

Learning to Mesh: Deep Reinforcement Learning for Automated Topology and Geometry Generation

Per-Olof Persson

Department of Mathematics, University of California, Berkeley
Mathematics Department, Lawrence Berkeley National Laboratory

with Arjun Narayanan, Yulong (Lewis) Pan, Will Thacher, Wanzhou Lei

Tetrahedron Workshop VIII

July 13, 2026



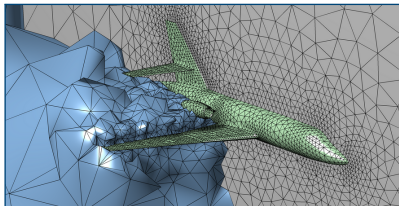
Outline

- 1 Introduction: Learning to Mesh
- 2 Part I: Learning to Repair Connectivity
- 3 Part II: Constructing Polygonal Meshes
- 4 Part III: Learning Geometry and Node Placement
- 5 Part IV: Escaping Local Optima in 3D
- 6 Conclusions

From Heuristics to Learning

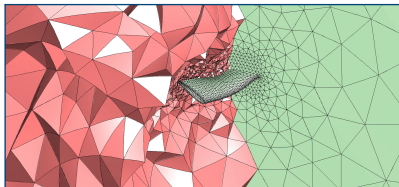
Why meshing matters

- Mesh quality controls **accuracy**, **stability**, and computational cost.
- Successful algorithms combine geometry, topology, and many special cases.
- Difficult domains still require expert choices and manual tuning.



A learning-based alternative

- Treat mesh generation as a **sequential decision process**.
- Replace a fixed sequence of heuristics by a learned policy.
- Let the agent improve through experience on procedurally generated problems.



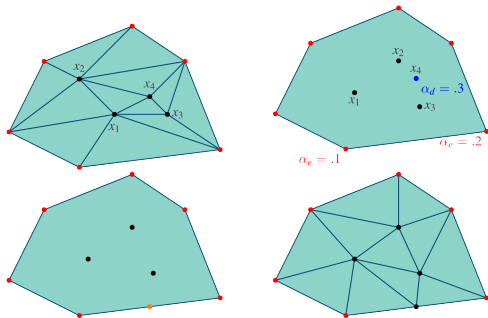
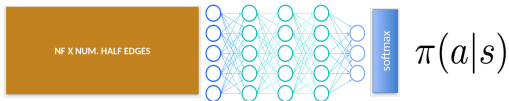
Why a Sequential Decision Process?

Meshing is not one prediction

- Each edit changes the actions available next.
- A locally attractive edit may block a better final mesh.
- Useful strategies are often **sequences** of insertions, deletions, moves, or flips.
- The number of vertices and elements changes during the process.

Reinforcement learning asks the right question:

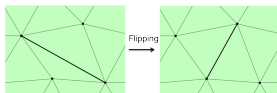
Which action should be taken now to maximize the quality of the final mesh?



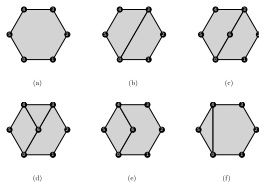
Four Mesh-Generation Games

Problem	Starting point	Learned actions	Primary goal
Connectivity repair	existing 2D mesh	flips, splits, collapses	regular topology
Polygon decomposition	boundary polygons	generic DCEL edits	chosen face type
Node placement	boundary + Delaunay	move, insert, delete	geometric quality
Tetrahedral repair	existing 3D mesh	2-3 and 3-2 flips	escape local optima

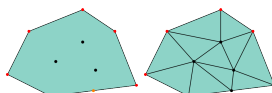
Connectivity repair



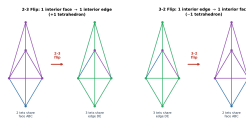
Polygon decomposition



Node placement



Tetrahedral repair



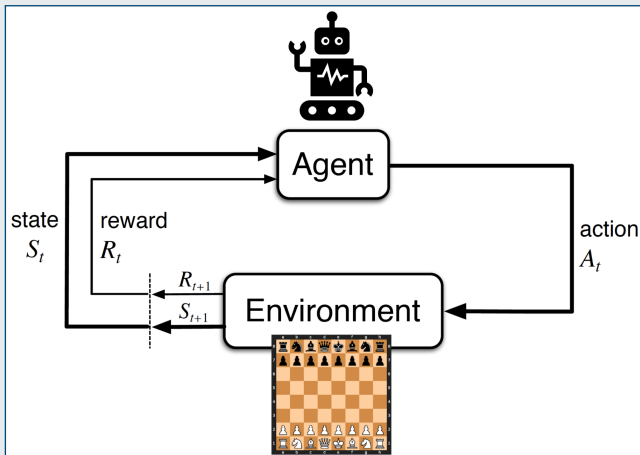
Reinforcement Learning: The Agent–Environment Loop

Learn through interaction

At time t :

- 1 Observe the mesh state s_t .
- 2 Sample an action a_t from the policy $\pi_\theta(a_t \mid s_t)$.
- 3 Apply the mesh edit.
- 4 Receive reward r_t and observe s_{t+1} .

The environment enforces geometric and topological validity.



States, Actions, Rewards, and Trajectories

The modeling choices

- **State** s_t : a graph representation of the current mesh.
- **Action** a_t : one valid local or global mesh edit.
- **Transition**: edit the mesh, then update connectivity and geometry.
- **Reward** r_t : improvement in a task-specific quality score.

The interaction produces a trajectory

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T).$$

$$\Pi(a_t | s_t, \theta)$$

Policy: Probability distribution
over actions

$$P(s_{t+1} | s_t, a_t)$$

State transition probability

$$\tau = s_0, a_0, \dots, s_H, a_H$$

State – action trajectory

$$R(\tau) = \sum_{t=0}^H R(s_t, a_t)$$

Cumulative returns of trajectory

The Objective: Final Quality, Not One-Step Accuracy

Cumulative reward and policy objective

The return measures cumulative reward:

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k.$$

$$\begin{aligned} U(\theta) &= \mathbb{E}[R(\tau); \Pi_\theta] \\ &= \sum_{\tau} P(\tau; \theta) R(\tau) \end{aligned} \qquad \theta^* = \arg \max_{\theta} U(\theta)$$

The policy objective is

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[G_0].$$

- γ controls how far ahead the policy looks.
- Telescoping quality rewards can make G_0 equal the final improvement.
- Delayed reward permits actions that only help after several edits.

Learning the Policy from Sampled Meshes

Policy-gradient idea

Increase the probability of actions that produced higher-than-expected returns:

$$\nabla_{\theta} J(\theta) = \mathbb{E} \left[\sum_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) A_t \right].$$

The advantage A_t compares an action with what was expected in the same state.

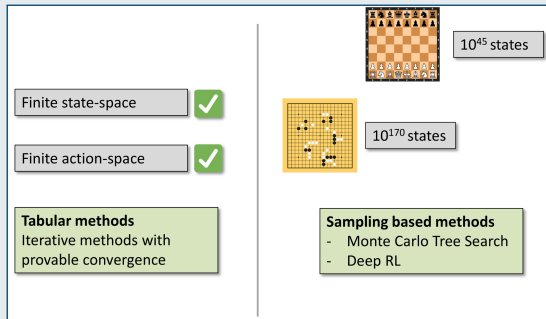
$$\begin{aligned} U(\theta) &= \sum_{\tau} P(\tau; \theta) R(\tau) \\ \nabla_{\theta} U(\theta) &= \sum_{\tau} \nabla_{\theta} P(\tau; \theta) R(\tau) \\ &= \sum_{\tau} P(\tau; \theta) \frac{\nabla_{\theta} P(\tau; \theta)}{P(\tau; \theta)} R(\tau) \\ &= \mathbb{E} [\nabla_{\theta} \log(P(\tau; \theta)) R(\tau)] \approx \frac{1}{m} \sum_{i=1}^m \boxed{\nabla_{\theta} \log(P(\tau^{(i)}; \theta))} R(\tau^{(i)}) \end{aligned}$$

Actor–Critic Training with PPO

Two learned functions

- **Actor:** the policy $\pi_{\theta}(a | s)$ scores valid mesh edits.
- **Critic:** $V_{\phi}(s)$ predicts the expected return and reduces gradient variance.
- **PPO:** reuses sampled trajectories while limiting destabilizing policy updates.

This is the common optimizer in all four projects; what changes is the mesh representation, action space, and reward.

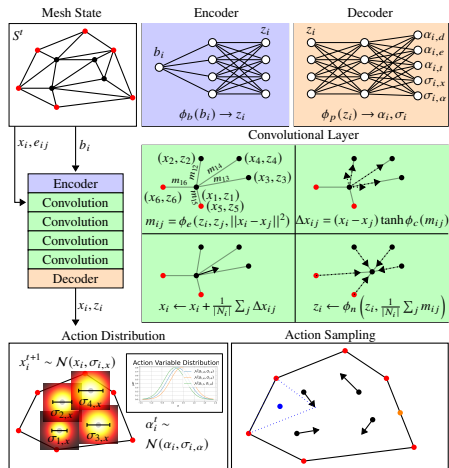


Why Graph Neural Networks?

Meshes are graphs, not arrays

- No preferred ordering of vertices, edges, or elements.
- Local neighborhoods vary in size and connectivity.
- Message passing shares one local rule across the entire mesh.
- Geometric features can be made translation-, rotation-, and scale-aware.
- The same network can act on meshes larger than those seen during training.

The network supplies the policy; the mesh data structure supplies valid local neighborhoods and edits.

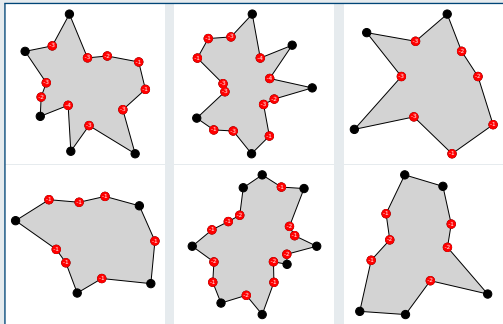


Self-Supervised Training on Random Geometries

No labeled action sequences required

- 1 Generate a random domain or perturb a known mesh.
- 2 Let the policy edit it for one episode.
- 3 Evaluate the resulting mesh with explicit quality metrics.
- 4 Update the actor and critic from the sampled experience.

The quality function is supervision: training does not require an expert to provide the correct move at every step.



Outline

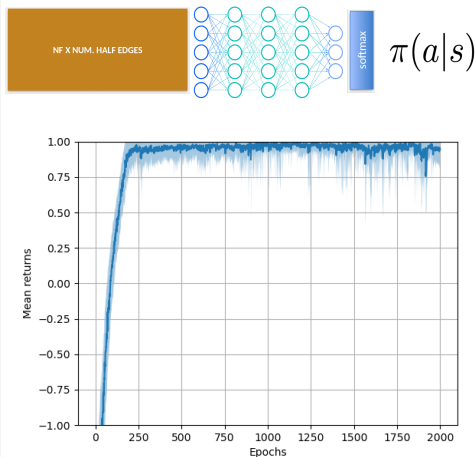
- 1 Introduction: Learning to Mesh
- 2 Part I: Learning to Repair Connectivity**
- 3 Part II: Constructing Polygonal Meshes
- 4 Part III: Learning Geometry and Node Placement
- 5 Part IV: Escaping Local Optima in 3D
- 6 Conclusions

The Connectivity Game

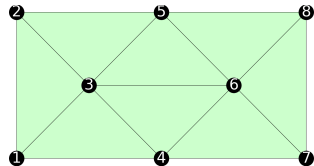
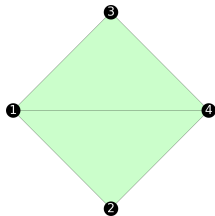
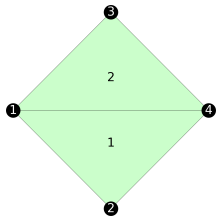
Improve an existing 2D mesh

- **State:** local mesh connectivity around a selected half-edge.
- **Actions:** triangle or quadrilateral topological operations.
- **Score:** deviation from desired vertex degree.
- **Training:** random polygons, PPO, and GPU-parallel environments.

The goal is not to imitate a meshing code, but to discover sequences that approach the best topology allowed by the boundary.



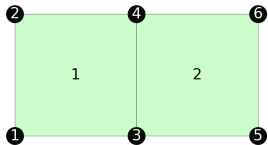
Mesh Editing Operations — Triangles



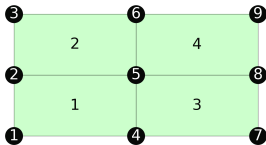
Edge flip: change connectivity without changing vertex set. **Edge split:** insert a new vertex on an edge. **Collapse:** merge two endpoints and remove an edge.

Together these operations can move, create, and cancel topological irregularities.

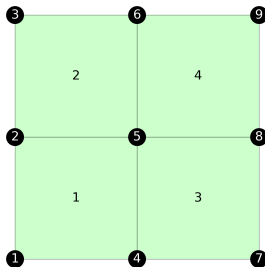
Mesh Editing Operations — Quadrilaterals



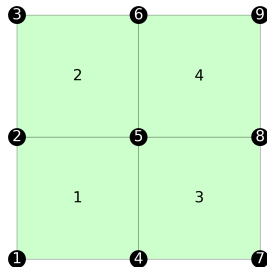
Local flip



Split-collapse



Global split



Global cleanup

Local edits rearrange irregularity; global edits add or remove an entire strip of quads.

The Score and Its Lower Bound

Vertex irregularity

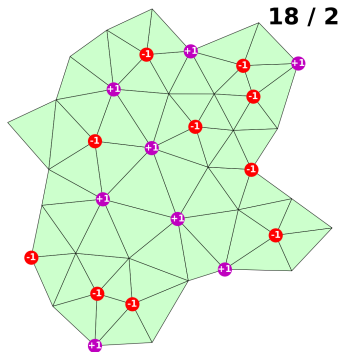
For desired degree d_i^* and current degree d_i ,

$$\Delta_i = d_i - d_i^*, \quad s = \sum_i |\Delta_i|.$$

The signed sum gives a topology-dependent lower bound:

$$s^* = \left| \sum_i \Delta_i \right| \leq s.$$

- s^* is invariant under the allowed mesh operations.
- Reaching $s = s^*$ certifies that the agent is optimal under this score.

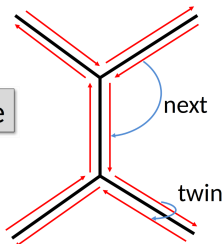


A Fixed-Size View of an Unstructured Mesh

Half-edge representation

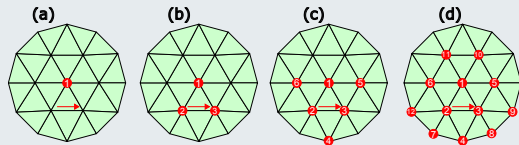
Pointer operations such as next, previous, and twin provide a consistent traversal of local topology.

Action: Half-edge + type



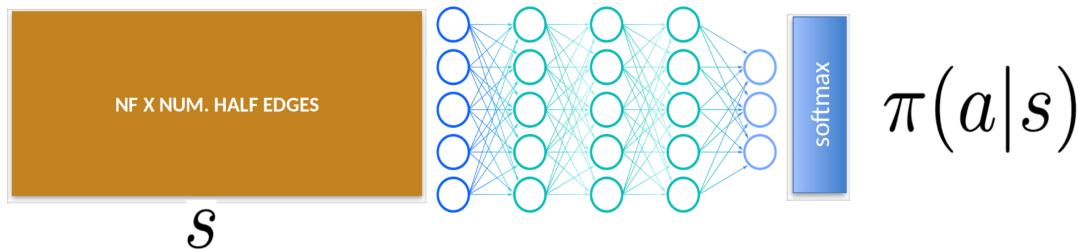
Local template

The policy observes an ordered, fixed-size neighborhood of vertex features around the chosen half-edge.



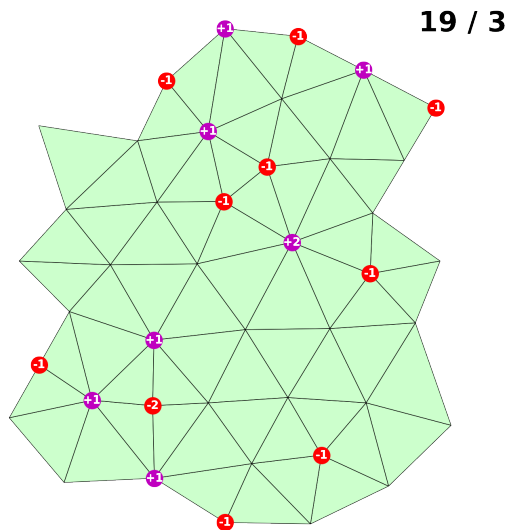
A Neural Network Learns the Edit Policy

- A shared half-edge feature extractor maps the local template to an embedding.
- The actor scores the admissible operations; invalid actions are masked.
- The critic predicts the expected remaining improvement.
- Self-play on random polygons exposes the agent to many irregularity patterns.



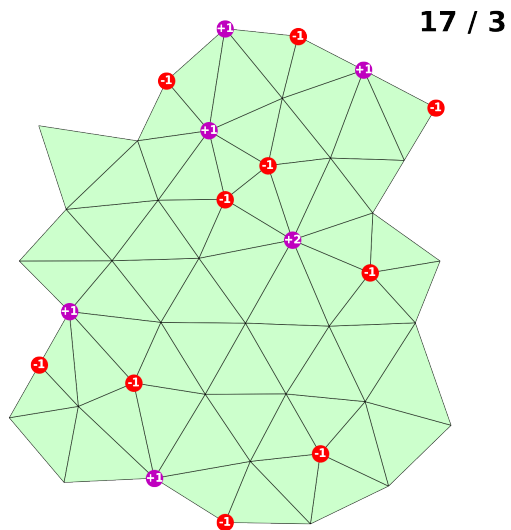
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 0 (out of 27)



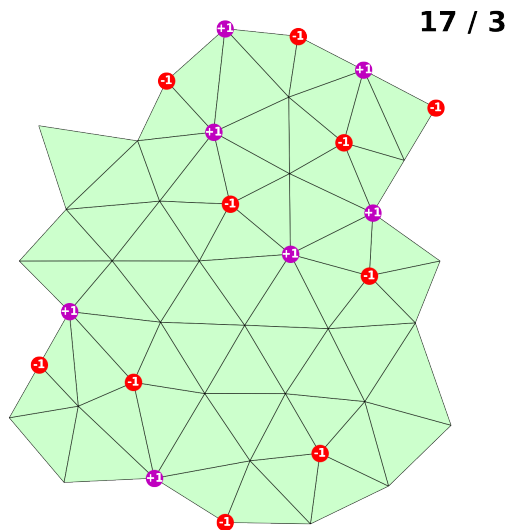
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 1 (out of 27)



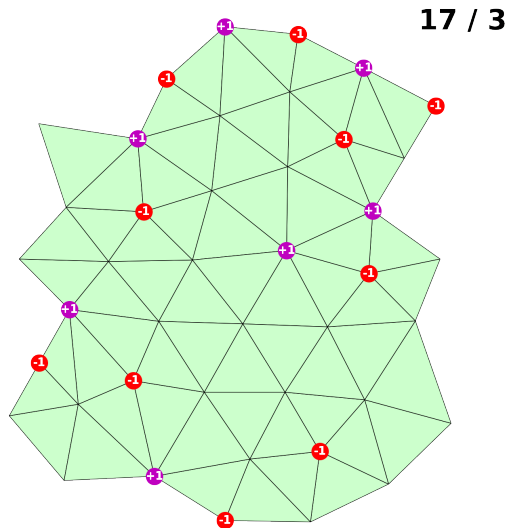
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 2 (out of 27)



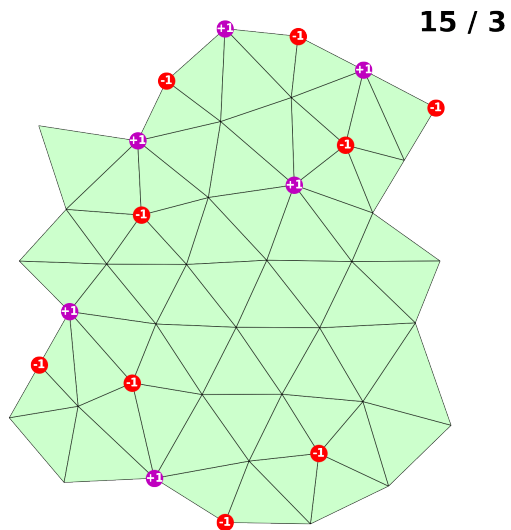
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 3 (out of 27)



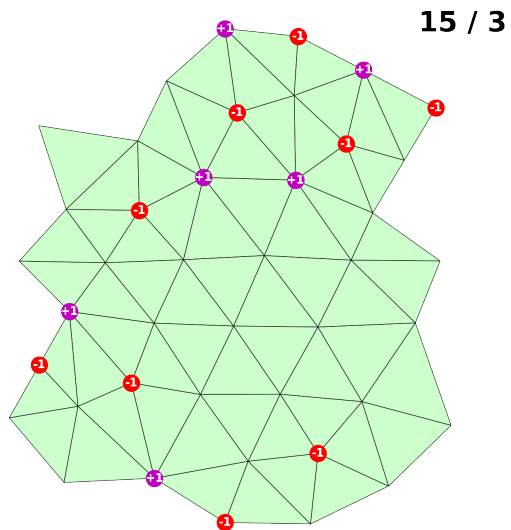
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 4 (out of 27)



