Computer-Aided Design 175 (2024) 103744

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Research Paper

Learning Topological Operations on Meshes with Application to Block Decomposition of Polygons

A. Narayanan [a,c,1,*], Y. Pan [b,c,2], P.-O. Persson [b,c,3]

[a] Department of Mechanical Engineering, University of California, Berkeley, CA 94709, United States
[b] Department of Mathematics, University of California, Berkeley, CA 94720, United States
[c] Mathematics Group, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, United States

ARTICLE INFO

Keywords:
Mesh generation
Reinforcement learning
Block decompositions

ABSTRACT

We present a learning based framework for mesh quality improvement on unstructured triangular and quadrilateral meshes. Our model learns to improve mesh quality according to a prescribed objective function purely via self-play reinforcement learning with no prior heuristics. The actions performed on the mesh are standard local and global element operations. The goal is to minimize the deviation of the node degrees from their ideal values, which in the case of interior vertices leads to a minimization of irregular nodes.

1. Introduction

Mesh generation is a crucial part of many applications, including the numerical simulation of partial differential equations as well as computer animation and visualization. While it can be discussed exactly what makes a mesh appropriate for a given situation, it is widely accepted that fewer number of irregular nodes lead to better quality meshes. Therefore, many mesh generation and mesh improvement methods have been proposed that aim to maximize the regularity of the mesh, in particular in the case of quadrilateral elements.

For triangular meshes, some of the most popular algorithms are the Delaunay refinement method [1] and the advancing front method [2]. The resulting meshes might be improved by local operations or smoothing, although typically based on element qualities rather than the regularity of the connectivities. Some quadrilateral mesh generators are also based on a direct approach, such as the paving method [3], but most use an indirect approach of creating quadrilateral elements from a triangular mesh. These methods include the popular Q-Morph method [4], element matching methods such as the Blossom-Quad method [5], and so-called regularization or mesh simplification methods which improve an initial mesh using various mesh modification techniques [6–9].

Although many of these mesh modification methods produce impressive results, we note that the algorithms for how they apply the various mesh operations are usually highly heuristic in nature [8,10]. This is expected, since finding an optimal strategy is a complex discrete optimization problem. Indeed, Canann et al. [11] highlight the need to "recognize and process patterns of irregularities that occur over larger groups of elements" in order to achieve meshes with desirable connectivity and they demonstrate the efficacy of complex mesh edits by composing sequences of multiple local operations. Developing such heuristics is challenging and laborious, and it is difficult to adequately explore the state space and action space. Identifying optimal sequences of mesh editing operations that improve mesh quality is not trivial and is highly dependent on the type of mesh being considered. For instance, the heuristics that optimize triangular meshes are different from the heuristics that optimize quadrilateral meshes. Further, traditional optimization methods such as mixed integer programming are computationally expensive and the computational cost grows exponentially with problem size making it challenging to adapt these methods to meshes of different size.

To address the above challenges, we explore deep reinforcement learning as a solution approach for this optimization problem. A major advantage of such an approach is that optimal sequences of actions can be discovered by the reinforcement learning agent purely by interacting with a mesh environment through self-play. This circumvents the requirement for human crafted heuristics. The agent can adapt to different mesh types by training it on an appropriate mesh environment. This provides a unified approach that can be applied to different mesh types. We formulate our problem in the language of reinforcement learning (RL) [12] wherein actions available to the agent are local mesh edit

* Corresponding author at: Department of Mechanical Engineering, University of California, Berkeley, CA 94709, United States.
E-mail addresses: arjun.narayanan@berkeley.edu (A. Narayanan), yllpan@berkeley.edu (Y. Pan), persson@berkeley.edu (P.-O. Persson).
[1] Graduate student, Department of Mechanical Engineering, University of California, Berkeley.
[2] Graduate student, Department of Mathematics, University of California, Berkeley.
[3] Professor, Department of Mathematics, University of California, Berkeley.

https://doi.org/10.1016/j.cad.2024.103744
Received 11 September 2023; Received in revised form 27 November 2023; Accepted 5 June 2024
Available online 12 June 2024

operations and the rewards are the improvement of mesh regularity as measured by a prescribed objective function. By converting our optimization problem into a sequential decision process consisting of local mesh editing operations, we avoid the exponential cost of global solution strategies like mixed integer programming.

In this work, we consider the case of planar straight-sided polygonal geometries. However, since our method is based purely on mesh connectivity, it may be applied to geometries with curved boundaries as well so long as the regularity of vertices on these boundaries is specified. We generate a coarse initial triangular mesh using the Delaunay refinement algorithm. In the case of quadrilateral meshes, we perform Catmull–Clark splits of the triangles, and we also introduce global mesh operations. One of these is the clean-up, which aims to reduce the total number of elements which is suitable for generation of block decompositions.

A key component of our framework is the employment of the half-edge data structure, which in particular allows us to define a convolutional operation on unstructured meshes. A neural network is trained to produce a probability distribution for the various actions on local neighborhoods of the mesh, i.e., a policy. The policy is sampled to determine the next operation to perform. A powerful property of our method is that it generalizes to both triangular and quadrilateral meshes with minimal modifications to account for the different actions available on these meshes. We limit action selection to local mesh neighborhoods, allowing the learned policy to generalize well to a variety of mesh types and sizes that were not present in the training data. We demonstrate our methods on several polygonal shapes, where we consistently obtain meshes with optimal regularity. Extension of this method to arbitrary polygonal elements is reserved for future work.

Machine learning has been applied to numerous mesh generation problems before. Pointer networks [13] have been used to generate convex hulls and Delaunay triangulations. Deep RL has been used to learn quadrilateral element extraction rules for mesh generation [14,15]. RL has also been employed to learn adaptive mesh refinement strategies [16,17]. In [18], RL was used to perform block decomposition of planar, straight-sided, axis aligned shapes using axis aligned cuts.

Our work differs from prior work in several key ways. Our objective function is purely based on the connectivity of the mesh and our framework aims to minimize the number of irregular vertices. We consider local topological edit operations as our action space. Our novel convolution operation on the half-edge data-structure provides a powerful, parameterized way of constructing state representations that encode neighborhood connectivity relationships. We employ a local neighborhood selection technique that allows us to generalize to different mesh sizes. These key features enable our method to work on both triangular and quadrilateral meshes of various sizes.

The half-edge data-structure is able to represent arbitrary polygonal shapes in 2D. Thus our state representation method naturally extends to all such polygonal shapes. Our reinforcement learning framework can be applied to these shapes so long as an appropriate action space is defined. We hypothesize that the technique can be extended to 3-dimensions by leveraging the equivalent of the half-edge data-structure in higher dimensions [19,20]. Prior work has explored the action space in 3D e.g. tetrahedral [21–23] and hexahedral meshes [24,25].

2. Problem statement

In the present work we are interested in optimizing the connectivity of triangular and quadrilateral meshes. The overall objective is to produce meshes where all the vertices have a specific number of incident edges. We refer to this as the *desired degree* of a vertex. A vertex whose degree is the same as the desired degree is called *regular*. A vertex whose degree is different from the desired degree is called *irregular*, with the difference between the degree and the desired degree being a measure of the *irregularity* of the vertex. Our framework allows the user to specify the desired degree on all vertices. The user is allowed to specify the desired degree of any newly introduced vertex.

While there exist robust algorithms for triangular and quadrilateral meshing such as Delaunay triangulation and paving, these algorithms are not designed to produce meshes with a specific connectivity structure. A common approach is to use these algorithms as a starting point and improve the connectivity of the mesh through various topological mesh editing operations [26]. We adopt this approach and frame our problem as a Markov Decision Process.

2.1. Objective function

Consider a mesh with N_v vertices. Let vertex i have degree d_i and desired degree d_i^*. Then its irregularity is $\Delta_i = d_i - d_i^*$. We compute a global score s as the L1 norm of s, which is a measure of the total irregularity in the mesh.

$$s = \sum_{i=1}^{N_v} |\Delta_i| \tag{1}$$

Clearly, a mesh with all regular vertices will have a score $s = 0$.

2.1.1. Heuristics to determine desired degree

Our heuristic for triangular (quadrilateral) meshes is based on achieving an interior angle of 60° (90°) in all elements. The desired degree of any vertex in the interior is 6 (4). The desired degree of a boundary vertex is chosen such that the average included angle in all elements incident on that boundary vertex is approximately 60° (90°) . The desired degree according to this heuristic can be expressed as,

$$d^* = \begin{cases} 360/\alpha & \text{interior vertex} \\ \max\left(\lfloor \theta/\alpha \rceil + 1, 2\right) & \text{boundary vertex} \end{cases} \tag{2}$$

where $\lfloor \cdot \rceil$ is the round to nearest integer operator, θ is the angle of the boundary at the vertex in question, and α is 60° (90°) for triangles (quadrilaterals). We observed that rounding to the nearest integer resulted in better performing models than using d^* as a continuous value on the boundary. According to this heuristic, the desired degree of a new vertex introduced on the boundary is set to 4 (3) since we assume that the edge on which the new vertex is introduced is a straight edge.

2.2. Topological operations on meshes

We define the following local operations on triangular meshes. See figure Fig. 1 for an illustration.

