EEC 289A - Project
Reinforcement Learning for Remeshing

Ahmed H. Mahmoud

1 Abstract:

In this work, we try to improve upon the work represented in [1] to add another layer of fine-grained control of the selection of the meshing operator used to enhance the quality of triangular mesh. It has been previous shown that a generic framework can be devised to improve various quality of initial mesh; a process that is important for many computer graphics and simulation applications. The framework consisted of a set mesh operators that can be applied in varying order to the input initial mesh. However, there is no consistent scheduling of such operators and it was left to the user to choose the right scheduling. In this report, we argue that it is possible to automate the scheduling by leveraging ideas from reinforcement learning and integrate it in the framework. Doing so, it is possible to apply the framework on large set of quality that no mapping into geometric primitives e.g., aligning mesh edges with an underlying directional field [2].

2 Introduction:

Processing of geometric models is a common practice in computer graphics [3], visualization, finite element analysis for simulation [4], and computer aided-design. These models are used as linear approximation for some underlying surface. These models or meshes could be acquired by laser scanner or modeling tool such as Autodesk Maya, ZBrush or Blender. Such process usually produces noisy meshes with artifacts that makes the processing of such meshes impossible. For that reason, geometry processing comes as an important stage in the 3D modeling acquisition pipeline [5] to improve the quality of the mesh and make the further processing of the models feasible. Improving the quality of initial mesh is also called *remeshing*.

Remeshing is a multi-objective optimization process where the objective requirement could be sometimes conflicting. That could be the reason why most of the existing remeshing techniques usually focus on one objective. However, there have been a rise on remeshing algorithms with multiple objectives [6, 7, 2].

The main remeshing objective we will focus on in this work is the non-obtuse of triangular meshes. Non-obtuse remeshing takes input mesh contains triangles with angles larger than 90° and returns an output mesh with all angles less than 90°. Non-obtuse meshes are vital for the correctness and convergence of numerical solution e.g., fast marching method [8]. It is been proven lately that it is always possible to obtain a non-obtuse triangulation with a polynomial bound over the number of input vertices [9]. Despite this theoretical results, there is not any remeshing algorithm with guarantees of producing non-obtuse remeshing. The framework devised in [1] can be considered as the state-of-art of non-obtuse remeshing as shown by empirical results. However, it is possible that the framework get stuck in certain configuration. The solution suggested in the paper is to re-run the algorithm with different random seed. The random seed will affect the decision over what meshing operator to use and on what vertex being used such that the stuck configuration may not be encountered again. This only holds a probabilistic guarantee that may not be encountered for different models or may need a large number of experiments.

Contribution: We propose in this report a way to formulate the problem as reinforcement learning problem i.e., MDP problem. Thus, it is possible to *learn* from previous experiments; and not just re-run the experiment with random seed and wish for better luck. Using this formulation, we experimented with different reinforcement learning algorithms and show that it is possible to guide the algorithm to achieve better results in the next run.

3 Remeshing Algorithm:

We describe here the remeshing framework. The framework is designed to serve various mesh quality or objectives. Since we are only concerned with non-obtuse remeshing, we tailored the framework description to serve the sole purpose of non-obtuse remeshing. The algorithm uses two nested loops; the outer loops is a *while*-loop that keeps repeating the same steps until maximum number of iterations is encountered, and the inner loop iterates over all five remeshing operators (described later). The maximum number of iterations is taken here to be 10. Each remeshing operator is applied on each obtuse angle. All the remeshing operators guarantee to either remesh the obtuse triangle into non-obtuse one or leave it as it is in case of failure.

Remeshing operators: Figure 1 shows the five operators. We first invest sometime to describe the first operator, *relocation*, and the later operators will be considered as extension of it.

