNs) (Papagiannopoulos et al., 2021; Vinyals et al., 2015; Yao et al., 2005; Z. Zhang et al., 2020), to sidestep those difficulties in algorithm development. However, these AI-based methods are still not mature and cannot replace standard mesh generation algorithms in the industry. There are a few reasons: 1) some methods have complex NN structures and are hard to train a robust generator; 2) their problem formulation makes the generator barely adaptable to complex geometric domains; 3) their training datasets are often low quality and imbalanced, causing low learning efficiency and compromised model performance.

The first two issues can be resolved by formulating the mesh generation problem into a sequential decision-making process (Pan, Huang, Cheng, & Zeng, 2021; Pan, Huang, Wang, et al., 2021). Thus, the mesh generation problem can be decomposed into how to generate a single element at a time; the NN structures can be simplified. For the last issue, instead of balancing existing datasets to improve the model performance, this paper proposes a quality function-based data generation method to achieve the optimal performance for the mesh generator. The quality function that measures the mesh quality is determined to filter out various examples covering all the possible meshing scenarios while ensuring the overall high mesh quality. A comprehensive dataset can be obtained and used to train the mesh generator approximated by a feedforward neural network (FNN).

The main contributions of this paper are summarized as follows:

- (1) the imbalanced data problem can be solved based on a well designed quality function, which dramatically improves the algorithm model performance and saves a lot of time for algorithms that need self-exploration.
- (2) because of the generation of high quality data, the mesh generation algorithm achieves full automation without human intervention and any extra cleanup operations, which reduces the complexity and benefits downstream applications.
- (3) comparing with several representative commercial software, the obtained algorithm demonstrates competitive performance in several aspects, including mesh regularity, high quality in various geometry domains, and the necessity of handling not expected elements (i.e., triangles).

The rest of the paper is organized as follows. Section 6.2 introduces the related work about imbalanced learning and mesh generation. Section 6.3 introduces the detailed implementation of quality function-based data generation for mesh generation algorithm. Section 6.4 evaluates the performance of the proposed method and comparing with other state-of-the-art meshing approaches. Section 6.5 discusses the important improvements of the proposed method to mesh generation and machine learning communities. Section 6.6 concludes this article and indicates some future directions.

## 6.2 Problem formulation and fundamentals

This section discusses the formulation of the mesh generation problem and fundamental techniques, including mesh generation and data generation.

### 6.2.1 Mesh generation

#### Existing methods

With the rapid progress of high-performance computing hardware, mesh generation methods are required to handle geometric domains with more complex shapes and higher resolution in reliable and fast fashions. Unstructured meshes are preferred in dealing with complex geometries (Bommes et al., 2013). Many methods can directly generate quadrilateral meshes (Shewchuk, 2012). They include 1) advancing front technique, which recursively generates elements from the domain boundary and updates its boundary inwardly by cutting the generated elements until the whole domain is filled with quadrilateral elements (Blacker & Stephenson, 1991; White & Kinney, 1997; Zeng & Cheng, 1993; Zhu et al., 1991); 2) modifying the quadtree background grid to conform to the domain boundaries (Baehmann et al., 1987; Liang & Zhang, 2012); 3) packing techniques, including square packing (Shimada et al., 1998) and circle packing (Bern & Eppstein, 2000); and 4) template-based mapping methods (Gengdong & Hua, 1996). Usually, the generated meshes need extra cleanup operations to handle irregular elements and even triangles for better quality. These operations can be dividing complex geometries into small regular regions to facilitate generating regular mesh

(C. Liu et al., 2017), reducing the singularity (Verma & Suresh, 2017), performing iterative topological changes (e.g., splitting, swapping, and collapsing elements) (Docampo-Sanchez & Haimes, 2019), and mesh adaptation (Verma & Suresh, 2018). Consequently, the algorithm complexity has increased; the meshing speed becomes slower.

To reduce algorithm complexity and increase meshing performance, some research efforts have been made on combining machine learning techniques with mesh generation. Nechaeva (2006) proposed an adaptive mesh generation algorithm based on self-organizing maps (SOM) (an unsupervised neural networks-based method), which adapted a given uniform mesh onto a target physical domain through mapping. The proposed method aimed to overcome the drawbacks of SOM in tackling inaccurate mesh in the domain border and mesh construction for non-convex domains. Vinyals et al. (2015) created a new neural architecture (Pointer networks) to solve combinatorial problems by NNs. It could also generate triangular meshes by outputting a set of triplets of integers (each forms a triangle) that correspond to the order of input points. Since it is not designed for meshing problems, the performance is not robust and not complete; the final mesh is partially covered with triangular elements and intersected edges. Papagiannopoulos et al. (2021) proposed a triangular mesh generation method using three NNs. It could predict the number of candidate inner vertices (to form triangular elements), coordinates of those vertices, and their connection relations with existing segments on the boundary, respectively. It is difficult to adapt to arbitrary and complex geometry domains with the fixed input size and complex network structures. Moreover, the meshing performance is seriously limited by the generator from which the training data comes.

### **Meshing problem formulation**

The mesh generation problem is formulated into a sequential decision making process (Pan, Huang, Wang, et al., 2021). Quadrilateral mesh in single-connected 2D domains is used as an example to demonstrate this process. The process is to discretize a geometric domain into quadrilateral elements, as shown in Figure 6.1. At each time step, an element  $Q_i$  (in red) is generated from the existing boundary (in blue),  $B_i$ , which is composed of piecewise linear segments and is denoted as a sequence of vertices  $[V_1, V_2, \dots, V_{N_i}]$ . The boundary is updated by cutting off the generated element and is then used to generate the next element until the remaining vertices on the boundary form the

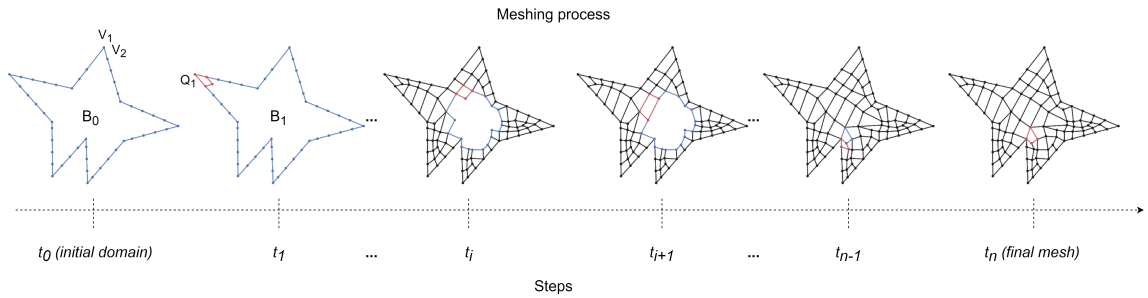


Figure 6.1: A sequence of decisions to complete the mesh. At each time step  $t_i$ , an element (in red) is extracted from the current boundary (in blue). The boundary is then updated by cutting off the element and serves as the meshing boundary in the next time step  $t_{i+1}$ . This process continues until the updated boundary becomes a quadrilateral element.

last element. The final mesh should meet several criteria: 1) each element is a quadrilateral; (2) the inner corner of each element should be between  $45^\circ$  and  $135^\circ$ ; (3) the aspect ratio (the ratio of opposite edges) and taper ratio (the ratio of neighbouring edges) of each quadrilateral should be within a predefined range; (4) the transition between a dense mesh and a coarse mesh should be smooth (Zeng & Cheng, 1993; Zeng & Yao, 2009).

The meshing procedure can be divided into four steps: 1) choosing a vertex (called reference vertex in this paper) from the boundary,  $B_i$ ; 2) constructing an element,  $Q_i$ , around the reference vertex; 3) removing the generated element, and; 4) updating the boundary. The first two steps are critical to the meshing success. Pan, Huang, Cheng, and Zeng (2021) selected the vertex on the boundary whose angle formed with its left and right vertices was the least among the remaining vertices, as the reference vertex, which demonstrated effective meshing performance. The optimal strategy can still be explored, such a dynamic selection. But it is not the focus of this paper.

Zeng and Cheng (1993) firstly found three basic boundary situations and defined three primitive rules: add one, two, and three edges, respectively, in corresponding situation to form an element. However, the difficulties are how to determine which rule should be selected and where vertices in the edge(s) should be located (Pan, Huang, Wang, et al., 2021). Yao et al. (2005) formally defined the reference vertex and its neighboring vertices as the input and the rule type and the location of the vertices as the output. They manually generated data about the specific input-output pairs and used an artificial neural network (ANN) to approximate the mapping relations. Pan, Huang, Cheng, and Zeng (2021); Pan, Huang, Wang, et al. (2021) further improved the input definition by considering

more environment information of the reference vertex, such as the vertices in the fan-shaped area formed by the reference point and its left and right vertices together. They also tried to use feed-forward neural networks (FNN) and reinforcement learning (RL) to automatically generate high quality samples (input-output pairs) and improve the accuracy of the mapping relations. Although the RL method has learned an effective meshing policy and makes the policy adaptable to complex geometries, there are some disadvantages: 1) the learning process is time-consuming; 2) the sample efficiency is low; 3) the randomness of the policy causes robustness issue for meshing. The reason behind those issues is that some boundary situations are rare, and the RL method requires more time to explore the state and action space to collect useful knowledge.