- **Edge Flip:** An interior edge in a triangular mesh can be deleted and the resultant quadrilateral can be re-triangulated across its other diagonal. This can be seen as "flipping" an edge between two possible states.
- **Edge Split:** Any edge in a triangular mesh can be split by inserting a new vertex on the edge and connecting it to the opposite vertices in the adjacent triangles.
- **Edge Collapse:** An interior edge in a triangular mesh can be collapsed resulting in the deletion of the two triangles associated with this edge.

Similarly, we define the following local operations on quadrilateral meshes. See figure Fig. 2 for an illustration.

- **Edge Flip:** An interior edge in a quadrilateral mesh can be deleted, and the resultant hexagon can be quad-meshed in two new ways. This can be seen as "flipping" an edge clockwise or counter-clockwise.
- **Vertex Split:** A vertex in a quad mesh can be split along an interior edge incident at that vertex. This results in the insertion of a new vertex and a new element into the mesh.

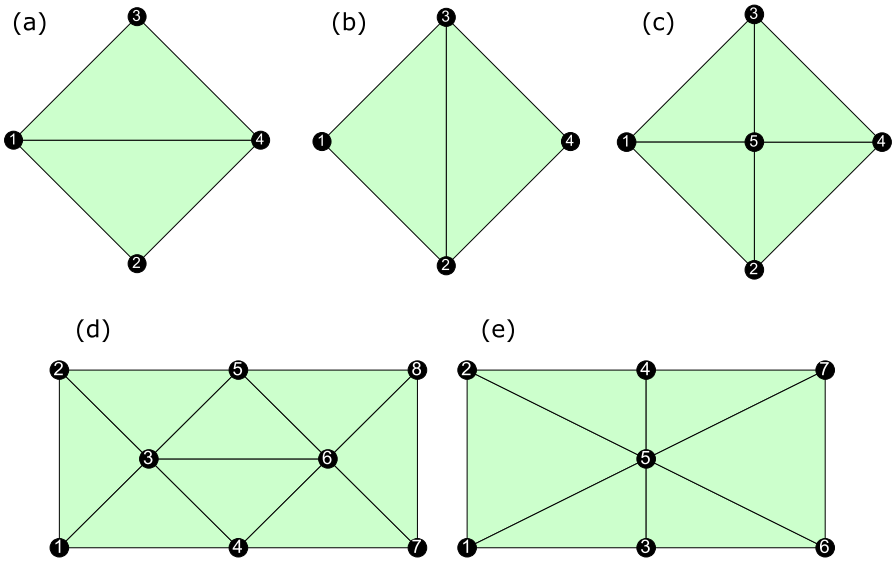


Fig. 1. Configuration (a) and (b) are related by an edge flip. Configuration (c) can be produced by splitting the interior edge in either (a) or (b). Collapsing the edge between vertex 3–6 in (d) produces (e).

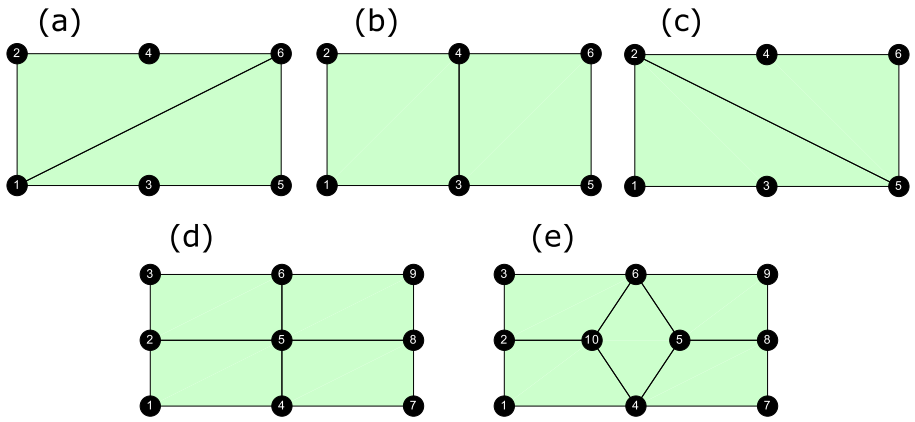


Fig. 2. Configuration (a) and (c) can be obtained from (b) via an edge flip. Configuration (e) is obtained from (d) via a vertex split, and the operation can be reversed via an element collapse.

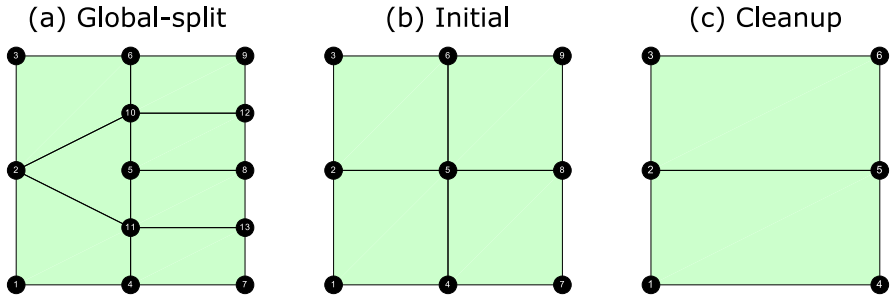


Fig. 3. Performing a global split on the edge between vertices 2 and 5 in the initial mesh (b) produces the mesh in (a). Alternatively, the sequence of edges between vertices 4–5–6 in the initial mesh (b) can be deleted by merging the neighboring elements, resulting in configuration (c).

- **Element Collapse:** A quadrilateral element can be collapsed along either diagonal by merging the two opposite vertices. The collapse operation can be seen as the inverse of the split operation defined above.

For quadrilateral meshes we also define the following global mesh editing operations. They are global in the sense that they can affect the topology of the mesh far away from where they are applied. See figure Fig. 3 for an illustration.

- **Global Split:** This operation splits an edge by inserting a quadrilateral element and introducing vertices on the edges in the two adjacent quadrilateral elements. The introduced vertices are hanging vertices — therefore we recover an all-quadrilateral mesh by propagating edges from the hanging vertices and sequentially splitting elements until the split terminates on a boundary.
- **Global Cleanup:** In some situations, global lines – which represent a sequence of edges – can be deleted by merging adjacent

elements. The global line either terminates on the boundaries of the mesh or forms a closed loop. We currently handle the situation where the global line terminates on the boundaries. (For meshes representing closed surfaces it would be important to consider the case of closed loops.) This operation results in the deletion of a sequence of vertices and elements. Vertices are distinguished into geometric and non-geometric vertices. Geometric vertices are those vertices which are integral in defining the geometry — these vertices cannot be deleted. The conditions under which we can perform this cleanup operation are (a) the end-points are on the boundary, are non-geometric, and have degree 3, and (b) all interior vertices are non-geometric and have degree 4. The cleanup operation is a powerful operation since it simplifies the problem and brings irregular vertices closer together. This strategy is particularly relevant for block decomposition of polygonal shapes.

3. Mesh representation and operations

3.1. The half-edge data structure

We employ the doubly-connected edge list (DCEL), also known as the half-edge data-structure, to represent our meshes. The advantage of the DCEL is that (a) it enables efficient implementations of the mesh editing operations described in Section 2.2, and (b) we utilize fundamental DCEL operations to represent the local topology in a given mesh region which is important to determine the appropriate action to be applied. The DCEL can be used to represent any planar, manifold mesh and as such allows our method to work on all such meshes. Extensions to the DCEL have been developed for non-manifold meshes and 3D volumetric meshes [19,20].

Briefly, the DCEL exploits the fact that each mesh edge is shared by exactly two mesh elements (except on the boundary). The DCEL represents each mesh edge as a pair of oriented half-edges pointing in opposite directions. Each half-edge contains a pointer to the counter-clockwise *next* half edge in the same element, and a pointer to the *twin* half-edge in the adjacent element. Each element contains a pointer to one of its half-edges (chosen arbitrarily) which induces an ordering on the half-edges in an element. Elements can be ordered by their global index in the mesh — this induces a global ordering on half-edges in the mesh. Each half-edge may be associated with a unique vertex. For triangles, we associate each half-edge with its opposite vertex. For quadrilaterals, we associate each half-edge with the vertex at its origin. The exact choice of the association does not matter as long as it is consistent. See Fig. 4 for an illustration. Further details about the DCEL can be found in a standard resource on computational geometry, for example [27].

3.2. Algorithmic complexity and parametrization of mesh editing operations

All of the local editing operations defined in Section 2.2 for triangles and quadrilaterals can be executed using the DCEL in constant time (assuming an upper bound on the maximum degree of a vertex). This is a powerful advantage offered by the DCEL compared to other mesh representations. For instance, when flipping a particular half-edge it is important to know which are the two neighboring elements across that edge — this is readily available in the DCEL.

The global operations defined for quadrilateral meshes in Section 2.2 requires connectivity editing operations that can propagate through several elements of the mesh before terminating. The algorithm for these operations scales linearly with the size of the mesh. In particular we disallow situations where global splits may result in the formation of loops or that do not terminate in a fixed number of iterations proportional to the size of the mesh. For the cleanup, we observe that every half-edge either lies on a cleanup path or does not. Performing a cleanup on a path does not affect the ability to cleanup other paths. Therefore all cleanups possible in a mesh can be performed by visiting every half-edge exactly once.

