GENERATE PLANE QUAD MESH WITH NEURAL NETWORKS AND TREE SEARCH (DRAFT)

A PREPRINT

Hua Tong*
Department of Modern Mechanics
University of Science and Technology of China
Hefei, China 230027
gt2k01@mail.ustc.edu.cn

November 27, 2021

ABSTRACT

The quality of mesh generation has long been considered a vital aspect in providing engineers with reliable simulation results throughout the history of the Finite Element Method (FEM). The element extraction method, which is currently the most robust method, is used in business software. However, in order to speed up extraction, the approach is done by finding the next element that optimizes a target function, which can result in local mesh of bad quality after many time steps. We provide *TreeMesh*, a method that uses this method in conjunction with reinforcement learning (also possible with supervised learning) and a novel Monte-Carlo tree search (MCTS) ([Coulom(2006)], [Kocsis and Szepesvári(2006)], [Browne et al.(2012)]). The algorithm is based on a previously proposed approach ([Pan et al.(2021)]). After making many improvements on DRL (algorithm, state-action-reward setting) and adding a MCTS, it outperforms the former work on the same boundary. Furthermore, using tree search, our program reveals much preponderance on seed-density-changing boundaries, which is common on thin-film materials.

1 Introduction

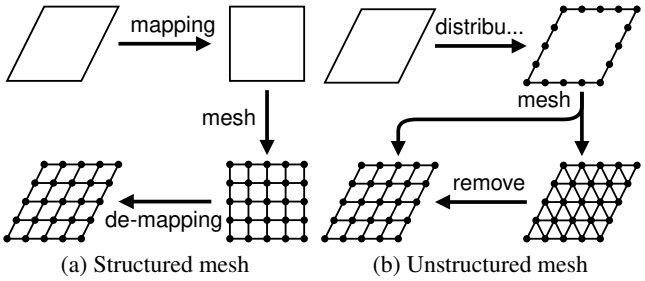
1.1 FEM mesh generation

In Figure 1, there are two techniques to mesh a given boundary: structured ([Thompson et al.(1998)]) and unstructured meshes ([Owen(1998)]). All elements in a structured mesh are first arranged in a regular way before being mapped to the real-world boundaries. As a result of the complex boundaries, the mesh quality may be poor. Unstructured mesh can be categorized into indirect and direct methods. Indirect methods require an intermediate triangular mesh, which is prone to instability and has a restricted number of vertices ([Garimella et al.(2004)]), whereas direct methods generate quadrilateral pieces directly, avoiding some potential problems ([Zeng and Cheng(1993)]). Our research is based on the element extraction method, which is the most popular of the direct methods ([Docampo-Sanchez and Haimes(2019)]).

1.2 Previous works

MCTS with deep neural networks (NN) can defeat world champions in the game of Go, according to previous research in [Silver et al.(2016)] and [Silver et al.(2017)]. NN may train with self-play from a random policy to a top player using Deep Reinforcement Learning (DRL). [Yao et al.(2005)] and [Pan et al.(2021)] also reported on the notion of utilizing the Artificial Neural Network (ANN) and DRL on FEM mesh generation. The Advantage Actor-Critic (A2C) approach, as shown in Figure 2, trains a policy and a value NN to provide solutions to diverse boundaries. After that, they perform behavior cloning by extracting experience (also known as data-set) and training another NN. With a self-evolving procedure, the supervised-learning-based NN *FreeMesh-S* can address new boundary challenges.

*Undergraduate Student, Theoretical and Applied Mechanics, Dynamics F1, 443 Huangshan Rd, Shushan District, Hefei, Anhui, China



(a) Structured mesh (b) Unstructured mesh

Figure 1: Two ways to mesh a given boundary. Indirect unstructured meshes have a transformation process while direct ones generate quadrilateral elements directly.

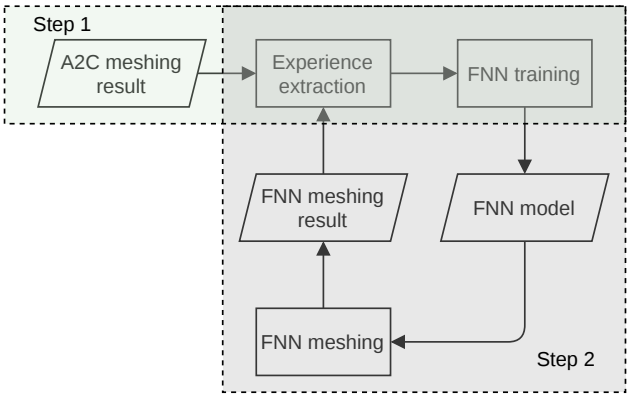


Figure 2: A previously proposed method to supervise an FNN with experience extracted from itself and another DRL program.