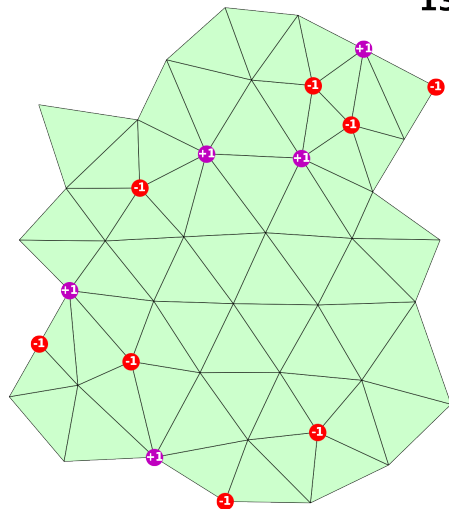
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 5 (out of 27)



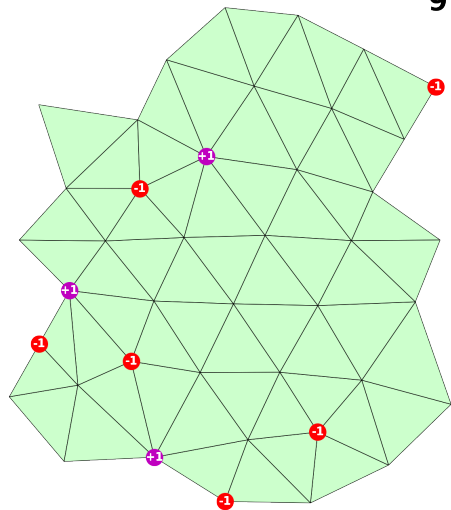
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 6 (out of 27)



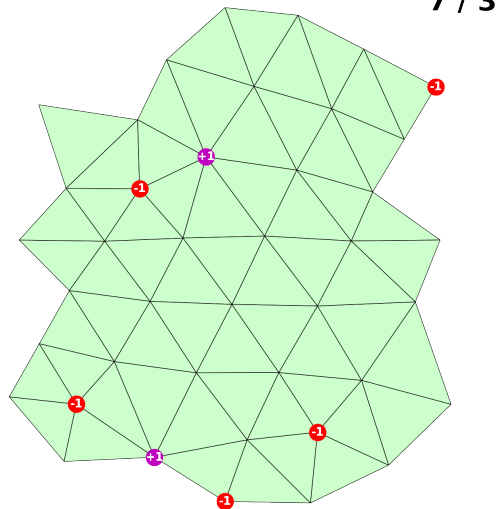
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 7 (out of 27)



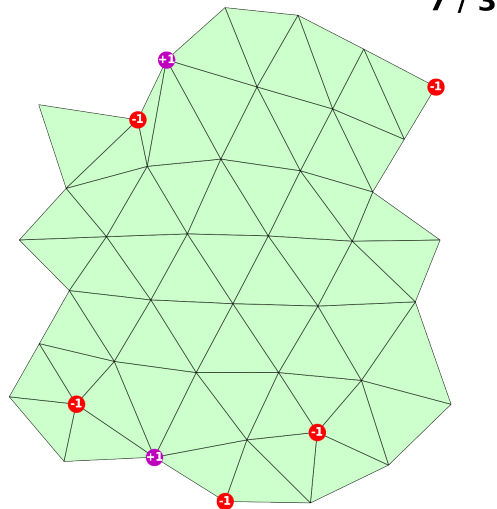
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 8 (out of 27)



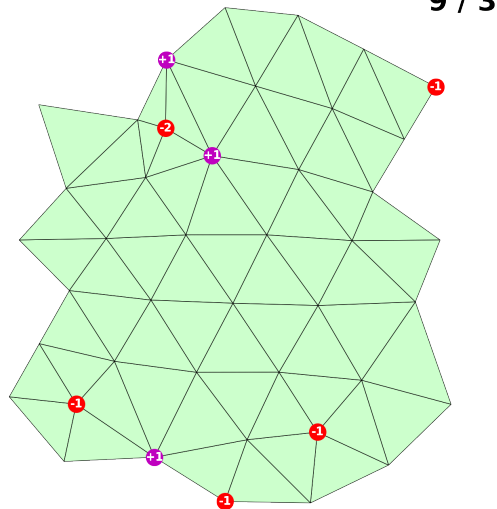
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 9 (out of 27)



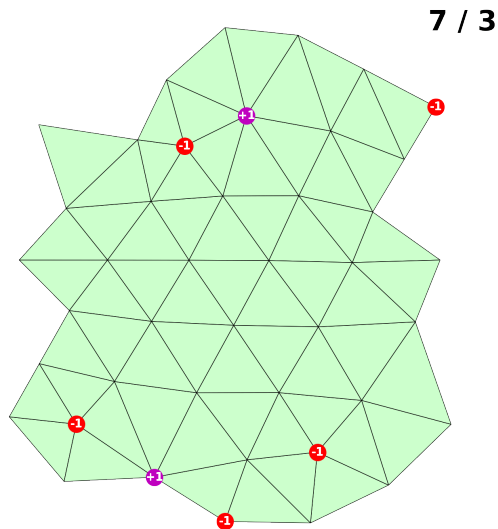
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 10 (out of 27)



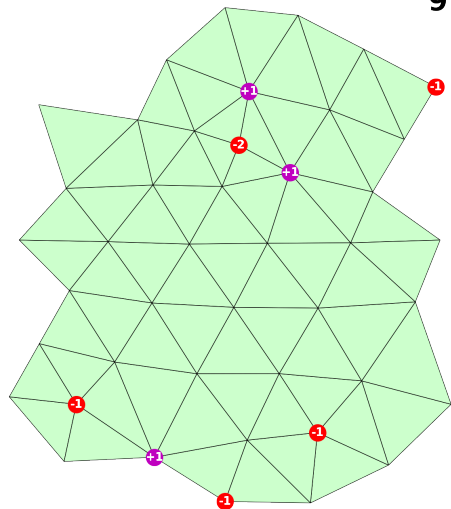
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 11 (out of 27)



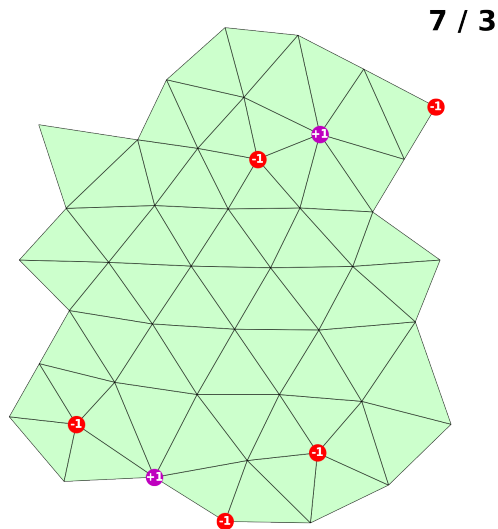
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 12 (out of 27)



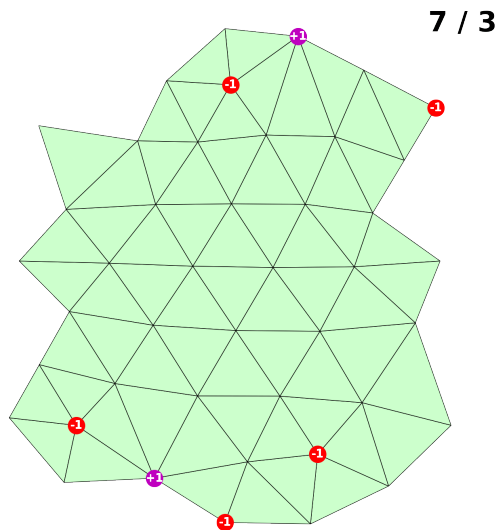
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 13 (out of 27)



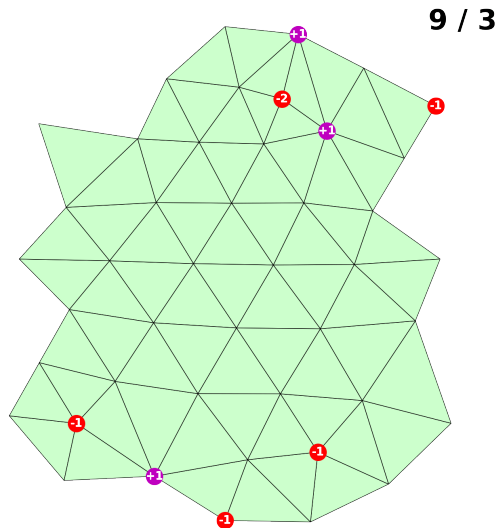
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 14 (out of 27)



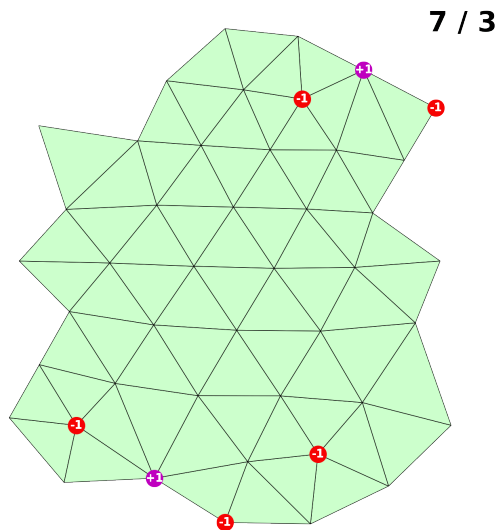
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 15 (out of 27)



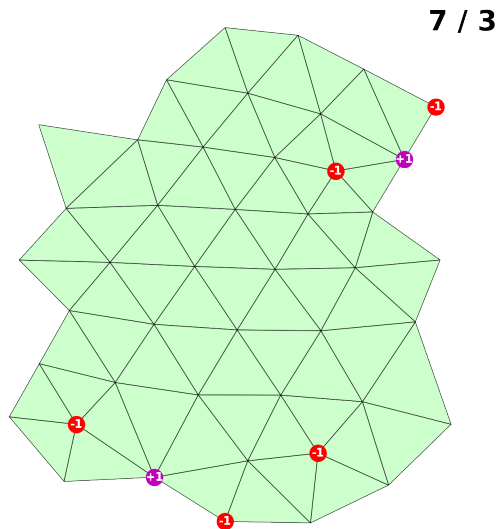
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 16 (out of 27)

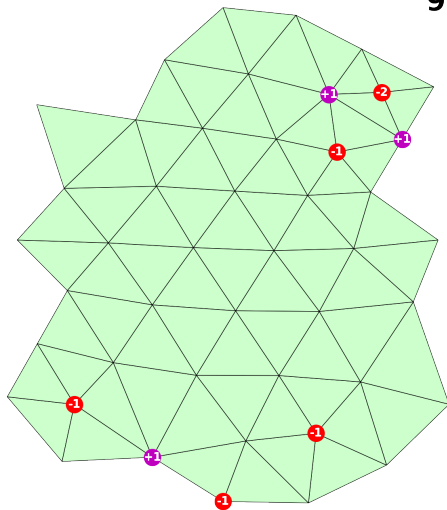


Triangular Mesh Optimization

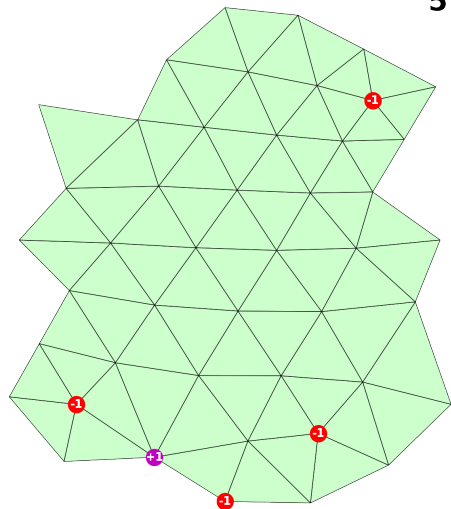
Triangular meshing
Example 1
Step 17 (out of 27)



Triangular meshing
Example 1
Step 18 (out of 27)

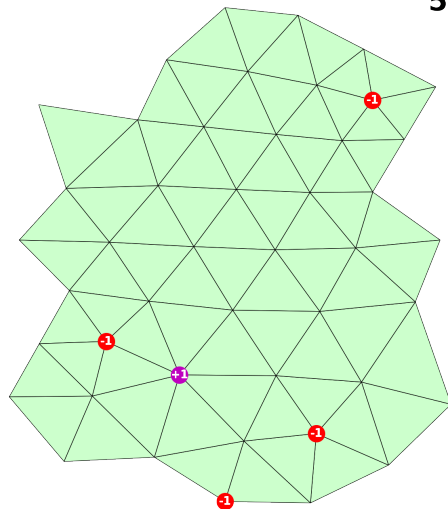


Triangular meshing
Example 1
Step 19 (out of 27)

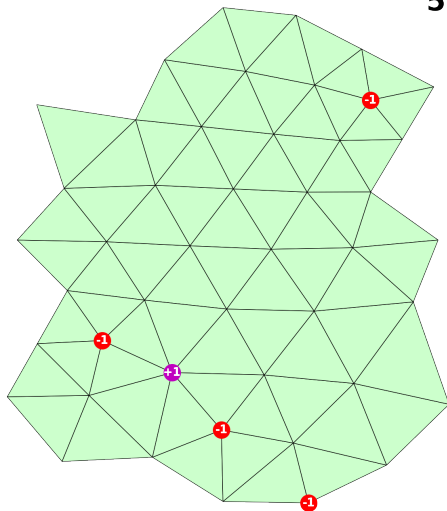


Triangular Mesh Optimization

Triangular meshing
Example 1
Step 20 (out of 27)

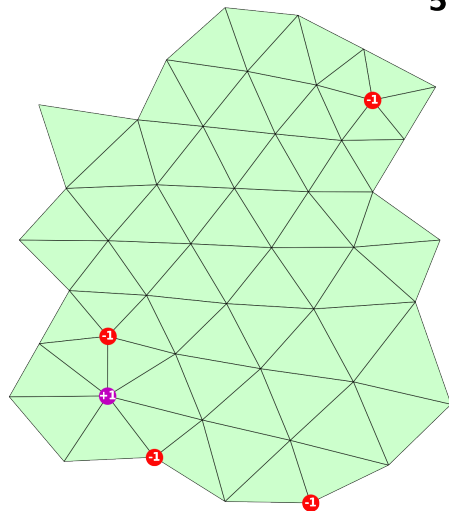


Triangular meshing
Example 1
Step 21 (out of 27)



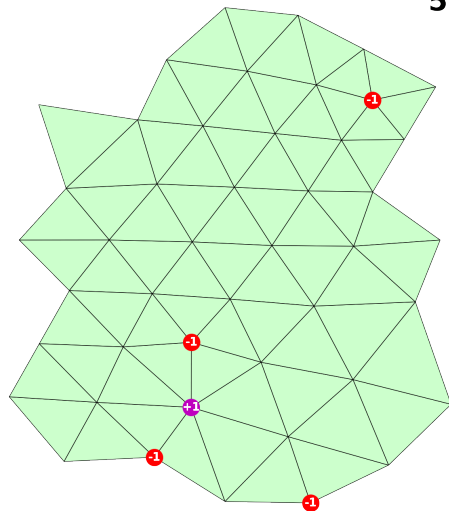
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 22 (out of 27)



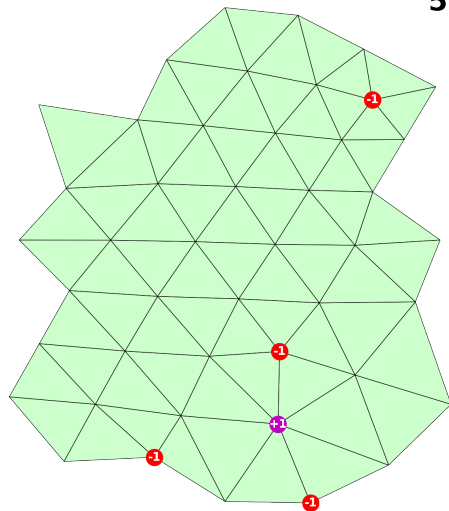
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 23 (out of 27)



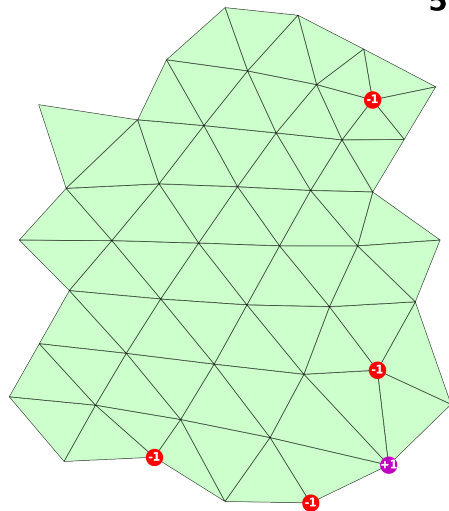
Triangular Mesh Optimization

Triangular meshing
Example 1
Step 24 (out of 27)

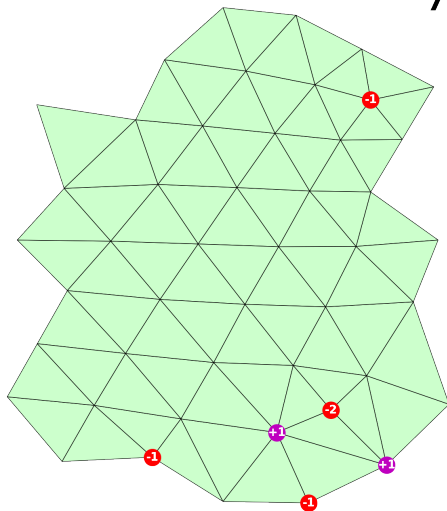


Triangular Mesh Optimization

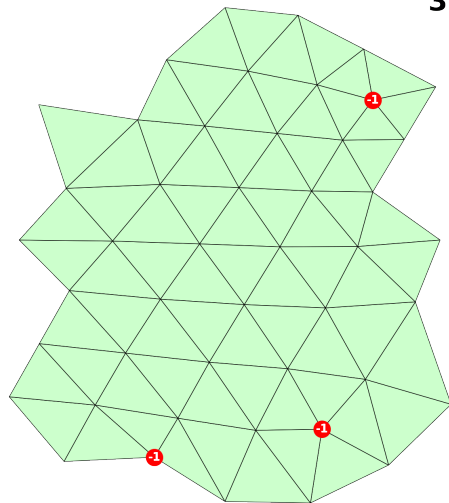
Triangular meshing
Example 1
Step 25 (out of 27)



Triangular meshing
Example 1
Step 26 (out of 27)

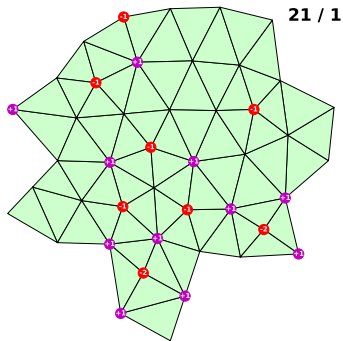


Triangular meshing
Example 1
Step 27 (out of 27)

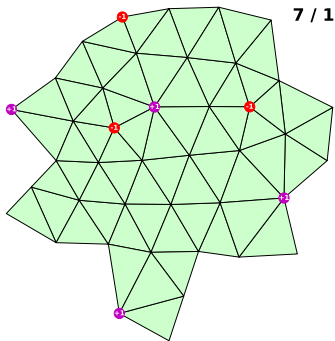


Triangular Mesh: One Complete Trajectory

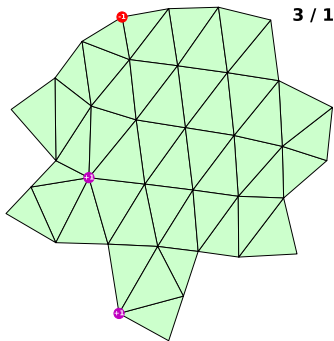
Three snapshots from a 20-sided polygon:



Initial



Intermediate



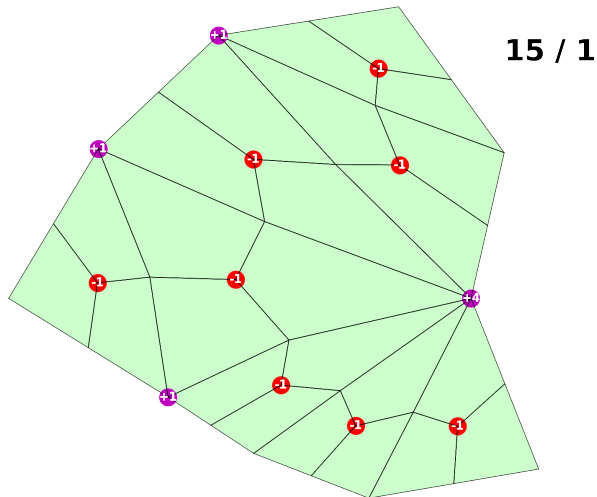
Final

Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 0 (out of 19)

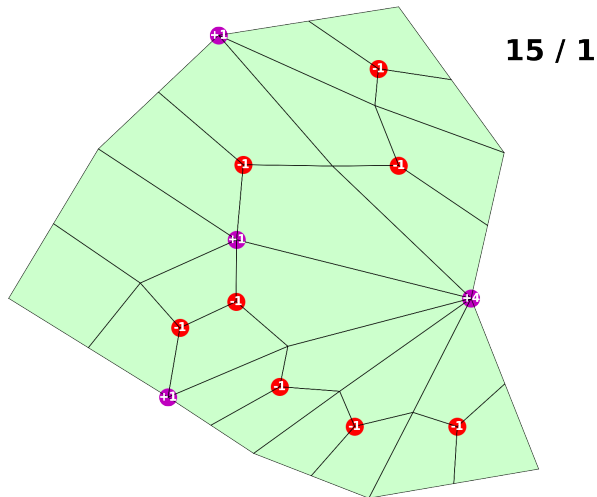


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 1 (out of 19)