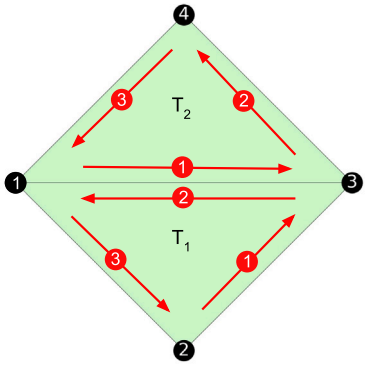


Fig. 4. Representing two triangular elements using the DCEL. The half-edges in each element are shown as red arrows. Each half-edge contains a pointer to the counter-clockwise *next* half-edge in the same element e.g. in triangle T_2, half-edge 2's *next* pointer points to half-edge 3. Half-edges in the interior of the mesh have a *twin* pointer to the half-edge in the adjacent element e.g. the *twin* of half-edge 1 in triangle T_2 is half-edge 2 in triangle T_1. We additionally associate each half-edge with a unique vertex in the element. For triangles we associate the vertex opposite a given half-edge e.g. half-edge 1 in triangle T_2 is associated with vertex 4. For quadrilaterals we associate the vertex at the origin of the half-edge. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Our framework optimizes a policy to perform sequences of mesh editing operations to achieve a given objective. All operations other than the global-cleanup are valid operations that can be learned by the policy. Whenever a global-cleanup is valid, it is always performed. We choose to do this because the cleanup simplifies the problem size and brings irregular vertices together making it easier to improve the connectivity of the mesh. A cleanup only deletes regular vertices according to our heuristic and never introduces any new irregular vertices in the mesh. Further, the cleanup is very useful in performing block decompositions of polygons.

We parametrise all the mesh editing operations in terms of half-edges. In a given mesh, specifying a particular half-edge and a particular type of edit determines an operation on the mesh. We have 3 operations per half-edge in the case of triangular meshes — flip, split, and collapse. Further, we have 5 operations per half-edge in the case of quadrilateral meshes — right-flip, left-flip, split, collapse, and global-split. There is some redundancy in this representation of actions on the mesh. For instance, flipping a half-edge and its twin are equivalent operations. We choose to retain this redundancy because (a) it fits in well with our half-edge framework, (b) the size of the state representation is larger only by a constant factor, and (c) it exposes the symmetries in the half-edge representation and may be seen as data augmentation in our state representation leading to more robust learning. Further, some actions – like the quadrilateral split – are not equivalent when performed on a half-edge and its twin.

4. Formulation as a reinforcement learning problem

4.1. Constructing the reward function

Clearly, a mesh with all regular vertices will have a score $s = 0$. Under the assumption of the heuristic described in Section 2.1.1, all the topological edit operations described in Section 2.2 are *zero-sum* leaving the quantity $s^* = |\sum_i \Delta_i|$ invariant for a given mesh. This does not hold true if we change the heuristic for the desired degree of newly introduced vertices from what we described in Section 2.1.1. If a mesh contains irregular vertices all of the same sign then its global score

Eq. (1) cannot be improved. s^* provides a lower bound on the score s,

$$s^* = \left| \sum_i^{N_v} \Delta_i \right| \leq \sum_i^{N_v} |\Delta_i| = s \tag{3}$$

We call s^* the *optimum score*. It is not clear if a score s^* can always be attained for a given mesh, however it serves as a useful measure of performance. The goal of our reinforcement learning framework is to learn sequences of actions that minimize s for a given mesh. In particular, consider a mesh M_t with score s_t at some time t. We now perform a mesh editing operation a_t on it to obtain mesh M_{t+1} with score s_{t+1}. Our agent is trained with reward r_t,

$$r_t = s_t - s_{t+1} \tag{4}$$

An agent starting with an initial mesh M_1 transformed through a sequence of n operations $a_1, a_2, \ldots, a_n$ collects reward $r_1, r_2, \ldots, r_n$. We consider the discounted return from state M_t as,

$$G_t = \sum_{k=t}^{n} \gamma^{k-t} r_k \tag{5}$$

with discount factor γ. (We use $\gamma = 1$ in all of our experiments.) Observe that the maximum possible return from this state is $G^* = s_t - s^*$. Thus, we consider the normalized return $\overline{G_t}$ as the *advantage* function to train our reinforcement learning agent,

$$\overline{G_t} = \frac{G_t}{s_t - s^*} \tag{6}$$

The return Eq. (5) collected on meshes of different sizes will be different simply because larger meshes tend to have more irregularities. By normalizing the return in Eq. (6), we ensure that actions are appropriately weighted during policy optimization. The mesh environment terminates when the mesh score $s_t = s^*$ or when a given number of mesh editing steps have been taken. We choose the maximum number of steps to be proportional to the number of mesh elements in the initial mesh.

While our current experiments are based on the objective described above, one could consider several modifications. Depending on the application, irregularities on the boundary may be more or less desirable than irregularities in the interior of the domain. The objective function can capture this difference in preference by weighting the contribution of boundary vertices and interior vertices differently when computing the score s. We could also consider improvement in element quality in the objective function. However, this would require a consideration of geometry along with topology and is reserved for future work.

4.2. Convolution operation on the DCEL data-structure

All of the actions, apart from the global-cleanup, affect the topology of the mesh locally. In order to determine if an action produces desirable outcomes in a particular neighborhood of the mesh, we need to understand the topology of this neighborhood. We require a representation of the local topology around each half-edge in order to select a suitable operation. In the language of reinforcement learning, this representation of the local topology is the *state* of a half-edge. The connectivity information in the immediate neighborhood of a half-edge is most relevant to determine the appropriate action to take in this neighborhood. We present here a convolution operation on the DCEL data-structure that encodes topological information around every half-edge. Indeed, this operation may be interpreted as a convolution on the graph induced by the half-edge connectivity. Iterative application of this convolution encodes topological information in a growing field-of-view around every half-edge. Further, this convolution operations can be efficiently implemented on modern GPU hardware.

Determining the appropriate action to take on a given half edge requires us to inspect the degree and irregularity of vertices in a neighborhood around the half-edge. Since the meshes we consider are unstructured, it is not immediately obvious which vertices to consider and in what order to consider them in. Our key observation is that the fundamental DCEL operations can be leveraged to construct a state representation for each half-edge that has a specific ordering. Our convolution operation requires two fundamental pieces of information both of which are easily available from the DCEL. For each half-edge we need to know the indices of (a) all the cyclic-next half-edges from the given element, and (b) the twin half-edges from the neighboring element. (a) is easily achieved by using the `next` operation repeatedly — 3 for triangles and 4 for quadrilaterals. (b) is fundamentally part of the DCEL data-structure.

As described in Section 3.1, there is a natural global ordering for all the half-edges in the mesh. Half-edges from the same element appear sequentially in this global ordering. If the half-edges are stored in this order, the cycle operation can be implemented efficiently as a sequence of matrix `reshape` operations which are provided by most array based programming languages. Consider a mesh with N_h half-edges with the state of each half-edge represented by an N_f dimensional vector. This data when stored in sequential order can be represented by a matrix $x \in \mathbb{R}^{N_f \times N_h}$. Algorithm 1 describes the cycle operation applied to this state matrix for triangular meshes. The extension to quadrilateral meshes or other polygonal meshes is straightforward. (We assume that n-dimensional arrays are stored in column-major order. We adopt a syntax that closely follows the Julia/MATLAB Programming Language.)

Algorithm 1 Cycle operation on triangular meshes

Input $\mathtt{x} \in \mathbb{R}^{N_f \times N_h}$
Output $\mathtt{y} \in \mathbb{R}^{3N_f \times N_h}$

```
x ← reshape(x, N_f, 3, :)
x1 ← reshape(x, 3N_f, 1, :)
x2 ← reshape(x[:, [2, 3, 1], :], 3N_f, 1, :)
x3 ← reshape(x[:, [3, 1, 2], :], 3N_f, 1, :)
y ← concatenate x1, x2, and x3 along the second
dimension (i.e. columns)
y ← reshape(y, 3N_f, :)
```

Information from twin half-edges is easily obtained by selecting the appropriate columns from the feature matrix. We use a learnable vector as the twin feature for edges on the boundary. This vector is part of the agent's parameter space and may be used by the agent to represent a useful signal indicating the boundary of the geometry. The same vector is used as the twin feature of all half-edges on the boundary. Our basic convolution operation involves cycling the current feature matrix, obtaining the features from the twin half-edges, and concatenating all of the features together. The resultant matrix is processed by a linear layer, followed by normalization and a non-linear activation function. We use LeakyReLU as our activation function. We refer to this operation as a *DCEL convolution block*. Under the operation of each convolution block, every half-edge receives information from all the half-edges within the same element and the twin half-edge from the adjacent element. After repeated application of such blocks, the final feature matrix will contain an encoding of the local topology in a field-of-view around every half-edge. The size of this field of view grows linearly with the number of convolution blocks. We use five convolution blocks in all of our experiments.

The initial feature matrix fed to the model is $x_0 \in \mathbb{R}^{2 \times N_h}$. Recall from Section 3.1 that each half-edge is associated with a vertex. The initial feature matrix consists of the degree and irregularity of the associated vertices for every half-edge. This initial feature matrix is projected to a high dimensional embedding space using a linear layer on which the convolution described above is applied. The final layer projects the features into an $N_a \times N_h$ matrix where N_a is the number of actions per half-edge.

