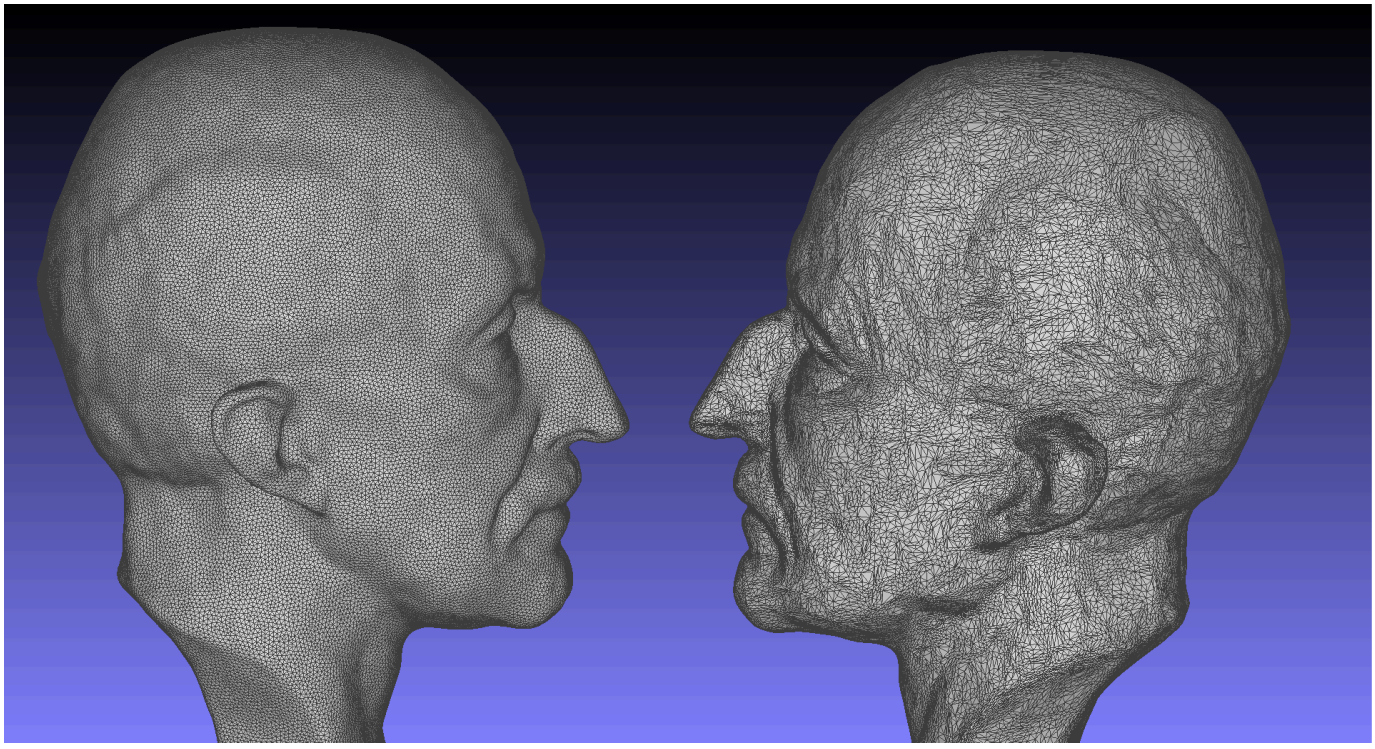
Project report on: “A Remeshing Approach to Multiresolution Modeling” by Mario Botsch, Leif Kobbelt, 2004

Project and report by: Amir Mann[1] and Ehud Gordon[2]

https://github.com/Amir-Mann/DGPIsotropicRemeshing



[1] amir.mann@campus.technion.ac.il, 207732132
[2] ehud.gordon@campus.technion.ac.il, 203861422

Table of Contents

Table of Contents **2**
Introduction **3**
Motivation 3
Algorithm 4
Application of Remeshing - Surface Evolver 4
Implementation details **6**
Half Edge Solution 6
Surface Evolver Solution 6
Results **7**
Area-Error & Runtime 7
Visual Comparison To Original Mesh and PMP 8
Inverting The Laplacian 11
Liquid Simulation Results 12
Appendix - running the code **14**

Introduction

Remeshing refers to the process of modifying the structure of a 3D triangular mesh to improve its quality or meet specific criteria, such as element uniformity and smoothness. This process is essential in various computational applications, where the initial mesh might be irregular and poorly shaped, or lack the necessary resolution for accurate simulations or analyses. In this work we focus on equalizing triangle areas. This is shown on the figure to the right.