This article intends to propose a data generation method that provides comprehensive data in advance to train an efficient mesh generator to resolve this problem. The data will cover all the possible boundary situations and their corresponding decisions. Therefore, this method reduces the computational burden for the model training; it relieves researchers from ineffectively searching heuristic knowledge for extra treatments; it offers a more direct pathway to obtain a mesh that satisfies the criteria.

## **6.2.2 Data generation**

### **Existing methods**

In real life environment, the data is unequally distributed; high quality data is rare to collect. The quantity and quality of the data decide the performance of data-driven machine learning algorithms. A large amount of data is essential to establish a complex decision boundary for classification problems and avoid overfitting problem, especially in deep-learning methods entailing many parameters (LeCun et al., 2015). Imbalanced data can easily compel the prediction of algorithms towards majority classes (He & Garcia, 2009). However, the minor classes are often more important and valuable. Data augmentation/synthesis are the techniques to enlarge data size by adding slightly modified samples or newly generated artificial data from the existing dataset (He & Garcia, 2009; Krawczyk, 2016).

Conventional methods concentrate on modifying the existing dataset to have a balanced distribution. Two common strategies are 1) removing examples from majority groups (undersampling) and 2) increasing new objects for minority groups (oversampling). The random undersampling method selects samples from the majority groups without replacement and removes them from the dataset. [X.-Y. Liu et al. \(2008\)](#) proposed two informed undersampling methods to solve the deficiency of information loss in the traditional random undersampling. [Yen and Lee \(2009\)](#) devised the undersampling method based on clustering (SBC) to avoid disjunct problem. Random oversampling method is a basic method and it randomly selects minority examples and replicates them to the dataset. Since this method can easily cause overfitting problem, synthetic sampling method are thus developed. [Chawla et al. \(2002\)](#) developed a method called SMOTE, to create artificial data based on the features space similarities between minority examples. As SMOTE was very successful in various applications, there were a few improvements to the SMOTE, including borderline-SMOTE ([Han et al., 2005](#)), safe-level-SMOTE ([Bunkhumpornpat et al., 2009](#)), and Adaptive Synthetic Sampling (ADS-SYN) ([He et al., 2008](#)). [Jo and Japkowicz \(2004\)](#) proposed cluster-based over-sampling (CBO) to resolve the small disjunct problem caused by within-class imbalance. Some researchers are using the structure information underlying the data to generate synthetic samples with better quality. [Xie et al. \(2015\)](#) tried to learn the cluster structure of the training samples and generated samples for minority groups based on the density. [C.-L. Liu and Hsieh \(2019\)](#) proposed a model-based synthetic sampling (MBS) to increase the data diversity by capturing the relationship between data features via regression models. Markov chain models are also utilized to synthesize time-varying stochastic data, such as win speed ([Shamshad et al., 2005](#)) and vehicle velocity in driving cycles ([Lee & Filipi, 2010](#)), by constructing transition matrices. [W. Yang and Nam \(2022\)](#) proposed a covariance matrix-based method that used a) the correlations between features obtained from the original dataset and b) random noise.

Many neural network-based data synthesis models are constantly proposed. [Habibie et al. \(2017\)](#) built a generative model for human motion samples based on variational autoencoder (VAE). [Y. Yang et al. \(2021\)](#) utilized deep neural networks to represent the feature space of data samples. Generative adversarial network (GAN) has been widely used for data synthesis due to its flexibility and efficiency in high dimensional datasets ([Tanaka & Aranha, 2019](#)). [Z. Wang et al. \(2018\)](#) combined

GAN with autoencoders and verified the synthetic data for the vibration signal of gearbox. [Xuan et al. \(2018\)](#) explored the integration of convolutional neural networks (CNN) and GAN on the generation of pearl images. Despite of its wide usage, the data generated by GAN based methods can be low quality if 1) the training of generator is not stable, 2) the size of the original dataset is small, and 3) the dimension and nonlinearity of the dataset is high ([Tanaka & Aranha, 2019](#); [W. Yang & Nam, 2022](#)).

Data augmentation can improve the algorithm performance by adding more valuable training data, creating data variability, preventing data scarcity, and reducing overfitting. It is also beneficial for increasing the generalization ability of the models and reducing the costs of collecting and labelling data. However, the performance improvement is limited because 1) the data characteristics are hard to discover and represent, which often depends on the researcher's degree of knowledge about a specific data environment; 2) the representative features of unseen situations are usually intrinsically missing from the existing dataset. Most simple transformations to the existing data will not fundamentally improve the algorithm performance and make the algorithm general to any situation. There is still a long way to devising a data generation method that captures the nature of the data, especially those with high dimension and nonlinearity.

### **Problem formulation**

To sidestep the limitation of the existing dataset, this paper intends to generate high quality data for the algorithm with a balanced distribution directly. The obtained algorithm is thus able to adapt to various situations and applies to the real-life environment. The general procedure is shown as follows:

- (1) formulates the control problem into state-action (i.e., input-output) pairs;
- (2) determines the feature space for each dimension in the state and action, respectively;
- (3) samples the state and action pairs in their feature spaces;
- (4) defines a quality function that can evaluate the state and action pairs, which usually requires a simulation environment to the problem;
- (5) specifies quality criteria to select qualified samples;
- (6) evaluates and balances collected samples.

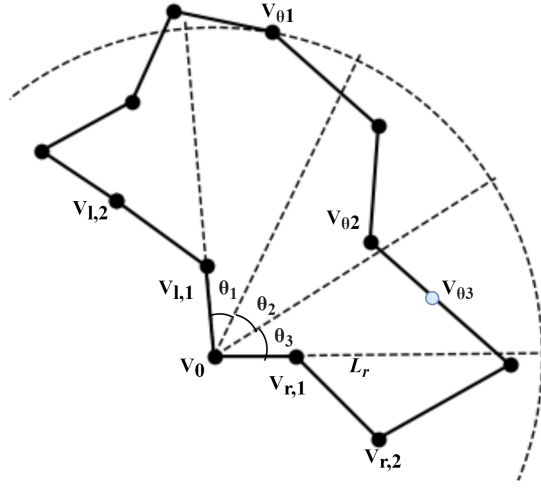


Figure 6.2: An example of the input with  $L_r = 4, n = 2, g = 3$  (Pan, Huang, Cheng, & Zeng, 2021).

Mesh generation problem is used as the example to demonstrate this procedure in this paper. It has been formulated into 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,

$$x_t = \{V_{l,n}, \dots, V_{l,1}, V_{r,1}, \dots, V_{r,n}, V_{\theta_1}, \dots, V_{\theta_g}\}. \quad (37)$$

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,  $V_{\theta_1}, \dots, V_{\theta_g}$  in the fan-shaped area  $\theta_1, \dots, \theta_g$  with radius  $L_r$ , where  $\theta_1 = \theta_2 = \dots = \theta_g$ . The output is formally defined as

$$y_t = [type_t, V_t], \quad (38)$$

where  $type_t \in \{0, 1, 2\}$ , which are corresponding to the three basic rules, respectively;  $V_t$  are the coordinates of the newly added vertex. An example of the input is shown in Fig. 6.2. A detailed description can be found in our previous work (Pan, Huang, Cheng, & Zeng, 2021).

With the formulations of input and output, the feature space for each dimension of the data can be easily determined. All kinds of boundary situations can be represented by equally sampling from those spaces. The quality function will evaluate all the possible actions for each situation, and



suitable actions can be selected via the specified criteria.

## 6.3 Quality function-based data generation for mesh generation

This section describes the quality function-based data generation method for the mesh generation algorithm. An overview of the data generation procedure will be introduced. Then, the main steps are detailed, including defining the quality function, sample balancing, and FNN training for the final mesh generation algorithm.

### 6.3.1 Data generation procedure

The general procedure of the data generation can be divided into three steps: 1) sample the input and output data according to the feature space of each dimension in the input and output, respectively; 2) set a quality threshold, and filter out samples whose quality is under the threshold; 3) balance the samples according to types. Finally, the data will be fed to an FNN model to train the mapping relations between the input and output. The overview of the data generation procedure for the mesh generation algorithm is shown in Figure 6.3.

The mesh generation problem is defined as a sequential decision making problem in the previous section. At each time step  $i$ , a partial boundary environment is taken as the input  $x_i$  (see Equation 37); the rule type and coordinates of the newly generated vertex are the output  $y_i$  (see Equation 38). This process will be repeated until the last element is formed. The partial boundary is continuously shaped and updated into various situations by the decisions. The meshing problem will be solved if the optimal decision is found for each situation. The idea behind data generation is to generate enough high quality solutions for all the possible situations; it then can be fed to train an FNN model. The dataset  $D$  is denoted as  $\{x_i, y_i\}_{i=1}^N$ .