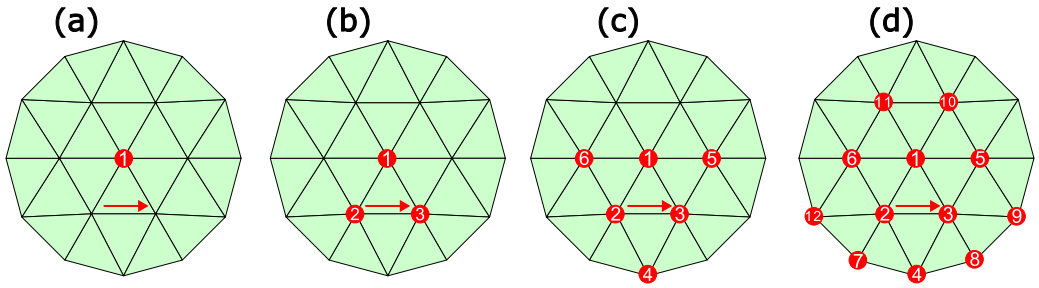


Fig. 5. Repeated application of the convolution operation produces an increasing field of view around every half-edge. For triangles, we associate every half-edge with the vertex opposite it in the same triangle (fig. a). A `cycle` operation gathers information from the remaining vertices in the element (fig. b). Repeated application of `twin` and `cycle` produces the ordered list of vertices in fig. (c) and (d).

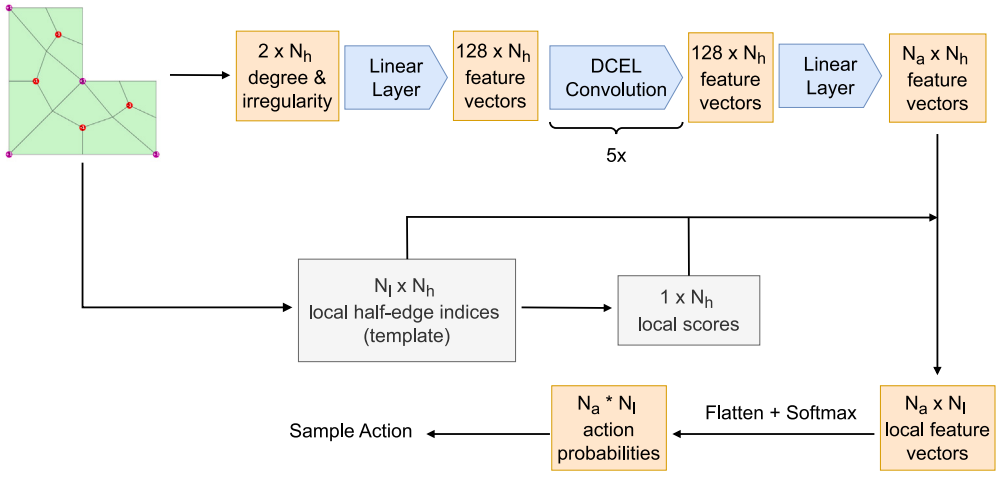


Fig. 6. Illustrating the action selection process on meshes. Half-edge features consisting of the degree and irregularity of the associated vertex are first projected to an embedding space. Convolution on the DCEL data-structure is performed on these embeddings. Local templates are constructed around all half-edges to obtain a local measure of mesh irregularity. Action selection is restricted to the local template with the highest measure of irregularity. The final feature matrix is flattened and passed through a softmax layer to obtain action probabilities (i.e. a policy). This distribution is sampled to determine the action to take.

4.2.1. Action selection by the agent

The size of meshes can vary as the agent manipulates the mesh. The total number of actions available to the agent varies with the size of the mesh. To ensure that the agent can generalize across different mesh sizes, we found it important that our policy is represented by a fixed sized vector representing the probabilities of selecting various actions.

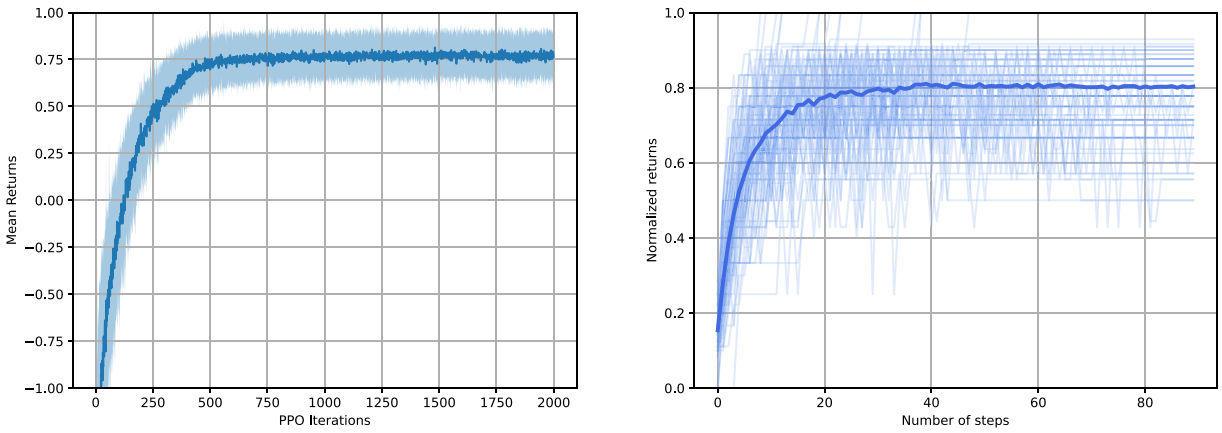
To do this, we generate a list of vertices which we call the *template* around each half-edge (see Fig. 5.) The template can be constructed using operations similar to the convolution described in Section 4.2. Initially, every half-edge has the index of the vertex it is associated with. After a *cycle* operation, every half-edge receives the indices of the vertices that are cyclic *next*. After a *twin* operation, every half-edge receives vertex indices from neighboring elements. Notice that there is some repetition in indices which can be avoided by selecting appropriate rows of the index matrix after a *cycle* or *twin* operation. These operations are repeatedly applied to grow the size of the template. We use dummy vertices if the template goes outside the boundary of the mesh. We then compute the score Eq. (1) restricted to each template. This is a measure of the local irregularity around every half-edge. The irregularity of dummy vertices is set to zero ensuring that they do not contribute to the score of the template. Action selection is then restricted to the half-edges in the template with the highest local score with ties broken randomly. Thus we consider an $N_a \times N_l$ subset of the output feature matrix from Section 4.2 where N_l is the number of half-edges in the template. This matrix is flattened and passed through a softmax layer to obtain a probability distribution over actions in the template. We sample from this distribution to take a step into a new mesh state. Fig. 6 shows an illustration of the action-selection process for a given mesh.

In the current implementation, convolution is performed on the entire mesh. Subsequently, the local irregularity measure is used to restrict action selection to the local template. While this works fine for our small examples, it would be computationally wasteful on larger meshes since convolution is performed on many half-edges that may not be included in the final template. In future implementations we will first choose the local template and only perform convolutions on this subset of half-edges.

The main purpose of constructing the template is to obtain a local measure of the irregularity score which enables *fast* selection of candidate regions. This ensures that action selection is restricted to regions with a high potential for reward. Further, we are able to consider a fixed size action space restricted to this candidate region, enabling the agent to generalize across mesh sizes. Alternate strategies may be adopted for candidate selection instead. For example, using breadth-first-search to find the K-nearest neighbors of a half-edge in the graph induced by the half-edge connectivity. The score Eq. (1) restricted to this K-neighborhood can be used as a measure of the local irregularity in the mesh.

4.2.2. Training the agent in self-play

The initial states for self-play are randomly generated polygonal shapes. We randomize the degree of polygon (i.e. number of sides of the polygon) between set bounds. We perform Delaunay refinement meshing of this shape and use that as the input to the triangular mesh agent. For the quadrilateral agent, we perform the Catmull–Clark splits on the triangles to get an all quad initial mesh.



(a) Average performance over training history

(b) Evaluating the trained agent on multiple rollouts

Fig. 7. (a) Performance of the triangle mesh agent over the training history. Solid line represents the average normalized return over 100 meshes evaluated periodically during training. Shaded region represents the 1-standard deviation envelope. (b) Performance of the trained agent over 100 rollouts. The agent incrementally improves mesh quality up to a certain number of steps. Notice that returns do not increase monotonically, indicating that a greedy strategy may not be effective for this problem.

The agent is allowed to interact with the mesh and perform operations on it for a finite number of steps or until the agent achieves the optimum score s^* whichever comes first. Because the size of the mesh environment can be variable, the *value* or expected reward that an agent can receive from a given mesh environment is variable and cannot be inferred purely from the local representation of state that we employ. This makes it challenging to use value function based algorithms such as Deep Q-Learning [28] or Soft Actor–Critic [29]. Therefore, we train our agent using an actor-only version of the Proximal Policy Optimization (PPO) [30] algorithm without a critic (i.e. value function) network. PPO is one of the most widely used deep-reinforcement learning algorithms. It falls within the family of policy-gradient algorithms [31] with additional constraints that prevent large changes in the policy distribution during training. This ensures training stability and uniform policy improvement over the course of training. We use Eq. (6) as the advantage function in the PPO algorithm. We add an entropy regularization to the loss function to avoid local minima and balance exploration with exploitation. Full details of the hyperparameters used are provided in Table 1 .

The agents were trained for approximately 24 h on a single Nvidia GTX 2080TI GPU. As the learning curves in Figs. 7(a) and 10(a) show, the agent reaches its optimal performance fairly quickly after which performance plateaus implying that the full 24 h was not necessary to train the model to a good level of performance. The inference cost of each step of the agent is on the order of a few milliseconds and is hardware dependent. Note that we did not optimize for model throughput.

