Isotropic Remeshing with Boundary and Feature Preservation

Simon Ghyselincks
sghyseli@cs.ubc.ca
University of British Columbia
Vancouver, BC, Canada

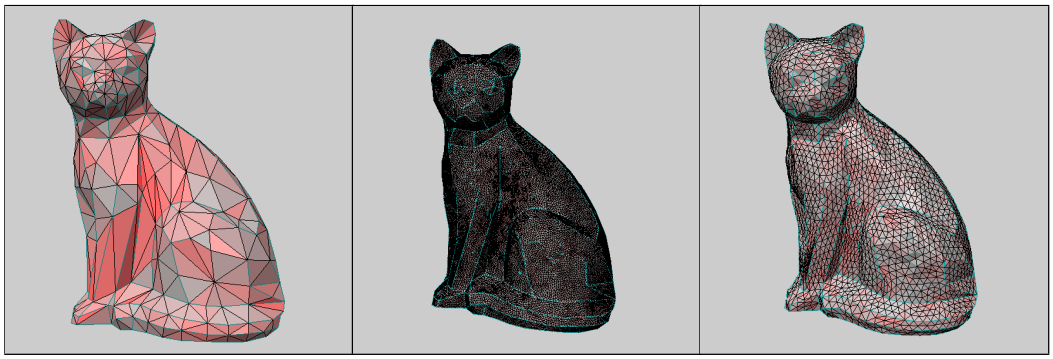

Figure 1: Isotropic remeshing results on open boundary mesh. (Left) Input mesh with irregular triangles. (Center) After uniform remeshing at fine scale. (Right) After uniform feature preserving remeshing at medium scale.

Abstract

This project presents an implementation of uniform explicit isotropic remeshing, designed to regularize 3D triangle meshes while preserving geometric features and boundary integrity. The algorithm follows the classic pipeline proposed by Botsch and Kobbelt (2004), synthesized with modern implementation details from the 2015 Master's thesis by Tanja. The system robustly handles edge splitting, collapsing, flipping, and tangential smoothing. Particular attention is given to the preservation of sharp creases and open boundaries, ensuring that the remeshing process does not degrade the topological genus or the visual fidelity of the input model.

1 Introduction

Remeshing is a fundamental operation in computer graphics, modeling, and simulation that aims to modify an existing mesh towards a set of desired qualities without significantly impacting its overall shape or features. The quality of a mesh is a combination of both geometric and human aesthetic factors, including face aspect ratios, detail levels and refinement, parallel lines, and uniformity. Many methods exist for remeshing; however, this work focuses on explicit methods, which apply local operations in a sequential manner. In contrast to a global parameterization approach, explicit methods are flexible, extendable, and rely upon simple local operations iteratively applied to the mesh until convergence. The quality of the final mesh may not be as high as a parameterization approach, but they are often faster and easier to implement [Tanja 2015].

Isotropic remeshing aims to produce a surface with equal length edges, resulting in a uniform distribution of triangles across the mesh. This is a popular choice, with a baseline method from Botsch and Kobbelt [2004]. Many variants exposing new features are described by Botsch et al. [2010], including anisotropic remeshing, curvature-adaptive remeshing, and volumetric remeshing. In this work, we implement the uniform isotropic remeshing algorithm with explicit feature and boundary preservation, allowing remeshing with open boundaries and sharp creases.

The work ahead follows the original algorithm from Botsch and Kobbelt [2004], with implementation details and improvements selected from the synthesis of the art found in Tanja [2015] Master's thesis. In advanced non-isotropic remeshing, the length of edges is adaptive; however, in this work, we focus on the uniform case where the objective length L is constant across the mesh.

2 Methodology

The remeshing algorithm operates using a set of local mesh operations, local operation validity checks, feature identification and preservation, and iterative looping with hyperparameters to control desirable convergence. Before combining all of these components together, we first describe the core local components and definitions, followed by the overall algorithm structure.