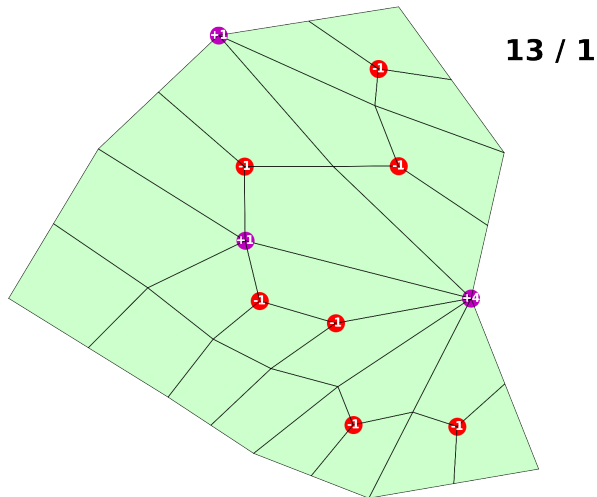


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 2 (out of 19)

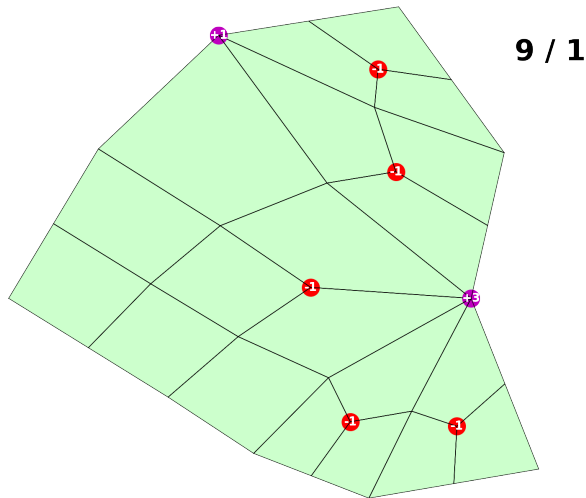


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 3 (out of 19)

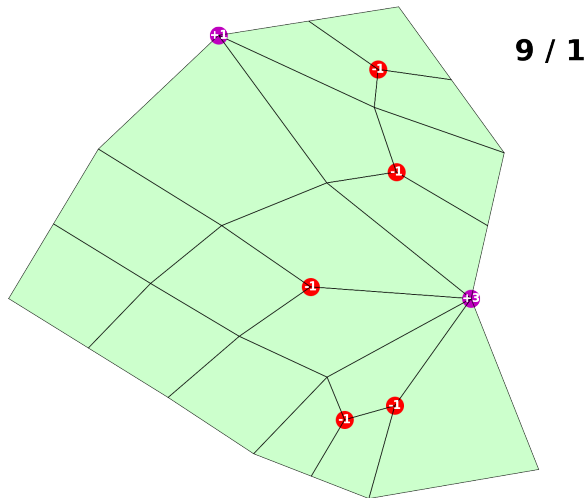


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 4 (out of 19)

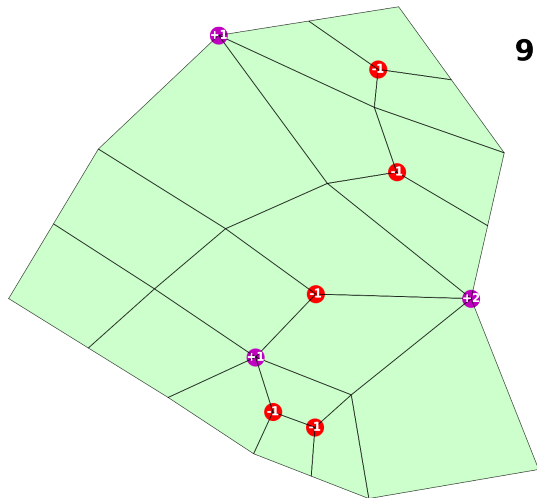


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 5 (out of 19)



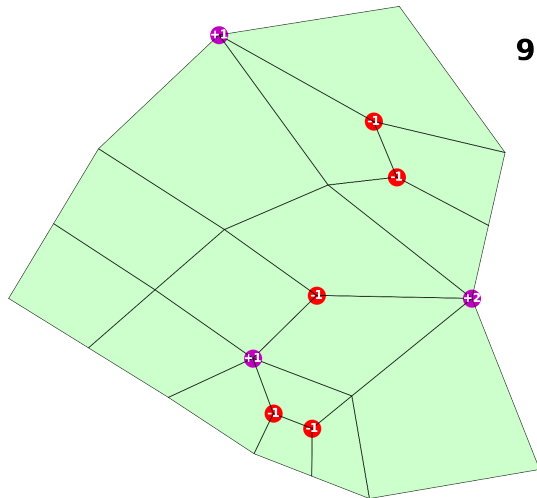
9 / 1

Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 6 (out of 19)



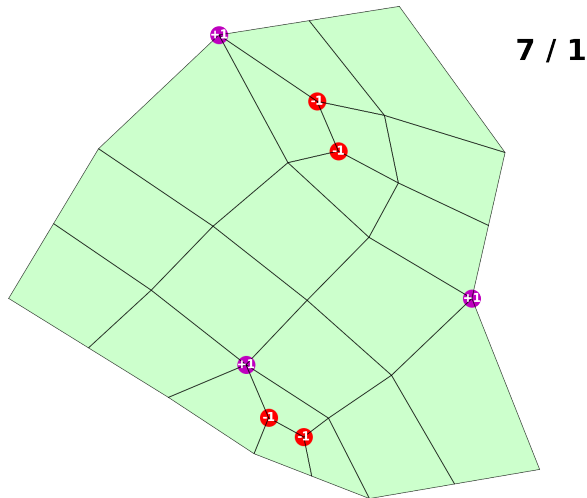
9 / 1

Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 7 (out of 19)

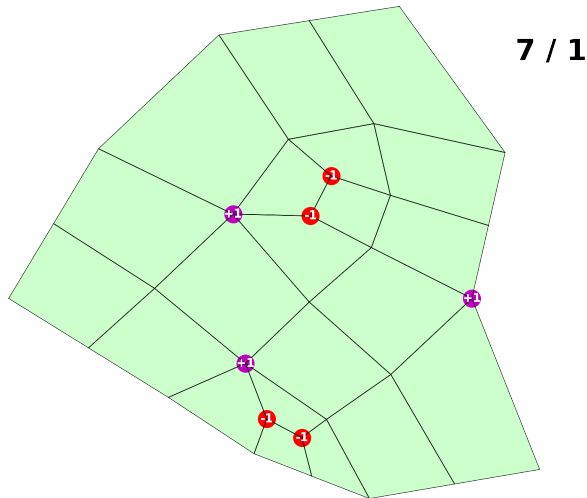


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 8 (out of 19)

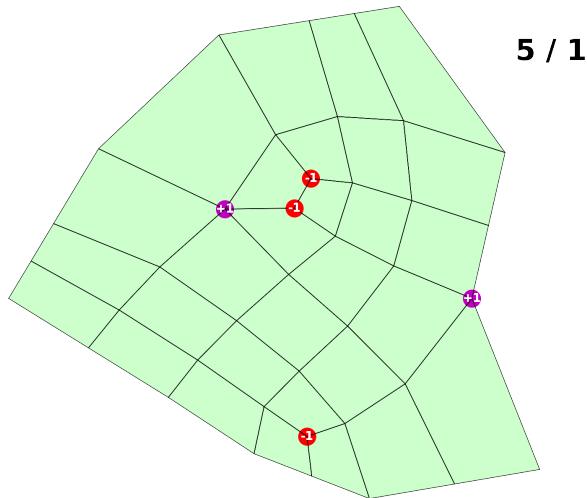


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 9 (out of 19)

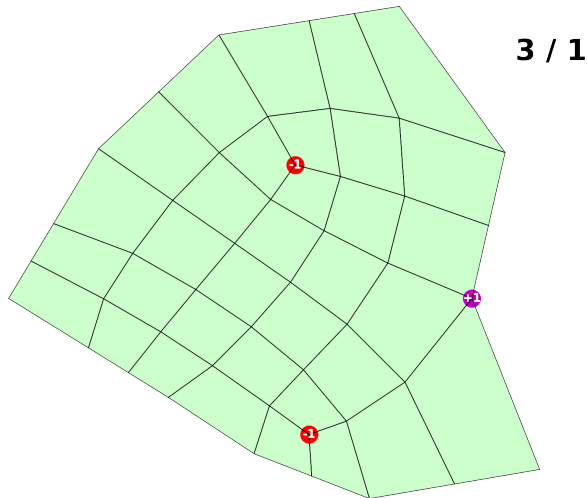


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 10 (out of 19)

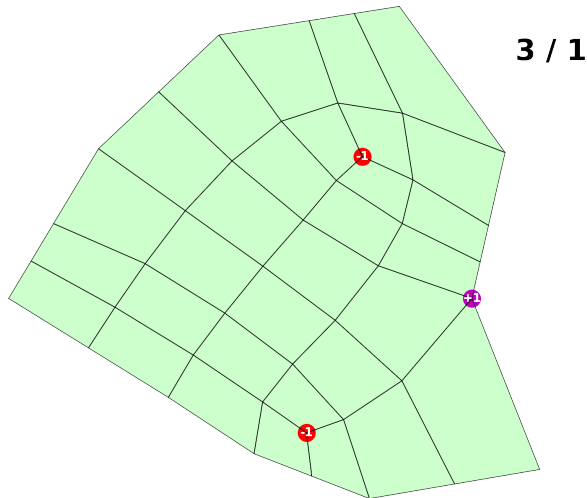


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 11 (out of 19)

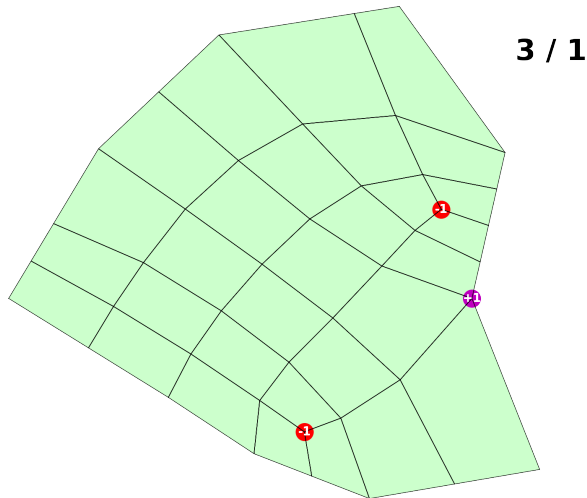


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 12 (out of 19)

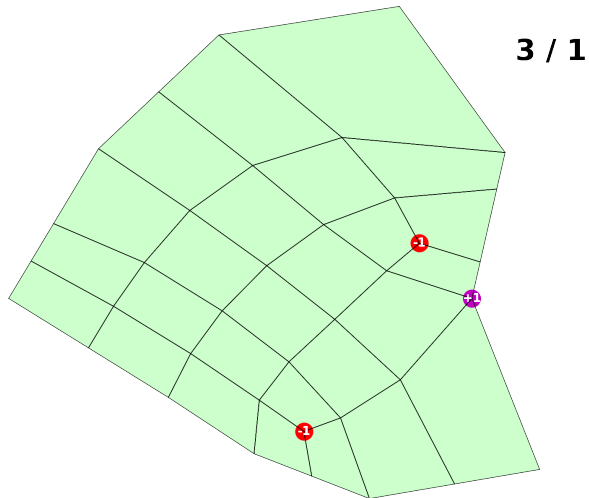


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 13 (out of 19)

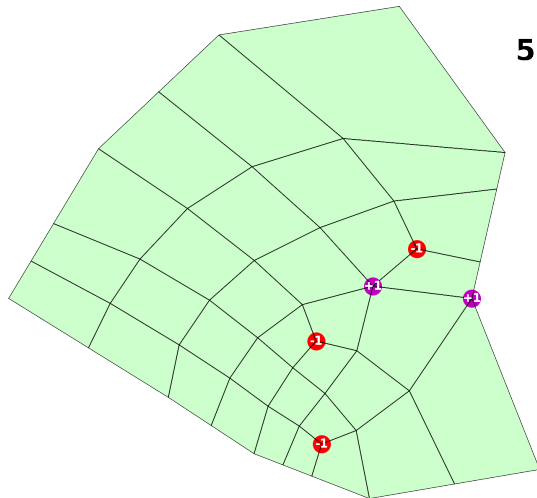


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 14 (out of 19)

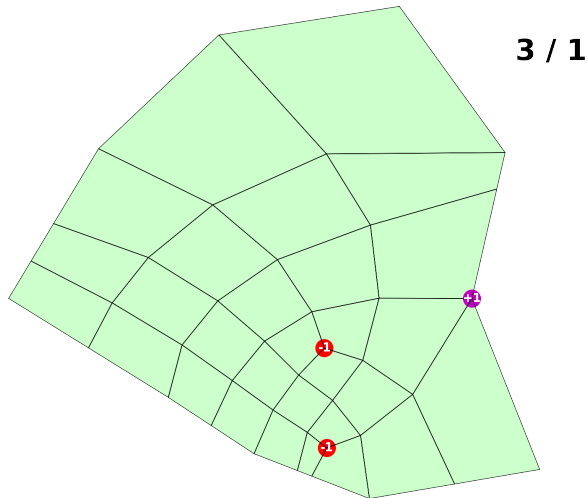


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 15 (out of 19)

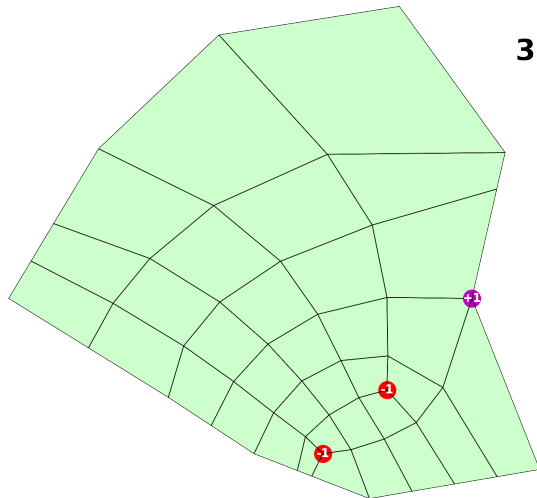


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 16 (out of 19)

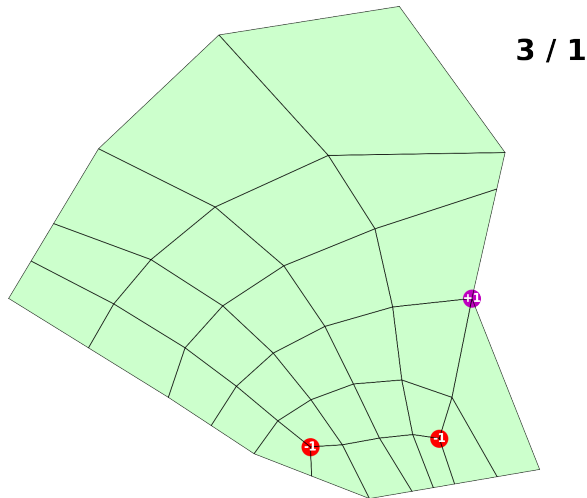


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 17 (out of 19)

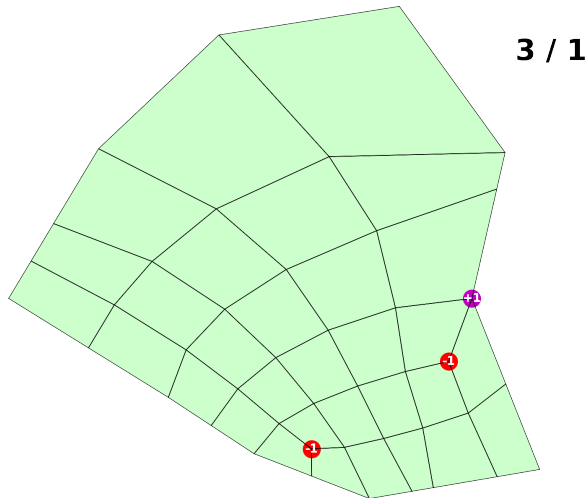


Quadrilateral Mesh Optimization

Block mesh decomposition

Example 1

Step 18 (out of 19)

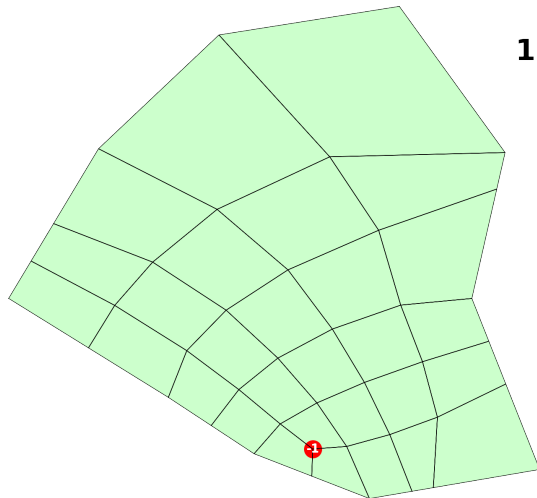


Quadrilateral Mesh Optimization

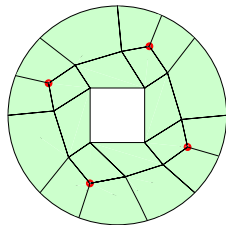
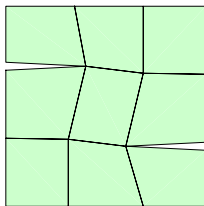
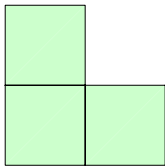
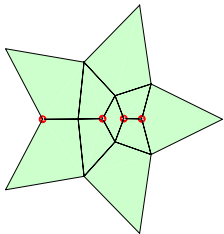
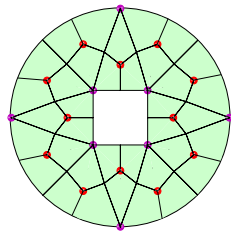
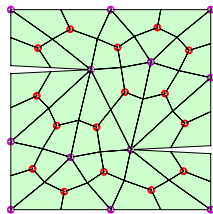
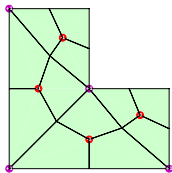
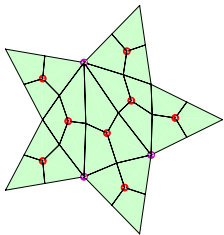
Block mesh decomposition

Example 1

Step 19 (out of 19)



Static Examples: More Complex Domains



Star-shaped domain

L-shaped domain

Double notch

Multiple boundaries

The same policy handles convex, non-convex, and multiply connected domains.

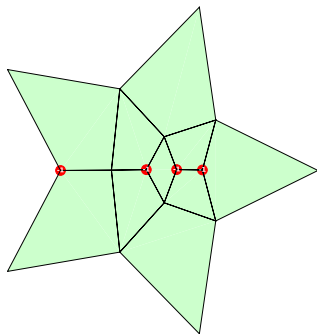
Part I Summary: Learned Topological Repair

What worked

- A mesh can be optimized by treating edits as actions in a game.
- Half-edge templates turn changing topology into fixed-size observations.
- The learned policy discovers long sequences and reaches close-to-optimal regularity.

What remains constrained

- The input must already be a valid triangle or quad mesh.
- The action set is specialized to each element type.
- Geometry is secondary and handled mainly through smoothing.



Outline

- 1 Introduction: Learning to Mesh
- 2 Part I: Learning to Repair Connectivity
- 3 Part II: Constructing Polygonal Meshes**
- 4 Part III: Learning Geometry and Node Placement
- 5 Part IV: Escaping Local Optima in 3D
- 6 Conclusions

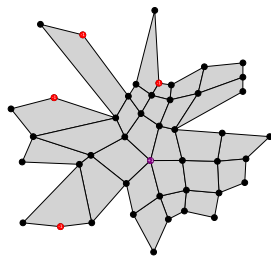
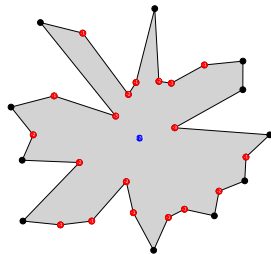
From Repairing a Mesh to Building One

Previous formulation: local repair

- Start from a triangle or quadrilateral mesh.
- Use element-specific operations.
- Primarily optimize vertex-degree regularity.

New formulation: constructive decomposition

- Start from the **boundary polygon itself**.
- Use one generic mesh-editing language.
- Request triangles, quads, hexagons, or another target face degree.
- Combine topological and geometric quality in the objective.



One Editing Language for Many Element Types

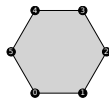
Four elementary DCEL edits

- **Edge insertion:** split a face into two.
- **Edge deletion:** merge adjacent faces.
- **Vertex insertion:** split an edge.
- **Vertex deletion:** remove a degree-2 vertex.

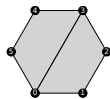
For target face degree f^* :

- $f^* = 3$: triangular decomposition,
- $f^* = 4$: quadrilateral decomposition,
- $f^* = 6$: hexagonal decomposition.

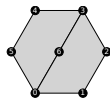
The operations stay fixed; only the objective and trained policy change.