1.3 Our approach

According to [Hornik(1991)], standard multi-layer feed-forward networks with activation function may arbitrarily well approximate any continuous function if there are enough hidden units available. Therefore, once enough experience has been extracted from Reinforcement Learning (RL), we can construct an MLP or ResNet ([Shan et al.(2016)]) architecture to solve new boundaries. To improve the quality of data-set in *FreeMesh-S*, *TreeMesh*, our semi-open-source program makes many improvements on *FreeMesh-S*. We replace the A2C DRL algorithm with the more powerful DPPO algorithm and add a brute force MCTS to find nearly-optimal meshes. We use a new state tensor, an action tensor with a smaller range, and a sophisticated reward function to speed up convergence. On the whole, *TreeMesh* aims to use DRL and MCTS to discover an optimal, or nearly-optimal, solution to some problematic boundaries in a large-scale boundary (e.g. sharp angles, bottleneck region, unevenly distributed segments, and holes) with a pre-trained model or from scratch. Further technical improvements in DRL and MCTS algorithms and solving procedures are described in Section 2, mesh results are exhibited in Section 3. We have released a single-threaded version on GitHub.

2 Methodology

2.1 Reinforcement learning in *TreeMesh*

Our method continues to use the actor-critic framework, including a policy network $\pi(S_t, \theta)$ and a value network $V_\pi(S_t, w)$. Both NN use the same input tensor, which includes standardized node coordinates and mesh progression. The value network evaluates the mean return value, whereas the policy network outputs the coordinates for a new element. On one hand, we use the back propagation of the value loss function to assist it in providing increasingly precise evaluations after receiving trajectories. On the other hand, we calculate the time difference error and the probability of choosing each action to back propagate the policy loss function. We adopt linear layers and leaky ReLUs in the NN architecture.

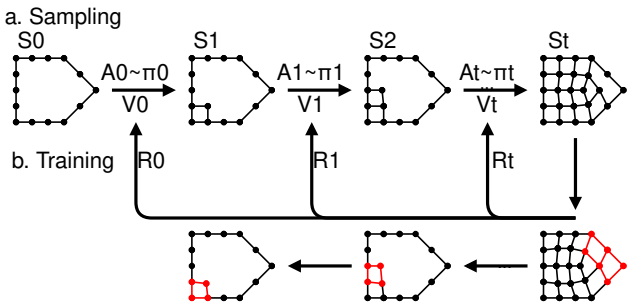


Figure 3: PPO workflow. **a.** The program is given a state and is asked to perform an action and predict cumulative rewards. A move is sampled using a multivariate normal distribution, with the action prediction serves as the mean value, a decay variance for exploration, and the raw reward serving as a criterion for the move's legality. **b.** Adaptive Laplacian Smoothing ([Xi et al.(2021)]) is used to optimize coordinates and recalculate each rewards. The loss functions are calculated using new rewards and the probability of sampling each moves to update parameters in NN.

NN in *FreeMesh-S* are trained with an actor-critic framework, which was introduced in [Konda and Tsitsiklis(2000)]. We use a distributed version of proximal policy optimization (DPPO), an steady off-policy algorithm based on actor-critic. According to [Schulman et al.(2017)], PPO has outperformed A2C, A2C+Trust Region, and CEM on seven MuJoCo environments, including HalfCheetah-v1, Hopper-v1, InvertedDoublePendulum-v1, InvertedPendulum-v1, Reacher-v1, Swimmer-v1, and Walker2d-v1. Each of N parallel actors collects T_i time steps of data in each episode. Then we calculate each trajectory, shuffle them, and back propagate mini-batches with Adam ([Kingma and Ba(2014)]), for K epochs. In Figure 3, a workflow of Algorithm 1 is shown. Lines 3 to 9 are referred to as the sampling section **a**, while lines 11 to 20 are referred to as the training section **b**.