2.1 Core Local Operations

The core principle for explicit isotropic remeshing is to define a set of local mesh operations which are combined in sequence to iteratively improve the model. In all implementations studied, the following four operations shown in Figure 2 are used:

(1) **Edge Split:** Edges longer than $\alpha \cdot L$ are split at their midpoint to create two new shorter edges.
(2) **Edge Collapse:** Edges shorter than $\beta \cdot L$ are collapsed, merging their two vertices into one and removing up to two faces.
(3) **Edge Flip:** Edges may be flipped according to a local valence optimization rule, improving the vertex regularity. Note that in 3D meshes, flipping an edge means replacing

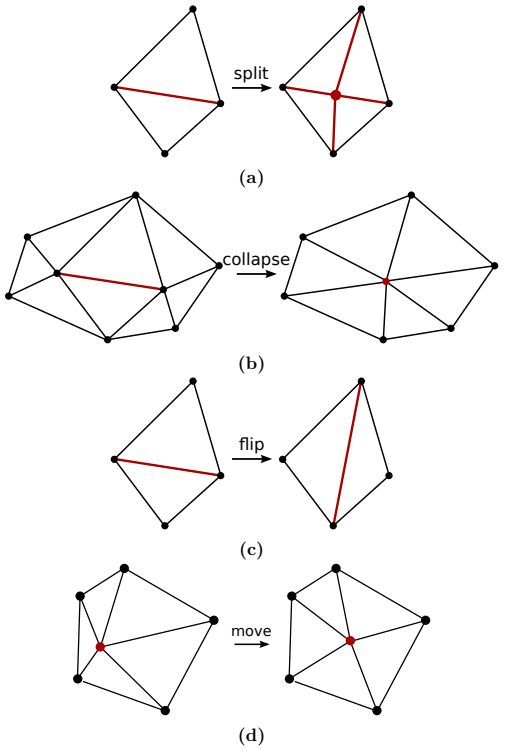

Figure 2: The four core local operations used in isotropic remeshing: (a) edge split, (b) edge collapse, (c) edge flip, and (d) tangential smoothing. Image adapted from [Tanja 2015].

it with the opposite diagonal in the quadrilateral formed by its two adjacent faces.

(4) **Tangential Smoothing:** Vertex positions are allowed to move in a restricted plane or domain to improve the overall triangle quality. In isotropic remeshing, this is typically done in the plane tangent to a local vertex normal.

In practice, the literature values of $\alpha = \frac{4}{3}$ and $\beta = \frac{4}{5}$ are used for edge splitting and collapsing thresholds respectively [Botsch and Kobbelt 2004].

2.2 Feature and Boundary Preservation

Using local operations for remeshing works directly towards the uniform isotropic goal; however, care must be taken to avoid destroying important geometry and for handling special edges along open boundaries. For this reason, these edges are explicitly defined on the mesh and maintained throughout the remeshing process. The core concept is that boundary and feature edges should be minimally disturbed, by restricting the local operations that can be applied to them.

A feature is defined as an edge with two faces whose normals form a dihedral angle sufficiently deviating from flat. Recall that in a flat configuration, the two face normals are parallel, while in a sharp crease or fold, they are not aligned. The angle formed by the two intersecting face planes is thresholded to estimate sharp local curvature that should be preserved.

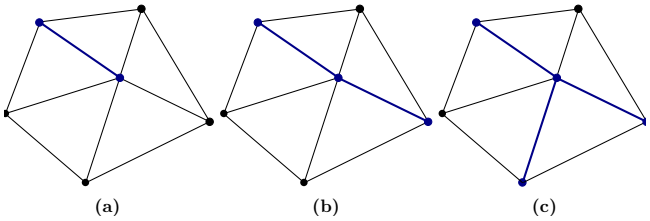

Figure 3: Feature edges are shown in blue, with two feature vertex classes shown in the center of the umbrella. (a) one edge, corner vertex; (b) two edges, edge vertex; (c) three edges, corner vertex. Image adapted from [Tanja 2015].