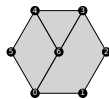
(a)



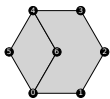
(b)



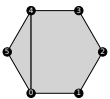
(c)



(d)



(e)



(f)

Rewarding Topology and Geometry

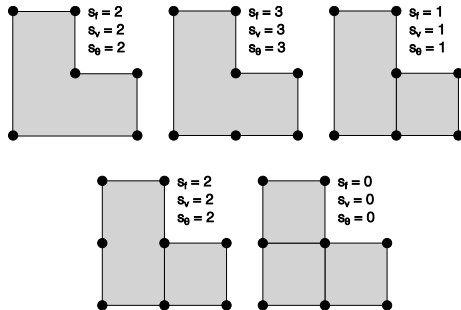
Three complementary scores

$$s_f = \sum_i |f_i - f^*|, \quad s_v = \sum_j |v_j - v_j^*|,$$
$$s_\theta = \sum_k \left| \frac{\theta_k - \theta^*}{\theta^*} \right|, \quad \theta^* = \frac{f^* - 2}{f^*} 180^\circ.$$

The combined objective is

$$s = w_f s_f + w_v s_v + w_\theta s_\theta.$$

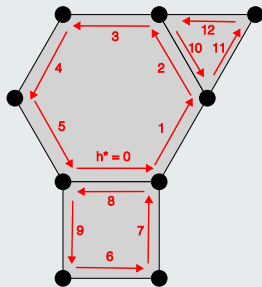
Local smoothing after each edit makes angle quality part of the environment transition.



Reusing the DCEL Idea with a Richer State

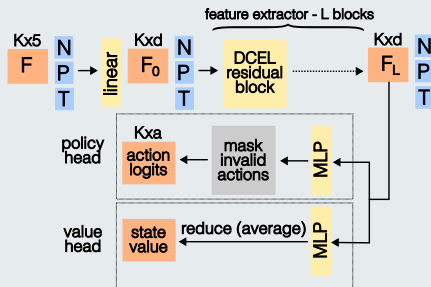
Local observation

- Select the half-edge with highest local irregularity.
- Build a K -nearest neighborhood by BFS.
- Record geometry, face degree, vertex degree, angle, and edge length.



Actor-critic network

- DCEL convolutions mix next, previous, and twin embeddings.
- Residual blocks increase the topological field of view.
- Invalid edits are masked before sampling.



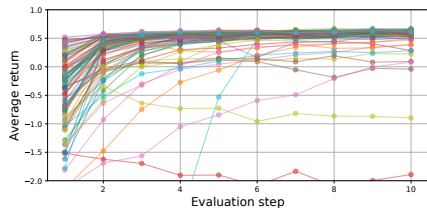
Training and Evaluation

Self-play on random polygons

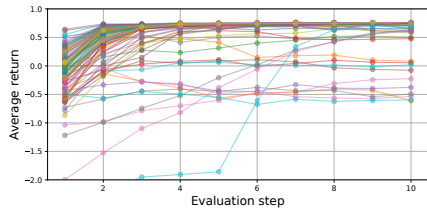
- Train a separate policy for each target degree f^* .
- Sample polygon size and geometry from a procedural distribution.
- Terminate on zero score or an edit limit.
- Tune PPO and network parameters with Optuna.
- Evaluate with best-of-30 stochastic rollouts.

What follows

- Colored markers identify remaining degree deviations.
- Clean animations show only the construction sequence.

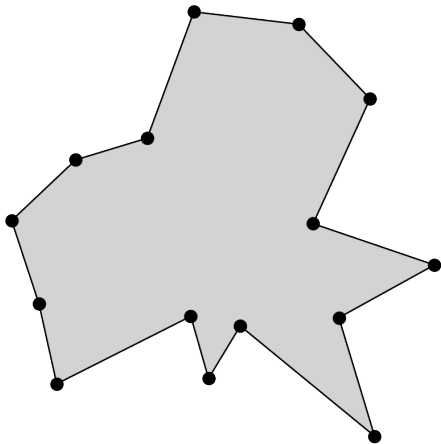


(a) $f^* = 4$

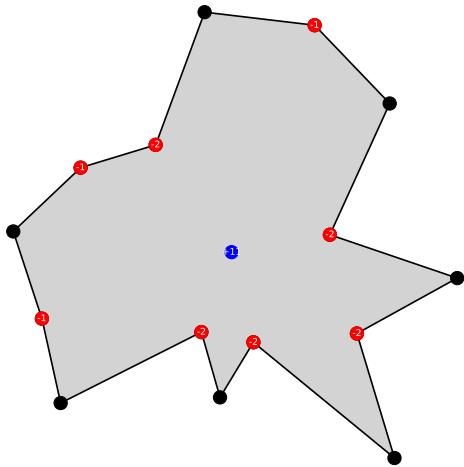


(b) $f^* = 3$

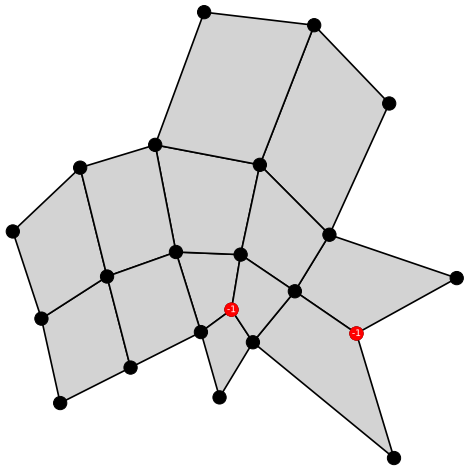
15-Sided Polygon to Quads: Construction



15-Sided Polygon to Quads: Result

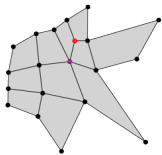


Boundary-only input

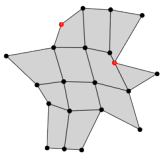


$$s_f = 0, s_v = 2, s_\theta = 10.4$$

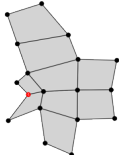
15-Sided Polygon to Quads: Sweep



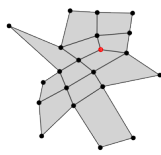
seed 01 $t=0$ $m=0$ $v=2$



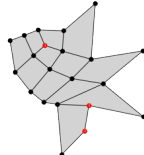
seed 02 $t=0$ $m=0$ $v=2$



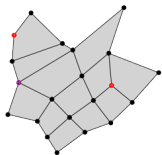
seed 03 $t=0$ $m=0$ $v=1$



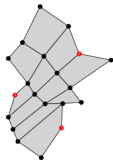
seed 04 $t=0$ $m=0$ $v=1$



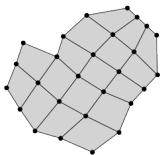
seed 05 $t=0$ $m=0$ $v=3$



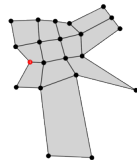
seed 06 $t=0$ $m=0$ $v=3$



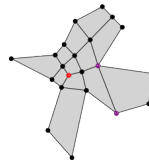
seed 07 $t=0$ $m=0$ $v=3$



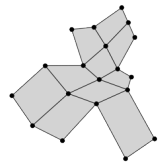
seed 08 $t=0$ $m=0$ $v=0$



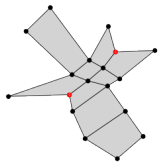
seed 09 $t=0$ $m=0$ $v=1$



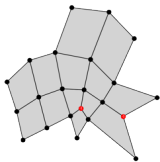
seed 10 $t=0$ $m=0$ $v=3$



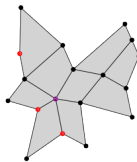
seed 11 $t=0$ $m=0$ $v=0$



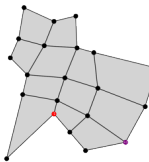
seed 12 $t=0$ $m=0$ $v=2$



seed 13 $t=0$ $m=0$ $v=2$

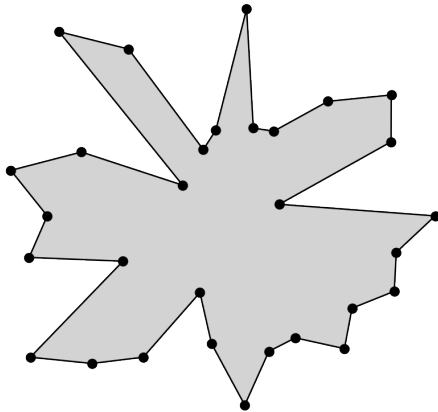


seed 14 $t=0$ $m=0$ $v=4$

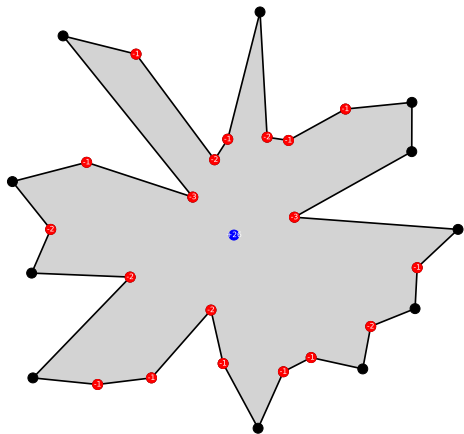


seed 15 $t=0$ $m=0$ $v=2$

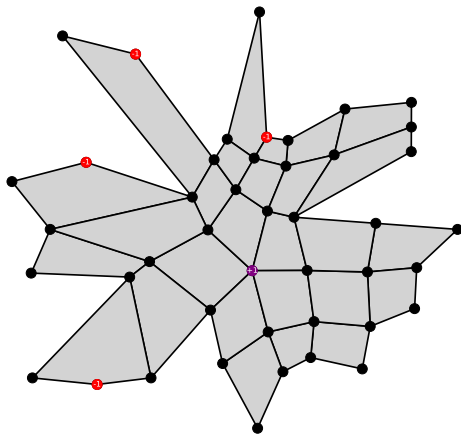
30-Sided Polygon to Quads: Construction



30-Sided Polygon to Quads: Result

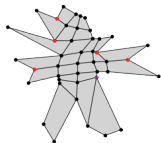


Larger input outside the easy regime

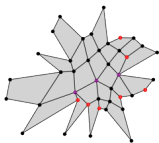


$$s_f = 0, s_v = 5, s_\theta = 26.5$$

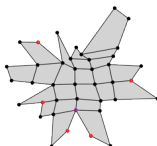
30-Sided Polygon to Quads: Sweep



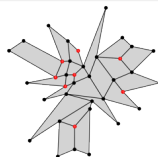
seed 01 $f=0$ $m=0$ $v=8$



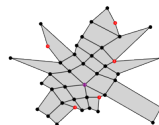
seed 02 $f=0$ $m=0$ $v=11$



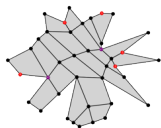
seed 03 $f=0$ $m=0$ $v=8$



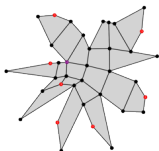
seed 04 $f=0$ $m=0$ $v=8$



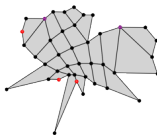
seed 05 $f=0$ $m=0$ $v=6$



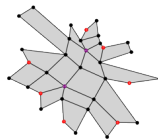
seed 06 $f=0$ $m=0$ $v=7$



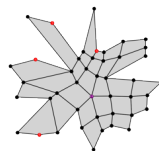
seed 07 $f=0$ $m=1$ $v=8$



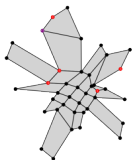
seed 08 $f=0$ $m=0$ $v=5$



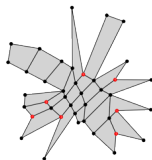
seed 09 $f=0$ $m=0$ $v=8$



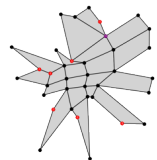
seed 10 $f=0$ $m=0$ $v=5$



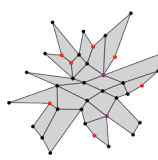
seed 11 $f=0$ $m=0$ $v=8$



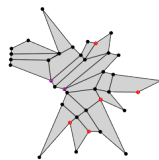
seed 12 $f=0$ $m=0$ $v=7$



seed 13 $f=0$ $m=1$ $v=8$

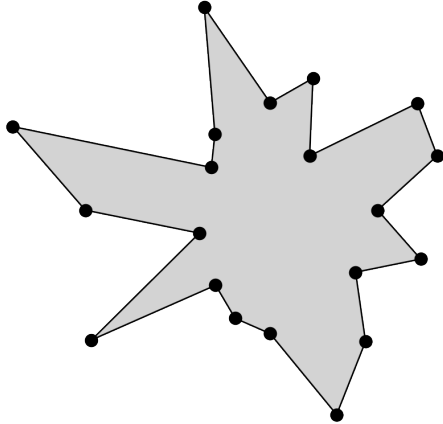


seed 14 $f=0$ $m=0$ $v=8$

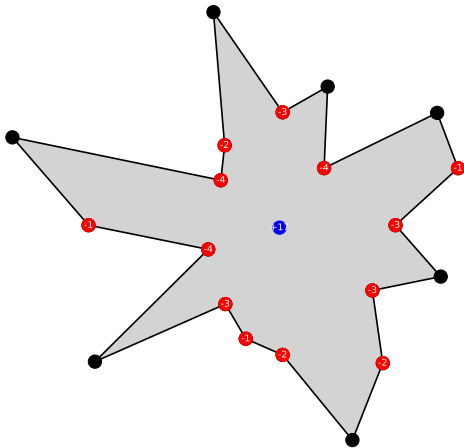


seed 15 $f=0$ $m=0$ $v=7$

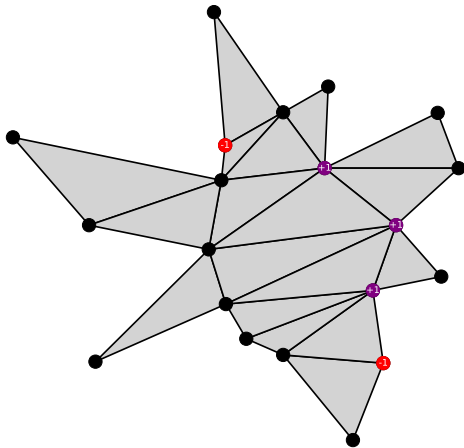
20-Sided Polygon to Triangles: Construction



20-Sided Polygon to Triangles: Result

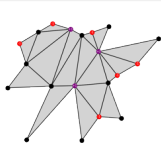


Boundary-only input

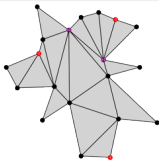


$$s_f = 0, s_v = 5, s_\theta = 22.1$$

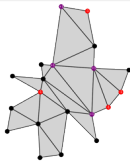
20-Sided Polygon to Triangles: Sweep



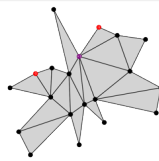
seed 01 $f=0$ $m=0$ $v=10$



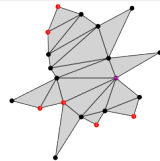
seed 02 $f=0$ $m=0$ $v=6$



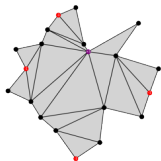
seed 03 $f=0$ $m=0$ $v=8$



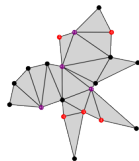
seed 04 $f=0$ $m=0$ $v=3$



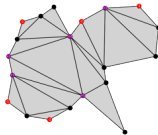
seed 05 $f=0$ $m=0$ $v=7$



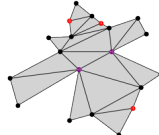
seed 06 $f=0$ $m=0$ $v=7$



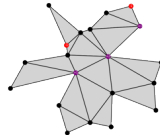
seed 07 $f=0$ $m=0$ $v=9$



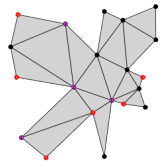
seed 08 $f=0$ $m=0$ $v=11$



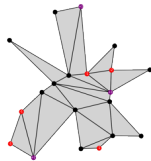
seed 09 $f=0$ $m=0$ $v=5$



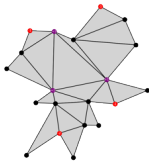
seed 10 $f=0$ $m=0$ $v=6$



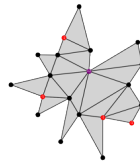
seed 11 $f=0$ $m=0$ $v=10$



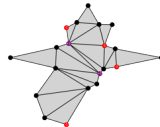
seed 12 $f=0$ $m=0$ $v=9$



seed 13 $f=0$ $m=0$ $v=8$

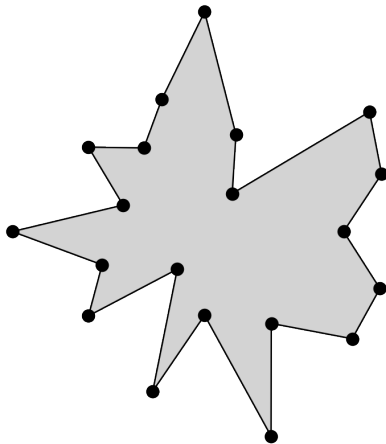


seed 14 $f=0$ $m=0$ $v=8$

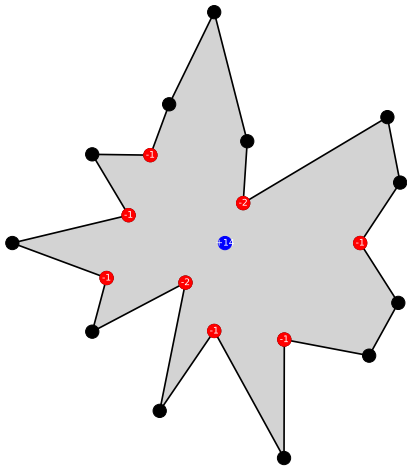


seed 15 $f=0$ $m=0$ $v=7$

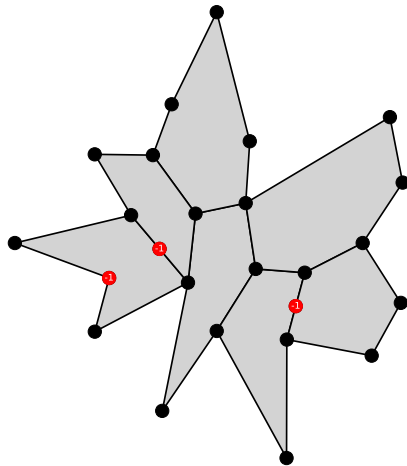
20-Sided Polygon to Hexagons: Construction



20-Sided Polygon to Hexagons: Result

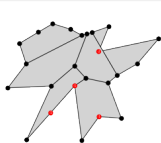


Boundary-only input

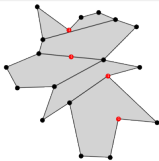


$$s_f = 0, s_v = 3, s_\theta = 14.4$$

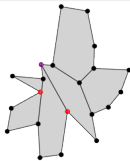
20-Sided Polygon to Hexagons: Sweep



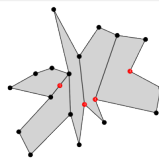
seed 01 $f=0$ $m=4$ $v=5$



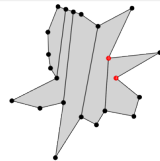
seed 02 $f=0$ $m=3$ $v=4$



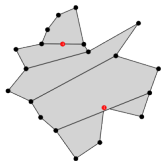
seed 03 $f=0$ $m=1$ $v=4$



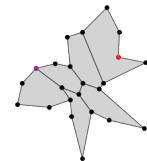
seed 04 $f=0$ $m=2$ $v=4$



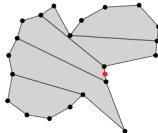
seed 05 $f=0$ $m=2$ $v=2$



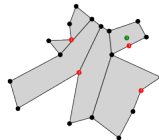
seed 06 $f=0$ $m=2$ $v=2$



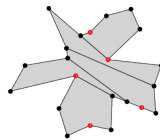
seed 07 $f=0$ $m=1$ $v=2$



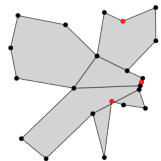
seed 08 $f=0$ $m=1$ $v=1$



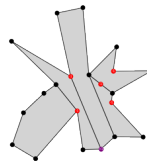
seed 09 $f=1$ $m=4$ $v=4$



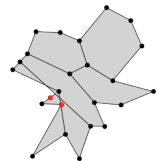
seed 10 $f=0$ $m=3$ $v=5$



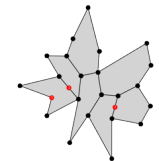
seed 11 $f=0$ $m=4$ $v=3$



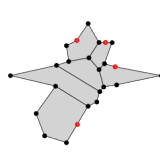
seed 12 $f=0$ $m=4$ $v=6$



seed 13 $f=0$ $m=2$ $v=2$



seed 14 $f=0$ $m=4$ $v=3$

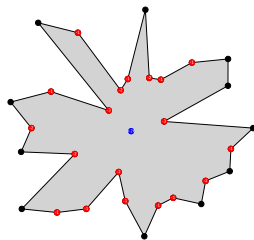


seed 15 $f=0$ $m=5$ $v=4$

Part II Summary: Constructive Decomposition

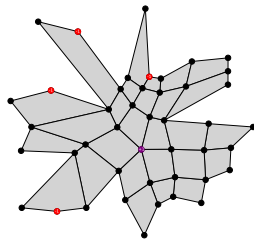
What changed

- The agent begins with the coarsest geometry, not a pre-existing mesh.
- One generic edit language supports multiple target element types.
- Face, vertex, and angle rewards couple topology with geometry.



What we learned

- Quadrilateral decompositions are consistently strong.
- Triangles are easy topologically, but the action bias limits geometric quality.
- Hexagons demonstrate flexibility but require stronger validity constraints.



