Adaptive Feature-Preserving Isotropic Remeshing

YUAN YAO, University of British Columbia

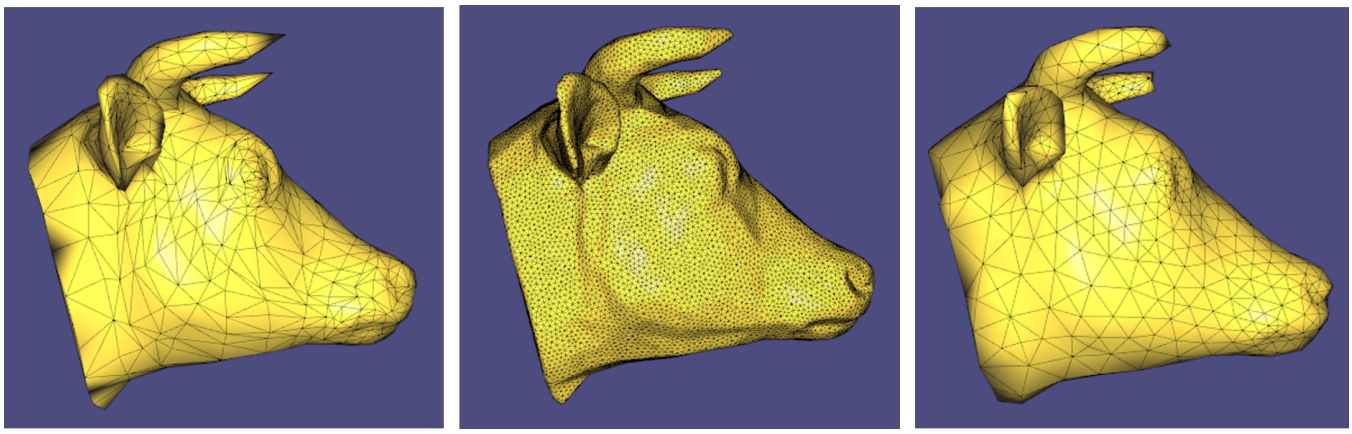

Fig. 1. **Isotropic Remeshing**. The left figure is the original 3D mesh of **cow head**. The medium one is the isotropic remeshing result. The right one is adaptive remeshing result.

Remeshing is a traditional topic in computational geometry. There are already many different mature algorithms in isotropic remeshing, anisotropic remeshing and even quadratic remeshing. In this work I mainly focus on implementing a classic incremental remeshing algorithm proposed by [Botsch and Kobbelt 2004]. Besides the basic approach, this work also tries to add feature-preserving and adaptive remeshing for improving the quality of final results. The results for standard isotropic remeshing is as good as in the original paper, however the adaptive method works but lack of details. Additionally, I noticed that the library I am using is libigl which is actually not suitable for this inremental remeshing algorithm as it's more often to be used for solving linear system.

Additional Key Words and Phrases: remeshing, isotropic, adaptive, feature-preserving

ACM Reference format:
Yuan Yao. 2019. Adaptive Feature-Preserving Isotropic Remeshing. *ACM Trans. Graph.* 36, 4, Article 1 (July 2019), 4 pages.
https://doi.org/

1 INTRODUCTION

Triangle and quad-dominant meshes are ubiquitously used in computer graphics and CAD applications to represent surfaces, either directly, or as the control grid for higher-order parametric or subdivision surfaces. Nowadays, meshes can be acquired with high accuracyusing multiple methods, for example, using three-dimensional (3D)laser scanners, RGBD cameras, dense reconstruction from multi-view stereo images and isosurface contouring. Using these acquire-draw meshes directly in downstream applications is difficult becausethey usually contain redundant data and have poor mesh quality.Remeshing is necessary to improve mesh quality while preserving the original geometry.

The most widely used triangular remeshing method is incremental isotropic remeshing which firstly proposed by [Botsch and Kobbelt 2004] and is a simplified version of [Kobbelt et al. 2000]. It produces results that are comparable to the ones by the original algorithm, but it has the advantage of being simpler to implement and of being robust. In particular, it does not need any parameterization nor the involved computation of (geodesic) Voronoi cells as some other variational methods do.