An angle threshold θ_f is used for classification such that any edge with two faces with normals $\mathbf{n}_1$ and $\mathbf{n}_2$ where

$$\frac{\mathbf{n}_1 \cdot \mathbf{n}_2}{|\mathbf{n}_1||\mathbf{n}_2|} < \cos(\theta_f)$$

is marked as a feature edge. Boundary edges with only one face are automatically marked as features.

After initial edge identification, the vertices are further categorized into three types:

(1) **SIMPLE** vertices are connected only to non-feature edges.
(2) **EDGE** vertices are connected to exactly two feature edges.
(3) **CORNER** vertices are connected to one, three, or more feature edges.

A diagram of the vertex types is shown in Figure 3.

2.3 Operator Sequence and Iteration

The described local operations are applied in sequence over the entire mesh, with one or more iterations of final tangential smoothing at the end of each pass to allow vertex relaxation to settle. Implementation details for each operation are described further below.

2.3.1 Mesh Splitting. Edges longer than $\alpha \cdot L$ are split at their midpoint. The splitting operation is additive with less topological risk, so any edge can be split, including feature and boundary edges. The newly created vertex is feature classified according to its emplacement edge. The two new lateral edges are not features and do not contribute to incident feature counts for the new vertex. If the original edge was a feature, the two new sub-edges are marked as features to preserve the crease, and the vertex is marked as an EDGE vertex, otherwise it is marked as SIMPLE. Extra considerations are taken for boundary edges where only a single face requires a lateral edge to be created.

2.3.2 Edge Collapse. The edge collapse mechanism appears simple in Figure 2(b); however, it is fraught with topological pitfalls. In the simple interior case, collapsing an edge removes the edge, merges its two vertices, and merges four of the lateral edges into two while removing two faces. The following rules are applied to ensure topological validity and feature preservation:

- Corner vertices are never collapsed, as it would remove the feature,
- Boundary vertices are not movable, so a collapse is a one-way merge towards the boundary vertex,
- Edge vertices draw a merge with simple directly into the edge to preserve the feature line,
- Simple vertices may collapse into any other vertex type except corner,
- All edge-edge and simple-simple collapses create a new vertex at the midpoint,
- Non-feature edges may not collapse two feature vertices together which would cause edge bending,
- Standard topological validity for tetrahedron and *link condition* checks are applied,
- Collapses that create overly long new edges as a result are thresholded and rejected,
- Merges that create degenerate small faces or flip face normals are rejected.

Special operations for boundary edges need to also be considered, since only some of the lateral edges and faces exist compared to the interior case. To simplify the operations while searching for bugs, the special cases where adjacent faces are boundary holes that are rare in practice are rejected if detected, although this could be improved in future work.

After a merge, all of the vertices and edges need to be re-classified as features. The merged vertices inherit the strongest feature restriction in the order CORNER > EDGE > SIMPLE. The merged edges inherit feature status if either of the original edges were features.

2.3.3 Edge Flipping. Edge flipping is a mechanism for distributing vertex valences across the mesh. The proposed edge flip is simulated to check if it improves the overall valence of the nearby vertices. If the flip reduces the total error from ideal valence (6 for interior vertices, 4 for boundary vertices), the flip is accepted.

$$\text{Error} = \sum_{v \in \{v_{1..4}\}} |(\text{valence}(v) - \text{target}(v))| \tag{1}$$

However, an edge flip can easily cause topological intersection issues, which are hard to isolate. A check is performed to verify if the flip alters the local face normals excessively and is rejected if so, preserving sharp features [Botsch et al. 2010].