Algorithm 1 Distributed Proximal Policy Optimization with clipping

1: **for** episode $i = 1, 2, \cdots, M$ **do**
2: **for** agent $j = 1, 2, \cdots, N$ **do**
3: Get an initial state S_1^{ij} (i and j are omitted hereafter);
4: **for** time step $t = 1, 2, \cdots, T$ **do**
5: Collect value network prediction $V_\pi(S_t, w)$
6: Sample action $A_t \sim \pi(|\ S_t, \theta)$;
7: Get the raw reward R_t^r and next state S_{t+1};
8: If done, break the loop;
9: **end for**
10: **end for**
11: Optimize coordinates with Adaptive Laplacian Smoothing ([Xi et al.(2021)]), and calculate optimized rewards R_t^o;
12: Calculate the TD error $\delta_t = \sum_{i=1}^{T-t-1} \lambda^i \left[R_t^o + V_\pi(S_{t+i+1}, w) - V_\pi(S_{t+i}, w)\right]$
13: Calculate cumulative rewards $U_t = \sum_{i=1}^{T} R_t^o$
14: **for** mini-batch number $k = 1, 2, \cdots, K$ **do** (k omitted hereafter)
15: **for** update epoch $l = 1, 2, \cdots, L$ **do**
16: Calculate policy loss $J_\pi = -\sum_t \min\left\{\frac{\pi_l^{\text{new}}(A_t|S_t,\theta)}{\pi^{\text{old}}(A_t|S_t,\theta)}\delta_t, \text{clip}\left[\frac{\pi_l^{\text{new}}(A_t|S_t,\theta)}{\pi^{\text{old}}(A_t|S_t,\theta)}, 1-\epsilon, 1+\epsilon\right]\delta_t\right\}$
17: Calculate value loss $J_V = \alpha \cdot \text{smooth\_L1\_loss}\left[V_\pi(S_t, w), U_t\right]$
18: Back propagate w, θ with $J_\pi + J_V$
19: **end for**
20: **end for**
21: **end for**

In compared to business software meshes and the mesh in [Pan et al.(2021)], the parameters w, θ we acquire after DPPO are strong enough to give a state-of-the-art mesh. Many investigations have revealed, however, that higher-quality meshes have higher variance, because the goal of reinforcement learning is for the agent to learn an optimal, or

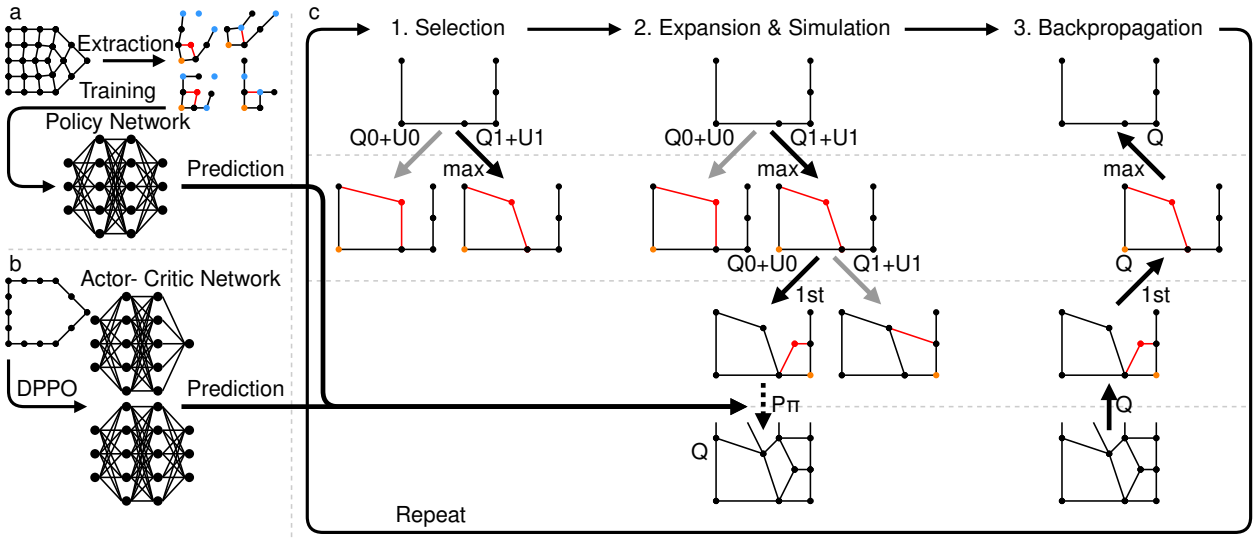


Figure 4: MCTS workflow 1. **a.** Behavior cloning is a quick and easy technique to create effective policies given limited data. **b.** We train an actor-critic network with the DPPO algorithm and store the policy model, either from scratch or from a pre-trained model. **c.** In Selection, each simulation traverses the tree by selecting the edge with the highest sum of an expected quality $\overline{Q}$ plus an Upper Confidence Bound (UCB) U based on a prior probability P and visit count N for that edge. In Expansion and Simulation, the leaf node is expanded and evaluated by the quality of one meshing to the end. In Back Propagation, quality Q is utilized to update all evaluations $\overline{Q}$ in the sub-tree under the current episode. Once the search is complete, search times and expected qualities are returned, and the user can choose the next move.

nearly-optimal, policy that maximizes the **expected** total reward, as described in [Russell and Norvig(2010)], the policy may never find the ideal solution with a high variance. Thus, we combine NN with an MCTS to search for coordinates surrounding projected action, resulting in more dependable findings.