Outline

- 1 Introduction: Learning to Mesh
- 2 Part I: Learning to Repair Connectivity
- 3 Part II: Constructing Polygonal Meshes
- 4 Part III: Learning Geometry and Node Placement**
- 5 Part IV: Escaping Local Optima in 3D
- 6 Conclusions

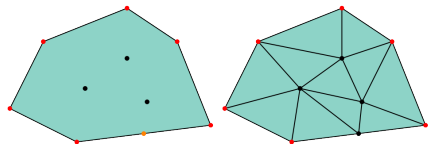
From Connectivity to Geometry

A complementary problem

- Previous agents primarily chose **connectivity**.
- Here the topology is recomputed by Delaunay triangulation.
- The policy learns where vertices should be placed and how many are needed.

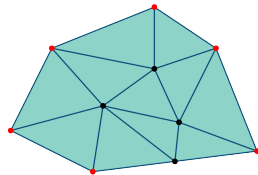
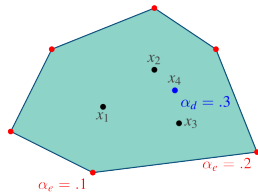
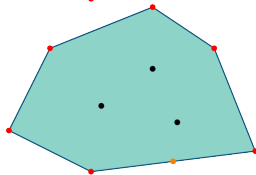
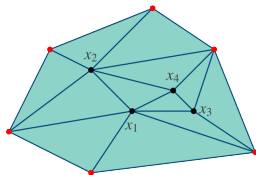
Available actions

- 1 Move an interior vertex.
- 2 Delete an interior vertex.
- 3 Insert at an edge midpoint.
- 4 Insert at a triangle centroid.



One Learned Mesh-Generation Step

- The state is the current vertex-edge graph.
- The stochastic policy proposes continuous position changes and discrete insert/delete events.
- Boundary vertices are fixed.
- Delaunay triangulation restores connectivity after every step.
- Repeating the step produces a mesh trajectory.



Geometric Quality Becomes the Reward

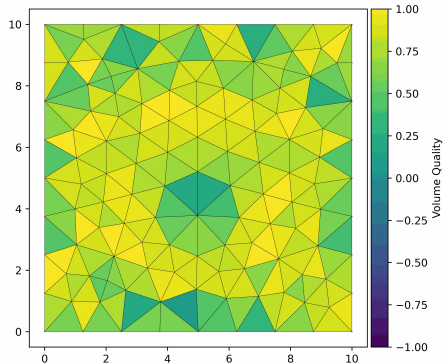
Four normalized quality measures:

- 1 Edge length q_e ,
- 2 angle quality q_a ,
- 3 element area q_v ,
- 4 shape quality $q_r = 2\rho_{\text{in}}/\rho_{\text{out}}$.

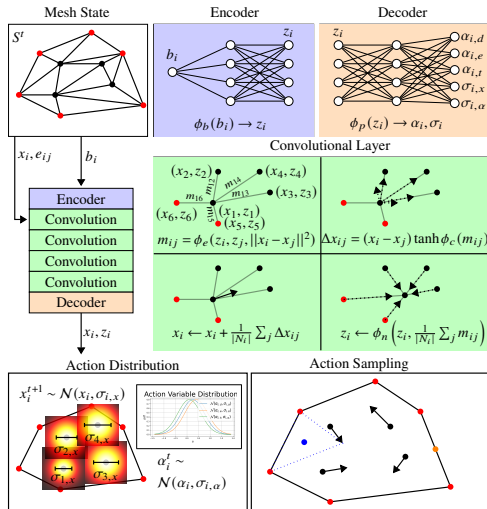
The score $S(s)$ is a weighted average, with reward

$$r_t = S(s_{t+1}) - S(s_t).$$

Therefore the episode return is the final improvement in mesh quality.



An Equivariant Mesh Generator



- Vertices are graph nodes; mesh edges carry messages.
- Position updates use relative coordinates and are translation- and rotation-equivariant.
- Interior equilateral regions are unchanged by construction.
- The position update is inspired by force-based DistMesh smoothing.
- PPO trains the stochastic edit policy and a shared-architecture critic.

Encoder, message passing, and action decoder

A Variable-Size Stochastic Action Space

The decoder predicts position and edit distributions at every vertex:

$$[x_i, \alpha_{i,d}, \alpha_{i,e}, \alpha_{i,t}, \sigma_{i,x}, \sigma_{i,\alpha}].$$

Continuous decision

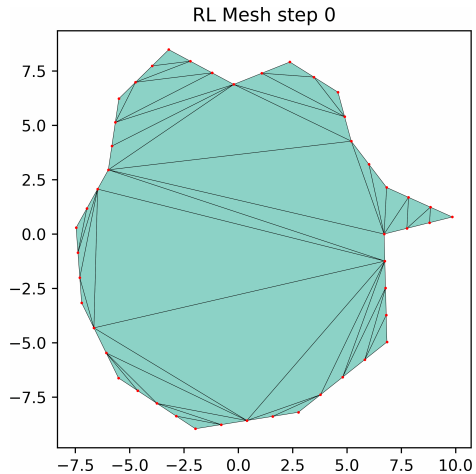
- Sample a new position for each movable vertex.
- Equivariance preserves the geometry of the problem.

Discrete decisions

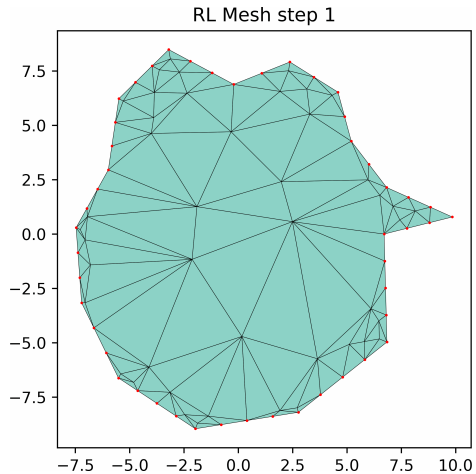
- $\alpha_{i,d} < \nu$: delete vertex i .
- $\alpha_{i,e} < \nu$: insert on an edge.
- $\alpha_{i,t} < \nu$: insert in a triangle.

The number of possible actions grows and shrinks with the mesh; the GNN shares parameters across all of them.