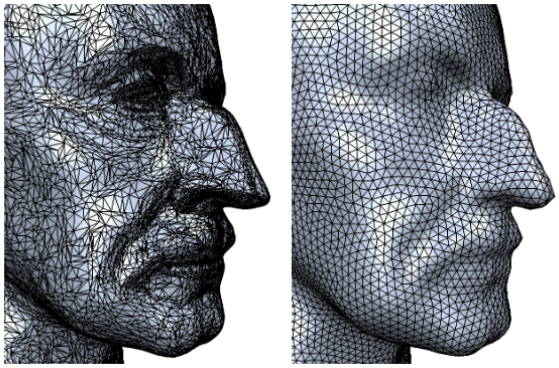
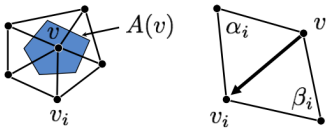


Figure from paper: [the paper's] remeshing technique yields very regular meshes and additionally equalizes the Voronoi areas of vertices.

Motivation

One of the key benefits of remeshing is its ability to produce well-shaped, symmetrical Laplacians. The Cotangent-Laplacian matrix, which is a common discretization of the continuous Laplace operator, plays a critical role in many computational geometry tasks. When the mesh has equally-sized triangles, the Laplacian becomes symmetric, as it is given for none diagonal entries by:

$$\Delta_{i,j} = \begin{cases} -\frac{cot(\alpha_{i,j}) + cot(\beta_{i,j})}{2A(v_i)} & i \neq j \end{cases}$$



Where the two angles opposing the $v_i - v_j$ edge are symmetric, and c only if the mesh has regular triangle sizes. This allows for more efficient numerical solutions using solvers like the banded Cholesky solver for sparse symmetric matrices.

Solving systems of equations involving the Laplacian is crucial for applications like shape smoothing and remodeling. In these applications, the Laplacian is compared to zero to determine how much a shape deviates from smoothness. By iteratively adjusting the mesh based on these comparisons, a smoother and more aesthetically pleasing shape can be achieved. The paper uses this concept as a tool for remodeling, demonstrating how adjusting the mesh based on the Laplacian can lead to more desirable geometric properties, with different geometric properties based on different orders of the laplacian taken Δ^k.

Figure 2: The order k of the energy functional defines the stiffness of the surface in the support region and the maximum smoothness C^{k-1} of the boundary conditions. From left to right: membrane surface ($k = 1$), thin-plate surface ($k = 2$), minimal curvature variation ($k = 3$).

Another importance of mesh quality is in finite element simulation, the quality of the mesh is paramount for accurate physics-based simulations. The finite element method (FEM) discretizes a physical system into smaller elements, where the mesh's quality determines how well the system is represented. A

well-constructed mesh ensures that the simulation captures the stress distribution, displacements, and other physical phenomena accurately. FEM guarantees that under the right conditions, including a good-quality mesh, the simulation results will be reliable and can be used to optimize designs before they are manufactured. Thus, controlling the mesh quality is essential to achieving the best results, as overly coarse or poorly shaped meshes can lead to inaccurate or misleading outcomes.

Algorithm

The paper presents the following method for remeshing a given mesh, with a target edge length of l:

1. Split at their midpoints all edges of length larger then $\frac{4}{3}l$
2. Collapse all edges of length smaller then $\frac{4}{5}l$
3. Flip edges in order to minimize valance variance
4. Relocate vertices to the weighted by area average center of their neighbors, on their tangent plane.

These 4 steps are then repeated for five iterations (in most cases) to produce a Voroni area equalized mesh.

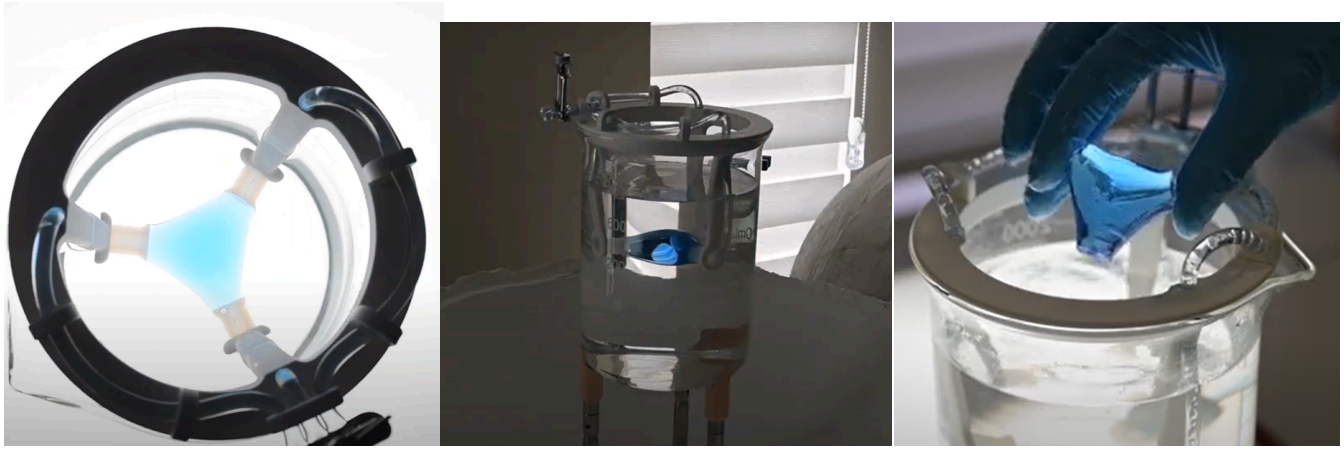
The area-based tangential smoothing is performed in the following manner: Each vertex p_i is assigned a "gravity" proportional to its area $A(p_i)$, and it is moved towards its gravity-weighted centroid g_i, which is projected back into the tangent plane of the surface, ensuring that the smoothing remains tangential:

$$g_i = \frac{1}{\sum_{p_j \in N(p_i)} A(p_i)} \cdot \sum_{p_j \in N(p_i)} A(p_i) p_i \,, \;\; p_i \leftarrow p_i + \lambda(I - n_i n_i^{\top})(g_i - p_i)$$

Except for the edge split action, all three actions can change the geometry of the mesh, however for fine enough meshes the re-meshed mesh remains close to the original geometry. This is important as it limits the use cases of this remeshing algorithm to ones in which fine-details may be slightly altered, while keeping the base-surface constant.

Application of Remeshing - Surface Evolver

Surface Evolver (SE) is an open source finite element based fluid simulation, used to simulate many systems and behaviors. In the Moran Bercovici lab at the Technion's Mechanics Faculty, SE is used to simulate a physical process done in the lab. The process starts with a polymer held by a plastic frame inside a flotation fluid. The polymer is then hardened, preserving its fluid shape and forming a solid 3D-printed object. This is illustrated in the figure below:



From left to right: An above view of the shape floating, A side view of the shape, A view of the printed shape after being hardened taken out of its container. Credit: Bercovici fluidic technologies lab.

In simulation, the polymer is represented by a triangle mesh. As part of our project, we've implemented remeshing in the Surface Evolver scripting language, with the purpose of improving the simulation process.

Implementation details

We have re-implemented the paper's algorithm in two environments, one, in Python, which is a half-edge representation of a mesh, and the other, in Surface Evolver, using the simulator's internal command scripting language.

Half Edge Solution

We've implemented from scratch a half-edge representation of a mesh which is inspired by an existing implementation by Santiago Caracciolo.
We've implemented the 4 basic operations required for remeshing and their relative integrity checks. In some cases, performing mesh operations destroys required properties of the mesh, like being manifold, orientability and non-overlapping triangles.
Using these operations, we've implemented the proposed remeshing algorithm, and tested it on a few non-regular meshes.

Surface Evolver Solution

We have implemented in SE's command script language the algorithm in order to apply it during a fluid simulation. The scripting language supports edge collapse, edge split and edge flipping with its basic operation, and supports vertex averaging without tangent projection.
We can get better than the projection results by taking the simulator's gradient step in order to return to a minimal energy shape after each averaging step.
An issue we encountered was when using the paper-suggested (and somewhat justified) constant for collapse / split, the mesh never settled down. Because of that, we have played around and found that 1.5 and 0.5 give better looking meshes and that the algorithm converges after 1-3 iterations. See more in the Results section.

So than the remeshing step as we defined it in the simulator is:

1. Split edges longer then $1.5l$.
2. Collapse edges shorter then $0.5l$.
3. Flip edges to equalize valance.
4. Average vertices and take a small gradient step.

With this remeshing step we then changed the simulation process to be:

1. Find a minimal energy shape for the current mesh.
2. Take a few remeshing steps. In the demos provided it is 2 steps.
3. If needed, refine the mesh to a smaller edge length using a few remeshing steps with smaller l and go back to 1.

Results

Area-Error & Runtime

We have computed the relative mean area error before and after the remeshing, in order to assess the quality of our algorithm. The relative mean area error is defined as:

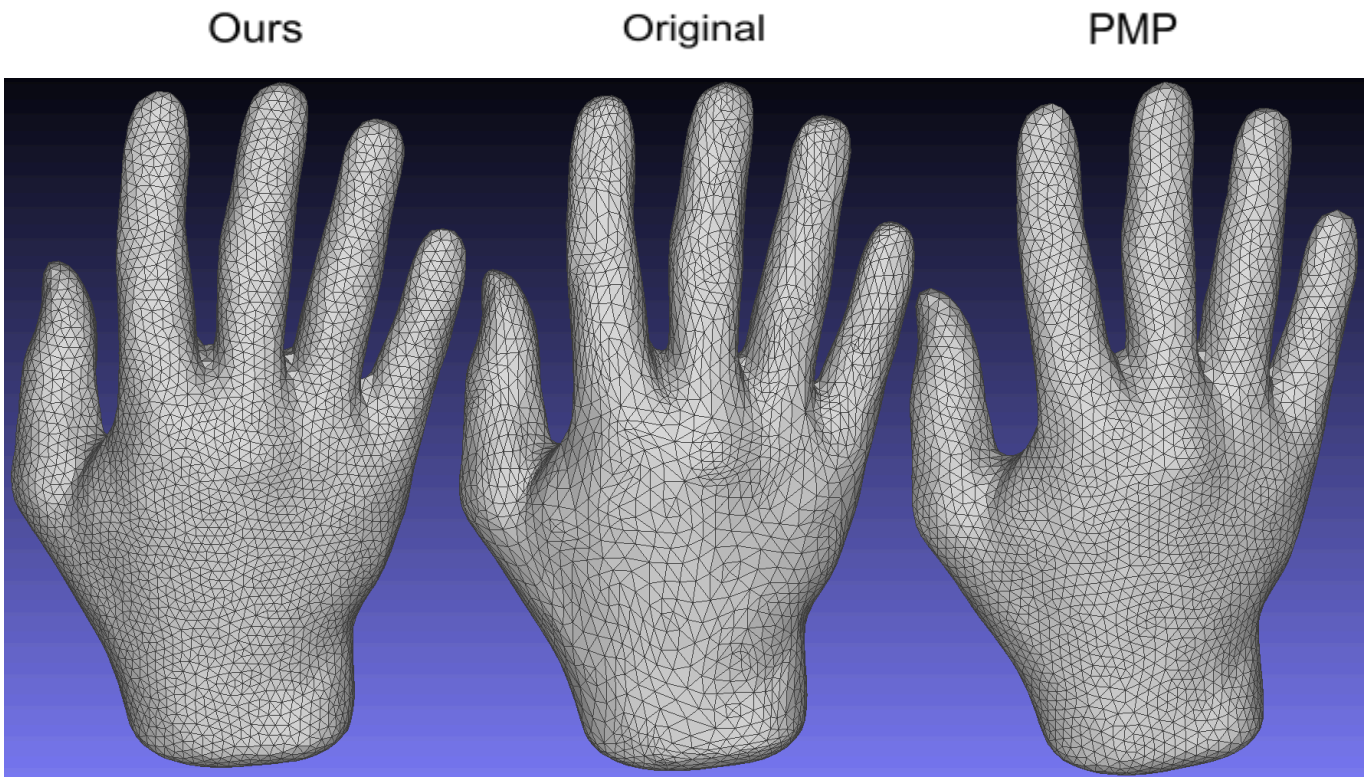
$$E = \frac{1}{|V|} \sum_{v \in V} \frac{|A(v) - \bar{A}|}{\bar{A}} \quad , \quad \bar{A} = \frac{1}{|V|} \sum_{v \in V} A(v)$$

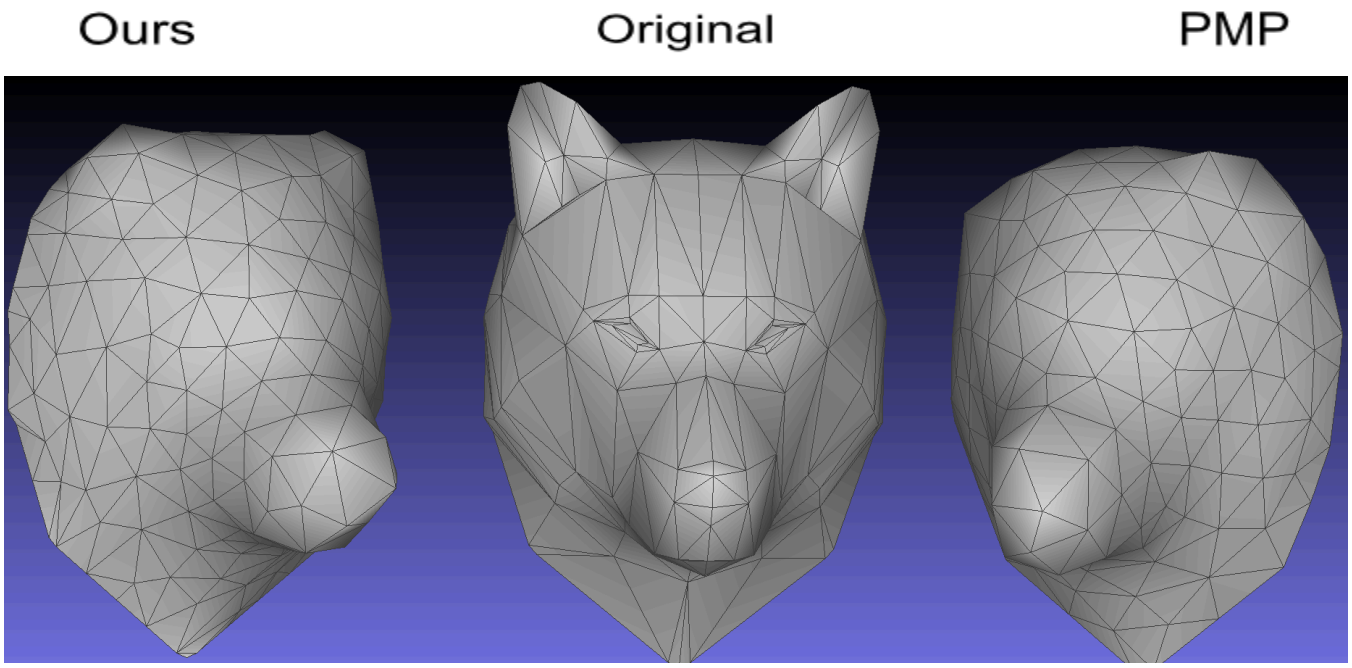
where $A(v)$ is the Voronoi area, and $\bar{A}$ is the average Vornoi-area over all vertices.
In order to ascertain the validity of our results, we've compared our results to an implementation by the PMP library, a well-tested and optimized mesh processing library in C++. As our implementation is in Python, our runtimes are longer. However, our implementation achieves slightly lower error rates, and as can be seen in the visualized meshes, our implementation retains slightly more details than the PMP implementation.