Table 1
Hyperparameter settings used to train the agent.

Hyperparameter	Value
PPO ϵ parameter	0.05
Minibatch size	128
Epochs per PPO iteration	5
Trajectories sampled per PPO iteration	200
Number of PPO iterations	2000
Weight of entropy loss	0.001
Learning rate	10^{-4}
Discount factor γ	1
Number of DCEL convolution blocks	5
DCEL convolution hidden layer size	128

5. Results

5.1. Triangular meshes

The triangular mesh agent was trained on random shapes consisting of 10 to 30 sided polygons. The initial mesh was a Delaunay refinement mesh generated by the Triangle package [1]. Fig. 7 shows the learning curves of our agent over training history, and the performance of the trained model over 100 rollouts. The average normalized single-shot performance over 100 meshes was about 0.81 ($\sigma = 0.11$). However, since the learned policy is stochastic, a simple way to improve the performance is to run the policy k times from the same initial state and pick the best mesh. Using $k = 10$ samples per mesh and averaging over 100 random meshes, the performance improved to 0.86 ($\sigma = 0.08$).

Table 2 demonstrates the generalization capability of the learned policy. By using a fixed sized local template, the same agent can be evaluated on meshes of various sizes with good results. We do observe some reduction in model performance on larger meshes. Irregularities tend to be separated by greater distances on larger meshes, requiring longer sequences of operations to effectively remove them. We illustrate an example of the trained agent's performance on a 40-sided polygon in Fig. 9. While the agent is able to eliminate some irregularities that are near each other, the final mesh still contains quite a few irregularities that are separated by large distances. We need to bring irregularities near each other in order to regularize them. This requires a complex sequence of moves that our agent is unable to learn. Further, since our state-representation relies on a local template, when irregularities are separated by large distances, our agent is unable to detect them through the local state representation.

We note that the normalized performance of the agent in the quadrilateral mesh environment is significantly better (see sec. Section 5.2.) Our experiments indicate that the use of global mesh editing operations such as the global-split and global-cleanup are instrumental in achieving this performance. The global cleanup, in particular, is effective at coarsening the mesh without introducing new irregularities. The cleanup operation brings irregularities closer to each other making it easier to combine them and regularize them. Defining such a cleanup operation for triangles is non-trivial and remains to be addressed through future work (see Fig. 8).

5.2. Quadrilateral meshes

The quadrilateral mesh agent was trained on random shapes consisting of 10 to 30 sided polygons. Fig. 10(a) shows the average normalized returns over training for the quadrilateral mesh agent. We observe that the agent quickly learns operations that significantly improves

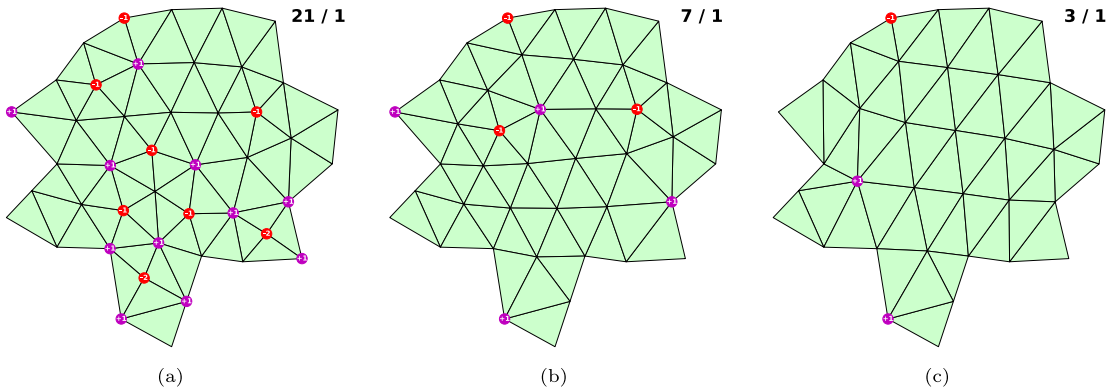


Fig. 8. Example rollout of the triangular mesh agent on a 20-sided polygon. Irregular vertices are marked in color, with the current score and optimum score shown at the top right for each figure. (a) is the initial Delaunay refinement mesh (b) is at an intermediate stage and (c) is the final mesh after 27 operations. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

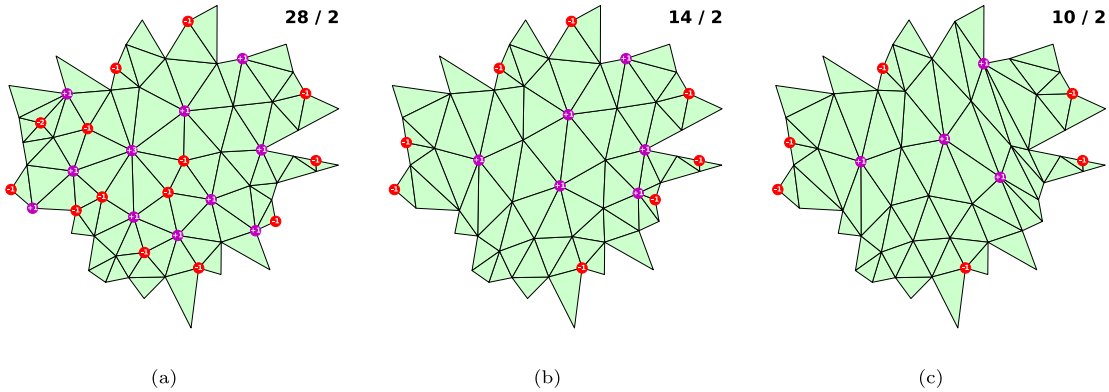


Fig. 9. Example rollout of the triangular mesh agent on a 40-sided random polygonal shape. (a) is the initial mesh, (b) is an intermediate state, and (c) is the mesh with the lowest score during policy rollout. Notice that the agent is able to improve the irregularity score of the mesh. However, the final mesh contains irregular nodes that are separated by several mesh elements. These irregularities require complex sequences of moves to bring them together and regularize them. Further, the agent has a limited field of view controlled by the size of the local template. Irregularities outside the local template will not be seen by the agent.

Table 2
Evaluating the triangle mesh agent on various sized random polygons. The agent was trained purely on 10–30 sided polygons but is able to generalize to other polygon sizes with minor deterioration in performance. The agent was evaluated by picking the best of 10 rollouts per geometry, with the statistics computed over 100 randomly generated shapes. The results demonstrate the effectiveness of using a fixed-sized local template which enables better generalization to different mesh sizes.

Polygon degree	Average	Standard deviation
3–10	0.83	0.19
10–20	0.87	0.08
20–30	0.83	0.10
30–40	0.78	0.08
40–50	0.75	0.07

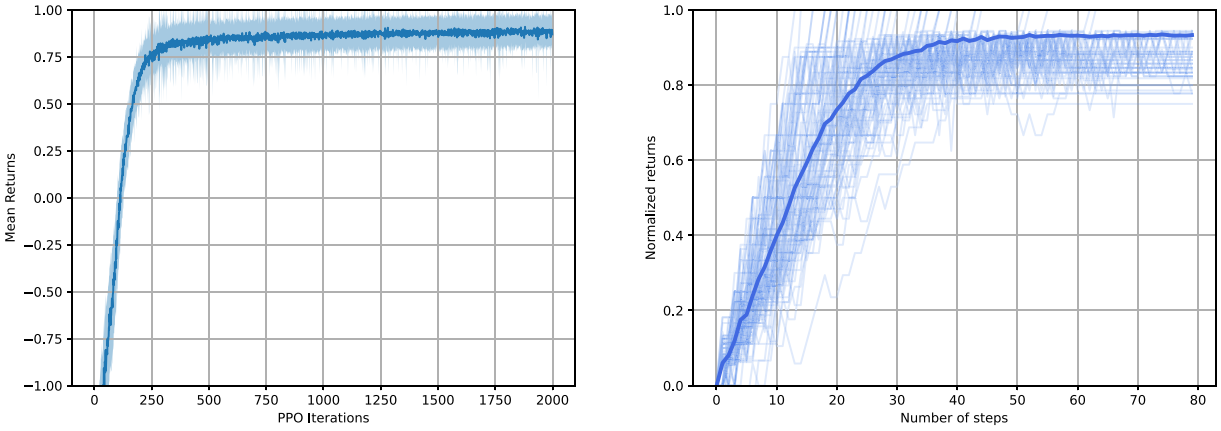
the connectivity of the mesh to nearly optimal. Performance was assessed periodically during training by evaluating the model on 100 randomly generated meshes. Fig. 10(b) shows the evaluation of the best performing model on 100 trajectories. We observe that performance depends on the maximum number of steps given to the agent up to a certain point. The average normalized single-shot performance over 100 meshes was about 0.95 ($\sigma = 0.05$.). Using $k = 10$ samples per mesh and averaging over 100 random meshes, the performance improved to 0.992 ($\sigma = 0.02$).

Since our state representation is a fixed-sized local template around a half-edge of interest, our model generalizes well to polygons that were not part of the training dataset. Table 3 shows the performance of a model trained on 10–20 sided polygons that is able to generalize to larger sized polygonal shapes. We do observe some drop in the performance of the agent when mesh sizes are increased. This is consistent with our observations for triangular meshes. Irregularities are often separated by several mesh elements in larger meshes, requiring longer range sequences of operations to regularize them. The complexity of these operations, coupled with the local nature of our state representation likely causes the deterioration in the agent's performance.