2.2 Monte-Carlo tree search in *TreeMesh*

We use MCTS to perform a brute-force search to check if there are any better meshes. The prediction point will be the center of a 5×5 dot array with a unit side length of 0.1. As shown in Figure 4 and Figure 5, the MCTS uses the policy network $\pi(|\ S_t, \theta)$ to guide its simulations. Each edge in the search tree holds a prior probability $\pi'(A_t \mid S_t)$ with the same mean value as $\pi(A_t \mid S_t, \theta)$. Each episode starts at the root and finds the child node that maximizes an upper confidence bound before it reaches a leaf node:

$$\begin{aligned} a_t &= \arg\max_{A_t} \left[Q(S_t, A_t) + U(S_t, A_t)\right] \\ &= \arg\max_{A_t} \left[U_0 + c_{\mathrm{PUCT}} \pi'(A_t \mid S_t) \frac{\sqrt{N_t}}{1 + N_{t+1}}\right] \end{aligned} \tag{1}$$

c_{PUCT} is a hyper-parameter that determines the proportion of UCB to expected quality. U_0 is the cumulative reward from time step 0. If the leaf node has been visited, the algorithm will extend all possible moves and choose the first child node. Finally, the algorithm will run all the way to the finish, updating nodes along the way. The whole pseudo code is in Algorithm 2.

2.3 State and action setting

As shown in Figure 6 (a), the state tensor has 18 dimensions, including:

1. $\left(\frac{x_{P0}-x_{\min}}{x_{\max}-x_{\min}}, \frac{y_{P0}-y_{\min}}{y_{\max}-y_{\min}}\right)$
2. $\left(\frac{x_{Pr1}-x_{\min}}{x_{\max}-x_{\min}}, \frac{y_{Pr1}-y_{\min}}{y_{\max}-y_{\min}}\right)$
3. (x'_{Pr2}, y'_{Pr2})
4. (x'_{Pl1}, y'_{Pl1})

Algorithm 2 Monte-Carlo tree search

function UCBSEARCH(s_0)
  Create a root node n_0 with $N(n_0) = 0, Q(n_0) = 0, n$'s parent $\leftarrow$ NULL
  while Within computational budget and ($500 \nmid$ Episode i or confirm continue) **do**
    $n \leftarrow$SELECTION(n_0)
    $n, Q =$ROLLOUT(n)
    UPDATE(n, Q)
  end while
  return n_0
end function
function SELECTION(n)
  while n is non-terminal ($N(n) \leq 1$) **do**
    $n \leftarrow \underset{n' \in \text{ children of } n}{\arg\max} [Q(S_t, A_t) + U(S_t, A_t)]$
  end while
  if $N(n) = 1$ **then**
    return EXPAND(n)
  else
    $N(n) = 1$
    return n
  end if
end function
function EXPAND(n)
  Add $n' \in$ children of n to n's children list
  return n's 1st child
end function
function ROLLOUT(n)
  while not done **do**
    $n \leftarrow \text{uniform}(n'\text{s child})$
  end while
  return node n, quality Q
end function
function UPDATE(n, Q)
  while n is not NULL **do**
    $N(n) = N(n) + 1$
    $Q(n) = \frac{Q(n)[N(n)-1]+Q}{N(n)}$
    $n \leftarrow n$'s parent
  end while
end function

5. (x'_{Pl2}, y'_{Pl2})
6. $(x'_{P\theta 1}, y'_{P\theta 1})$
7. $(x'_{P\theta 2}, y'_{P\theta 2})$
8. $(x'_{P\theta 3}, y'_{P\theta 3})$
9. $\frac{m}{N}$
10. $\frac{n}{N}$

As demonstrated in Figure 6 (b), from 1 to 6 are coordinates of $Pr2, Pl1, Pl2, P\theta 1, P\theta 2, P\theta 3$, standardized by coordinates of $P0$ and $Pr1$ (by dragging $P0$ to O, and $Pr1$ to $(1, 0)$). The standardization procedure is as follows:

$$\begin{aligned}\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} &= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{(x_{P0}-x_{r1})^2+(y_{P0}-y_{r1})^2}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{(x_{P0}-x_{r1})^2+(y_{P0}-y_{r1})^2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{P0} \\ 0 & 1 & -y_{P0} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \\ &= E \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \end{aligned} \tag{2}$$