The lower and upper value bound of each dimension in the input  $X$  is denoted as

$$X_{L,U} = ([x_{l,1}, x_{u,1}], \dots, [x_{l,n}, x_{u,n}]),$$

where  $x_{l,j}$  and  $x_{u,j}$  determines the lower and upper value bound for  $j$ th dimension of  $X$ ;  $n$  is the

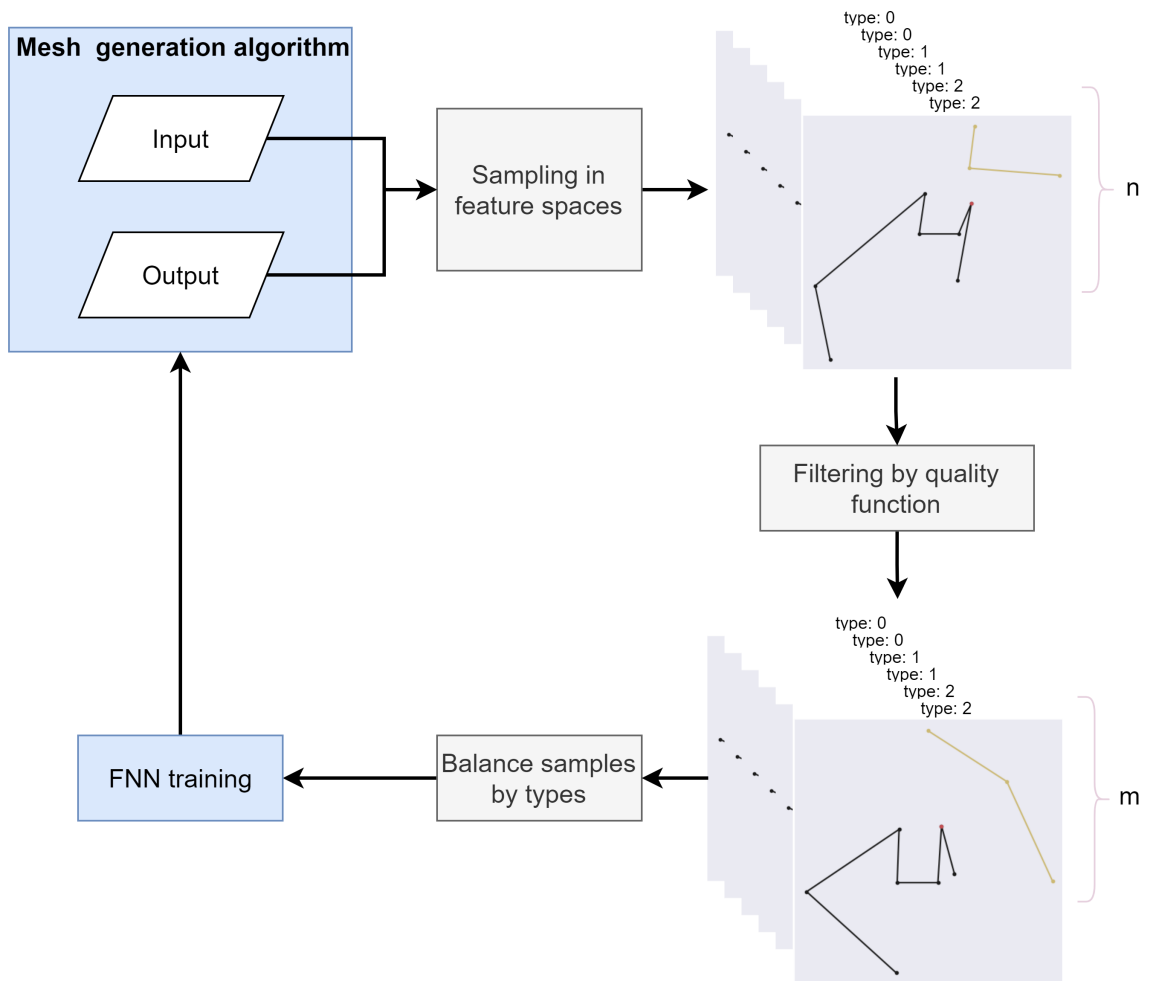


Figure 6.3: Quality function-based data generation procedure for mesh generation. The mesh generation algorithm consists of a set of input-output pairs.

size of the input dimensions. Similarly, let  $Y_{L,U} = ([y_{l,1}, y_{u,1}], \dots, [y_{l,n}, y_{u,n}])$  denotes the lower and upper value bound of each dimension in the output  $Y$ . Then, a large number of input-output pairs  $D$  can be sampled using uniform distribution. The details of the algorithm are shown as Algorithm 3.

---

**Algorithm 3** Sampling input-output pairs

---

**Require:** input and output value bounds  $X_{L,U}$  and  $Y_{L,U}$ , sample number  $N$ ;

1:  $D \leftarrow \square$

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**

---

### 6.3.2 Quality function for performance measurement

The goodness of the solution to a problem should be determined and measurable. Significantly, for a sequential decision making problem, how the solution contributes to the final goal at each step should be specified. In this article, the quality function measures the performance of the stepwise solution. After a large amount of samples is generated, the quality function is then used to evaluate them.

The objective of mesh generation is to achieve overall high mesh quality and completeness with finite quadrilateral elements, as mentioned in the previous section. The quality function will stepwise measure: 1) the validness of input and output themselves, if the vertices in the input form intersected segments or if the output has the intersections with the input. If there is such a case, the quality is set to 0. 2) the joint quality of the formed element and the updated boundary if the input and output are valid. Consequently, the quality function  $\eta(x_i, y_i)$  is defined as follows:

$$\eta(x_i, y_i) = \begin{cases} 0, & \text{invalid input and output;} \\ (1 - \alpha)\eta_i^e + \alpha\eta_i^b, & \text{otherwise;} \end{cases} \quad (39)$$

where  $\eta_i^e$  refers to the element quality measured by its edges and angles situation;  $\eta_i^b$  indicates the quality of the remaining boundary measured by both the quality of the angles formed between the newly generated elements and the boundary, and the shortest distance of the generated vertex to its

surrounding edges;  $\alpha$  adjusts the weights between the element quality and remaining boundary quality, which experimentally sets as 0.618 because the remaining boundary quality is more important to guarantee the overall mesh quality than the instant element quality.

The detailed definition and introduction of element quality  $\eta_i^e$  and boundary quality  $\eta_i^b$  can be found in our previous work (Pan, Huang, Cheng, & Zeng, 2021; Pan, Huang, Wang, et al., 2021). The reason to use these two qualities is that they represent the trade-off between the generated element and the remaining boundary. Since the generated element constantly shapes the remaining boundary, it may cause a very difficult situation to form a good quality element or even a valid element in the future if pursuing a perfect element in the current situation. The trade-off is critical to guarantee the overall mesh quality and mesh completeness.

Many high quality samples can be selected by setting a quality threshold  $\tau$  to the measurement results. The details of the algorithm are shown as Algorithm 4. Those samples  $D_\tau$  cover: 1) all the possible situations (i.e., the input) that the mesh generation may encounter; 2) all the possible high quality solutions (i.e., the output) to each situation. This is fundamental to train a comprehensive mesh generator for various geometric domains while maintaining overall high mesh quality.

---

**Algorithm 4** Filtering samples by quality threshold

---

**Require:** sample dataset  $D$ , quality threshold  $\tau$

```

1:  $D_\tau \leftarrow \square$ 
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

```

---

### 6.3.3 Sample balancing

Since the first dimension of the output is the type of the rules,  $y_{0,i} = type_i$ , it has a discrete value range,  $type_i \in \{0, 1, 2\}$ . To maintain a balanced dataset  $D_b = \{x_i, y_i\}_{i=1}^M$ , the total amount of samples for each type need to be specified as  $\gamma M$ ,  $(1 - 2\gamma)M$ , and  $\gamma M$ , respectively. This article experimentally uses  $\gamma = 1/3$ , which ensures that the sample numbers for each type are uniformly distributed.

### 6.3.4 FNN training

The finally obtained dataset  $D_b$  will be fed to train an FNN model. The model captures the mapping relations between the input and output and serves the final mesh generator. The network structure is the same as the one in (Pan, Huang, Wang, et al., 2021). The loss function, mean square error (MSE), is used to evaluate the model prediction error. The performance evaluation will be conducted in the following section.

## 6.4 Experiment results

In this section, the effectiveness of the proposed method, FreeMesh-DG, will be demonstrated by a few types of experiments: data diversity verification and meshing performance comparison. The implementation details will be introduced first, and then the experimental evaluation follows.

### 6.4.1 Implementation details

We set  $n = 3$  and  $g = 3$  in the input formulation (in Equation 37). The input is composed of vertices; each vertex is represented by a 2D polar coordinate system,  $(r, \theta)$ . The lower and upper bounds for each axis are  $[0, 1]$  and  $[0, \pi]$ , respectively. The lower and upper bounds for each dimension in the output are  $[0, 1]$ ,  $[0, 1]$ ,  $[0, \pi]$ , respectively. The optimal quality threshold  $\tau$  and the size of final balanced dataset  $M$  will be analyzed in the following section. The network structure of the FNN has 5 hidden layers as  $[64, 128, 64, 32, 16]$ , which is evaluated by Pan, Huang, Wang, 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 RAM. A total of 1 million time steps are used for training the policy.