Figs. 11 and 12 show some example rollouts on various polygon sizes. Note that the "optimal" mesh produced by the agent in Fig. 12(c) contains an irregular vertex with degree 2 on the bottom boundary. This mesh is considered optimal according to our objective function Eq. (1) since its score is equal to the optimal score s^* for this configuration. However, there are many applications wherein a configuration such as Fig. 12(d) is preferred, with the irregularity moved to the interior of the domain. This latter configuration is achieved via a post-processing step by applying an edit similar to the global split. Observe that both of these configurations have exactly the same objective score and are thus considered equivalent by our metric. If irregular vertices on the boundary are not preferred, they may be fixed by post-processing steps. Alternatively, the objective function may be modified by using a higher weight on irregularities on the boundary. This can encourage the agent to learn to move irregularities away from the boundary into the interior of the domain.

5.3. Generalization to new geometries

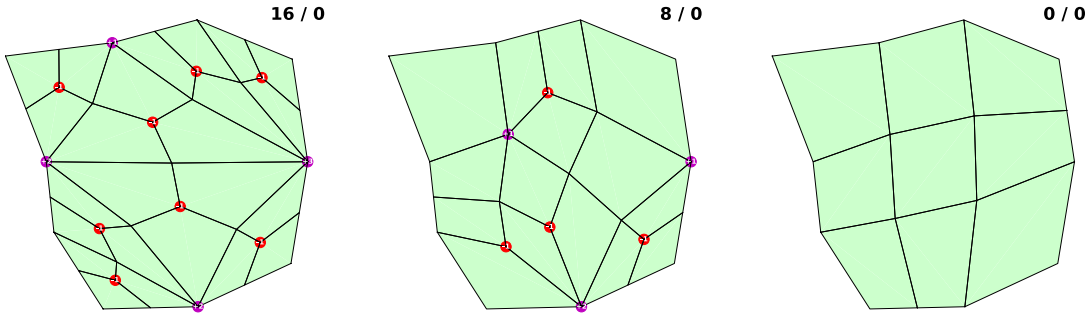
Figs. 13 and 14 show zero-shot transfer to never before seen geometries like L-shape, star-shape, etc. The agent is able to handle geometries



(a) Average performance over training history

(b) Evaluating the trained agent on multiple rollouts

Fig. 10. (a) Performance of the quadrilateral mesh agent over the training history. Solid line represents the average normalized returns evaluated over 100 meshes. Shaded region represents the 1-standard deviation envelope. The curve demonstrates that the agent is able to achieve good performance quite rapidly, and the learning remains stable over many training iterations. (b) Evaluating the trained model over multiple rollouts. Solid line represents the average performance of 100 rollouts. The graph demonstrates that increasing the maximum number of operations available to the agent has a big impact on performance initially, but only up to a certain point. Returns do not increase monotonically, highlighting that a greedy strategy may not be effective in this setting.



(a) (b) (c)

Fig. 11. Example rollout for a 10-sided polygon. Irregular vertices are marked in color. The mesh score and optimal score are shown at the top right for each figure. (a) is the initial mesh after Delaunay triangulation and Catmull–Clark splits, (b) is at an intermediate stage, and (c) is the final mesh after 18 operations. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
Evaluating the quadrilateral mesh agent on various sized random polygons. The agent was trained purely on 10–20 sided polygons, but is able to generalize to larger polygonal shapes with minor deterioration in performance. The agent was evaluated by picking the best of 10 rollouts per geometry, with the statistics computed over 100 randomly generated polygonal shapes. Using a fixed-sized local template enables stronger generalization to different sized meshes.

Polygon degree	Average	Standard deviation
10–20	0.98	0.03
20–30	0.97	0.06
30–40	0.94	0.14
40–50	0.91	0.13

with re-entrant corners and notches which were not explicitly part of the training space. Further, our model, which was trained exclusively on genus-0 geometries with no interior holes is able to generalize zero-shot to genus-1 shape consisting of a square hole in a circular shape.

We highlight the use of our approach in block decomposition of complex shapes into coarse quadrilateral elements. The global cleanup operation is particularly effective for this application as it is effective at coarsening the geometry without introducing new irregularities, and bringing existing irregularities closer to each other which makes it simpler to regularize them.

6. Conclusions

We presented here a method that learns to improve the connectivity of triangular and quadrilateral meshes through self-play reinforcement learning without any human input. A key contribution of this work is a parameterized method to generate a representation of the local topology in mesh neighborhoods. This enables appropriate selection of standard topological mesh editing operations which result in the reduction of irregular vertices in the mesh. Our method is built on the DCEL data-structure which allows the same framework to work on any planar 2D mesh with the discussion in this paper restricted to triangular and quadrilateral meshes.

When optimizing for connectivity, it is recommended to work with the coarsest possible mesh that captures the details of the geometry being considered. Optimal strategies to regularize meshes consist of sequences of operations that combine irregularities together. It is easier to regularize coarser meshes because irregularities are relatively closer to each other on these meshes. Irregularities often become isolated on finer meshes, and require longer sequences of complex operations to regularize them. Both triangles and quadrilateral meshes can be globally refined without introducing irregular vertices. Some applications, like numerical simulation, demand finer meshes for the sake of simulation accuracy. Thus, once the connectivity of a coarse mesh has

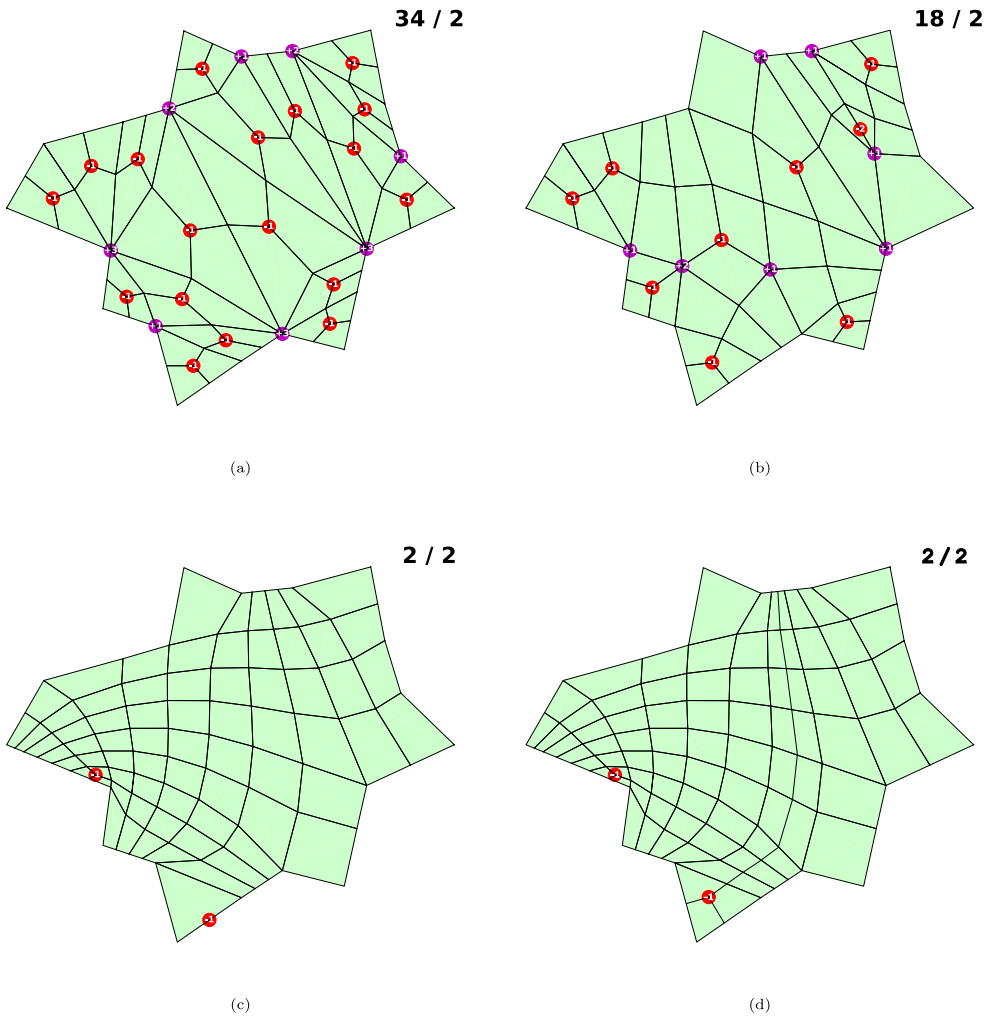


Fig. 12. The same agent as before is able to optimize a 20-sided polygonal shape using 40 operations. (a) initial mesh, (b) intermediate mesh, (c) final mesh produced by the agent, (d) degenerate vertices on the boundary can be post-processed using an operation similar to the global split. Note that meshes (c) and (d) have the same score as measured by Eq. (1) thus our agent does not prefer one over the other. If the application demands that there be no degenerate vertices, these irregularities can be eliminated through a final post-processing step.

been optimized, it can be easily refined to the desired resolution while maintaining its regularity.