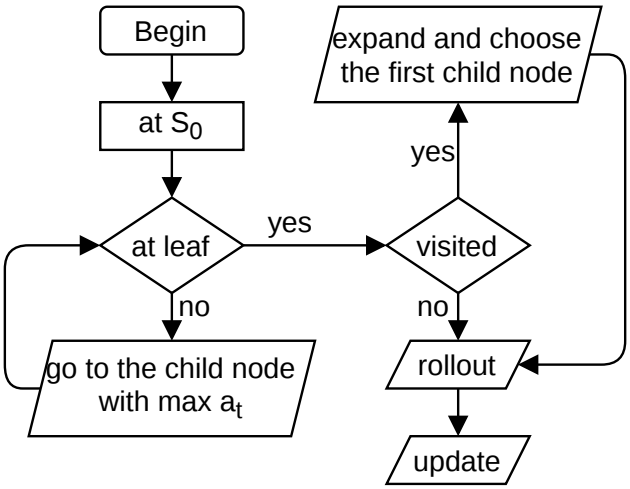
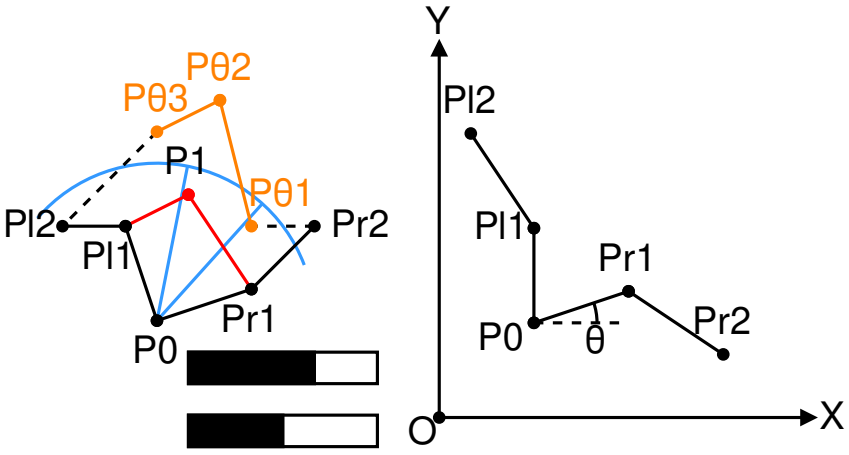


Figure 5: MCTS workflow 2. Notice that the first "no" corresponds to step 1 in Figure 4, from the first "yes" to "roll out" is step 2, and "update" relates to step 3.



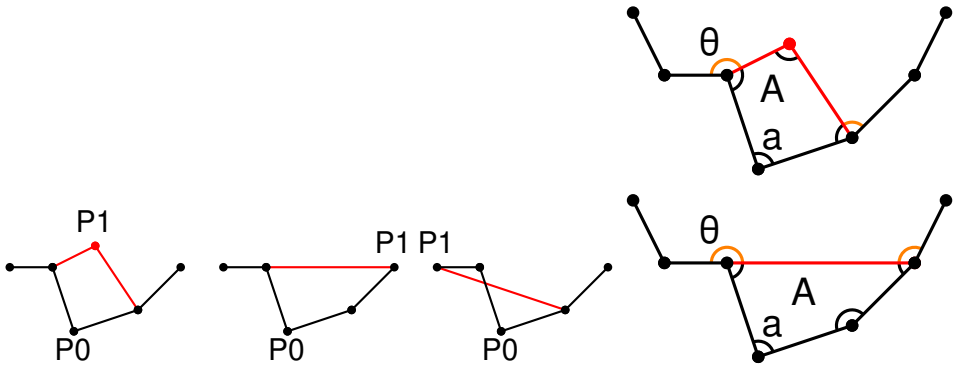
(a) State setting (b) Coordinate transformation

Figure 6: A state tensor with 18 dimensions and a sketch showing the coordinate transformation. The tensor includes 12 standardized coordinates, 4 re-scaled coordinates, and 2 progress indicators to aid in the evaluation of the value network. 3 steps are required to standardize coordinates: drag $P0$ to O, rotate $P0Pr1$ to x-axis, and re-scale $Pr1$ to $(1, 0)$.

There are a few notations that need to be clarified. From 1 to 6, Pli and Prj indicate the ith point on the left hand (clockwise) and the jth point on the right hand (anti-clockwise) of $P0$, respectively. These points are particularly relevant to the selection of point $P1$. All points on the current boundary except $P0, Pl1, Pr1$ are traversed to find $P\theta 1, 2, 3$. They are each in one of the three trisections of $\angle Pr1P0Pl1$ and are close enough to point $P0$. (a composite function is used to sort out the three points). To speed up convergence, the three points are clipped to a circle with a center $P0$ and a radius of 3 (in the transformed coordinate system).

7 and 8 are coordinates of $P0$ and $Pr1$, re-scaled by coordinates of the top right and bottom left corner points on the original boundary.

9 and 10 indicates the progress of meshing. m is the number of sides of the remaining polygon, n is the number of meshed quadrilaterals, and N is the maximum step permitted in meshing. The actor-critic network outputs a value and an action. The action is a 2-dimensional tensor of the next point (x', y'), and the value $V_\pi(S_t, w) = \mathbb{E}(U_t \mid S_t)$ is an expectation of cumulative rewards. Because the minimum angle for all boundaries is smaller than π, both coordinates are constrained in $[0, 1.5]$ with a hyperbolic tangent function, the new point is most likely in the first quadrant. An inverse matrix E^{-1} will be used to transform the new point back to the previous coordinate system.