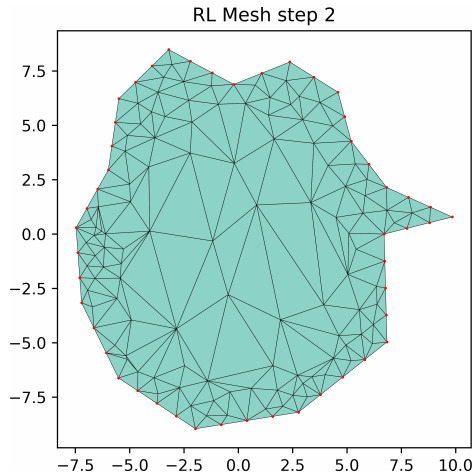
Learned Node Placement: Example 1



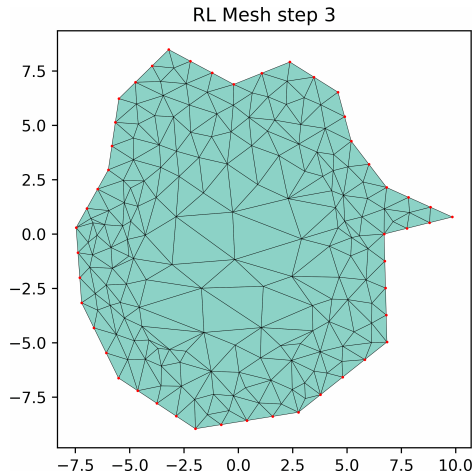
Learned Node Placement: Example 1



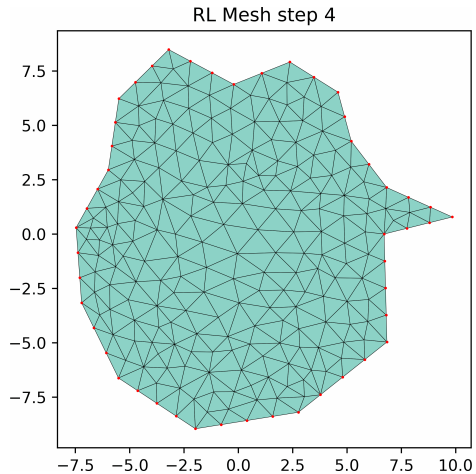
Learned Node Placement: Example 1



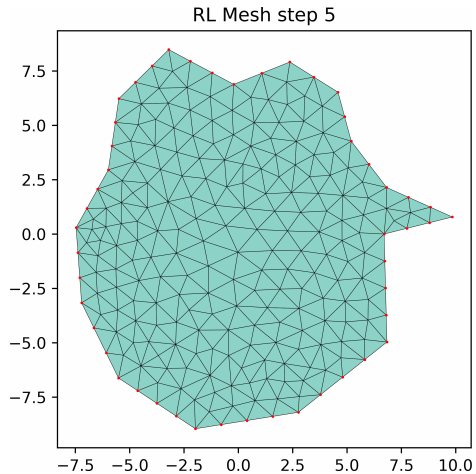
Learned Node Placement: Example 1



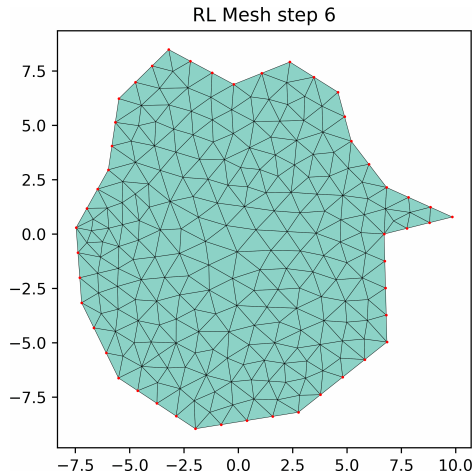
Learned Node Placement: Example 1



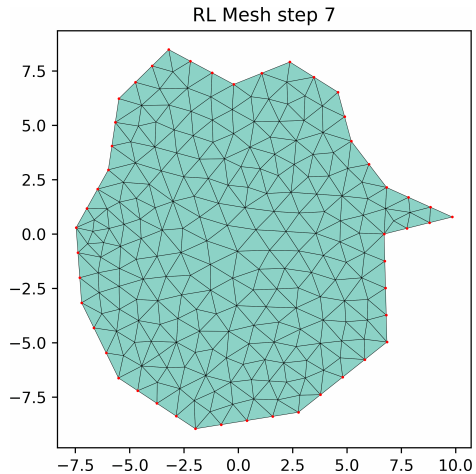
Learned Node Placement: Example 1



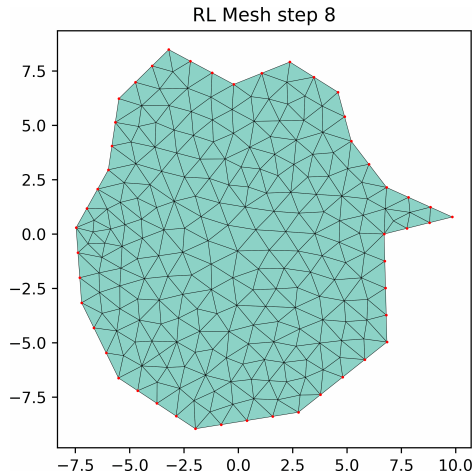
Learned Node Placement: Example 1



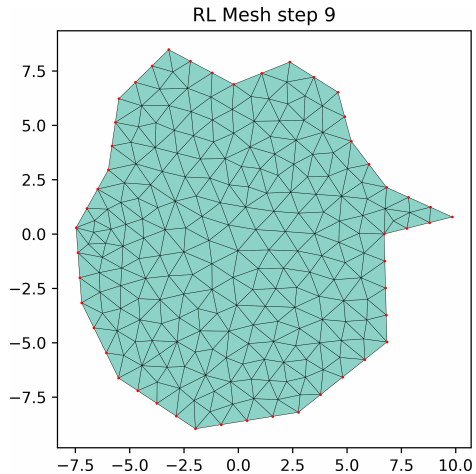
Learned Node Placement: Example 1



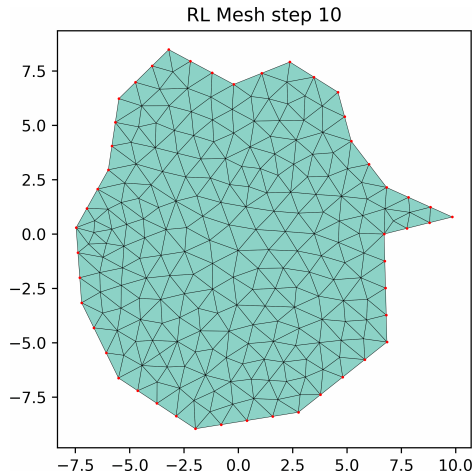
Learned Node Placement: Example 1



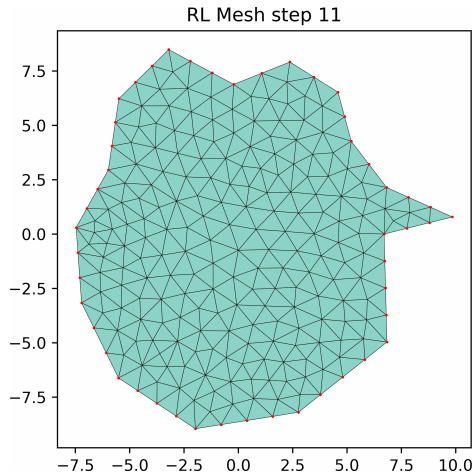
Learned Node Placement: Example 1



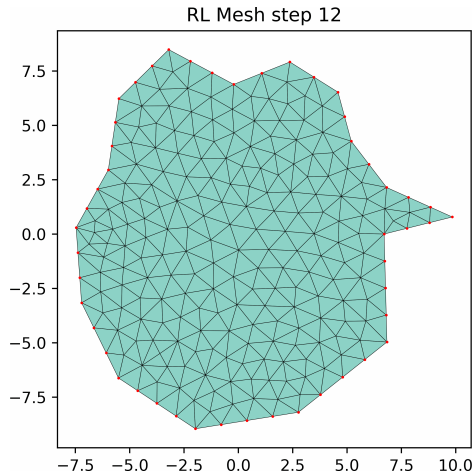
Learned Node Placement: Example 1



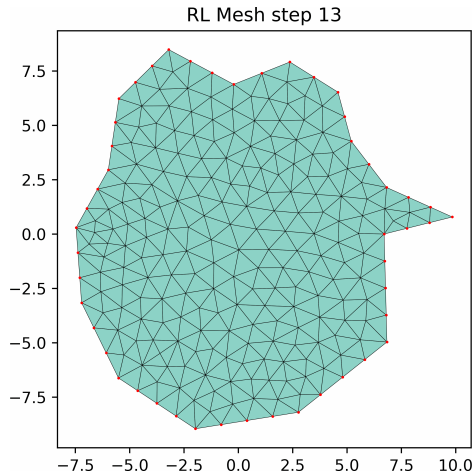
Learned Node Placement: Example 1



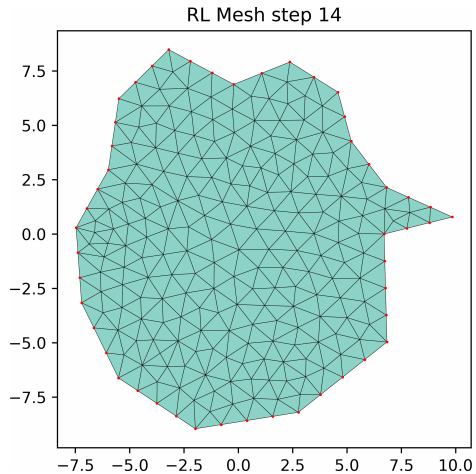
Learned Node Placement: Example 1



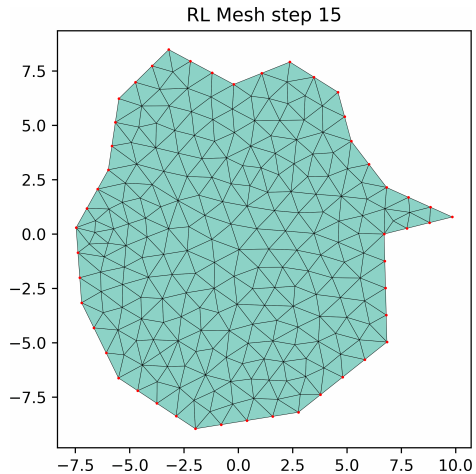
Learned Node Placement: Example 1



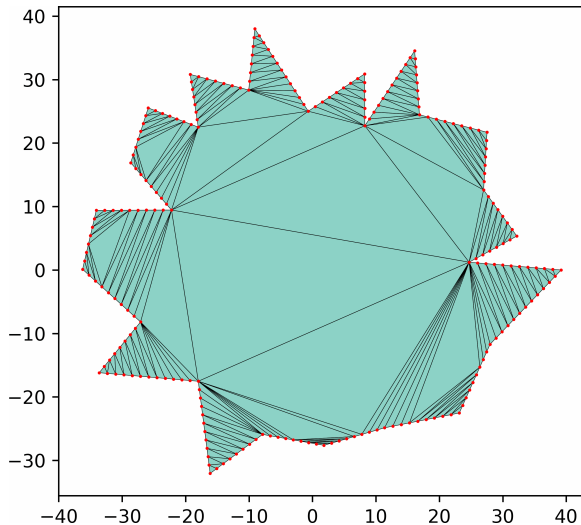
Learned Node Placement: Example 1



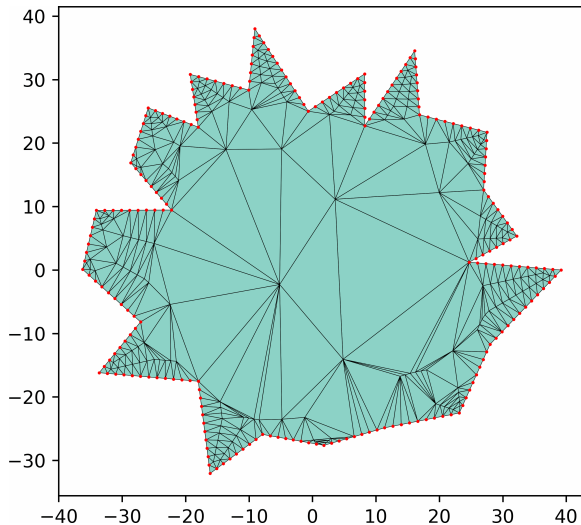
Learned Node Placement: Example 1



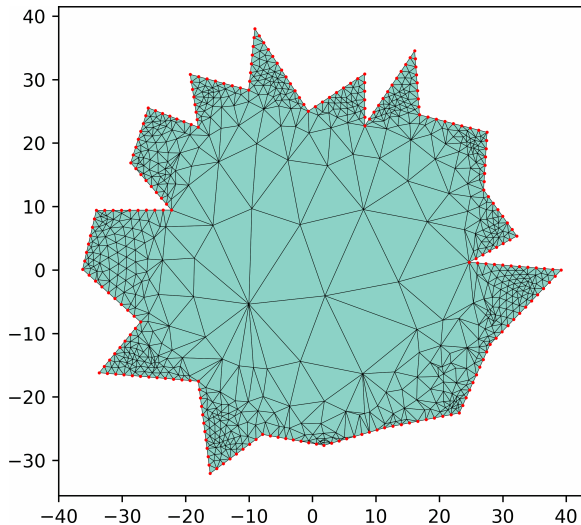
Learned Node Placement: Example 2



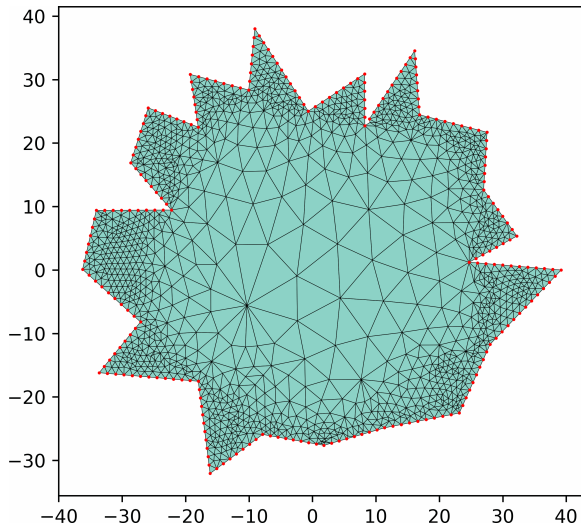
Learned Node Placement: Example 2



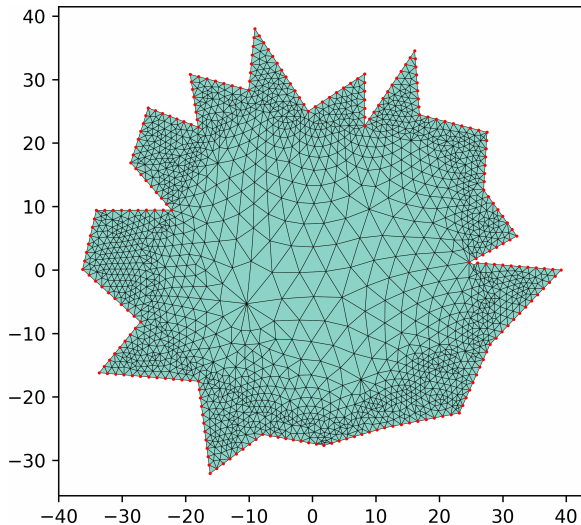
Learned Node Placement: Example 2



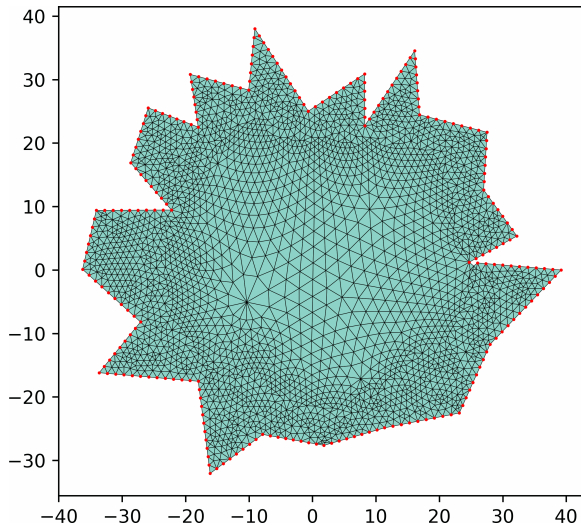
Learned Node Placement: Example 2



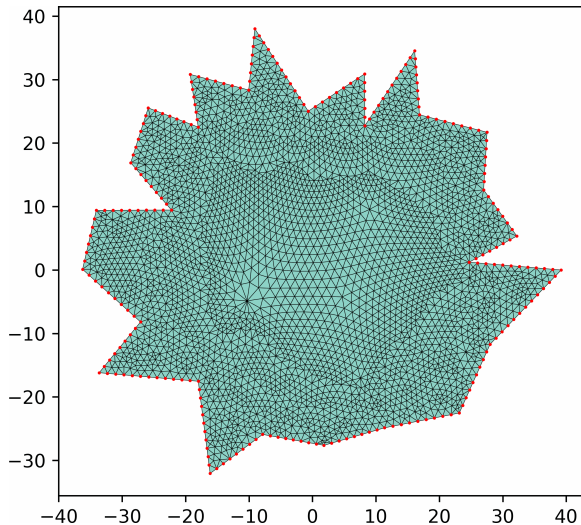
Learned Node Placement: Example 2



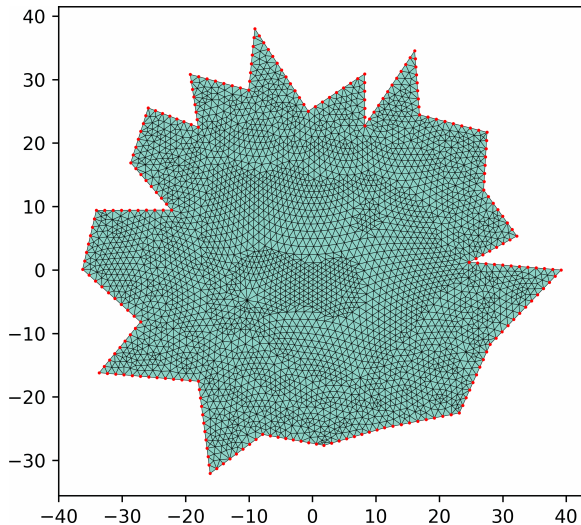
Learned Node Placement: Example 2



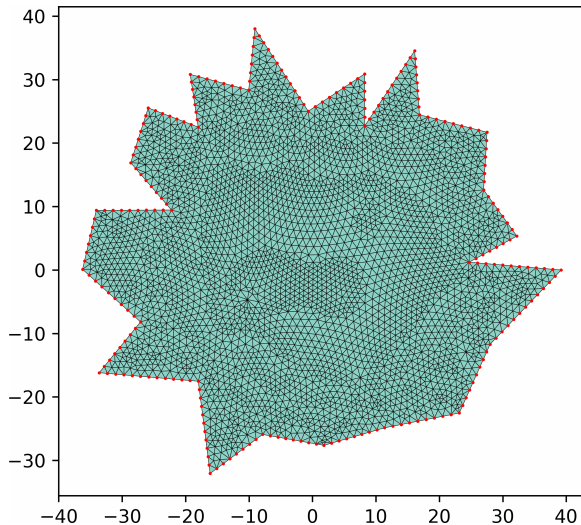
Learned Node Placement: Example 2



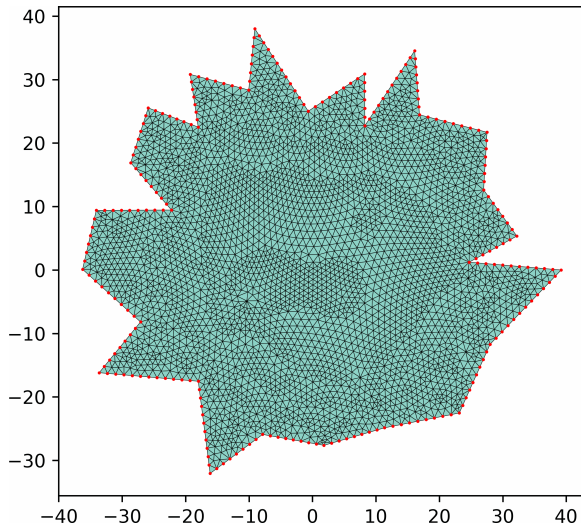
Learned Node Placement: Example 2



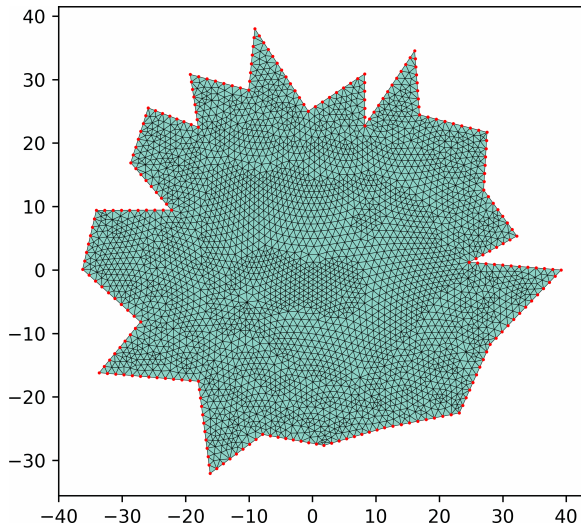
Learned Node Placement: Example 2



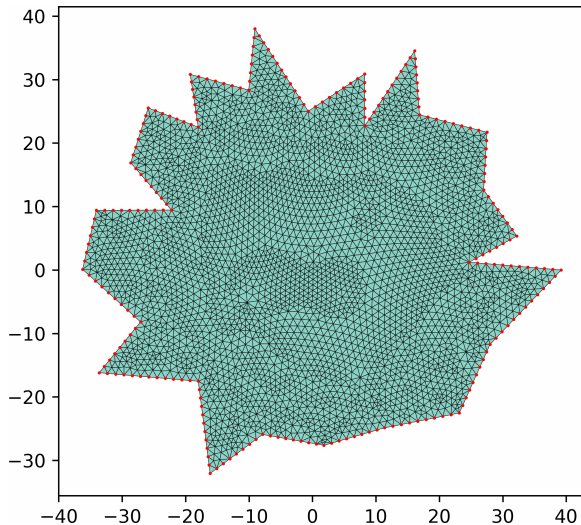
Learned Node Placement: Example 2



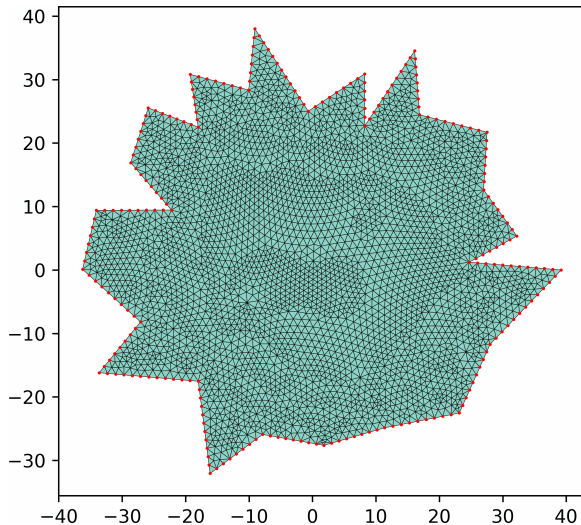
Learned Node Placement: Example 2



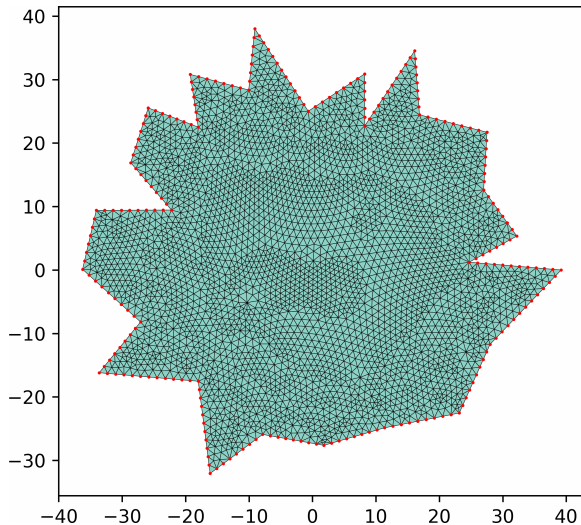
Learned Node Placement: Example 2



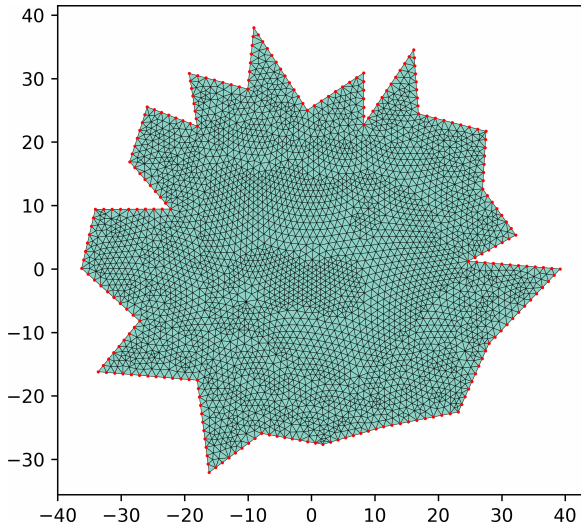
Learned Node Placement: Example 2



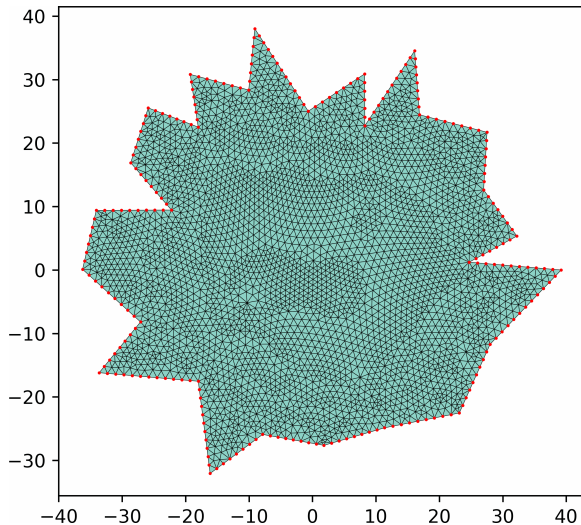
Learned Node Placement: Example 2



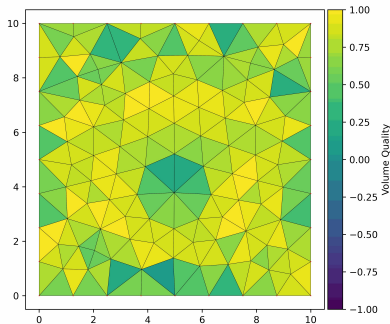
Learned Node Placement: Example 2



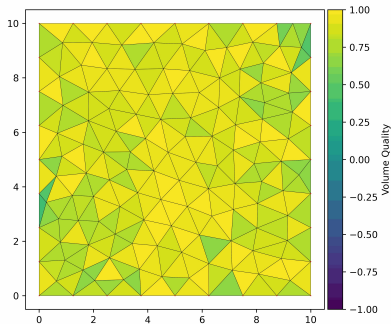
Learned Node Placement: Example 2



The Reward Controls the Result



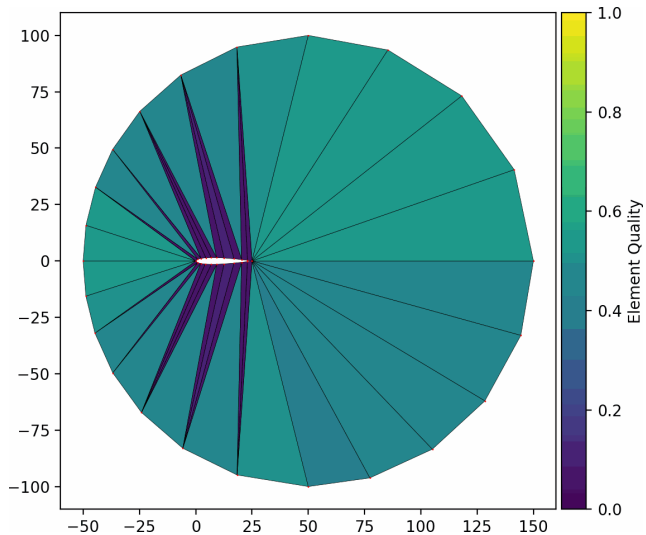
$w_a = w_e = 0.5$: good angles and edge lengths



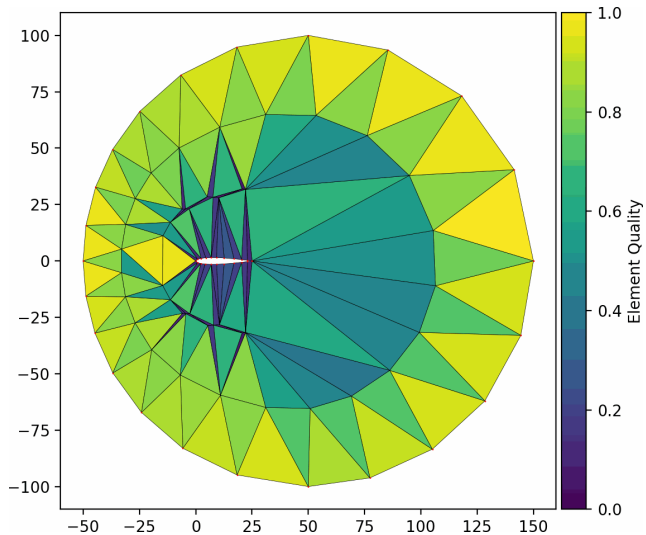
$w_v = 1$: more uniform element areas

- Tailoring the reward reliably changes the learned meshing preference.
- Stochastic evaluation can outperform deterministic action selection.
- No retraining labels are needed when the desired metric changes.