2.3.4 Tangential Smoothing. The final step applies a tangential smoothing or lateral displacement to the mesh points to improve mesh quality. This step is performed iteratively many times over the course of the remeshing sequence so it is important that it preserves mesh volume and shape. To achieve this, the feature lines and mesh volume are locally preserved by restricting vertex movement based on feature classification and vertex normals. A vertex normal is computed as the angle-weighted average of incident face normals [Surazhsky and Gotsman 2003].

$$\mathbf{n}_v = \frac{\sum_{f \in \text{faces}(v)} \theta_f \mathbf{n}_f}{\sum_{f \in \text{faces}(v)} \theta_f}$$

This direction forms the basis for a local tangent plane, such that the vertex should be constrained to not move along its normal direction which would likely change the mesh volume. To determine the unconstrained direction of movement, a variation on the Laplacian-Beltrami operator is used, where a weighted set of barycentric neighbor positions are averaged to determine a target position. The use of barycentric weights improves area uniformity and prevents nested vertex structures [Botsch et al. 2010]. The vertex $\mathbf{v}_i$ movement direction $\mathbf{c}_i$ is given as

$$\mathbf{c}_i = \frac{\sum_{f \in N(f)} A_f \mathbf{b}_f}{\sum_{f \in N(f)} A_f} - \mathbf{v}_i$$

where $N(f)$ is the set of neighboring faces with area A_f and barycenter $\mathbf{b}_f$. Thus leading to an attractive force towards triangles that are larger than average.

Finally the normal component is removed by projection and a damping coefficient λ is applied to control stepsize and oscillation as suggested by Botsch and Kobbelt [2004].

$$\mathbf{v}_i' = \mathbf{v}_i + \lambda \left(I - \mathbf{n}_v \mathbf{n}_v^T\right) \mathbf{c}_i$$

Special Considerations for Boundary and Features. CORNER vertices are not allowed to move at all to preserve features, including during smoothing. EDGE vertices must only move along feature edges to preserve the crease; a 1D equivalent of line weights is used to decide motion and it is projected onto the feature edge. Since boundary edges are also features, this handles their motion as well.

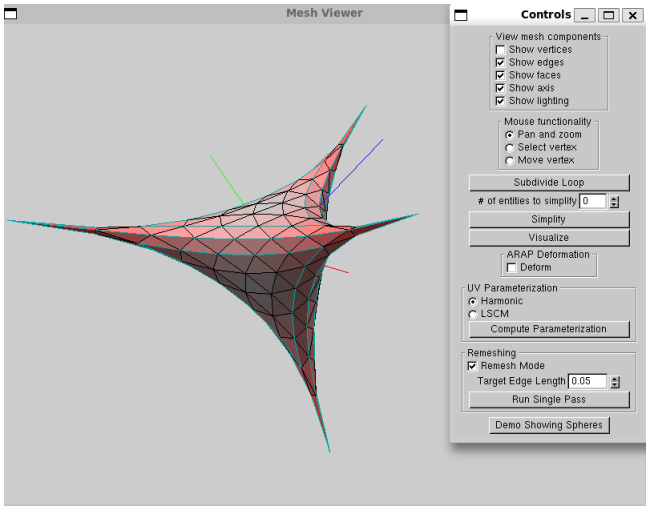

Figure 4: Screenshot of the remeshing GUI, showing the input mesh (left) with red color indicating triangle deviation from equilateral condition and feature edges highlighted in blue (feature threshold of 25°).

3 Implementation Details

The algorithm was implemented entirely within the UBC CPSC 524 'minimesh' framework, using C++ and CMake. Eigen is used as an external library with no other mesh utilities or external dependencies. The code has been structured to interface with a simple GUI for applying successive iterations and monitoring mesh quality. A screenshot of the GUI is shown in Figure 4.

Core mesh hyperparameters can be adjusted in the class header file including interior/exterior target valence, λ smoothing factor,

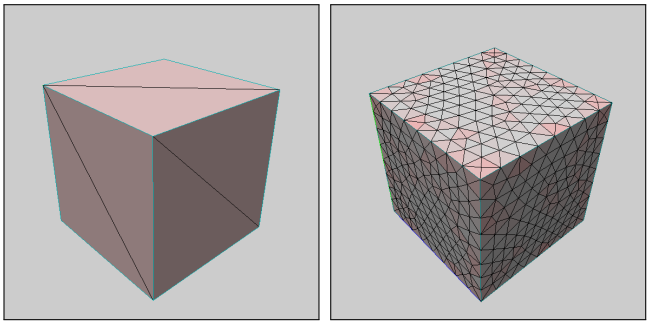

Figure 5: Remeshing comparison on the cube model. Triangle quality shown in red, with feature edges shown in blue.