### Quality threshold analysis

After a large amount of input-output samples are generated, the quality threshold  $\tau$  needs to be determined to select high quality solutions. Five metrics are selected to represent the basic characteristics of the dataset. They are element quality ( $\eta^e$  in Equation 39), boundary quality ( $\eta^b$ ), quality

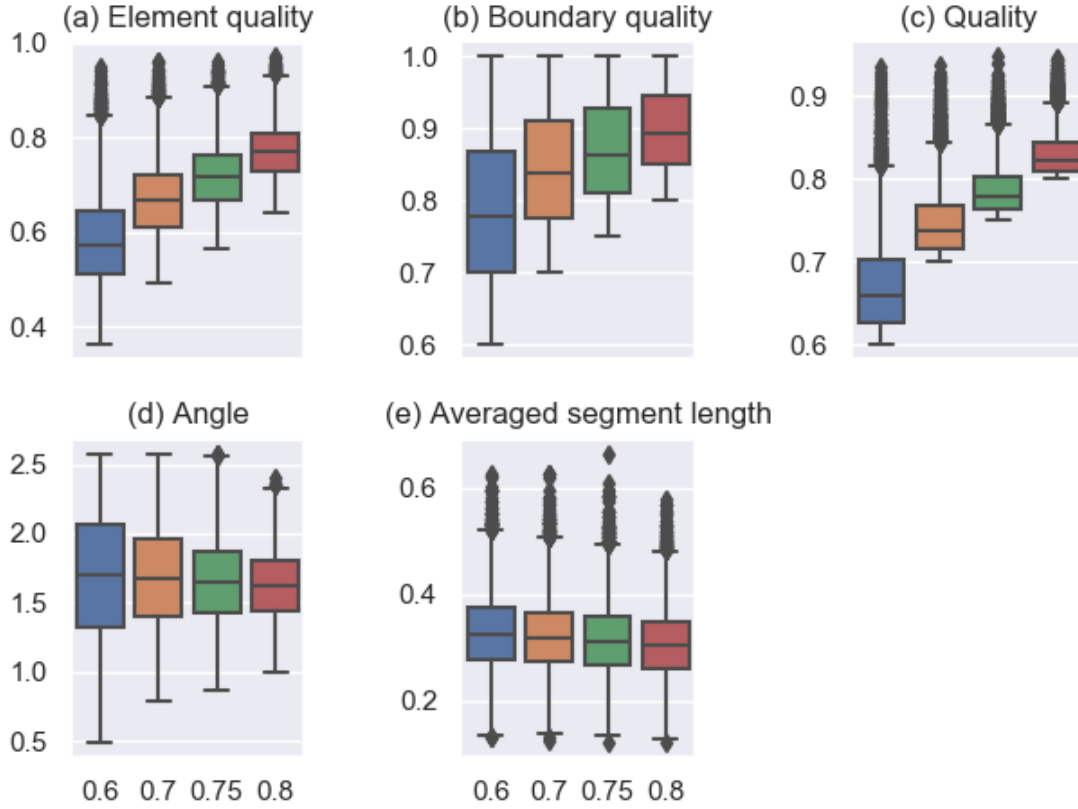


Figure 6.4: Comparison of datasets with four levels of quality thresholds,  $\tau \in \{0.6, 0.7, 0.75, 0.8\}$ . Five kinds of metrics are used to represent the characteristics of the dataset, including element quality ( $\eta^e$  in Equation 39), boundary quality ( $\eta^b$ ), quality ( $\eta$ ), angle (i.e.,  $\angle V_{i,1} V_0 V_{r,1}$  in the input), and averaged segment length.

( $\eta$ ), angle (i.e.,  $\angle V_{i,1} V_0 V_{r,1}$  in the input), and averaged segment length. Four kinds of quality thresholds,  $\tau \in \{0.6, 0.7, 0.75, 0.8\}$ , are identified and their generated datasets are evaluated on the five metrics. The comparison results are shown in Fig. 6.4. With the increase of the quality threshold, the values of three kinds of quality indices are correspondingly increased but their value ranges are decreasing. The values of the averaged segment length are slight decreased with almost fixed value range. The mean values of all the angle indices are around  $90^\circ$ ; but the value ranges are shrinking. The general principle to select the appropriate quality threshold is to guarantee the high quality while maximize the example diversity, such as the angle distribution. Therefore, we choose  $\tau = 0.7$  as a compromised threshold to select the final dataset for the optimal model.

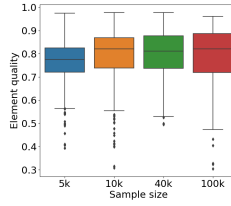


Figure 6.5: Comparison of element quality with four levels of sample size,  $M \in \{5e3, 1e4, 4e4, 1e5\}$ . Element quality is used to measure the meshing results with different levels of sample size.

### Sample size analysis

The sufficient dataset size to achieve optimal algorithm model performance is analyzed in this section. Certainly, a large amount of data could facilitate the establishment of a clear decision boundary for the targeting problem. However, it will not significantly improve the algorithm model performance but increase the computational burden if the sample size exceeds the minimum required for useful and representative statistics. Four levels of sample sizes are selected and compared for the optimal algorithm model performance, such as  $M \in \{5e3, 1e4, 4e4, 1e5\}$ . The element quality ( $\eta^e$  in Equation 39) is used as the performance indicator. The comparison results are shown in Fig. 6.5. The sample size  $M = 5e3$  has the smallest mean element quality comparing with the other three levels, which indicates the least performance. As shown in the remaining three levels, with the increase of sample size, there is no significant element quality improvement; the outliers are fewer, which shows better stability of the algorithm. Therefore, the sample size  $M = 4e4$  is used in this paper because of its balance of algorithm model performance and computational cost.

## 6.4.2 Evaluation

### Data diversity verification

Many NN-based methods are using existing mesh generators as the data sources to obtain the training data. For instance, Papagiannopoulos et al. (2021) utilized the output data of Gmsh, a mesh generator that had implemented Blossom-Quad algorithm (Remacle et al., 2012), to train their NNs. However, the diversity and balance of the obtained data are significantly limited, which significantly compromises the final model performance. To examine the diversity and balance of the data,

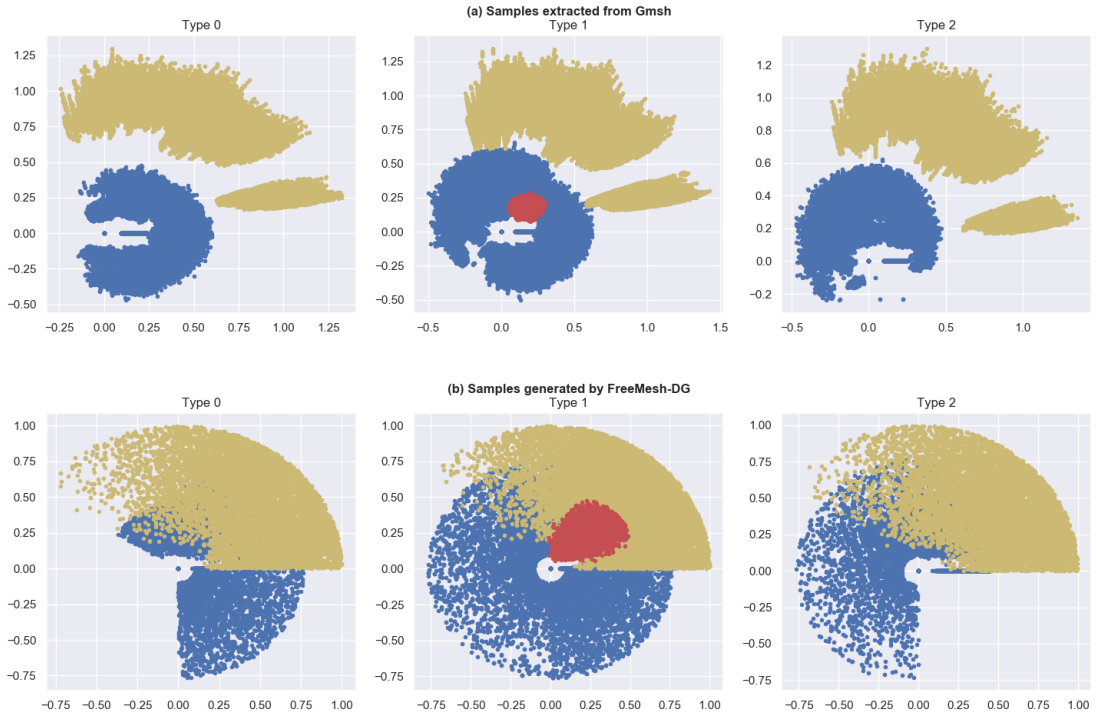


Figure 6.6: Comparison of vertex distribution of samples. The first row (i.e., Subfigure (a)) is the distribution of all the vertices in the input-output samples extracted from Gmsh; The second row (i.e., Subfigure (b)) is the vertex distribution of samples generated by FreeMesh-DG with quality threshold  $\tau \geq 0.7$ . Type 0, 1, and 2 correspond to the three basic rules in the output. Only type 1 needs generating 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 vertex.

we extract training samples from a domain meshed via Gmsh and compare it with the proposed method. The sample extraction process can be found in our previous work (Pan, Huang, Wang, et al., 2021).