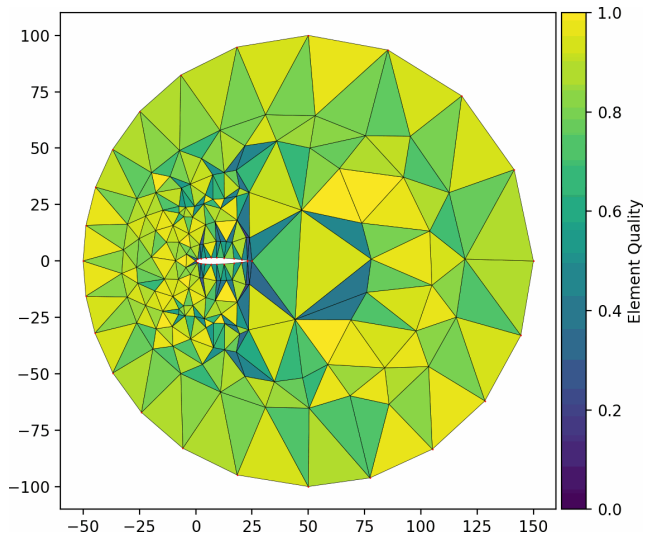
NACA Airfoil: Learned Construction



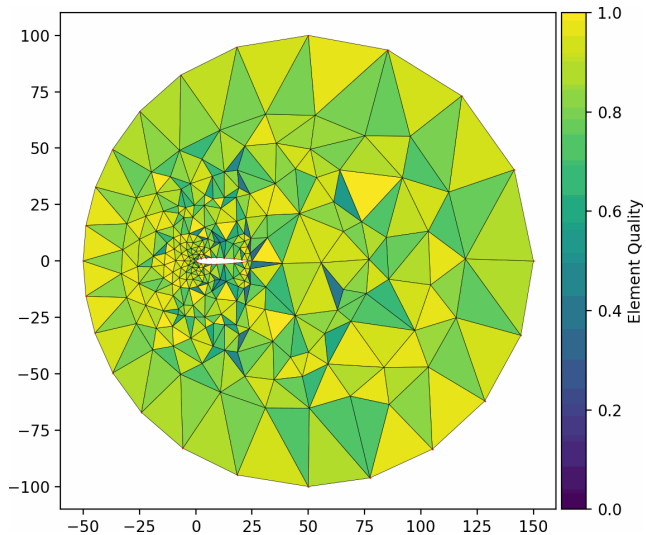
NACA Airfoil: Learned Construction



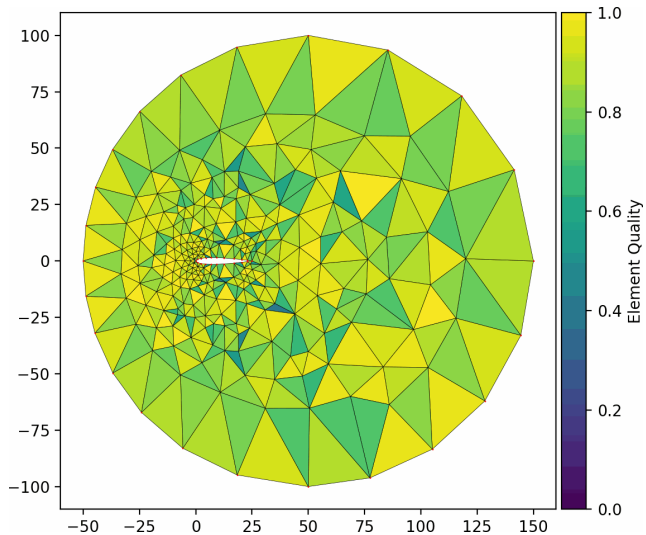
NACA Airfoil: Learned Construction



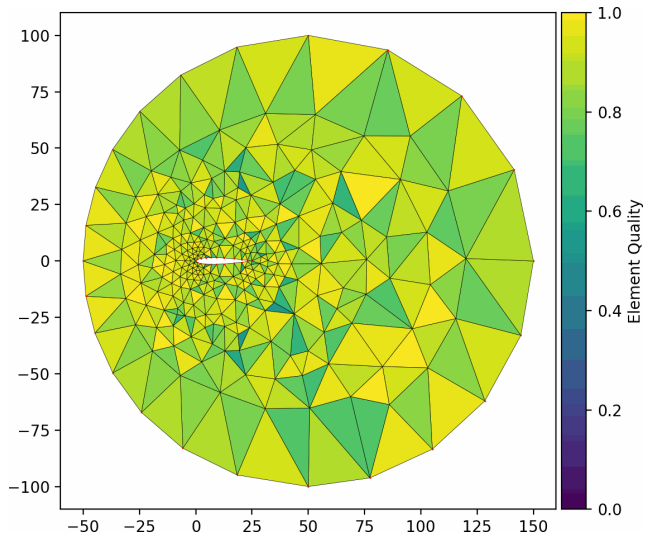
NACA Airfoil: Learned Construction



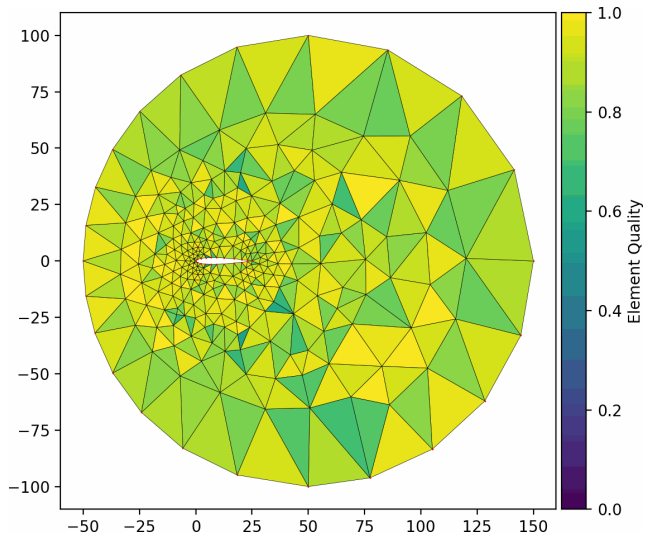
NACA Airfoil: Learned Construction



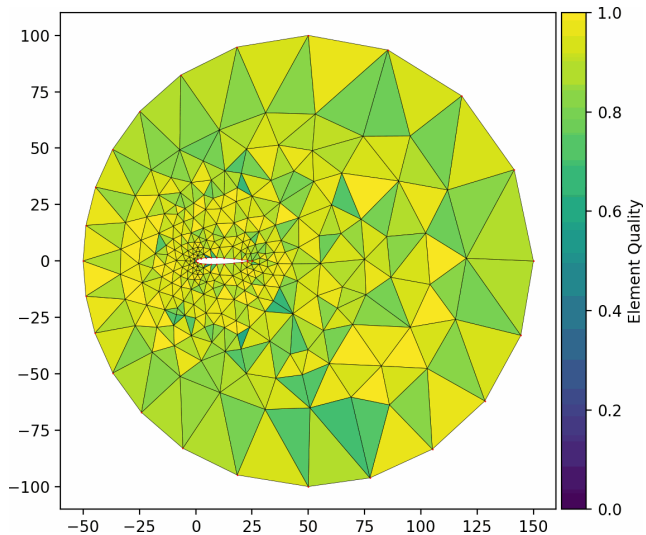
NACA Airfoil: Learned Construction



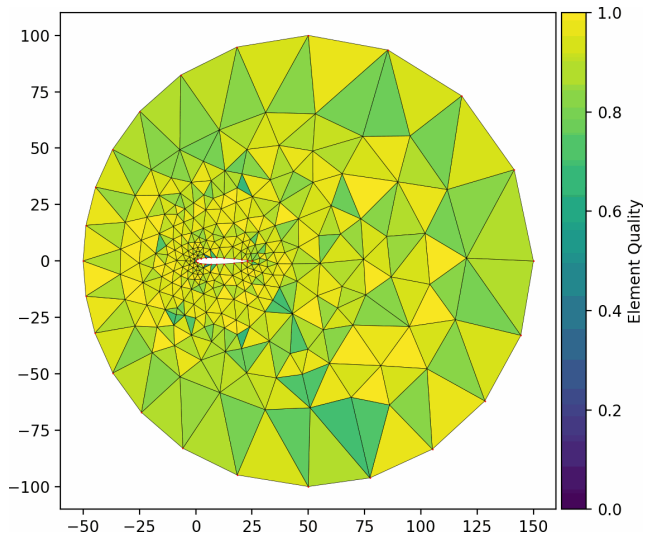
NACA Airfoil: Learned Construction



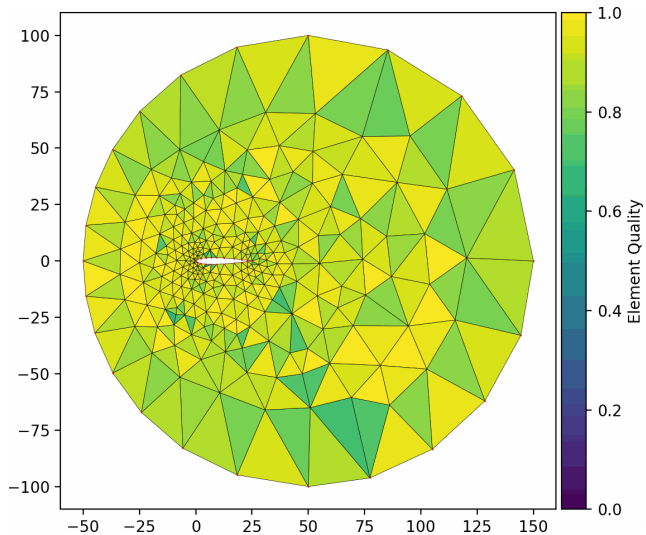
NACA Airfoil: Learned Construction



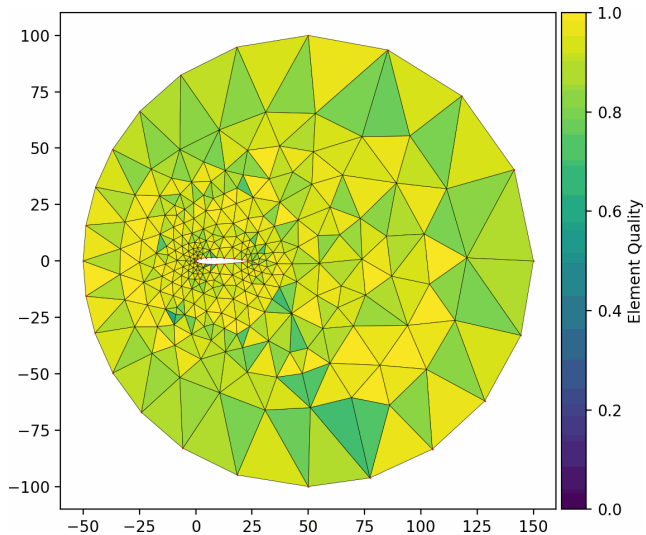
NACA Airfoil: Learned Construction



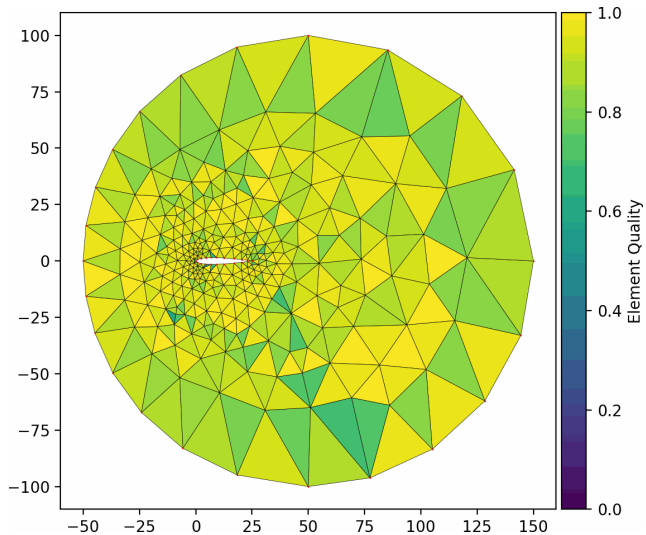
NACA Airfoil: Learned Construction



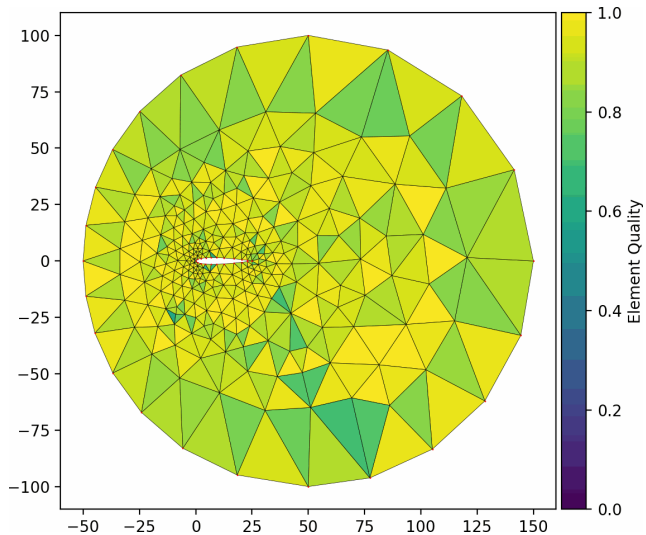
NACA Airfoil: Learned Construction



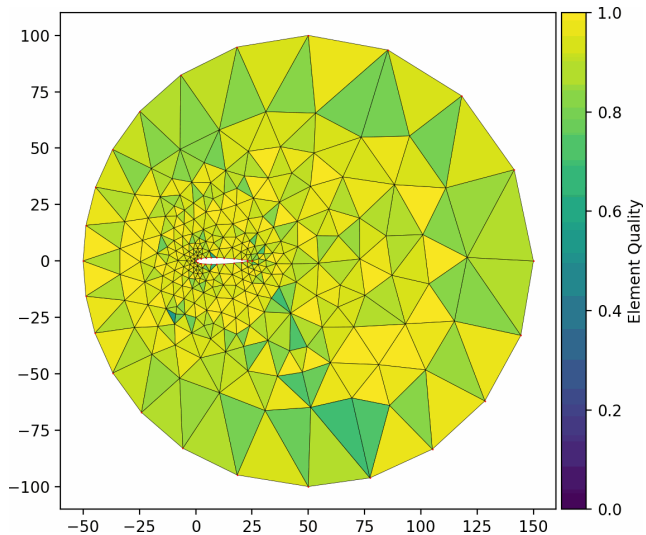
NACA Airfoil: Learned Construction



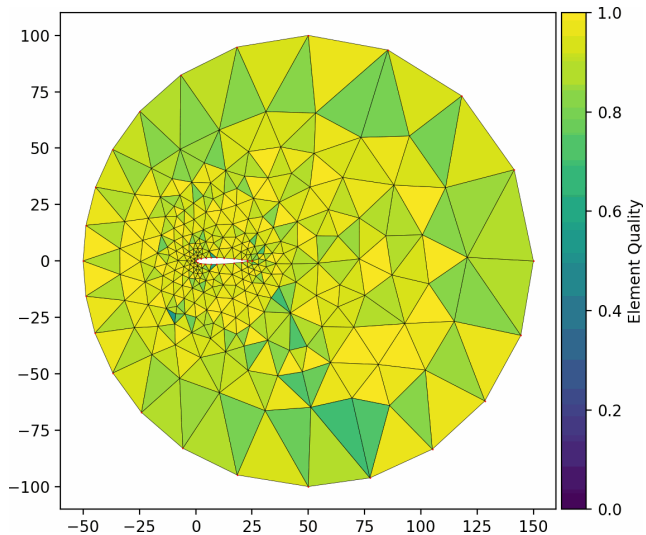
NACA Airfoil: Learned Construction



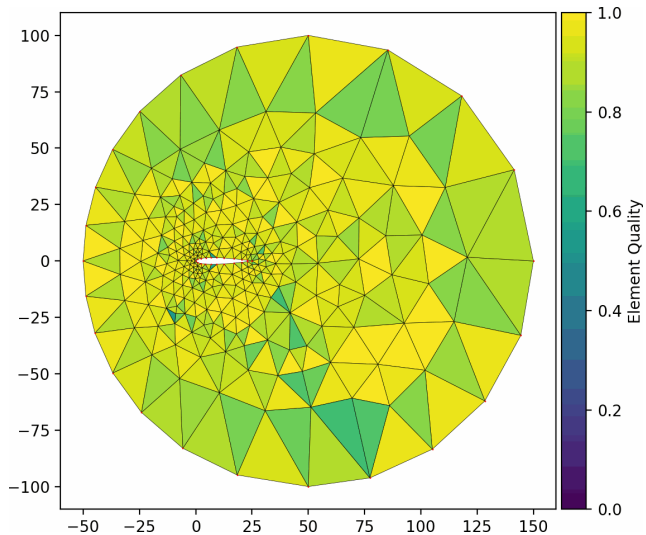
NACA Airfoil: Learned Construction



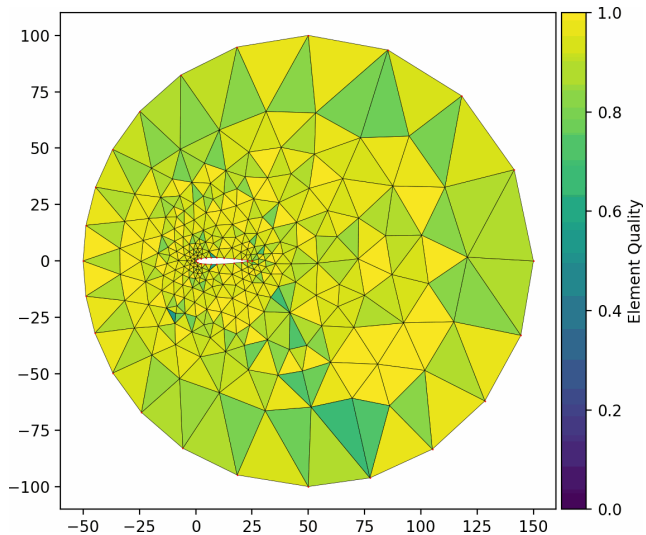
NACA Airfoil: Learned Construction



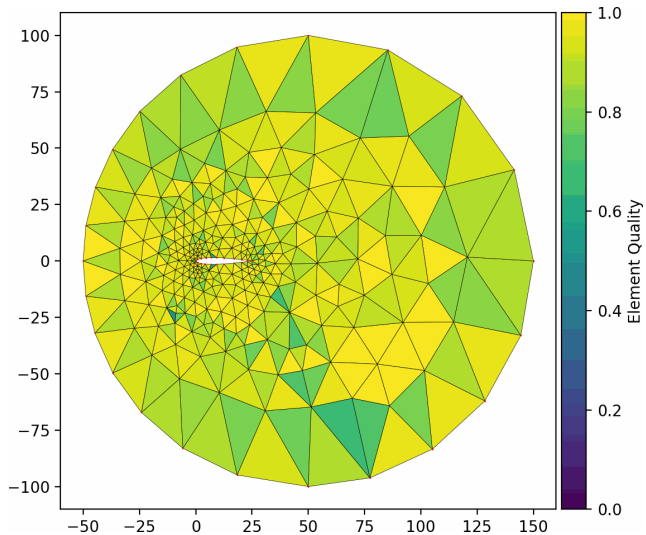
NACA Airfoil: Learned Construction



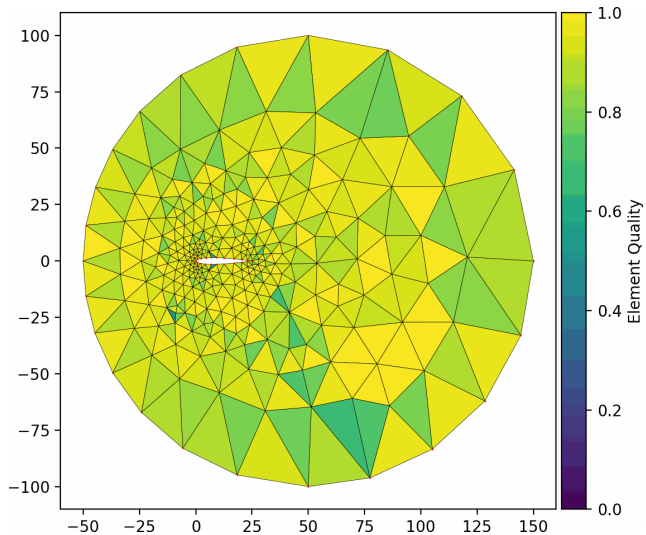
NACA Airfoil: Learned Construction



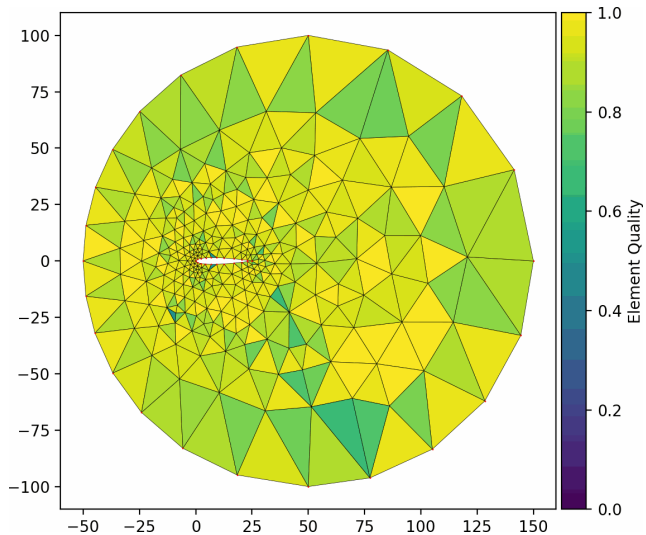
NACA Airfoil: Learned Construction



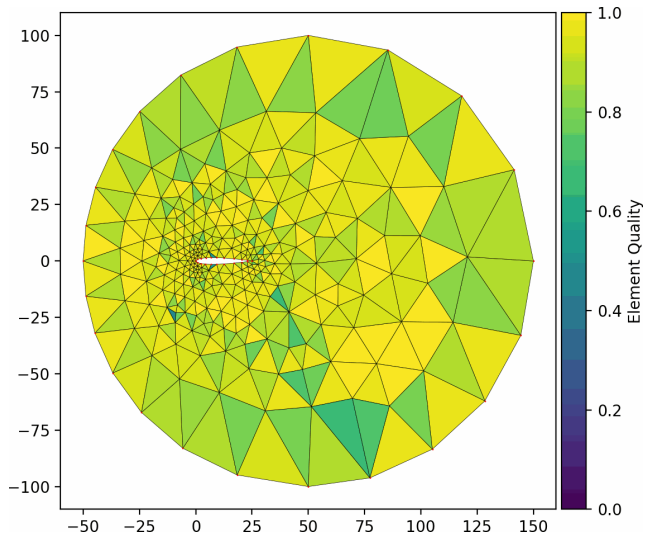
NACA Airfoil: Learned Construction



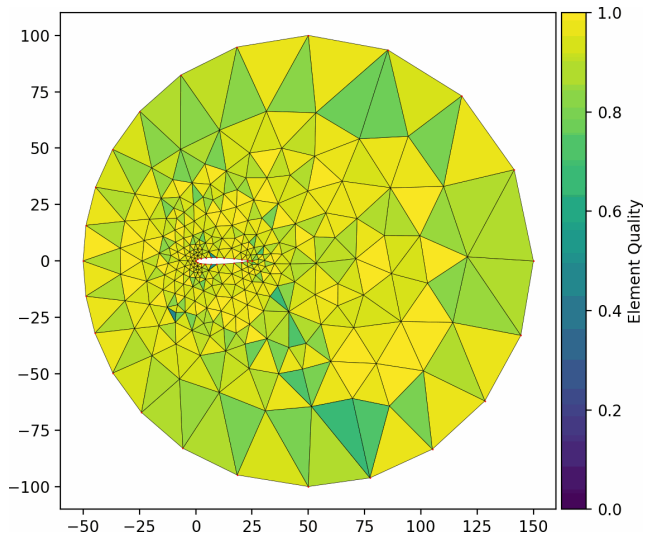
NACA Airfoil: Learned Construction



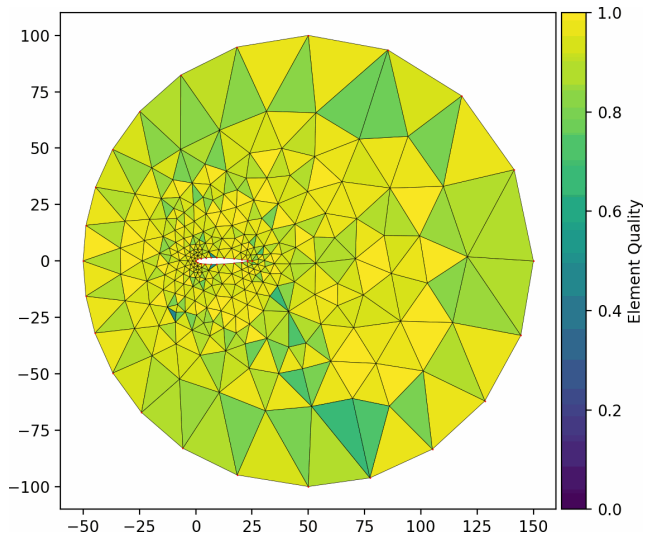
NACA Airfoil: Learned Construction



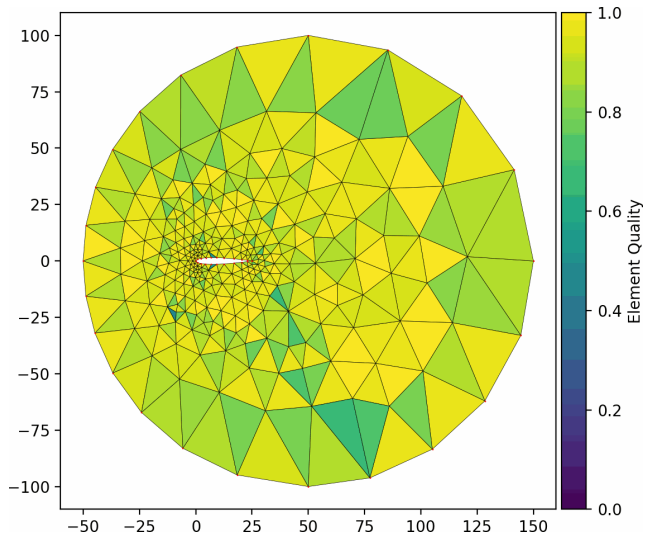
NACA Airfoil: Learned Construction



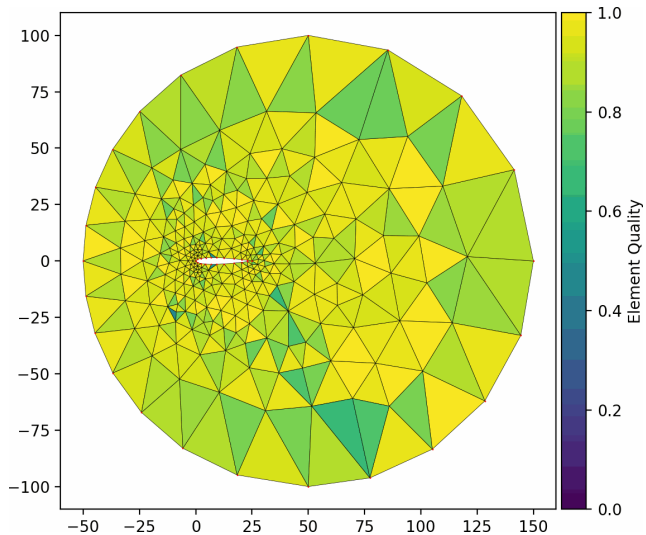
NACA Airfoil: Learned Construction



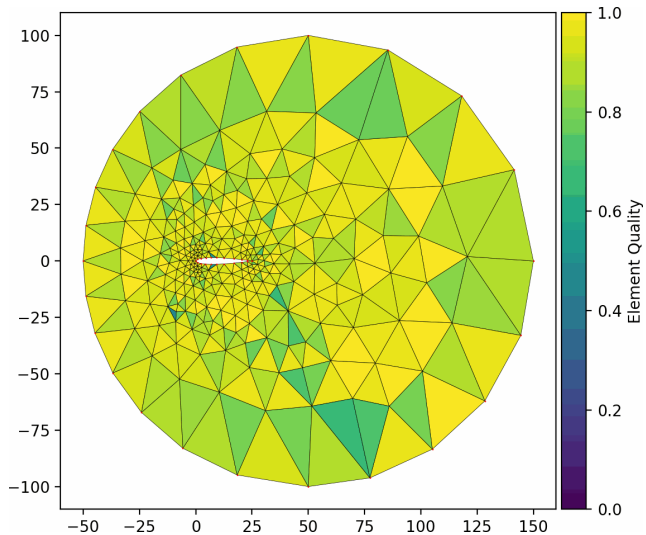
NACA Airfoil: Learned Construction



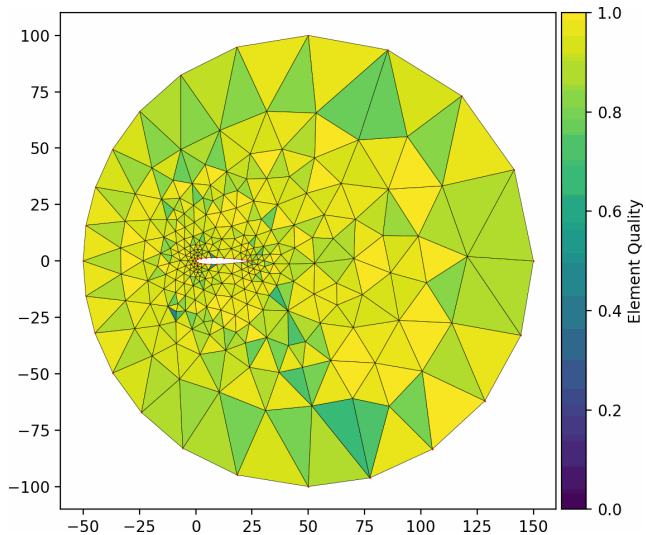
NACA Airfoil: Learned Construction



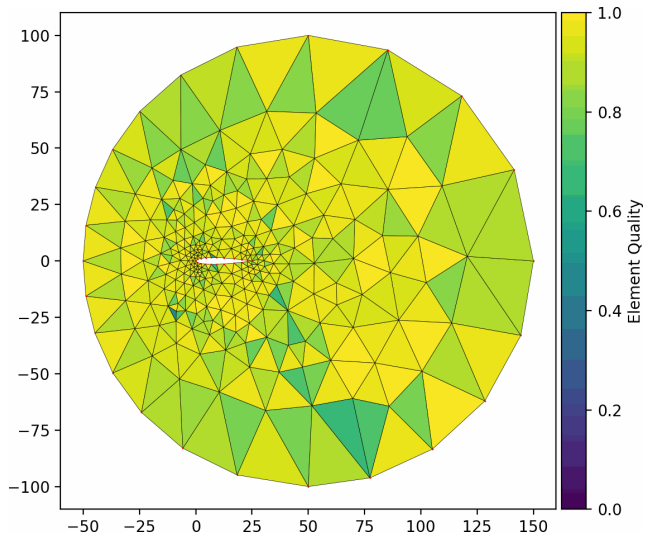
NACA Airfoil: Learned Construction



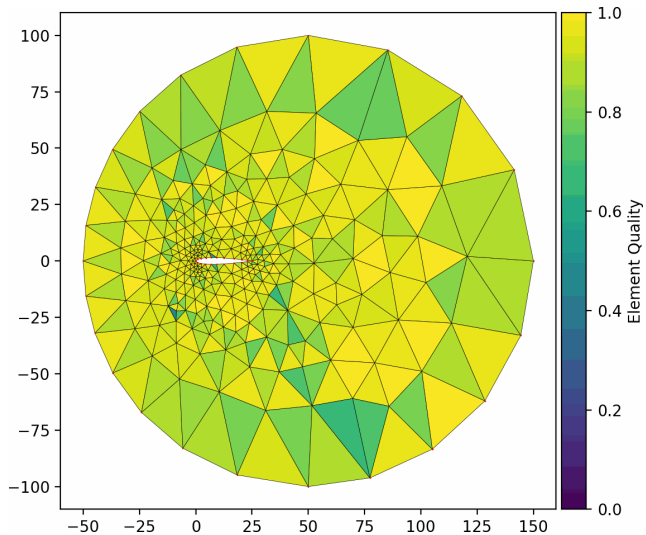
NACA Airfoil: Learned Construction



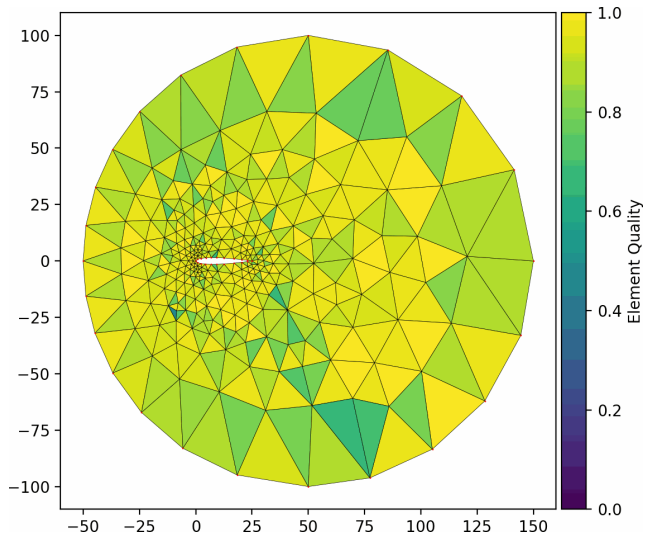
NACA Airfoil: Learned Construction



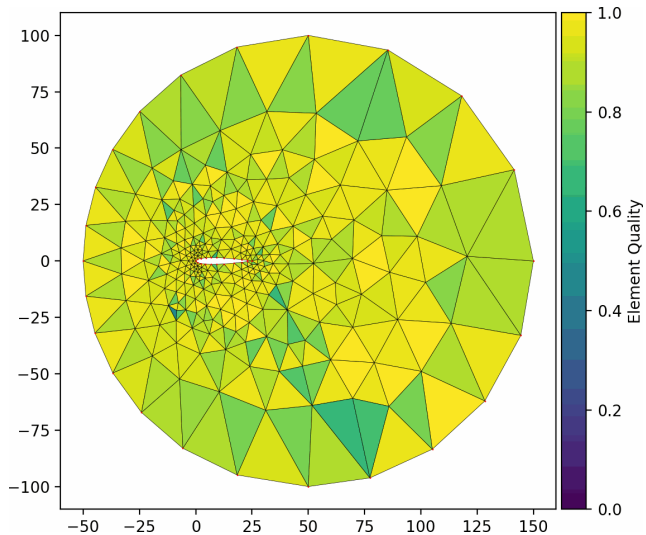
NACA Airfoil: Learned Construction



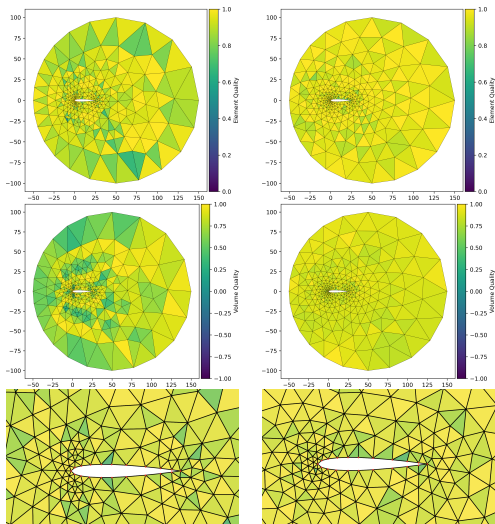
NACA Airfoil: Learned Construction



NACA Airfoil: Learned Construction



NACA Airfoil: Comparison with DistMesh



Learned policy

DistMesh

- The policy was trained on random polygons, not airfoils.
- The learned mesh is broadly acceptable on this out-of-distribution geometry.
- DistMesh remains stronger near the sharp trailing edge.
- Generalization across geometry and resolution remains the central challenge.

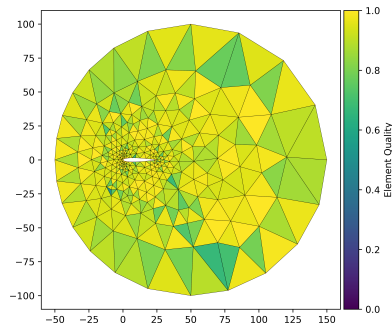
Part III Summary: Learning Geometry

What worked

- A GNN can jointly move, insert, and delete mesh vertices.
- Delaunay triangulation cleanly separates learned geometry from connectivity.
- Explicit quality rewards produce interpretable meshing preferences.

Current limitations

- Performance is strongest near the training distribution.
- Classical methods still handle difficult boundaries more robustly.
- A complete learned mesher must ultimately coordinate topology and geometry.



Outline

- 1 Introduction: Learning to Mesh
- 2 Part I: Learning to Repair Connectivity
- 3 Part II: Constructing Polygonal Meshes
- 4 Part III: Learning Geometry and Node Placement
- 5 Part IV: Escaping Local Optima in 3D**
- 6 Conclusions

Why Tetrahedral Topology Is a Harder Test

From 2D demonstrations to a 3D capability test

- Tetrahedral quality depends strongly on **connectivity**, even with fixed vertices.
- Valid 2–3 and 3–2 flips give a compact, purely topological action space.
- Greedy improvement stops when every single valid flip looks worse.
- Reaching a better topology may require a deliberate temporary loss in quality.

The question is no longer only whether RL can choose good edits:

Can it learn to make a sacrifice now in order to escape a local optimum later?

greedy plateau



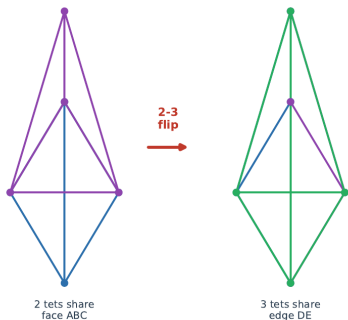
temporary quality dip



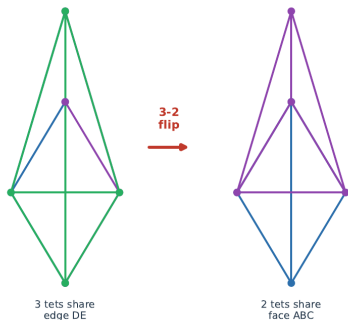
better topology

Setting: Quality Optimization by Local Flips

2-3 Flip: 1 interior face → 1 interior edge
(+1 tetrahedron)



3-2 Flip: 1 interior edge → 1 interior face
(-1 tetrahedron)



- Actions: **2-3 flips** (interior faces), **3-2 flips** (valence-3 edges)
- Fixed vertices — **topology only**

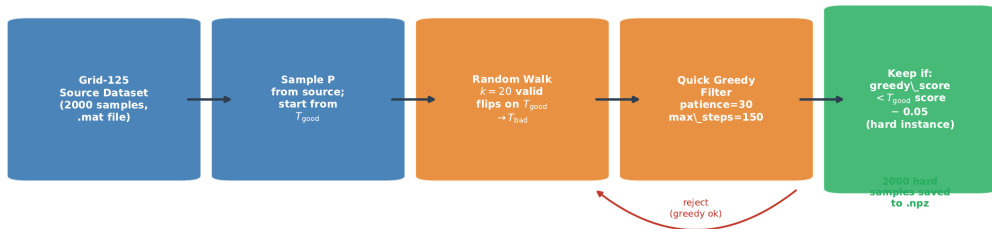
Per-tet quality: $q = \frac{12(3V)^{2/3}}{\sum_{\ell} \ell^2}$

Global score:

$$Q_{\text{softmin}} = -\tau \log \frac{1}{|T|} \sum_t e^{-q_t/\tau}, \quad \tau = 0.05$$

Greedy Flipping Gets Stuck — a Hard-Instance Benchmark

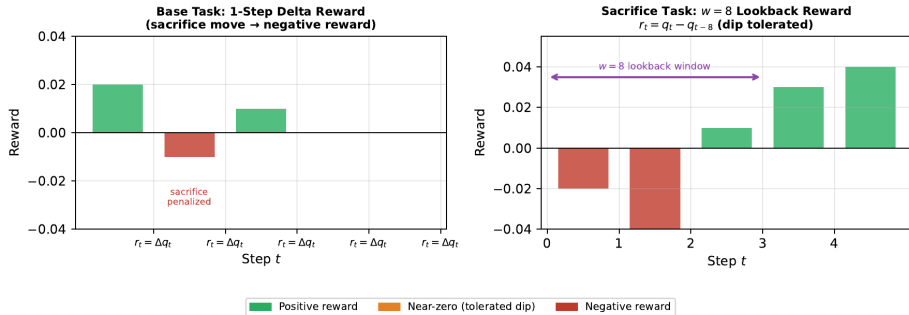