1. **Relocation** is applied on one vertex and seeks to move this vertex such that all triangles share it as a head are non-obtuse. It starts with removing the vertex and all edges connected to it. In order to fill in the void with another vertex and produce non-obtuse triangles, the new vertex should be outside all diameter circles of the edges surrounding the newly created void (shown in red in Figure 1(a)). This will prevent any the angle at the head of the vertex (apex) to be non-obtuse. Additionally, in order to prevent the base angle from being obtuse, the new vertex should be one the right side of the two half-planes constructed at the two ends of each edge surrounding the void as shown in Figure 2(a). Similar geometric primitives can be constructed in order to prevent the angle from deterioration below certain limit (θ_{min}). Such additional primitives are shown in Figure 2(b) while Figure 2(c) shows the complete set of geometric primitives used in order to make sure the new vertex is in the region that will guarantee non-obtuse triangles to all triangles connected to the new vertex. Such primitives represent a mapping from the objective function to simple geometric primitives that are easy to construct and work on. Next, we sample the void randomly respecting the feasible region of sampling represented by the geometric primitives. It is possible that the primitives overlap in such a way that no new sample is possible to be drawn inside the void. At such cases, we quit and move to another obtuse triangles. Otherwise, we connect the new vertex to the vertices surrounding the void.

2. **Ejection** removes two vertices and add one as shown in Figure 1(b). It follows the same steps as for Relocation in terms of constructing the primitives and sampling one vertex.

3. **Injection** adds new vertex inside a bad triangles/batch. Here we do not remove any vertices. Instead we remove edges (shown as dotted edges in (Figure 1(c)) and this created a void to work on. One or both of the removed edges belongs to an obtuse triangles. The same process of constructing the primitives and sampling one vertex inside the void are followed to produce non-obtuse triangulation.

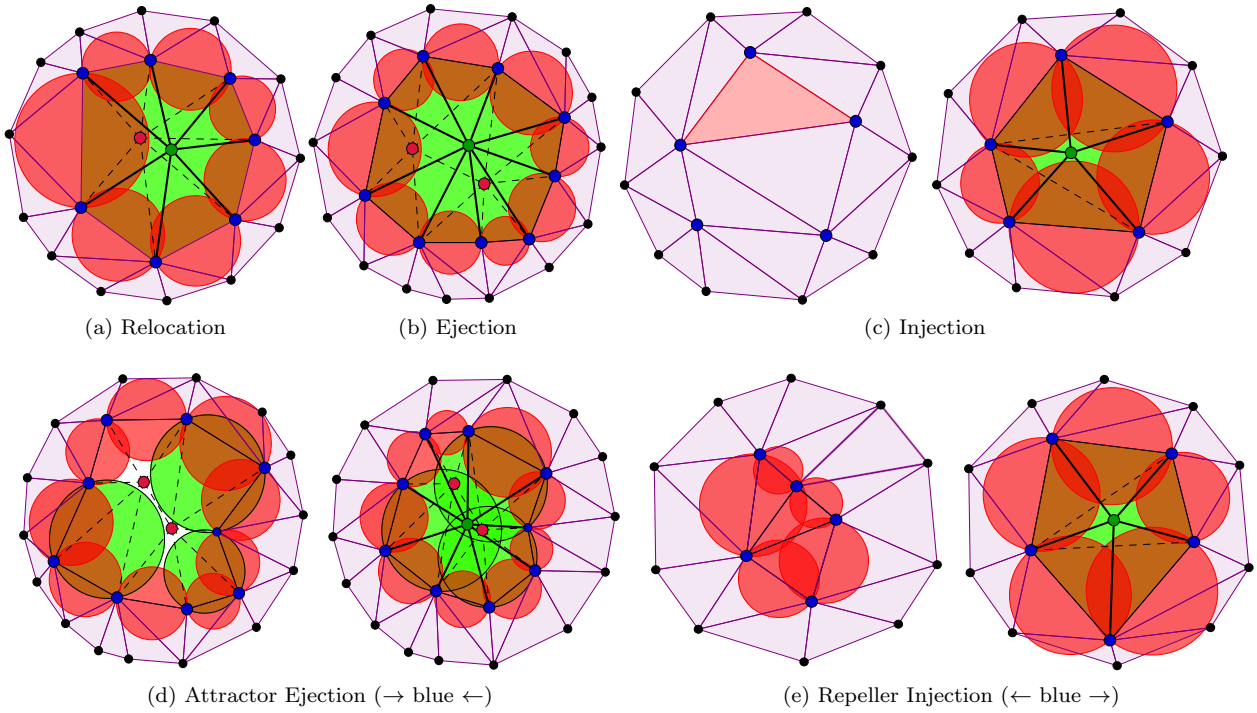
4. **Attractor Ejection:** is an extension to Ejection in which the void vertcies (shown as blue in Figure 1(d)) and moved closer to the center of the void. This helps closes up a wide void such that the primitives overlap and a sample can be drawn to produce non-obtuse triangles.