The comparison results of vertex distributions of two methods are shown in Fig. 6.6. Since all the input-output pairs in the samples use a polar coordinate system with the reference vertex as the center, the data presents circular shapes. 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 smaller and more unbalanced in the samples from Gmsh (in Fig. 6.6 (a)) than via FreeMesh-DG (in Fig. 6.6 (b)) across three type of outputs. Those imbalances and limited value ranges will cause critical situations less covered.



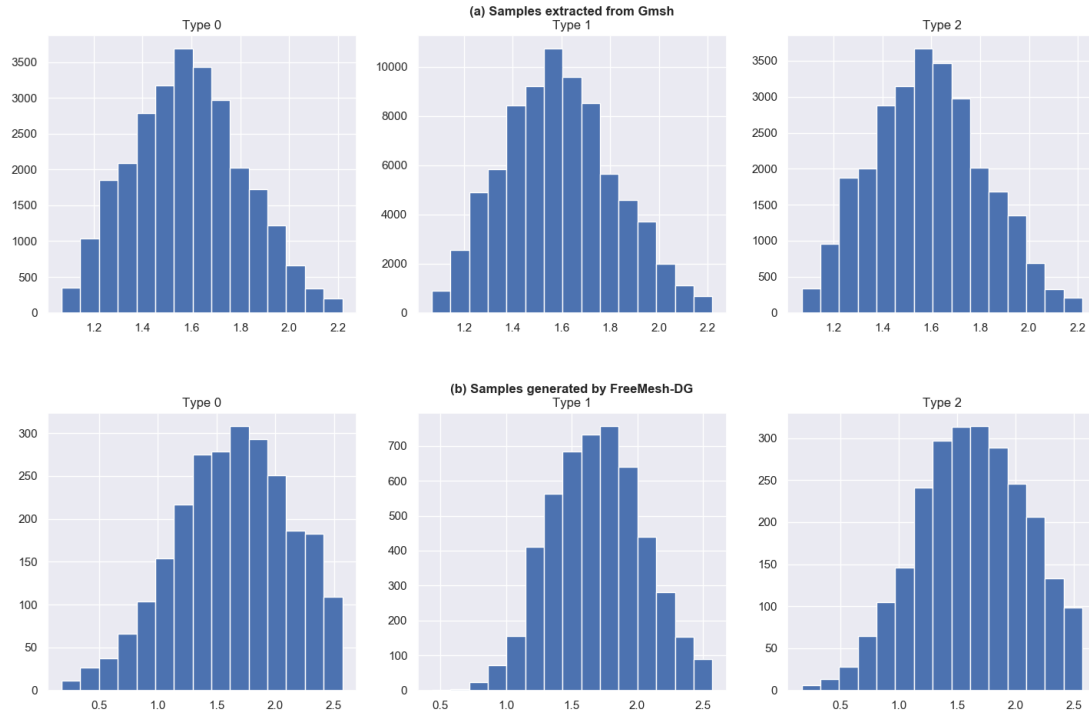


Figure 6.7: Comparison of angle distribution of samples. The angle ranges from 0 to  $\pi$ . Type 0, 1, and 2 correspond to the three basic rules in the output. Subfigure (a) is the angle distribution of samples from Gmsh; Subfigure (b) is the vertex distribution of samples generated by FreeMesh-DG with quality threshold  $\tau \geq 0.7$ .

Additionally, we examine the angle distribution in the samples of two methods, as shown in Fig. 6.7. All kinds of angles in the boundary should be covered, which facilitates the algorithm to deal with them. Both methods have a similar normal distribution with a mean of around  $90^\circ$ . The difference is that FreeMesh-DG (in Fig. 6.7 (b)) covers more smaller angles in the samples than via Gmsh (in Fig. 6.7 (a)), which can increase the accuracy and robustness of the algorithm in those situations.

Consequently, the proposed method can produce more diverse and balanced data than using existing data generators. This is usually because 1) the problematic situations for meshing seldomly occur in the existing methods; 2) the methods themselves cannot handle those situations appropriately. FreeMesh-DG can directly generate the data for various situations by uniformly sampling the feature spaces while maintaining high quality.

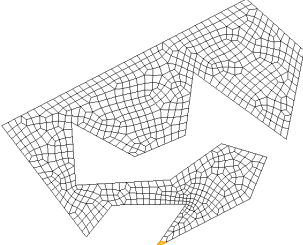
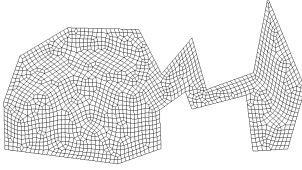
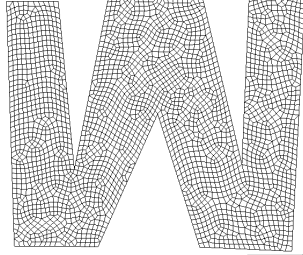
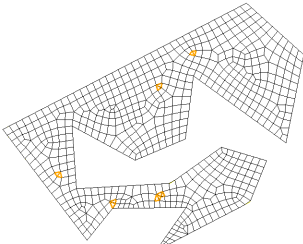
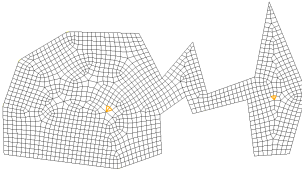
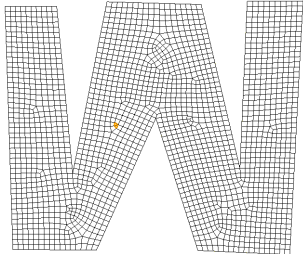
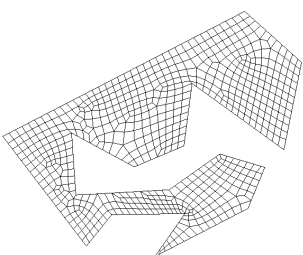
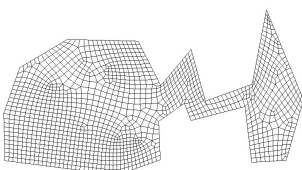
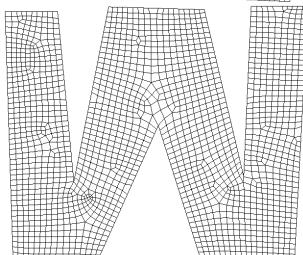
## Performance comparison

To evaluate the meshing performance, we compare the quality of meshes by FreeMesh-DG against two widely adopted meshing approaches over three predefined 2D domain boundaries (i.e., domains D1-3). These domains contain different features, including sharp angles, bottleneck regions, unevenly distributed edges, and holes, to increase the testing diversity. The two famous approaches are Blossom-Quad (Remacle et al., 2012) and Pave (Blacker & Stephenson, 1991; White & Kinney, 1997).

The meshing results are shown in Table 6.1. Although all the methods have completed the mesh for the three domains, Blossom-Quad and Pave methods are mixed with triangles marked in yellow. Their differences are 1) Blossom-Quad is problematic in handling domains with sharp boundary angles; and 2) Pave has the issue in the interior area of domains where two updated boundaries are approaching together. Extra cleanup operations are usually required to eliminate triangles or bad quality elements. FreeMesh-DG is the only method that has full quadrilateral elements, which makes it cleanup free.

The quantitative measurement of the meshing performance for the three methods are shown in Table 6.2. Eight common quality metrics are used, such as singularity, element quality ( $\eta^e$  in Equation 39),  $|MinAngle - 90|$ ,  $|MaxAngle - 90|$ , scaled Jacobian, stretch, taper, and the number of triangles (#triangles) (Pan, Huang, Wang, et al., 2021). The measurement results are averaged over the three domains. FreeMesh-DG achieves the best performance in the indices of singularity, taper, and triangle. The smaller singularity value indicates better regularity, which is beneficial for accurate numerical simulations. The smaller taper value represents that the mesh elements have more regular shapes like a square. The involvement of triangles can slow down the meshing speed because additional treatments are required to deal with them. The Pave method achieves the best performance in other indices (i.e., element quality, min and max angles, stretch, and scaled Jacobian). The larger Scaled Jacobian value means more regular quadrilateral elements. The performance of Blossom-Quad is at a suboptimal level and it only outperforms the Pave method in having less triangles. It is a common problem for all the indirect methods because of the preliminary triangulation (Remacle et al., 2013).

Table 6.1: Meshing results comparison

Algorithms	Domain 1	Domain 2	Domain 3
Blossom-Quad			
Pave			
FreeMesh-DS			

The elements in yellow represent existing triangles in the meshes.