(a) Three possible moves

(b) Variables considered in judgement

Figure 7: At any given time step, there are three possible moves. All of the moves generate a new quadrilateral, with the exception of the first, which has no effect on the number of edges, and the other two, which reduce the number of edges by two. As a result, the starting boundary should have an even number of nodes.

2.4 Reward setting

The reward function is a critical signal for the NN since it allows the NN to determine whether a movement is positive or harmful. We propose an improved reward function upon [Pan et al.(2021)]. This option delivers superior convergence performance, according to empirical analysis. Noticing that on Algorithm 1 line 7 and line 11, we calculate a raw reward R_t^r and an optimized reward R_t^o. The raw reward, with a range $\{-5\} \cup [0, 1]$, is purely for judging a movement in Figure 7 (a), or give a punishment reward (e.g. in Figure 7 (a), the third move is illegal, the other two moves are legal):

$$R_t^r = \begin{cases} -5, \text{illegal} \\ \sqrt[4]{\prod_{i=1}^{4} \left[\frac{l_i}{\sqrt{A_t}}^{\operatorname{sgn}\left(\sqrt{A_t} - l_i\right)} \right] \left(1 - \frac{|\, a_j - \frac{\pi}{2} \,|}{\frac{\pi}{2}} \right)}, \text{legal} \end{cases} \tag{3}$$

At time step t, the three raw rewards will be multiplied by their respective factor $\prod_{i=1}^{2} \left[\frac{\min\left(\theta_k, \frac{\pi}{3}\right)}{\frac{\pi}{3}} \right]$, and the highest-valued move will be chosen. It's worth noticing that we can also allow the NN to decide which action to take, but this will make convergence more difficult.

On line 11, we have collected a trajectory $R_0^r, R_1^r, \cdots, R_T^r$. Assume there are a total of T_0 potential steps (with the remaining $T - T_0$ rewards being -5). Then, using Adaptive Laplacian Smoothing ([Xi et al.(2021)]), we optimize all internal coordinates and update every effective reward to $R_t^o = \frac{R_t^{r'} \eta}{T_0}$. η is a hyper-parameter with a value of 500 that controls the cumulative rewards in the $[100, 1000]$ range. The penalty for illegal rewards remains at -5.

3 Result

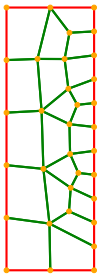
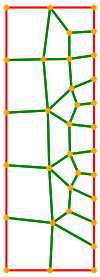
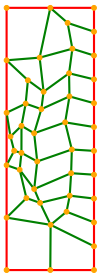
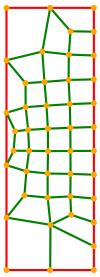
3.1 Some DPPO results

Here, we exhibit our work on two boundaries: a "$11 \rightarrow 5$" seed-density-changing rectangle, and a pentagonal plate that was studied in [Pan et al.(2021)].

Table 1 compares *TreeMesh* (DPPO algorithm used only) with ABAQUS on a "$11 \rightarrow 5$" seed-density-changing boundary. Equation 3 is used to compute the mean quality of the optimized mesh. We outperform ABAQUS in terms of quality by 13.8573%.

Table 2 compares *TreeMesh* (DPPO algorithm used only) with ABAQUS and *FreeMesh-S* on a pentagonal boundary. We outperform ABAQUS in terms of quality by 5.3204%, and *FreeMesh-S* by 1.4822% in quality, and by around 90% in time.

Table 1: A "11 $\rightarrow$ 5" seed-density-changing boundary with 4 MLP layers.

Program	Raw Mesh	Optimized Mesh	Quality	Quantity	Time
ABAQUS			0.7036	22	$\approx 1s$
TreeMesh			0.8011	34	$\approx 10^3 s$

3.2 A DPPO+MCTS time line

A more detailed experiment is conducted on the first boundary (Table 1), with a 3-residual-block architecture. In 1500s, we output the raw mesh created by the DPPO algorithm. Weight files at three time points are rendered to execute a brute force MCTS. At this boundary, a better mesh is discovered and shown in Figure 8's upper right corner. It has been appraised as a state-of-the-art mesh by some engineers.

3.3 Empirical Analysis of DPPO training reward in Table 1 and Table 2

We continued the training for approximately 4 hours. Figure 9 shows the smoothed training reward on both boundaries. The co-variance matrix of the multivariate normal distribution starts at $0.05I$ and decays at a constant rate of 0.999 per 100 episodes during the training. This slows convergence slightly but allows the policy NN to explore more policies. The policy NN will eventually stop at a nearly-optimal policy if the initial learning rate is properly set ($\sim 10^{-6}$) and decays at the same rate. Here, however, we remain the learning rate constant (10^{-6}) to accelerate convergence.