	original avg. area error	**Ours avg. area error**	**PMP avg. area error**	**Ours runtime**	**PMP runtime**
cat (18k)	15%	12%	10%	58.52 sec	0.13 sec
iphi_bad (10k)	68%	11%	13%	37.06 sec	0.12 sec
Hand (8K)	38%	11%	13%	34.10 sec	0.06 sec
cow (6K)	76%	12%	14%	25.06 sec	0.05 sec
wolf_head (300)	71%	16%	13%	1.15 sec	0.02 sec

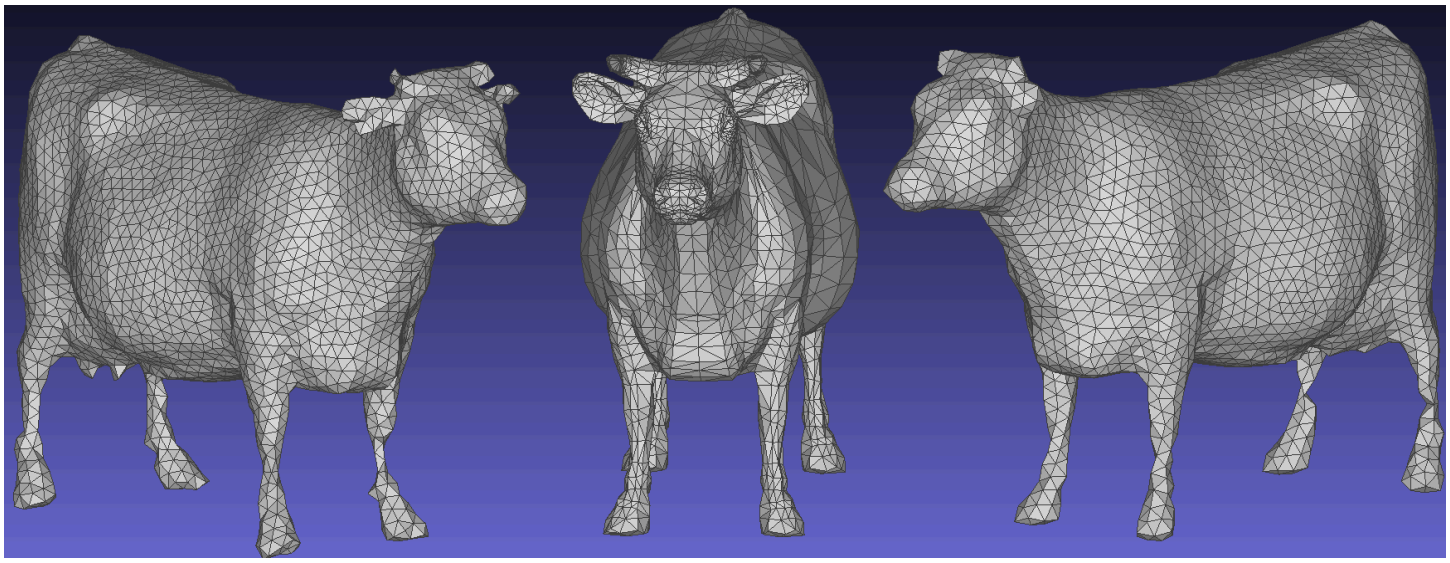
Visual Comparison To Original Mesh and PMP