Table 6.2: Averaged mesh quality metrics over the three domains

Metrics	Blossom-Quad	Pave	FreeMesh-DG
Singularity (L)	$322.67 \pm 114.11$	$78.67 \pm 17.15$	<b><math>73.33 \pm 13.57</math></b>
Element quality (H)	$0.75 \pm 0.11$	<b><math>0.87 \pm 0.10</math></b>	$0.83 \pm 0.12$
$ MinAngle - 90 $ (L)	$6.86 \pm 6.77$	<b><math>3.33 \pm 3.33</math></b>	$5.87 \pm 7.13$
$ MaxAngle - 90 $ (L)	$19.06 \pm 10.43$	<b><math>9.33 \pm 12.54</math></b>	$11.95 \pm 11.70$
Scaled jacobian (H)	$0.93 \pm 0.07$	<b><math>0.97 \pm 0.11</math></b>	$0.96 \pm 0.08$
Stretch (H)	$0.82 \pm 0.07$	<b><math>0.90 \pm 0.07</math></b>	$0.86 \pm 0.08$
Taper (L)	$0.11 \pm 0.10$	$0.05 \pm 0.11$	<b><math>0.05 \pm 0.08</math></b>
Triangle (L)	$0.67 \pm 0.94$	$6 \pm 5.29$	<b><math>0 \pm 0</math></b>

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.

## 6.5 Discussion

The proposed method, FreeMesh-DG, has exhibited its competitive performance against the other two meshing algorithms and achieved the optimal performance in three quality indices: singularity, taper, and triangle. Meanwhile, it demonstrates some further significance from the following aspects.

FreeMesh-DG can fulfill various quality requirements of downstream applications by customizing the quality function specifically. Different applications have distinct simulation tasks that require the mesh with diverse topological and geometric features. Since existing mesh generators can only produce meshes with their own characteristics, many post-processing techniques are hence needed, such as remeshing and mesh adaptation (Verma & Suresh, 2017). However, the development of those extra techniques is expensive and time-consuming; they will also slow down the meshing speed and increase the algorithm complexity. FreeMesh-DG provides a novel path to directly design the desired mesh by only adjusting the quality function. The generated data can then be fed to train the target algorithm within short period of time, which is valuable in real-life situations.

Many real-life problems can be resolved in the same fashion of FreeMesh-DG. The problem can be formulated into a sequential decision making process; a few primitive rules (i.e., state-action pairs) are identified; a quality function is then used to select high quality samples; a neural network can be finally trained to capture the mapping relations between states and actions using the obtained data. In this way, a lot of human efforts could be saved to obtain an efficient and robust algorithm.

## 6.6 Conclusion

To overcome the high quality data scarcity and imbalance problem, this paper presents a quality function-based data generation method for automatic quadrilateral mesh generation algorithm, FreeMesh-DG. The mesh generation is defined as a sequential decision making problem and consists of a set of state-action (i.e., input-output) pairs. A large amount of data is equally sampled from the feature spaces of the input and output, which intends to cover all the possible input situations and output solutions. Then those samples are evaluated and selected by a well designed quality function that determines if the output is a qualified solution to the input. The extensive experiments

demonstrate that the obtained algorithm based on the generated data is competitive with two representative mesh generation approaches and even outperforms them in three importance measurement indices.

This paper has further discussed the potential of the FreeMesh-DG to the mesh generation and algorithm design community. Several future works can be conducted, such as exploring quality functions for different requirements of downstream applications, and the extension of this method to the 3D mesh generation problem.

# Chapter 7

## Conclusions and future works

### 7.1 Conclusions

The objective of this thesis was threefold: 1) design smart element extraction rules that are adaptable to complex geometries; 2) design a self-learning mechanism to acquire robust and high-quality element extraction rules without human intervention; and 3) investigate the correlations between the input and output of the rule, and increase the sampling and learning efficiency of extraction rules.

As was reviewed in Chapter 2, numerous techniques have been proposed over the years to generate quadrilateral meshes. We have focused on the element extraction methods because of their high mesh quality, especially along the domain boundary. Conventionally, element extraction methods rely heavily on heuristic operations to guarantee the overall mesh quality, which is extremely difficult, expensive, and time-consuming for human algorithm designers. In Chapter 4, a self-learning system, FreeMesh-S, is proposed to generate quadrilateral elements by automatically acquiring robust and high-quality element extraction rules. First, according to the recursive logic, the extraction rules are categorized into three types (i.e., adding 1, 2, and 3 edges, respectively). These three primitive rules are smart and sufficient to mesh for any complex geometries. Then, a novel learning scheme formed by A2C and FNN is used to construct the input-output relations automatically. The experiment results demonstrate that derived element extraction rules can be adaptive to various boundary shapes, achieve competitive performance with the other three popular meshing methods,

and outperform them at Scaled Jacobian and Taper indices. Simultaneously, FreeMesh-S requires the minimal and simplest geometric knowledge.

To further increase the algorithm automation and reduce human intervention, a sole RL-based learning method, FreeMesh-RL, is proposed to obtain the element extraction rules. A partial observation of the environment is designed as the state to eliminate the agent’s dependency on global information and thus achieve adaptability to arbitrary environments. A well designed reward function incorporates the element quality, boundary quality, and mesh density together, which essentially contributes to the overall high mesh quality and mesh completeness. Extensive experiments demonstrate that the proposed method can be adaptable to different environments and achieve competitive performance compared with representative commercial meshing software while no complex extra operations are further needed.

High quality data scarcity and imbalanced distribution are the main challenges for optimal sample efficiency and algorithm performance. To overcome the issues, Chapter 6 proposes a quality function-based data generation method for automatic quadrilateral mesh generation algorithm. As the mesh generation consists of a set of state-action (i.e., input-output) pairs, a large amount of data can be uniformly sampled from their feature spaces. The purpose is to obtain all the possible input situations and output solutions. A quality function is designed to evaluate if the output is a qualified solution to each input in the samples. Compared with two representative mesh generation approaches, the trained algorithm based on the generated data achieves optimal performance in importance measurement indices. Moreover, the algorithm development time is much faster than the previous two methods.

## 7.2 Future works

Many perspectives for future works exist.

- (1) Extension of the quadrilateral techniques to 3D: The proposed three methods have thoroughly studied the formulation of primitive extraction rules of quadrilateral elements and their automatically obtaining via reinforcement learning and data generation techniques. The 3D mesh generation problem could be similarly cast into an MDP process and its primitive rules can

be formulated.

- (2) Build a benchmark framework based on quadrilateral mesh generation for analyzing different machine learning algorithms: As explained in 6.5, the mesh generation has the potential to be a benchmark problem for theoretically analyzing the performance of various RL algorithms, and understanding many RL topics, such as state representation and reward specification. A benchmark framework can thus be built for such objectives. Many supervised algorithms can be analyzed and evaluated by this framework as well.
- (3) Combining data generation technique with reinforcement learning algorithms: Sample efficiency is a common challenge for RL algorithms. The quality function-based data generation method could provide samples sufficiently. The combination of these two methods to reduce exploration time for RL algorithms could be investigated.



# References

- Akhras, G. (2000). Smart materials and smart systems for the future. *Canadian Military Journal*, 1(3), 25–31.
- Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., ... Zaremba, W. (2017). Hindsight experience replay. *arXiv preprint arXiv:1707.01495*.
- Atalay, F. B., Ramaswami, S., & Xu, D. (2008). Quadrilateral meshes with bounded minimum angle. In *Proceedings of the 17th international meshing roundtable* (pp. 73–91). Springer.
- Baehmann, P. L., Wittchen, S. L., Shephard, M. S., Grice, K. R., & Yerry, M. A. (1987). Robust, geometrically based, automatic two-dimensional mesh generation. *International Journal for Numerical Methods in Engineering*, 24(6), 1043–1078.
- Bern, M., & Eppstein, D. (2000). Quadrilateral meshing by circle packing. *International Journal of Computational Geometry & Applications*, 10(04), 347–360.
- Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., ... others (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.
- Blacker, T. D., Owen, S. J., Staten, M. L., Quadros, W. R., Hanks, B., Clark, B. W., ... others (2016). *Cubit geometry and mesh generation toolkit 15.2 user documentation* (Tech. Rep.). Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).
- Blacker, T. D., & Stephenson, M. B. (1991). Paving: A new approach to automated quadrilateral mesh generation. *International journal for numerical methods in engineering*, 32(4), 811–847.
- Bommes, D., Lévy, B., Pietroni, N., Puppo, E., Silva, C., Tarini, M., & Zorin, D. (2013). Quad-mesh generation and processing: A survey. , 32(6), 51–76.

- Brewer, M. L., Diachin, L. F., Knupp, P. M., Leurent, T., & Melander, D. J. (2003). The mesquite mesh quality improvement toolkit. In *Imr*.
- Bunkhumpornpat, C., Sinapiromsaran, K., & 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* (pp. 475–482).
- Capuano, G., & Rimoli, J. J. (2019). Smart finite elements: A novel machine learning application. *Computer Methods in Applied Mechanics and Engineering*, 345, 363–381.
- Catmull, E., & Clark, J. (1978). Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-aided design*, 10(6), 350–355.
- Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16, 321–357.
- Chedid, R., & Najjar, N. (1996). Automatic finite-element mesh generation using artificial neural networks-part i: Prediction of mesh density. *IEEE Transactions on Magnetics*, 32(5), 5173–5178.
- Chen, R. T., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018). Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*.
- Chew, L. P. (1989). Constrained delaunay triangulations. *Algorithmica*, 4(1-4), 97–108.
- Daniels, J., Silva, C. T., Shepherd, J., & Cohen, E. (2008). Quadrilateral mesh simplification. *ACM transactions on graphics (TOG)*, 27(5), 1–9.
- Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 3844–3852.
- Docampo-Sanchez, J., & Haimes, R. (2019). Towards fully regular quad mesh generation. In *Aiaa scitech 2019 forum* (p. 1988).
- Dolšák, B. (2002). Finite element mesh design expert system. In *Applications and innovations in intelligent systems ix* (pp. 3–14). Springer.
- Dong, Q., Gong, S., & Zhu, X. (2018). Imbalanced deep learning by minority class incremental rectification. *IEEE transactions on pattern analysis and machine intelligence*, 41(6), 1367–1381.