References

[Coulom(2006)] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[Kocsis and Szepesvári(2006)] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[Browne et al.(2012)] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[Pan et al.(2021)] Jie Pan, Jingwei Huang, Yunli Wang, Gengdong Cheng, and Yong Zeng. A self-learning finite element extraction system based on reinforcement learning. *AI EDAM*, pages 1–29, 2021.

[Thompson et al.(1998)] Joe F Thompson, Bharat K Soni, and Nigel P Weatherill. *Handbook of grid generation*. CRC press, 1998.

[Owen(1998)] Steven J. Owen. A survey of unstructured mesh generation technology. In *INTERNATIONAL MESHING ROUNDTABLE*, pages 239–267, 1998.

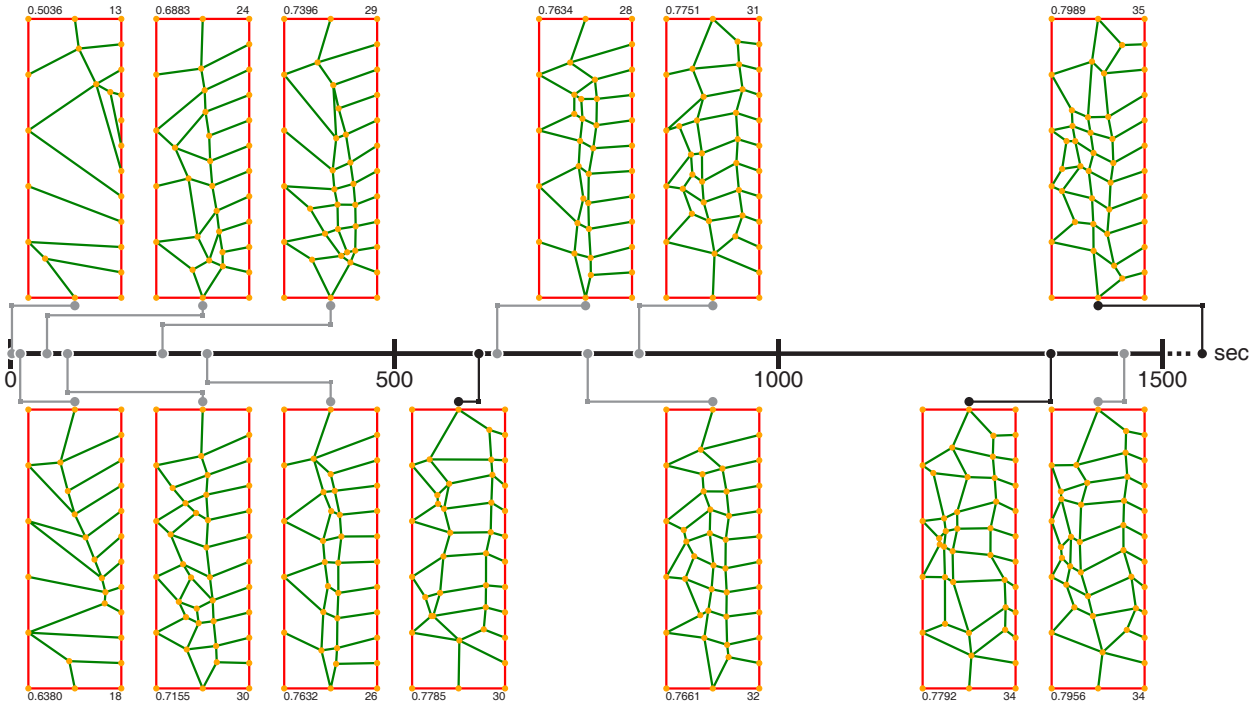
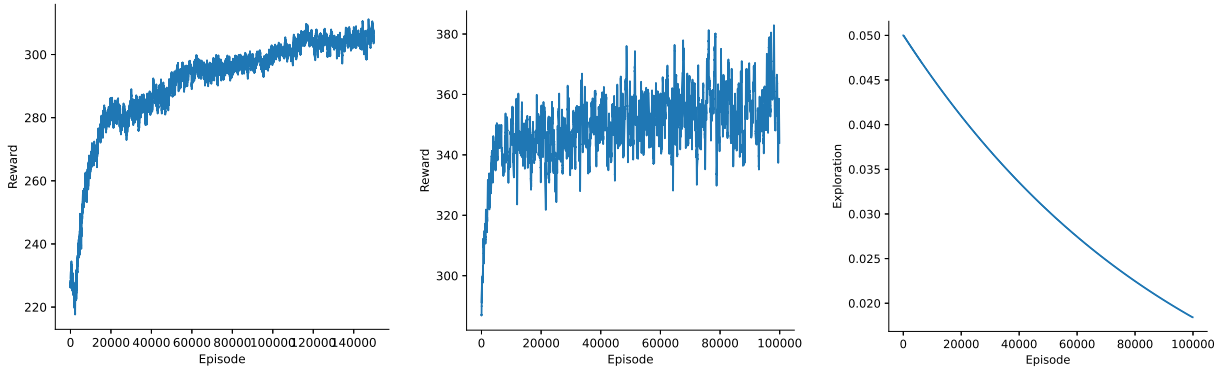