Sacrifice Dataset Generation Pipeline ($k = 20$, threshold=0.05, target $N=2000$)



- Random-walk $k = 20$ flips away from T_{good} ; keep instance if greedy stalls ≥ 0.05 below
- $N = 2000$ instances ($5 \times 5 \times 5$ cube grids, ≈ 400 tets) — mean greedy gap **0.213**

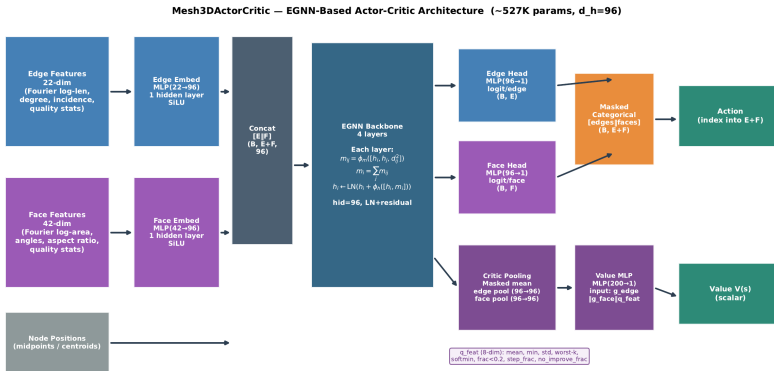
Key Idea: Pay Now, Gain Later — the Lookback Reward

Reward Signal Design: Base vs Sacrifice Task



- 1-step reward $r_t = \Delta Q_{\text{softmin}} \times 10$ penalizes every dip — sacrifice moves never learned
- Lookback: $r_t = (Q_{\text{softmin}}(q_t) - Q_{\text{softmin}}(q_{t-w})) \times 10$, $w = 8$
- Dips shorter than $w \rightarrow$ near-zero reward; recovery $\rightarrow$ positive — same actions, same network

Policy: an Equivariant GNN on the Flip Graph

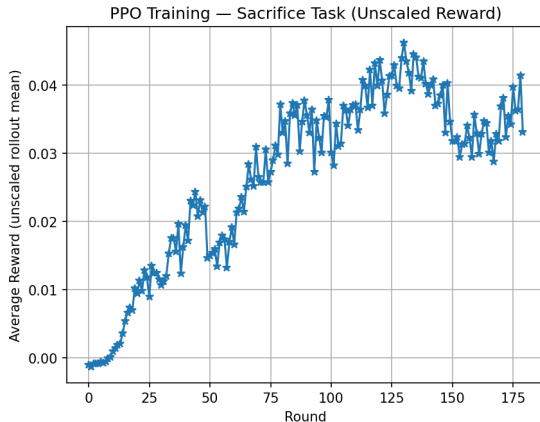


- Nodes = candidate actions (edges + faces) — **4 EGNN layers**, $d_h = 96$, **527K parameters**
- Masked categorical over $E + F$ logits · critic: pooled embedding + 8 quality features

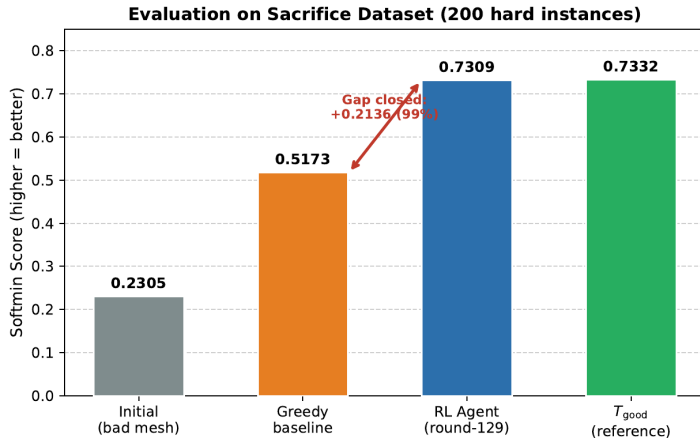
Training: Standard PPO, One V100, 3.5 Hours

envs × rollout × rounds	$32 \times 50 \times 180$
discount / GAE	$\gamma = 0.985$ / $\lambda = 0.93$
clip / epochs / minibatch	0.15 / 10 / 128
learning rate	$2 \times 10^{-4} \rightarrow 10^{-5}$
entropy coefficient	$0.004 \rightarrow 0.0012$
episode cap / patience	150 / 50 steps

- Reward peaks at round 130 — **round-129**
checkpoint used at evaluation

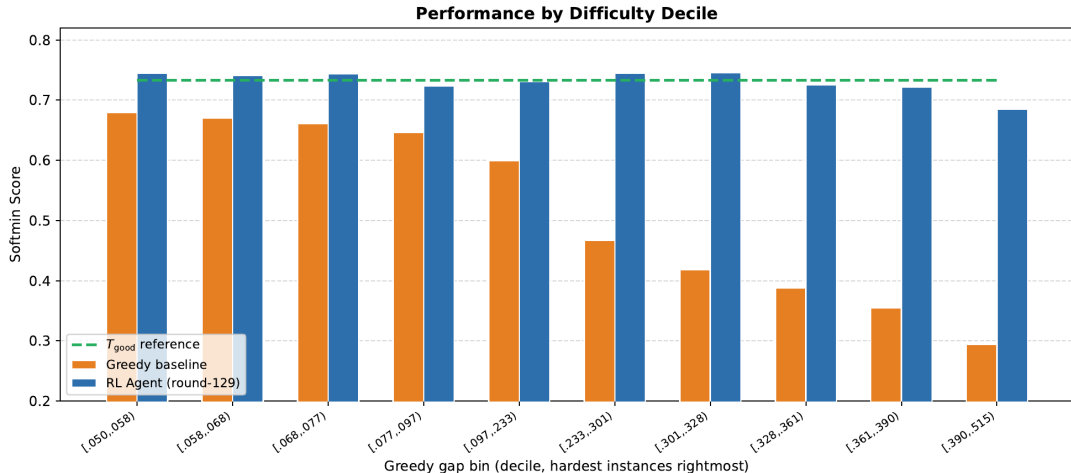


Result: 99% of the Greedy Gap Closed



- 200 held-out hard instances: greedy 0.517 $\rightarrow$ agent **0.731** vs. T_{good} 0.733
- Agent beats greedy on **195/200** · final meshes: **372.9 tets**, below the 384-tet reference

The Harder the Instance, the Bigger the Win



- Hardest decile: greedy 0.29 vs. agent 0.68 (+0.39) — agent ≈ 0.72 –0.75 in most bins

Part IV Summary: Learning to Sacrifice

- **Reward design, not architecture:** an 8-step lookback unlocks sacrifice behavior.
- **Local, scale-free features + equivariance:** nothing in the policy is tied to cube geometry.
- **Hard instances benefit most:** the agent closes 99% of the greedy gap.
- **Unexpected structural benefit:** 372.9 vs. 384 tetrahedra at near-reference quality.
- In progress: sphere meshes for zero-shot transfer and retraining.

Code: github.com/wanzhoulei/3d-meshing-run

Outline

- 1 Introduction: Learning to Mesh
- 2 Part I: Learning to Repair Connectivity
- 3 Part II: Constructing Polygonal Meshes
- 4 Part III: Learning Geometry and Node Placement
- 5 Part IV: Escaping Local Optima in 3D
- 6 Conclusions**

One Framework, Four Different Meshing Problems

	Representation	What the policy learns	Main result
I	half-edge template	repair connectivity	near-optimal 2D regularity
II	DCEL neighborhood	construct connectivity	triangles, quads, hexagons
III	geometric mesh graph	place and count vertices	quality-driven node placement
IV	equivariant flip graph	cross quality barriers	99% of greedy gap closed

The common pattern is

mesh state $\longrightarrow$ GNN policy $\longrightarrow$ valid edit $\longrightarrow$ quality reward.

The scientific choices are the representation, action language, and objective.

Conclusions and Open Problems

What these studies demonstrate

- Meshing can be posed as sequential decision-making, not only as a fixed heuristic pipeline.
- Self-supervision from explicit quality metrics can discover nontrivial editing strategies.
- Graph representations allow policies to operate on changing, unstructured meshes.
- Reward design determines which behavior is learnable—including temporary sacrifices.

Toward a complete learned mesher

- Couple topology and geometry in one policy.
- Enforce validity and robustness outside the training distribution.
- Generalize across element types, resolution, geometry, and dimension.
- Use learned policies alongside classical algorithms where each is strongest.

References and Codes

- [1] A. Narayanan, Y. Pan, and P.-O. Persson. *Learning topological operations on meshes with application to block decomposition of polygons*. Computer-Aided Design 175, 103744 (2024); arXiv:2309.06484.
- [2] A. Narayanan. *Automated mesh generation using graph neural networks and reinforcement learning*. Ph.D. thesis, University of California, Berkeley (2026).
- [3] W. Thacher, P.-O. Persson, and Y. Pan. *Optimization of a Triangular Delaunay Mesh Generator using Reinforcement Learning*. Computer-Aided Design 189, 103964 (2025); arXiv:2504.03610.
- [4] J. Schulman et al. *Proximal Policy Optimization Algorithms*. arXiv:1707.06347 (2017).

Part 1: Quad/triangle improvement using flips: <https://github.com/ArjunNarayanan/Quadrl>

Part 2: Polygonal decompositions: <https://github.com/ArjunNarayanan/MeshRL>

Part 4: Tetrahedral improvement using flips: github.com/wanzhoulei/3d-meshing-run

Thank you!!