A major advantage of artificial intelligence is its ability to discover heuristics that are too laborious and cumbersome for humans to identify, formulate, and prescribe. There are several areas in mesh generation where the automatic discovery of such heuristics can significantly aid engineers in their work. We hope that this paper demonstrates one such use-case.

7. Future work

There are several exciting directions of future research that we highlight here,

- **Incorporating value function:** Most deep reinforcement learning methods benefit from having a value function as this can help speed-up training. Our current formulation makes it challenging to estimate state value because we employ a local representation of state that does not provide sufficient information to estimate global value. Addressing this challenge is the focus of our current work.
- **Policy improvement with tree search:** our learned policy is stochastic and may be combined with e.g. Monte Carlo Tree Search (MCTS) [32] to efficiently search for optimal meshes for a specific geometry. The performance improvements that we observe from our naive best-of-k method in Sections 5.1 and 5.2 indicates that MCTS could be effective at improving the performance of our trained model. Such an approach would be similar to the AlphaZero [33] system.
- **Optimizing for element quality:** to achieve this, our model would need to additionally receive geometric information (e.g. vertex coordinates) as input. This can be easily achieved by including the coordinates of vertices as part of the input features to our model (see Section 4.2). We expect that the coordinates need to be normalized e.g. affine transform half-edges (and all vertices in its template) to a normalized coordinate system (e.g. $[0, 1]$.)
- **Extension to 3D:** We expect that our method can leverage the equivalent of the half-edge data-structure in 3D [19] to learn topological mesh editing operations on tetrahedral and hexahedral meshes. Determining optimal sequences of operations in 3D is highly challenging, and a self-learning method would have significant use.

We anticipate that extension to 3D applications can have significant utility and will likely attract further research interest. We discuss here some challenges that we foresee and possible methods to address these challenges.

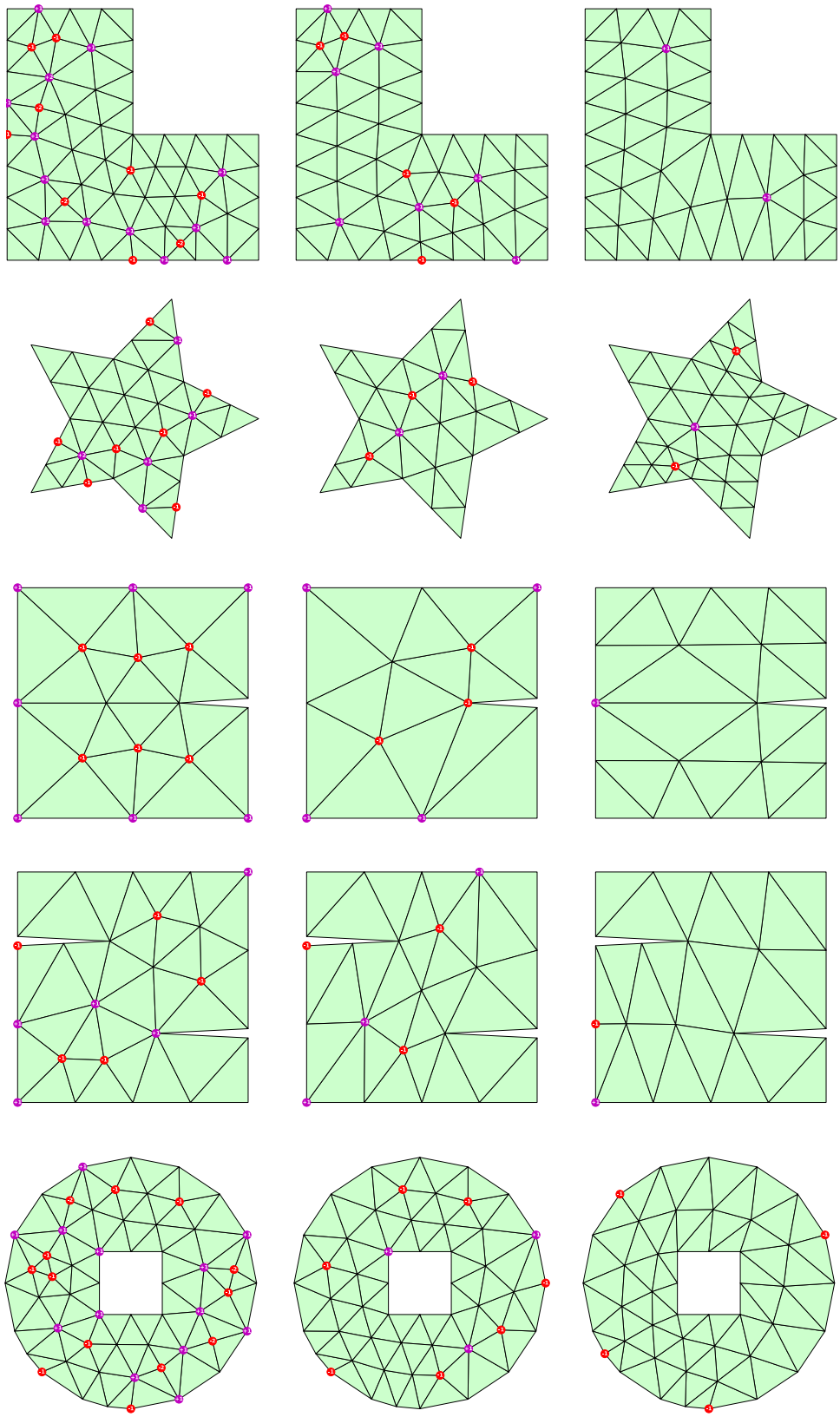


Fig. 13. The triangular mesh agent demonstrates zero-shot transfer on geometries that were not seen during training, including geometries with re-entrant corners like the star shape and the single and double-notch domains. First column is the initial mesh, second column is the intermediate mesh as the agent optimizes connectivity, and the third column is the final mesh after optimization. Since our model is based purely on connectivity, we can directly transfer the model onto geometries with holes even though such geometries were never seen during training. Notice that in several of the examples including the L-domain, single-notch, and the square hole in the circle, the model achieves the optimal score and the remaining irregularities cannot be eliminated as they are intrinsic to the geometry.

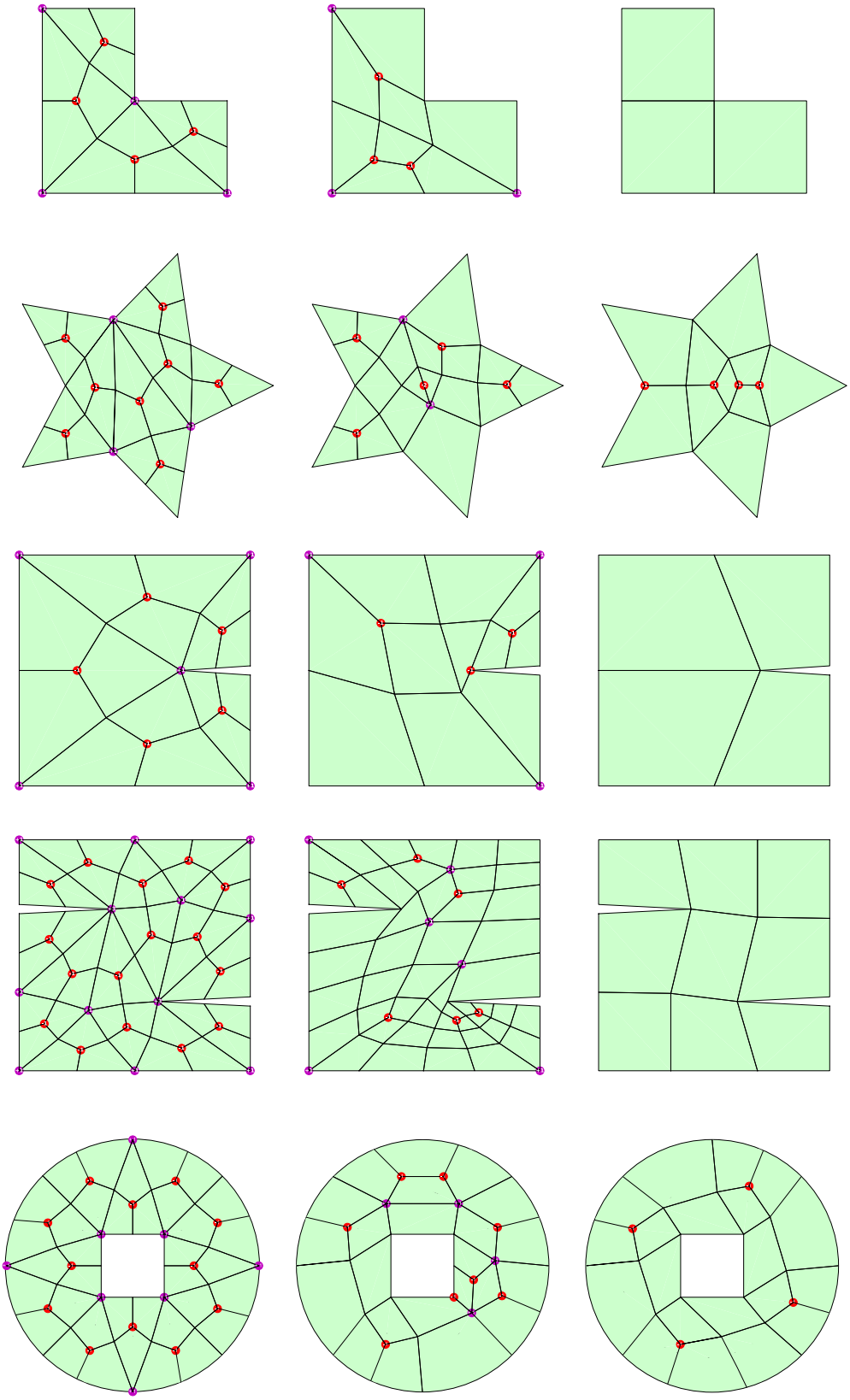


Fig. 14. The quadrilateral mesh agent demonstrates zero-shot transfer on geometries that were not seen during training, including geometries with re-entrant corners like the star shape and the single and double-notch domains. First column is the initial mesh, second column is the intermediate mesh as the agent optimizes connectivity, and the third column is the final mesh after optimization. Since our model is based purely on connectivity, we can directly transfer the model onto geometries with holes even though such geometries were never seen during training. We are particularly interested in coarse meshes representing block decompositions of more complex shapes. The cleanup operation is particularly useful in achieving coarse meshes. This is most evident in the single notch and double notch example in row 3 and 4. Notice that the star-shaped domain and the circular mesh with a square hole contain intrinsic irregularity that cannot be improved upon with our prescribed operations and heuristic.