In this project, I mainly do such works:

- Implement a basic incremental remeshing algorithm. This algorithm consists of five steps including: split edge, collapse edge, flip edge, tangential relaxation and project to surface.
- I add a feature preserving strategy. The feature vertices/edges are computed at very beginning and are preserved along the whole pipeline with vertices data structure.
- I also tried to modify it to be adaptive, which means the length in smooth region is large but short in areas near sharp features.

2 APPROACH

In this section, I mainly describe the methods and pipeline I use for the remeshing. The key objective for the isotropic remeshing is to set all the edge length be the same and all the vertices have perfect valence(6 for interior vertex, 4 for boundary vertex). Refer to [Botsch and Kobbelt 2004], I follow the traditional procedure: split edge, collapse edge, flip edge, tangential relaxation and projection

0730-0301/2019/7-ART1 $15.00
https://doi.org/

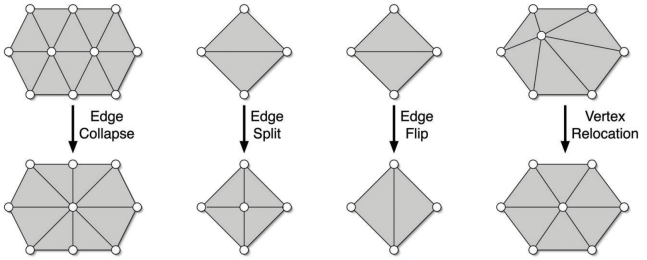

Fig. 2. Baisc Operations. Edge collapse is to collapse an edge and merge the two vertices together. Edge split would insert a new vertex which decrease the length of the original edge. Edge filp can change the valence. Vertex relocation move the vertex to a reasonable position which could be used for recover the original mesh.

to surface. The input to the whole method is only the target length for the final mesh's edges. The output would be a mesh which tries to satisfy this length. Besides, I also add two more functions: feature-preserving and adaptation which can help to improve the performance.

The incremental remeshing method separates the whole pipeline into five steps: split long edge, collapse short edge, flip edge, tangential relaxation and project back to surface. The function of the first two steps are to cut or merge the edges to make them at the similar level with the target length. Then we flip the edges which can make the valence better. Finally, to recover the original surface and details, the method project the vertex back to the original mesh. Figure 2 shows how the basic operations in this method works. The rest of this section will explain more details about these and other functions I add.

Split edges. Given the input target length, we firstly compute a upper bound and lower bound for the further computation by

$$low = (4/5) * length, \quad high = (4/3) * length \tag{1}$$

Then we traverse all the edges, and split is if it's longer than *high*. At the mean time, keep updating all the data structures to preserve the consistency. The new inserted vertex is on the middle of the edge. Now we have many edges no more than *high*, for those edges which are still long, they will be splitted in the later iteration.

Collapse edges. After splitting edges, we get rid of some long edges, so it's time to deal with short edges. Similarly. for those edges shorter than *low*, we collapse it by merging the two vertices and remove one of them. Actually this could cause many problems like non-manifold or triangle flipping.

Flip edge. To flip edge, it has to make the valence perform better. So the measurement for a potential edge flipping is

$$\begin{aligned} deviation = {} & abs(valence(a) - target\_val(a)) \\ & + abs(valence(b) - target\_val(b)) \\ & + abs(valence(c) - target\_val(c)) \\ & + abs(valence(d) - target\_val(d)) \end{aligned} \tag{2}$$

Here $target\_val$ means the target valence of the vertex, if it's an interior vertex, it's 6, if on boundary, it's 4. a,b,c,d are 4 vertices on the two triangles adjacent to the edge. If the deviation become smaller, which means the valence is closer to perfect settings, we flip this edge, otherwise keep it. So we traverse all the edges and do this check and flip the edge or not.

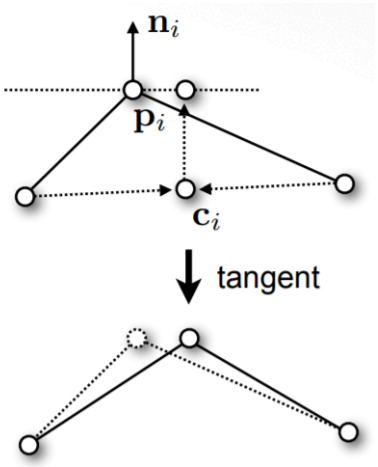

Fig. 3. Tangent relaxation illustration. Move the new vertex along the normal of the original vertex.