- Drummond, C., & Holte, R. C. (2000). Exploiting the cost (in) sensitivity of decision tree splitting criteria. In *Icml* (Vol. 1).
- Duan, Y., Chen, X., Houthoofd, R., Schulman, J., & Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning* (pp. 1329–1338).
- Dulac-Arnold, G., Mankowitz, D., & Hester, T. (2019). Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*.
- Ebeida, M. S., Karamete, K., Mestreau, E., & Dey, S. (2010). Q-tran: a new approach to transform triangular meshes into quadrilateral meshes locally. In *Proceedings of the 19th international meshing roundtable* (pp. 23–34). Springer.
- Fan, W., Stolfo, S. J., Zhang, J., & Chan, P. K. (1999). Adacost: misclassification cost-sensitive boosting. In *Icml* (Vol. 99, pp. 97–105).
- Garimella, R. V., Shashkov, M. J., & Knupp, P. M. (2004). Triangular and quadrilateral surface mesh quality optimization using local parametrization. *Computer Methods in Applied Mechanics and Engineering*, 193(9-11), 913–928.
- Gengdong, C., & Hua, L. (1996). New method for graded mesh generation of quadrilateral finite elements. *Computers & structures*, 59(5), 823–829.
- Geuzaine, C., & Remacle, J.-F. (2009). Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International journal for numerical methods in engineering*, 79(11), 1309–1331.
- Gordon, W. J., & Hall, C. A. (1973). Construction of curvilinear co-ordinate systems and applications to mesh generation. *International Journal for Numerical Methods in Engineering*, 7(4), 461–477.
- Gottesman, O., Johansson, F., Komorowski, M., Faisal, A., Sontag, D., Doshi-Velez, F., & Celi, L. A. (2019). Guidelines for reinforcement learning in healthcare. *Nature medicine*, 25(1), 16–18.
- Grondman, I., Busoniu, L., Lopes, G. A., & Babuska, R. (2012). A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6), 1291–1307.

- Gupta, K. (2020). *Neural mesh flow: 3d manifold mesh generation via diffeomorphic flows*. University of California, San Diego.
- Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning* (pp. 1861–1870).
- Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., . . . others (2018). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.
- Habibie, I., Holden, D., Schwarz, J., Yearsley, J., & Komura, T. (2017). A recurrent variational autoencoder for human motion synthesis. In *28th british machine vision conference*.
- Han, H., Wang, W.-Y., & Mao, B.-H. (2005). Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing* (pp. 878–887).
- Hanocka, R., Hertz, A., Fish, N., Giryas, R., Fleishman, S., & Cohen-Or, D. (2019). Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4), 1–12.
- He, H., Bai, Y., Garcia, E. A., & 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)* (pp. 1322–1328).
- He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9), 1263–1284.
- Huang, C., Li, Y., Loy, C. C., & Tang, X. (2016). Learning deep representation for imbalanced classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 5375–5384).
- Jadid, M. N., & Fairbairn, D. R. (1994). The application of neural network techniques to structural analysis by implementing an adaptive finite-element mesh generation. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 8(3), 177–191. doi: 10.1017/S0890060400001979
- Jo, T., & Japkowicz, N. (2004). Class imbalances versus small disjuncts. *ACM Sigkdd Explorations Newsletter*, 6(1), 40–49.
- Joe, B. (1995). Quadrilateral mesh generation in polygonal regions. *Computer-Aided Design*, 27(3),

209–222.

- Johnen, A. (2016). *Indirect quadrangular mesh generation and validation of curved finite elements* (Unpublished doctoral dissertation). Université de Liège, Liège, Belgique.
- Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., ... others (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873), 583–589.
- Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: a survey. *Journal of artificial intelligence research*, 4, 237–285.
- Kahneman, D. (2011). *Thinking, fast and slow*. Macmillan.
- Khan, S. H., Hayat, M., Bennamoun, M., Sohel, F. A., & Togneri, R. (2017). Cost-sensitive learning of deep feature representations from imbalanced data. *IEEE transactions on neural networks and learning systems*, 29(8), 3573–3587.
- Knupp, P. M. (2000). Achieving finite element mesh quality via optimization of the jacobian matrix norm and associated quantities. part ii—a framework for volume mesh optimization and the condition number of the jacobian matrix. *International Journal for numerical methods in engineering*, 48(8), 1165–1185.
- Knupp, P. M., Ernst, C., Thompson, D. C., Stimpson, C., & Pebay, P. P. (2006). *The verdict geometric quality library* (Tech. Rep.). Sandia National Laboratories.
- Kohonen, T. (2012). *Self-organization and associative memory* (Vol. 8). Springer Science & Business Media.
- Krawczyk, B. (2016). Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4), 221–232.
- Kukar, M., Kononenko, I., et al. (1998). Cost-sensitive learning with neural networks. In *Ecai* (Vol. 15, pp. 88–94).
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, 521(7553), 436–444.
- Lee, T.-K., & Filipi, Z. S. (2010). Synthesis and validation of representative real-world driving cycles for plug-in hybrid vehicles. In *2010 IEEE vehicle power and propulsion conference* (pp. 1–6).
- Lesort, T., Díaz-Rodríguez, N., Goudou, J.-F., & Filliat, D. (2018). State representation learning for control: An overview. *Neural Networks*, 108, 379–392.

- Lévy, B., & Liu, Y. (2010). L p centroidal voronoi tessellation and its applications. *ACM Transactions on Graphics (TOG)*, 29(4), 1–11.
- Li, Y. (2018). Deep reinforcement learning. *arXiv preprint arXiv:1810.06339*.
- Liang, X., Ebeida, M. S., & Zhang, Y. (2010). Guaranteed-quality all-quadrilateral mesh generation with feature preservation. *Computer Methods in Applied Mechanics and Engineering*, 199(29-32), 2072–2083.
- Liang, X., & Zhang, Y. (2012). Matching interior and exterior all-quadrilateral meshes with guaranteed angle bounds. *Engineering with Computers*, 28(4), 375–389.
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.
- Littman, M. L. (2015). Reinforcement learning improves behaviour from evaluative feedback. *Nature*, 521(7553), 445–451.
- Liu, C., Yu, W., Chen, Z., & Li, X. (2017). Distributed poly-square mapping for large-scale semi-structured quad mesh generation. *Computer-Aided Design*, 90, 5–17.
- Liu, C.-L., & Hsieh, P.-Y. (2019). Model-based synthetic sampling for imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 32(8), 1543–1556.
- Liu, X.-Y., Wu, J., & Zhou, Z.-H. (2008). Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2), 539–550.
- Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., & Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61, 523–562.
- Manevitz, L., Yousef, M., & Givoli, D. (1997). Finite-element mesh generation using self-organizing neural networks. *Computer-Aided Civil and Infrastructure Engineering*, 12(4), 233–250.
- Mazyavkina, N., Sviridov, S., Ivanov, S., & Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 105400.
- McCarthy, K., Zabar, B., & Weiss, G. (2005). Does cost-sensitive learning beat sampling for classifying rare classes? In *Proceedings of the 1st international workshop on utility-based data mining* (pp. 69–77).

- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., . . . Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937).
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- Nechaeva, O. (2006). Composite algorithm for adaptive mesh construction based on self-organizing maps. In *International conference on artificial neural networks* (pp. 445–454). Springer.
- Niv, Y. (2019). Learning task-state representations. *Nature neuroscience*, 22(10), 1544–1553.
- Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., . . . others (2019). Behaviour suite for reinforcement learning. *arXiv preprint arXiv:1908.03568*.
- Owen, S. J. (1998). A survey of unstructured mesh generation technology. In *Imr* (Vol. 239, p. 267).
- Owen, S. J., Staten, M. L., Canann, S. A., & Saigal, S. (1999). Q-morph: an indirect approach to advancing front quad meshing. *International journal for numerical methods in engineering*, 44(9), 1317–1340.
- Pan, J., Huang, J., Cheng, G., & Zeng, Y. (2021). Reinforcement learning for automatic quadrilateral mesh generation: a soft actor-critic approach. *Under review*.
- Pan, J., Huang, J., Wang, Y., Cheng, G., & Zeng, Y. (2021). A self-learning finite element extraction system based on reinforcement learning. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 1–29. doi: 10.1017/S089006042100007X
- Papagiannopoulos, A., Clausen, P., & Avellan, F. (2021). How to teach neural networks to mesh: Application on 2-d simplicial contours. *Neural Networks*, 136, 152–179.
- Park, C., Noh, J.-S., Jang, I.-S., & Kang, J. M. (2007). A new automated scheme of quadrilateral mesh generation for randomly distributed line constraints. *Computer-Aided Design*, 39(4), 258–267.
- Pébay, P. P., Thompson, D., Shepherd, J., Knupp, P., Lisle, C., Magnotta, V. A., & Grosland, N. M. (2008). New applications of the verdict library for standardized mesh verification pre, post, and end-to-end processing. In *Proceedings of the 16th international meshing roundtable* (pp. 535–552).
- Peters, J., & Reif, U. (1997). The simplest subdivision scheme for smoothing polyhedra. *ACM*