5. **Repeller Injection:** is an extension of Injection in which the void vertices are moved away from the center of the void. This could be helpful in widening a tight void and thus a new vertex can be sampled to produce non-obtuse triangles inside the void.

We notice that all operators seek monotonic improvement and do not allow degradation. This is a key feature that does not hold in previous art.

4 Reinforcement Learning Formulation:

Here we extend the algorithm described above and formulate the problem as MDP problem. The algorithm contains all the important elements needed to improve an input mesh but lacks a coherent scheduling of the operators. The algorithm in the present form seeks to apply the operators with lowest magnitude of change i.e., Relocation and keep trying more aggressive operators at the end. Since it is not possible to fully understand the function being optimized, we turn to reinforcement learning as an aid to help choose a better scheduling.



(a) Relocation (b) Ejection (c) Injection

(d) Attractor Ejection (→ blue ←) (e) Repeller Injection (← blue →)

Figure 1: Sampling operators: red disks are exclusion regions; light green disks/polygons are inclusion regions; dotted lines and dark red vertices are the removed elements and vertices; thick dark lines and dark green vertices are the new elements and vertices; blue are the neighbor vertices. (a) Relocation of a vertex from the exclusion region to the feasible region. (b) Ejection of two vertices, one of which is associated with a bad element. (c) Injection of a new vertex. A low quality (red-ish) triangle is destroyed along with two other triangles such that the new vertex is not irregular (3-valent/4-valent). (d) Attractor Ejection where the intersection of inclusion disks is empty; relocating the neighbor vertices towards the ejected vertices to pack a feasible region. (e) Repeller Injection of new vertex where the exclusion disks cover the void completely; relocating the neighbor vertices farther away from the void creates a feasible region.

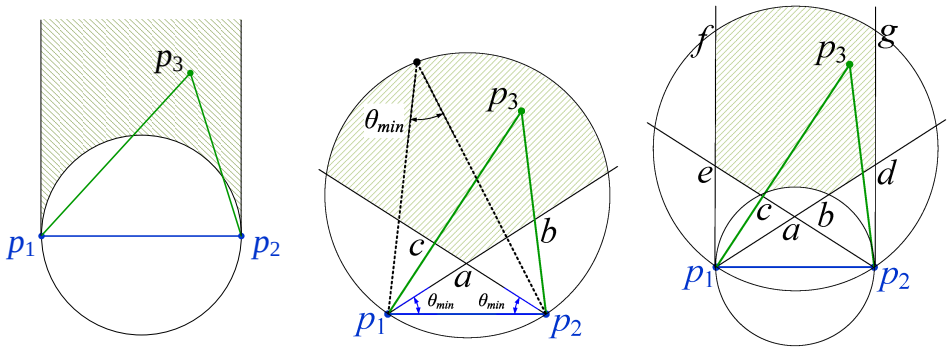
The environment here is presented by the mesh vertices, current number of obtuse-triangles, and ratio of failed to successful attempts for each operator. These three parameters are enough to described the current state of the environment and there is no need to trace the history of the vertices movement. That way, the problem can be treated as MDP problem. The agent goal is reduce the number of obtuse triangles through taking one action at a time represented by one of the remeshing operators. The problem is treated as an episodic tasks through which the agent trying to reach the optimal policy.

For each action taken, it is successful in eliminate at least one obtuse triangle, we give a reward of one. Otherwise, the reward is always zero. We first experimented with giving negative rewards (-1) but this made the reward always negative. This could probably have some relation tied to the geometry of the problem. When final episode is represented by reaching non-obtuse mesh in which case a reward of 100 is given. If after number of episodes proportional to the number of initial mesh vertices is reached without reaching non-obtuse mesh, we give a reward of -100 and re-initialize the mesh.