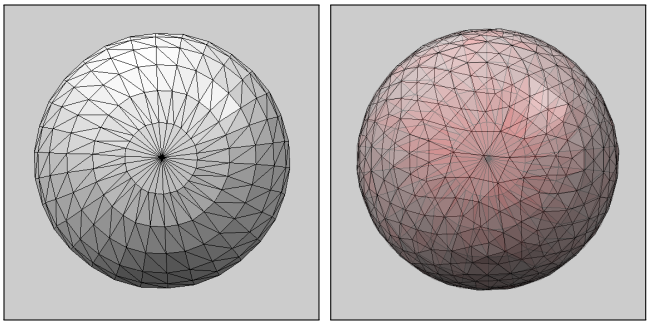

Figure 6: Overlaid remeshing comparison on the sphere model. Note the original model is overlaid on the remesh to show volume preservation.

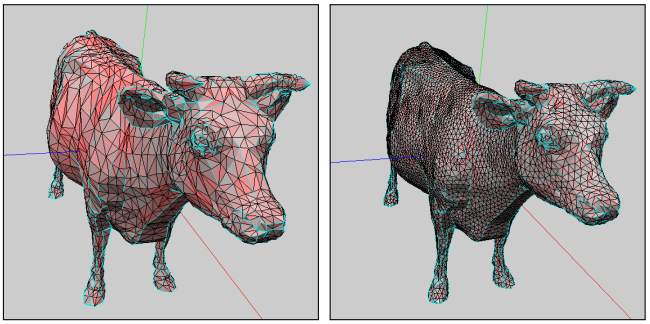

Figure 7: Cow remeshing results. Triangle quality shown in red, with feature edges shown in blue.

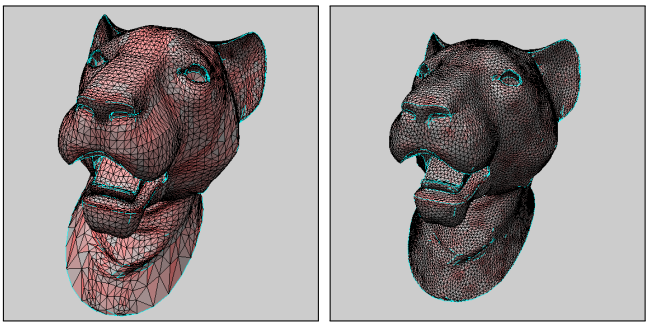

Figure 8: Lion remeshing results. Triangle quality shown in red, with feature edges shown in blue.

feature angle threshold, and edge-flip threshold. Element-wise target remeshing for total vertex or edge count is done using a similar approach to optimization algorithms whereby the target length L is adjusted up or down based on whether the current edge count is above or below the desired count, with early stopping at convergence.

4 Results

The implementation was tested on classical animal and shape benchmarks including cube, sphere, cow, camel, lion, and cat models. The results show good convergence towards uniform edge lengths, with feature and boundary preservation working as intended. Figures 5 and 6 show geometric analysis results. The feature preservation is on full display for the cube model where the corners and edges have stayed perfectly sharp throughout 15 iterations. The triangle quality has improved overall, with most triangles being close to equilateral. The sphere results do not involve any feature edges; however, the original mesh has badly conditioned triangles meeting at a dense vertex pole and no explicit vertex constraints to prevent volume changes. However, after convergence, the overlaid original mesh closely matches the remeshed surface, indicating good volume preservation.

The cow and lion models shown in Figures 7 and 8 respectively show good feature preservation on organic shapes with many sharp creases. Before feature preservation was implemented, the cow horns would have significant rounding and erosion in their detail. The lion manifold demonstrates good boundary preservation, with the open mouth, eyes, and collar region all remaining intact with significant improvement in triangle quality.