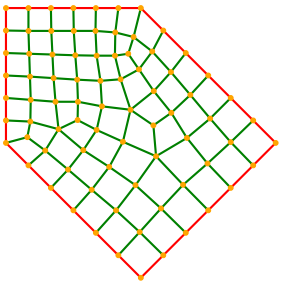
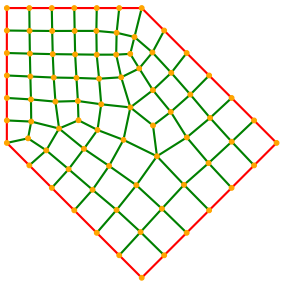
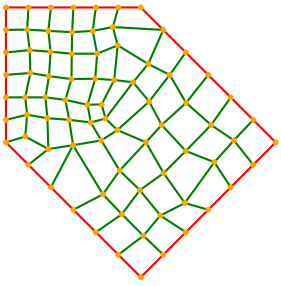
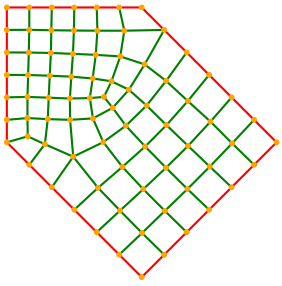
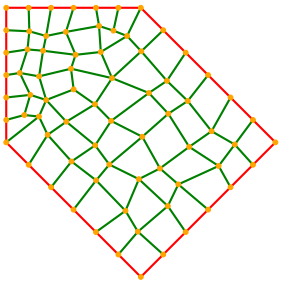
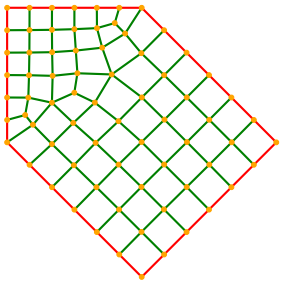
Figure 8: Mesh methods with 3 residual blocks and the MCTS method in a "11 $\rightarrow$ 5" seed-density-changing boundary. DPPO results are linked with grey lines, while MCTS results are linked with black lines.



(a) Seed-density-changing rectangle (b) Pentagonal plate (c) Exploration decay

Figure 9: Smoothed training reward on two boundaries after 150k episodes and 100k episodes respectively, and exploration decay. The improvement on training reward is a joint effect of both a better policy and a decaying exploration.

Table 2: A pentagonal boundary with 4 MLP layers.

Program	Raw Mesh	Optimized Mesh	Quality	Quantity	Time
ABAQUS			0.8646	64	$\approx 1s$
FreeMesh-S			0.8973	63	$\approx 10^4 s$
TreeMesh			0.9106	57	$\approx 10^3 s$

[Garimella et al.(2004)] Rao V Garimella, Mikhail J Shashkov, and Patrick M Knupp. Triangular and quadrilateral surface mesh quality optimization using local parametrization. *Computer Methods in Applied Mechanics and Engineering*, 193(9-11):913–928, 2004.

[Zeng and Cheng(1993)] Young Zeng and Gengdong Cheng. Knowledge-based free mesh generation of quadrilateral elements in two-dimensional domains. *Computer-Aided Civil and Infrastructure Engineering*, 8(4):259–270, 1993.

[Docampo-Sanchez and Haimes(2019)] Julia Docampo-Sanchez and Robert Haimes. Towards fully regular quad mesh generation. In *AIAA Scitech 2019 Forum*, page 1988, 2019.

[Silver et al.(2016)] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[Silver et al.(2017)] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[Yao et al.(2005)] S Yao, B Yan, B Chen, and Yong Zeng. An ann-based element extraction method for automatic mesh generation. *Expert Systems with Applications*, 29(1):193–206, 2005.

[Hornik(1991)] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2): 251–257, 1991.

[Shan et al.(2016)] Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 255–262, 2016.

[Konda and Tsitsiklis(2000)] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[Schulman et al.(2017)] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Kingma and Ba(2014)] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Xi et al.(2021)] Ning Xi, Yinjie Sun, Lei Xiao, and Gang Mei. Designing parallel adaptive laplacian smoothing for improving tetrahedral mesh quality on the gpu. *Applied Sciences*, 11(12):5543, 2021.

[Russell and Norvig(2010)] Stuart Russell and Peter Norvig.
Artificial Intelligence: A Modern Approach. Prentice Hall, 3 edition, 2010.