Below we compare our results to the original mesh and to the PMP output, for different models. Notice in the cow's model, how our model preserves more detail in the horns compared to the PMP

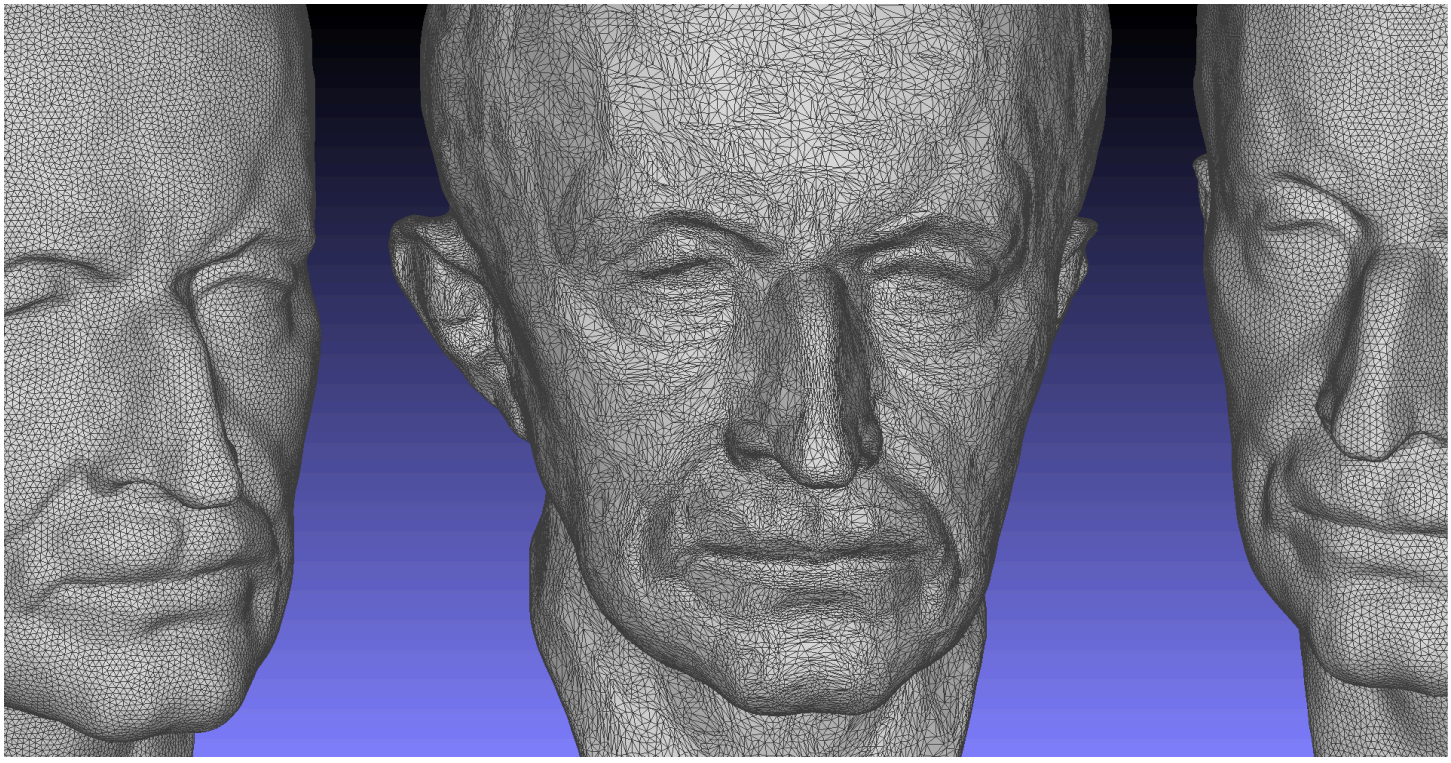


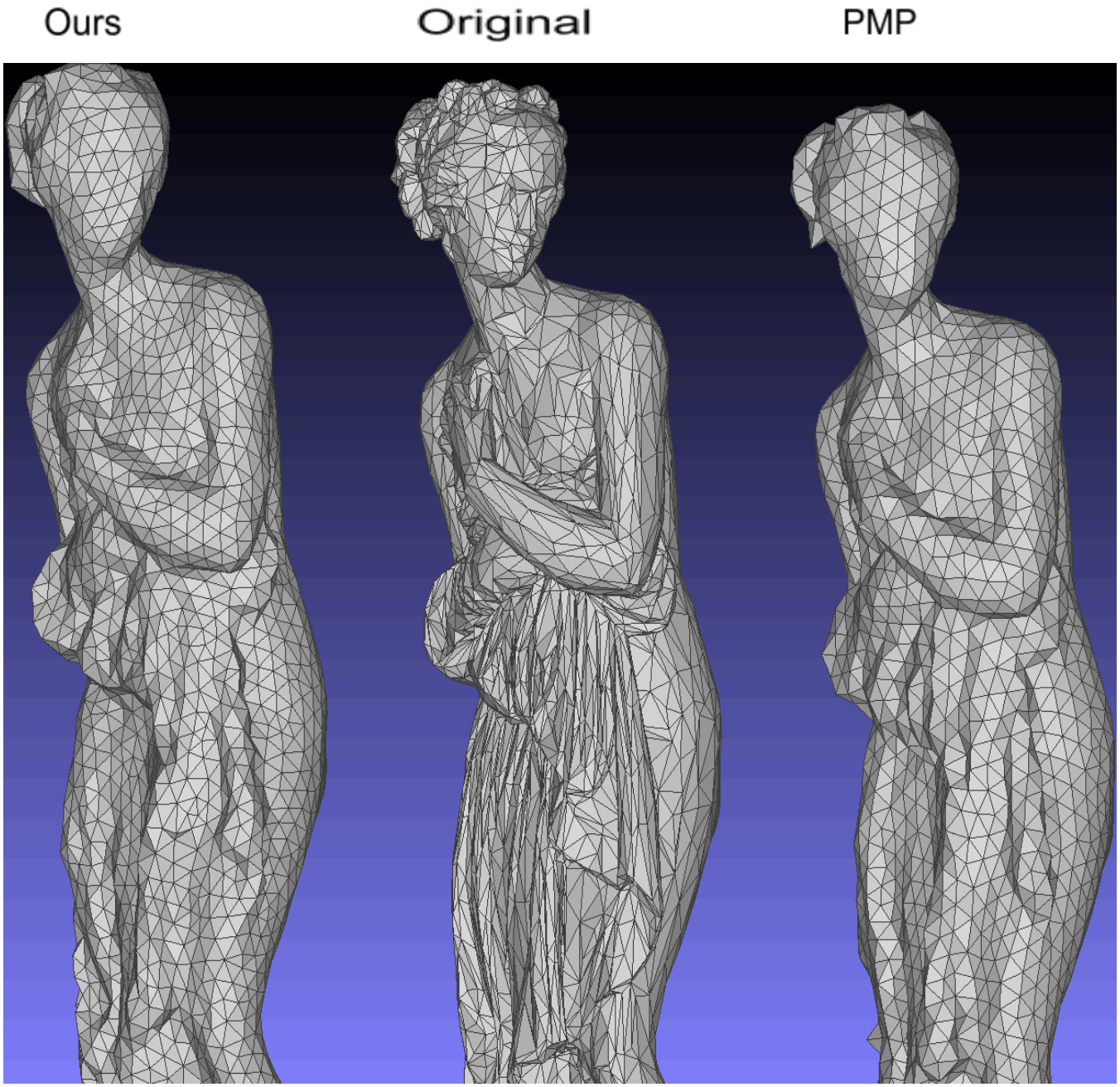


Ours Original PMP



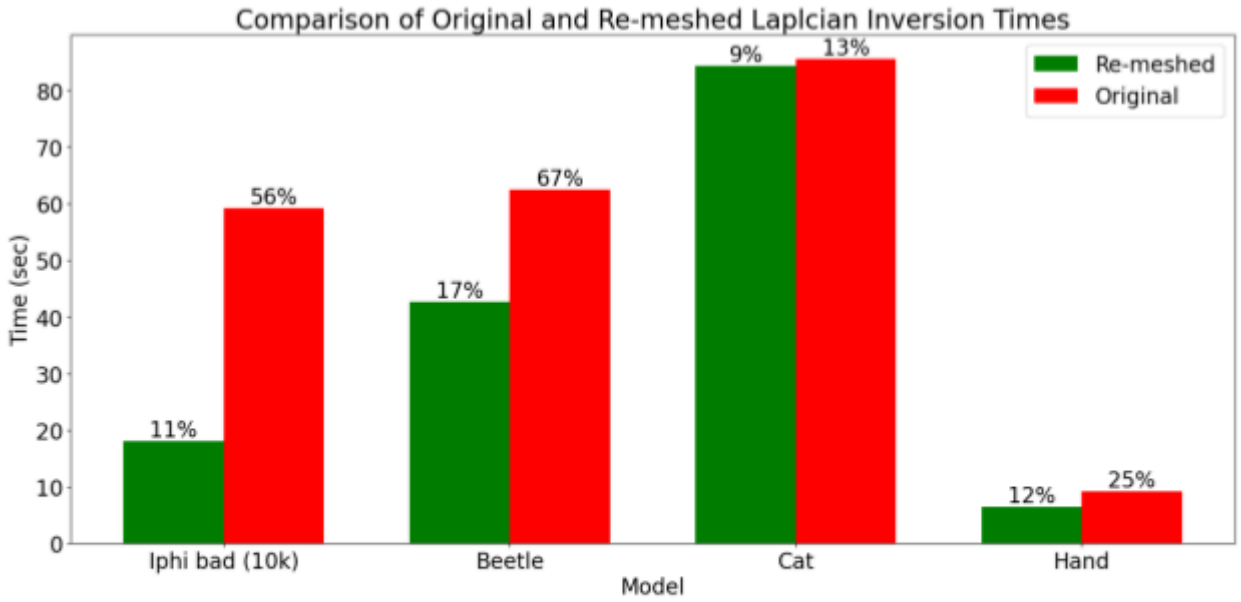
Ours Original PMP




Ours
Original
PMP

Inverting The Laplacian