Tangential relaxation. It applies an iterative smoothing filter to the mesh. Here, the vertex movement has to be constrained to the vertex tangent plane in order to stabilize the following projection operator. Let p be an arbitrary vertex in the current mesh, let n be its normal, and let q be the position of the vertex as calculated by a smoothing algorithm:

$$q = \frac{1}{|N(p)|} \sum_{p_j \in N(p)} p_j \tag{3}$$

The new position p' of p is then computed by projecting q onto p's tangent plane:

$$p' = q + nn^T(p - q) \tag{4}$$

This step is also illustrated in figure 3.

Surface projection. This step is way more important as it will largely recover the surface from original mesh 4. We project each vertex back to the original mesh by finding the closest triangle and doing interpolation. Then treat these vertices as the final locations. This can help to approximate the original mesh. To accelerate the process, we use AABB tree for searching.

Feature preserving. To preserve the features in the original mesh, I add a feature list and maintain it through the whole pipeline. To determine a feature edge, I compute the angle between the normals of the vertices adjacent to the edge and use this to set if the edge is feature or not.

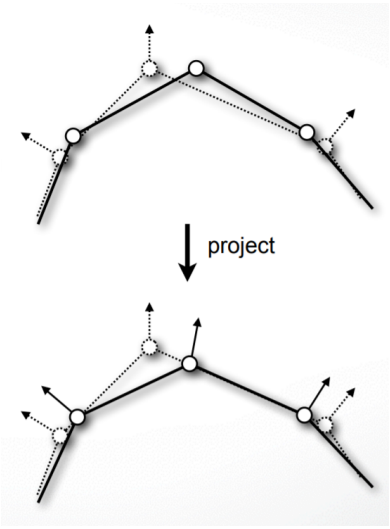

Fig. 4. Project back to surface by finding the closest triangle on the original mesh. After finding the closest triangle, do the barycentric interpolation.

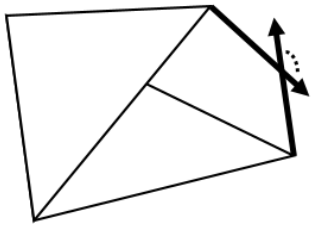

Fig. 5. Feature determination. Compute the angle between the normals of the vertices adjacent to the edge, if it's larger than a threshold, then set the edge and attibuted vertices to be features.

Adaptive. I also add a simple adaptive remeshing scheme. The key idea for adaptive remeshing is to set different target lengths on different areas. To know whether a vertex is in a smooth or sharp region, we need to use principle curvature k_{min} and k_{max}. Therefore the target length I set is

$$local\_target\_length = \frac{|k_{min}|}{|k_{max}|} * target\_length \tag{5}$$

Here target_length is the length we set as the input. This observation is not always correct, but can indeed generate some adaptive results.

3 RESULTS

In this section, I firstly describe more details on implementation and experiment settings. Then I present some visual results using different settings and method.

3.1 Implementation

I use **libigl** [] as the third party library whose data is simple vertex-face lists rather classic half-edge data structure. It's more general but actually not suitable for this algorithm as it is harder to represent the topology although it can easily provide global information and be more efficient for some global computation. To help with topology update, I add a new array which store the neighbor edges for a directed edge in a directional way. This largely help to find the neighbor edge and update other topology stuff.

However, **libigl** indeed provide some useful build-in function which I used. They have an interface for edge collapse and an AABB tree-based point to surface function which help to project back on original surface. It also integrate a easy-to-use GUI which I can change and modify it easily.

For the algorithm implementation, I did not check if the collapsed edge would become longer than *high* and I do not add any other constraint for edge flipping as these things are not that necessary and would cause some flipping triangles sometimes. The feature preserving part is a little tricky, I only care about the projection to feature edge but not check if edge collapse and other parts affect that. The results are already good enough and can observe some features get preserved.

3.2 Experiment

For this section, I will present experiment on different models, settings and their corresponding results. All the data are from the given models by the course website.