The algorithm successfully regularizes the mesh, bringing the majority of interior vertices to valence 6. The boundary loops remain watertight, and sharp features (like the corners of a cube) are preserved throughout the decimation and smoothing phases.

5 Performance and Limitations

Overall the algorithm can process meshes of moderate size (10k-100k triangles) in reasonable time (seconds to minutes total) on a standard laptop CPU. The single iterations are quite responsive in the GUI, allowing for interactive length adjustment and monitoring of iterative progress and convergence for a chosen set of hyperparameters. In practice, the local mesh operations are difficult to tune and prone to errors, especially within the edge merging operation. The many topological checks and feature preservation rules make the code complex and hard to debug; this is a significant drawback of these methods. However, once the basic operations are stable, the algorithm is well suited to making extensions and improvements.

Hyperparameters for generating good quality results are given in Table 1. These were selected based on literature values and some empirical tuning given the particular combination of implementation details.

Table 1: Hyperparameters used for remeshing experiments.

Hyperparameter	Value
Edge split threshold (α)	$\frac{4}{3}$
Edge collapse threshold (β)	$\frac{4}{5}$
Smoothing factor (λ)	0.4
Feature angle threshold	25°
Edge flip normal deviation threshold	30°

One of the largest limitations of the current implementation is the lack of rigorous self-intersection checks after local operations. These are hard to make in practice, and the tendency is for mesh coarsening to cause problems with intersecting faces, especially in the fine detail regions around feature edges. To overcome this challenge, a mesh coarsening step using a global approach such as QEM could be used for preprocessing before applying the remeshing. Additional geometry checks were recommended by Botsch et al. [2010] but there was not enough time to implement them all robustly. In practice, edge lengths that are smaller than or equal to the starting average perform well enough.

The mesh vertices and faces still have a tendency to drift over many iterations and could benefit from a global back-projection step back to the original mesh surface [Botsch et al. 2010]. This would help preserve volume and shape over many iterations of smoothing, especially in high-curvature regions. Parts of an adaptive length field were added to the code base in preparation for future work; however, the scope of the project did not allow for full implementation.

6 Conclusion and Future Work

This project successfully implements a robust isotropic remesher for manifold meshes with or without boundaries. While the current implementation uses a uniform target length, the architecture supports adaptive sizing. Future work would involve characterizing the mesh curvature and adapting a local edge length field based upon the curvature and the local feature structure. Additional geometric guards on the edge collapse and flip operations would improve robustness, along with a global back-projection step to preserve volume and shape over many iterations. However, the current implementation already produces good quality remeshed models with preserved features and boundaries, demonstrating the effectiveness of explicit local operations for isotropic remeshing.

Use of AI and LLMs

Google Gemini 3 Pro and Claude Code were used to assist in the preparation of the project code. Claude Code was assigned most of the GUI enhancements, although it required significant intervention at times. Google Gemini 3 Pro was used to create the initial header file and base .cpp template from instructions in the Botsch and Kobbelt [2004] paper. Although it functions well as a pair programmer, it does not have a good grasp of the complex mesh operation structures and full topological risks in operations; it also suggests inferior methods for tangential smoothing. It was used to help with some of the data structures and for restructuring some areas of the code that required extensions due to feature-based conditional logic. Overall, the use of LLMs sped up some of the boilerplate code writing and GUI enhancements, however the core algorithmic implementation and direction was fully done by the author.

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*. 185–192.

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. 2010. *Polygon mesh processing*. AK Peters/CRC Press.

Vitaly Surazhsky and Craig Gotsman. 2003. Explicit surface remeshing. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. Eurographics Association, 20–30.

Munz Tanja. 2015. *Curvature Adaptive Remeshing*. Master's Thesis. Bournemouth University. https://nccastaff.bournemouth.ac.uk/jmacey/MastersProject/MSc15/08Tanja/report.pdf