5 Numerical Results:

We applied several reinforcement learning algorithm over the stated problem. The algorithm were built on top of LibPGRL [10] which fleixible implementation of several reinforcement learning application using C++. We experimented the above model using the following algorithms:

- Actor Critic with NeuralNet approximator
- SARSA with NeuralNet approximator and Softmax policy with decaying temperature
- SARSA with NeuralNet approximator and e-Greedy policy with constant epsilon



(a) Nonobtuse. (b) $\angle p_i \geq \theta_{\min}$. (c) Both objectives.

Figure 2: Example primitives for resampling p_3 to form $\triangle p_1 p_2 p_3$: inclusion regions (green), exclusion regions (black boundaries).

- SARSA with NeuralNet approximator and e-Greedy policy with decaying epsilon
- QLearning with NeuralNet approximator and Softmax policy with decaying temperature
- QLearning with NeuralNet approximator and e-Greedy policy with constant epsilon
- QLearning with NeuralNet approximator and e-Greedy policy with decaying epsilon

For each algorithm, a run consisted of 1M steps which are averaged over 20 runs. For consistency over all algorithms, we used $\lambda = 0.9$, $\kappa = 10^{-4}$, $\epsilon = 0.5$. We reached this values through experimenting with different values and tuning up and down. The code can be found under "https://github.com/Ahdhn/EEC289A/tree/master/Project/code.

6 Conclusion:

References

[1] Ahmed Abdelkader, Ahmed H. Mahmoud, Ahmad A. Rushdi, Scott A. Mitchell, John D. Owens, and Mohamed S. Ebeida. A constrained resampling strategy for mesh improvement. *Computer Graphics Forum*, 36(5):189–201, July 2017. doi: 10.1111/cgf.13256. URL "http://escholarship.org/uc/item/5347s75h. Proceedings of the Symposium on Geometry Processing.

[2] Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. Instant field-aligned meshes. *ACM Trans. Graph.*, 34(6):189:1–189:15, October 2015. ISSN 0730-0301. doi: 10.1145/2816795.2818078. URL "http://doi.acm.org/10.1145/2816795.2818078.

[3] Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. *Recent Advances in Remeshing of Surfaces*, pages 53–82. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi: 10.1007/978-3-540-33265-72.

[4] K Ho-Le. Finite element mesh generation methods: a review and classification. *Computer-aided design*, 20(1):27–38, 1988.

[5] Fausto Bernardini and Holly Rushmeier. The 3d model acquisition pipeline. *Computer Graphics Forum*, 21(2):149–172, 2002. ISSN 1467-8659. URL "http://dx.doi.org/10.1111/1467-8659.00574.

[6] K. Hu, D. M. Yan, D. Bommes, P. Alliez, and B. Benes. Error-bounded and feature preserving surface remeshing with minimal angle improvement. *IEEE Transactions on Visualization and Computer Graphics*, 23(12):2560–2573, Dec 2017. ISSN 1077-2626. doi: 10.1109/TVCG.2016.2632720.

[7] Xingyi Du, Xiaohan Liu, Dong-Ming Yan, Caigui Jiang, and Hui Zhang Juntao Ye. Field-aligned isotropic surface remeshing. *submitted to COMPUTER GRAPHICSForum*, Seb 2017. doi: 10.13140/RG.2.2.36141.18408.

[8] Dong-Ming Yan and Peter Wonka. Non-obtuse remeshing with centroidal voronoi tessellation. *IEEE transactions on visualization and computer graphics*, 22(9):2136–2144, 2016.

[9] Christopher J. Bishop. Nonobtuse triangulations of pslgs. *Discrete & Computational Geometry*, 56(1):43–92, Jul 2016. ISSN 1432-0444. doi: 10.1007/s00454-016-9772-8. URL "https://doi.org/10.1007/s00454-016-9772-8.

[10] Douglas Aberdeen, Olivier Buffet, Jin Yu, and Fabio Pakk Selmi-Dei. Libpgrl library. URL "https://code.google.com/p/libpgrl.