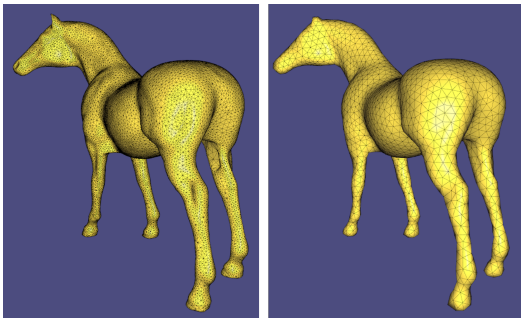

Fig. 6. Remeshing results. The left one is the original mesh, the right one is the remeshing result using target length 0.02. The result is almost isotropic but lost some information because of the high target length.

Visual results. As figure 6 shows, my implementation can indeed achieve isotropic remeshing as expected. It takes 5 6 seconds to get such results in near 4 iterations.

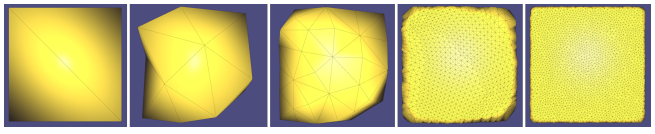

Fig. 7. Results of remeshing a cube in different iterations. From left to right: the original mesh, after 1 iteration, after 2 iterations, after 5 iterations, after 10 iterations.

Iteration. Iteration indeed is very important for some cases. Like figure 7, the first two iterations' results are largely different from the original because of the small amount of vertices. And it also shows the power of projection back to surface, which can still recover the original information after many iterations.

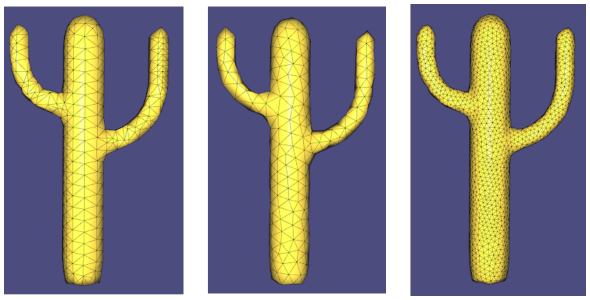

Fig. 8. Results of cactus in different target lengths.

Target length. As our input of the algorithm is target length, it could affect the final results in a very critical way. As figure 8 illustrates, the smaller the target length is, the more details the final result can recover.

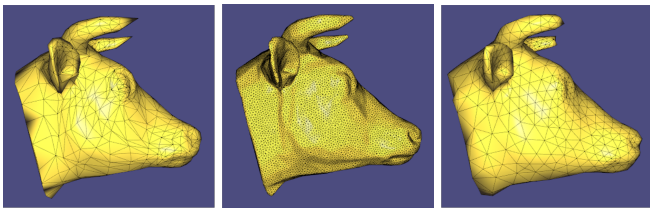

Fig. 9. Results of adaptive remeshing(same as teaser). The right one, which is the result of adaptive remeshing, has indeed some adaptation but still cannot recover the original details. The problem is that the threshold and metric I use for setting different target length are not almost correct.

Adaptive. As I mentioned before, I only tried a very simple and quick metric to determine if the target length for different vertices and edges. However the metric is not always correct, therefore the results can have some adaptation but cannot reproduce the original details. The results are shown in 9.

4 CONCLUSION

Remeshing is a super funny task, it integrate many of the topology operations and also some differential geometry knowledge and closest point problem. This work implemented a very standard and efficient incremental remeshing algorithm and tries to add some more functions. Although the adaptive results are not that good, it's also worth trying new metric to determine the best target length in different region. There are also some potential future works and direction like remeshing using graph convolutional network which can highly speedup and may resolve flipping triangle problem as it can self modify the topology. Additionally, it would also be interesting to transfer this method into quadratic remeshing.

REFERENCES

Mario Botsch and Leif Kobbelt. 2004. A remeshing approach to multiresolution modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM, 185–192.

Leif P Kobbelt, Thilo Bareuther, and Hans-Peter Seidel. 2000. Multiresolution shape deformations for meshes with dynamic vertex connectivity. In *Computer Graphics Forum*, Vol. 19. Wiley Online Library, 249–260.