- Transactions on Graphics (TOG)*, 16(4), 420–431.
- Prautzsch, H., & Chen, Q. (2011). Analyzing midpoint subdivision. *Computer Aided Geometric Design*, 28(7), 407–419.
- Quadros, W., Ramaswami, K., Prinz, F., & Gurumoorthy, B. (2004). Laytracks: a new approach to automated geometry adaptive quadrilateral mesh generation using medial axis transform. *International journal for numerical methods in engineering*, 61(2), 209–237.
- Remacle, J.-F., Henrotte, F., Carrier-Baudouin, T., Béchet, E., Marchandise, E., Geuzaine, C., & Mouton, T. (2013). A frontal delaunay quad mesh generator using the l norm. *International Journal for Numerical Methods in Engineering*, 94(5), 494–512.
- Remacle, J.-F., Lambrechts, J., Seny, B., Marchandise, E., Johnen, A., & 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(9), 1102–1119.
- Roca, X., & Loseille, A. (2019). *27th international meshing roundtable* (Vol. 127). Springer.
- Rushdi, A. A., Mitchell, S. A., Mahmoud, A. H., Bajaj, C. C., & Ebeida, M. S. (2017). All-quad meshing without cleanup. *Computer-Aided Design*, 85, 83–98.
- Sarrate Ramos, J., Ruiz-Gironés, E., & Roca Navarro, F. J. (2014). Unstructured and semi-structured hexahedral mesh generation methods. *Computational technology reviews*, 10, 35–64.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Shamshad, A., Bawadi, M., Hussin, W. W., Majid, T., & Sanusi, S. (2005). First and second order markov chain models for synthetic generation of wind speed time series. *Energy*, 30(5), 693–708.
- Shewchuk, J. R. (2012). Unstructured mesh generation. *Combinatorial Scientific Computing*, 12(257), 2.
- Shimada, K., Liao, J.-H., & Itoh, T. (1998). Quadrilateral meshing with directionality control through the packing of square cells. In *Imr* (pp. 61–75). Citeseer.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., . . . others (2016).



- Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587), 484–489.
- Silver, D., Singh, S., Precup, D., & Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, 103535.
- Slotnick, J., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., & Mavriplis, D. (2014). *Cfd vision 2030 study: a path to revolutionary computational aerosciences*. National Aeronautics and Space Administration, Langley Research Center.
- Sun, Y., Kamel, M. S., Wong, A. K., & Wang, Y. (2007). Cost-sensitive boosting for classification of imbalanced data. *Pattern recognition*, 40(12), 3358–3378.
- Suresh, K., & Verma, C. S. (2019). *Singularity reduction in quadrilateral meshes* (No. 20190325647).
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: an introduction*. MIT press.
- Tam, T., & Armstrong, C. G. (1991). 2d finite element mesh generation by medial axis subdivision. *Advances in engineering software and workstations*, 13(5-6), 313–324.
- Tanaka, F. H. K. d. S., & Aranha, C. (2019). Data augmentation using gans. *arXiv preprint arXiv:1904.09135*.
- Thompson, J. F., Soni, B. K., & Weatherill, N. P. (1998). *Handbook of grid generation*. CRC press.
- Verma, C. S., & Suresh, K. (2017). A robust combinatorial approach to reduce singularities in quadrilateral meshes. *Computer-Aided Design*, 85, 99–110.
- Verma, C. S., & Suresh, K. (2018).  $\alpha$ mst: A robust unified algorithm for quadrilateral mesh adaptation in 2d and 3d. *Computer-Aided Design*, 103, 47–60.
- Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., ... others (2017). Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*.
- Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. In *In advances in neural information processing systems* (Vol. 28, pp. 2692–2700).
- Wang, N., Zhang, Y., Li, Z., Fu, Y., Liu, W., & Jiang, Y.-G. (2018). Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the european conference on computer vision (eccv)* (pp. 52–67).
- Wang, W. Y., Li, J., & He, X. (2018). Deep reinforcement learning for nlp. In *Proceedings of the*

- 56th annual meeting of the association for computational linguistics: Tutorial abstracts* (pp. 19–21).
- Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning* (pp. 1995–2003).
- Wang, Z., Wang, J., & 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.
- Wei, Q., Wang, L., Liu, Y., & Polycarpou, M. M. (2020). Optimal elevator group control via deep asynchronous actor–critic learning. *IEEE transactions on neural networks and learning systems*, *31*(12), 5245–5256.
- Wen, C., Zhang, Y., Li, Z., & Fu, Y. (2019). Pixel2mesh++: Multi-view 3d mesh generation via deformation. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 1042–1051).
- White, D. R., & Kinney, P. (1997). Redesign of the paving algorithm: Robustness enhancements through element by element meshing. In *6th international meshing roundtable* (Vol. 10, p. 830).
- Wu, G., & Chang, E. Y. (2005). Kba: Kernel boundary alignment considering imbalanced data distribution. *IEEE Transactions on knowledge and data engineering*, *17*(6), 786–795.
- Wu, Y., Liao, S., Liu, X., Li, Z., & Lu, R. (2021). Deep reinforcement learning on autonomous driving policy with auxiliary critic network. *IEEE Transactions on Neural Networks and Learning Systems*.
- Xie, Z., Jiang, L., Ye, T., & 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* (pp. 3–18).
- Xuan, Q., Chen, Z., Liu, Y., Huang, H., Bao, G., & Zhang, D. (2018). Multiview generative adversarial network and its application in pearl classification. *IEEE Transactions on Industrial Electronics*, *66*(10), 8244–8252.
- Yang, J., Dzanic, T., Petersen, B., Kudo, J., Mittal, K., Tomov, V., . . . others (2021). Reinforcement

- learning for adaptive mesh refinement. *arXiv preprint arXiv:2103.01342*.
- Yang, W., & Nam, W. (2022). Data synthesis method preserving correlation of features. *Pattern Recognition, 122*, 108241.
- Yang, Y., Zha, K., Chen, Y.-C., Wang, H., & Katabi, D. (2021). Delving into deep imbalanced regression. *arXiv preprint arXiv:2102.09554*.
- Yao, S., Yan, B., Chen, B., & Zeng, Y. (2005). An ann-based element extraction method for automatic mesh generation. *Expert Systems with Applications, 29*(1), 193–206.
- Yen, S.-J., & Lee, Y.-S. (2009). Cluster-based under-sampling approaches for imbalanced data distributions. *Expert Systems with Applications, 36*(3), 5718–5727.
- Zeng, Y. (2004). Environment-based formulation of design problem. *Journal of Integrated Design and Process Science, 8*(4), 45–63.
- Zeng, Y. (2015). Environment-based design (ebd): A methodology for transdisciplinary design+. *Journal of Integrated Design and Process Science, 19*(1), 5–24.
- Zeng, Y., & Cheng, G. (1991). On the logic of design. *Design Studies, 12*(3), 137–141.
- Zeng, Y., & Cheng, G. (1993). Knowledge-based free mesh generation of quadrilateral elements in two-dimensional domains. *Computer-Aided Civil and Infrastructure Engineering, 8*(4), 259–270.
- Zeng, Y., & Yao, S. (2009). Understanding design activities through computer simulation. *Advanced Engineering Informatics, 23*(3), 294–308.
- Zhang, K., Cheng, G., & Xu, L. (2019). Topology optimization considering overhang constraint in additive manufacturing. *Computers & Structures, 212*, 86–100.
- Zhang, L., Cheng, L., Li, H., Gao, J., Yu, C., Domel, R., ... Liu, W. K. (2021). Hierarchical deep-learning neural networks: finite elements and beyond. *Computational Mechanics, 67*(1), 207–230.
- Zhang, Q., Yang, L. T., Chen, Z., & Li, P. (2018). A survey on deep learning for big data. *Information Fusion, 42*, 146–157.
- Zhang, Z., Wang, Y., Jimack, P. K., & Wang, H. (2020). Meshingnet: A new mesh generation method based on deep learning. In *International conference on computational science* (pp. 186–198). Springer.

- Zhao, D., Chen, J., Zheng, Y., Huang, Z., & Zheng, J. (2015). Fine-grained parallel algorithm for unstructured surface mesh generation. *Computers & Structures*, *154*, 177–191.
- Zhou, Z.-H., & Liu, X.-Y. (2005). Training cost-sensitive neural networks with methods addressing the class imbalance problem. *IEEE Transactions on knowledge and data engineering*, *18*(1), 63–77.
- Zhou, Z.-H., & Liu, X.-Y. (2010). On multi-class cost-sensitive learning. *Computational Intelligence*, *26*(3), 232–257.
- Zhu, J., Zienkiewicz, O., Hinton, E., & Wu, J. (1991). A new approach to the development of automatic quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, *32*(4), 849–866.
- Ziebart, B. D. (2010). *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University.