- A suitable action space needs to be clearly defined for 3D mesh types. It is preferable that these actions are local in nature to minimize computational cost. Prior work defining actions on tetrahedral meshes [21–23] and hexahedral meshes [24,25] will be useful to consider in this regard.
- Our framework parameterizes mesh editing operations in terms of geometric primitives. For 2D, our geometric primitive was the half-edge. Selecting a half-edge and an associated edit operation unambiguously identifies an edit operation on a mesh. A similar parametrization needs to be developed for 3D meshes to adopt our framework. We anticipate that both half-edges and *half-faces* would be required to parametrise mesh editing operations in 3D. For example, the popular 2–3 face swap operation in tetrahedral meshing can be parameterized by selecting a half-face and prescribing the swap operation on it.
- We require an extension of the convolution operation on the 3D mesh data-structures. In 3D, there is a connectivity structure on half-faces along with half-edges [19]. The convolution operation needs to operate on both of these connectivities.
- Selecting a candidate region in which to evaluate the reinforcement learning agent will be equally important in 3D applications due to the large variations in the number of elements. We anticipate that an objective score similar to Eq. (1) can provide a simple way of evaluating regions with the potential for significant quality improvement.
- In 2D, the enclosed angle at a boundary vertex provides a simple method of identifying the desired degree. Specifying the desired connectivity, particularly on boundaries, is essential to obtaining a well-defined method in 3D.

CRediT authorship contribution statement

A. Narayanan: Conceptualization, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Y. Pan:** Conceptualization, Formal analysis, Investigation, Methodology, Writing – original draft, Writing – review & editing. **P.-O. Persson:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported in part by the Director, Office of Science, Office of Advanced Scientific Computing Research, U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] Shewchuk Jonathan Richard. Delaunay refinement algorithms for triangular mesh generation. Comput Geom 2002;22(1–3):21–74.
[2] Peraire Jaime, Vahdati Morgan, Morgan Ken, Zienkiewicz Olgierd C. Adaptive remeshing for compressible flow computations. J Comput Phys 1987;72(2):449–66.
[3] Blacker Ted D, Stephenson Michael B. Paving: A new approach to automated quadrilateral mesh generation. int J Numer Methods Eng 1991;32(4):811–47.
[4] Owen Steven J, Staten Matthew L, Canann Scott A, Saigal Sunil. Q-Morph: an indirect approach to advancing front quad meshing. int J Numer Methods Eng 1999;44(9):1317–40.
[5] Remacle J-F, Lambrechts J, Seny B, Marchandise E, Johnen A, Geuzainet C. Blossom-quad: a non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm. Internat J Numer Methods Engrg 2012;89(9):1102–19 http://dx.doi.org/10.1002/nme.3279,.
[6] Daniels Joel, Silva Cláudio T, Shepherd Jason, Cohen Elaine. Quadrilateral mesh simplification. ACM Trans Graph (TOG) 2008;27(5):1–9.
[7] Tarini Marco, Pietroni Nico, Cignoni Paolo, Panozzo Daniele, Puppo Enrico. Practical quad mesh simplification. In: Computer graphics forum, vol. 29, Wiley Online Library; 2010, p. 407–18.
[8] Akram Muhammad Naeem, Xu Kaoji, Chen Guoning. Structure simplification of planar quadrilateral meshes. Comput Graph 2022.
[9] Bommes David, Lévy Bruno, Pietroni Nico, Puppo Enrico, Silva Claudio, Tarini Marco, et al. Quad-mesh generation and processing: A survey. In: Computer graphics forum, vol. 32, Wiley Online Library; 2013, p. 51–76.
[10] Docampo-Sánchez Julia, Haimes Robert. A regularization approach for automatic quad mesh generation. In: 28th International meshing roundtable. Zenodo. 2020.
[11] Canann Scott A, Muthukrishnan SN, Phillips RK. Topological improvement procedures for quadrilateral finite element meshes. Eng Comput 1998;14(2):168–77.
[12] Sutton Richard S, Barto Andrew G. Reinforcement learning: An introduction. MIT Press; 2018.
[13] Vinyals Oriol, Fortunato Meire, Jaitly Navdeep. Pointer networks. Adv Neural Inf Process Syst 2015;28.
[14] Pan Jie, Huang Jingwei, Wang Yunli, Cheng Gengdong, Zeng Yong. A self-learning finite element extraction system based on reinforcement learning. AI EDAM 2021;35(2):180–208.
[15] Pan Jie, Huang Jingwei, Cheng Gengdong, Zeng Yong. Reinforcement learning for automatic quadrilateral mesh generation: A soft actor–critic approach. Neural Netw 2023;157:288–304.
[16] Yang Jiachen, Dzanic Tarik, Petersen Brenden, Kudo Jun, Mittal Ketan, Tomov Vladimir, et al. Reinforcement learning for adaptive mesh refinement. In: International conference on artificial intelligence and statistics. PMLR; 2023, p. 5997–6014.
[17] Służalec Tomasz, Grzeszczuk Rafał, Rojas Sergio, Dzwinel Witold, Paszyński Maciej. Quasi-optimal hp-finite element refinements towards singularities via deep neural network prediction. Comput Math Appl 2023;142:157–74.
[18] DiPrete Benjamin C, Garimella Rao V, Cardona Cristina Garcia, Ray Navamita. Reinforcement learning for block decomposition of CAD models. 2023, arXiv preprint arXiv:2302.11066.
[19] Dobkin David P, Laszlo Michael J. Primitives for the manipulation of three-dimensional subdivisions. In: Proceedings of the third annual symposium on computational geometry. 1987, p. 86–99.
[20] Dyedov Vladimir, Ray Navamita, Einstein Daniel, Jiao Xiangmin, Tautges Timothy J. AHF: Array-based half-facet data structure for mixed-dimensional and non-manifold meshes. Eng Comput 2015;31:389–404.
[21] Shewchuk Jonathan Richard. Two discrete optimization algorithms for the topological improvement of tetrahedral meshes. 65, 2002, p. 2–7, Unpublished manuscript.
[22] Klingner Bryan Matthew, Shewchuk Jonathan Richard. Aggressive tetrahedral mesh improvement. In: Proceedings of the 16th international meshing roundtable. Springer; 2007, p. 3–23.
[23] Freitag Lori A, Ollivier-Gooch Carl. Tetrahedral mesh improvement using swapping and smoothing. Internat J Numer Methods Engrg 1997;40(21):3979–4002.
[24] Ledoux Franck, Shepherd Jason. Topological modifications of hexahedral meshes via sheet operations: a theoretical study. Eng Comput 2010;26:433–47.
[25] Tautgesa Timothy J, Knoopb Sarah E. Topology modification of hexahedral meshes using atomic dual-based operations. Algorithms 2003;11:12.
[26] Docampo-Sanchez Julia, Haimes Robert. Towards fully regular quad mesh generation. In: AIAA scitech 2019 forum. 2019, p. 1988.
[27] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. Computational geometry algorithms and applications. Spinger; 2008.
[28] Mnih Volodymyr, Kavukcuoglu Koray, Silver David, Graves Alex, Antonoglou Ioannis, Wierstra Daan, et al. Playing atari with deep reinforcement learning. 2013, arXiv preprint arXiv:1312.5602.
[29] Haarnoja Tuomas, Zhou Aurick, Abbeel Pieter, Levine Sergey. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International conference on machine learning. PMLR; 2018, p. 1861–70.
[30] Schulman John, Wolski Filip, Dhariwal Prafulla, Radford Alec, Klimov Oleg. Proximal policy optimization algorithms. 2017, arXiv preprint arXiv:1707.06347.
[31] Sutton Richard S, McAllester David, Singh Satinder, Mansour Yishay. Policy gradient methods for reinforcement learning with function approximation. Adv Neural Inf Process Syst 1999;12.
[32] Coulom Rémi. Efficient selectivity and backup operators in Monte-Carlo tree search. In: International conference on computers and games. Springer; 2006, p. 72–83.
[33] Silver David, Hubert Thomas, Schrittwieser Julian, Antonoglou Ioannis, Lai Matthew, Guez Arthur, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science 2018;362(6419):1140–4.