One of the motivations for remeshing a model is to get a symmetric Laplacian, since inverting a symmetric matrix is faster. Note - as done in the paper, after remeshing, the average Voroni area is taken to be the Voronoi-area for all vertices in the Laplacian of the remeshed model. This is done to get a symmetric Laplacian. To compare the inversion times of the matrices, we have used the Simplicial LDLT algorithm for the symmetric matrix, which is efficient for sparse symmetric matrices. For the original, non-symmetric Laplacian, we have used the SparseLU algorithm, which is efficient for general sparse matrices. The results are presented in the plot below:

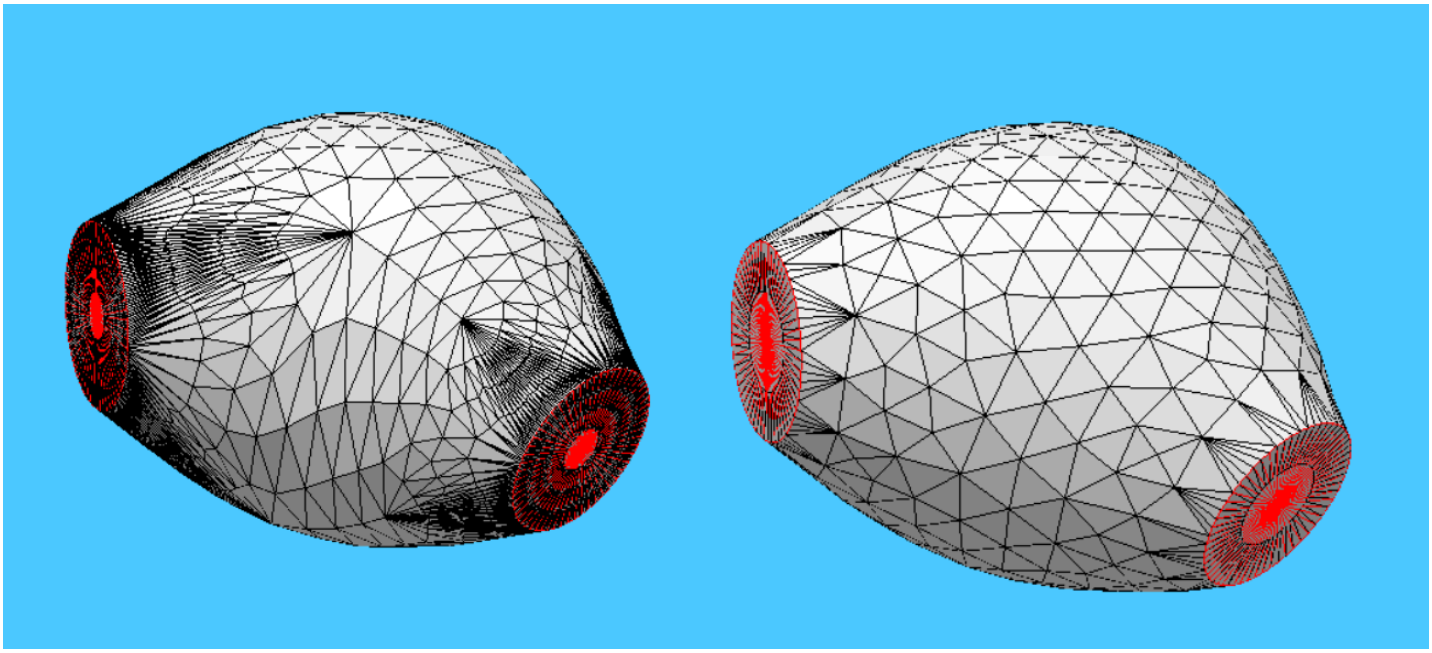


Comparing Laplacian inversion run-times for Original mesh vs. remeshed model. On top of each bar is displayed the relative mean area error of the model.

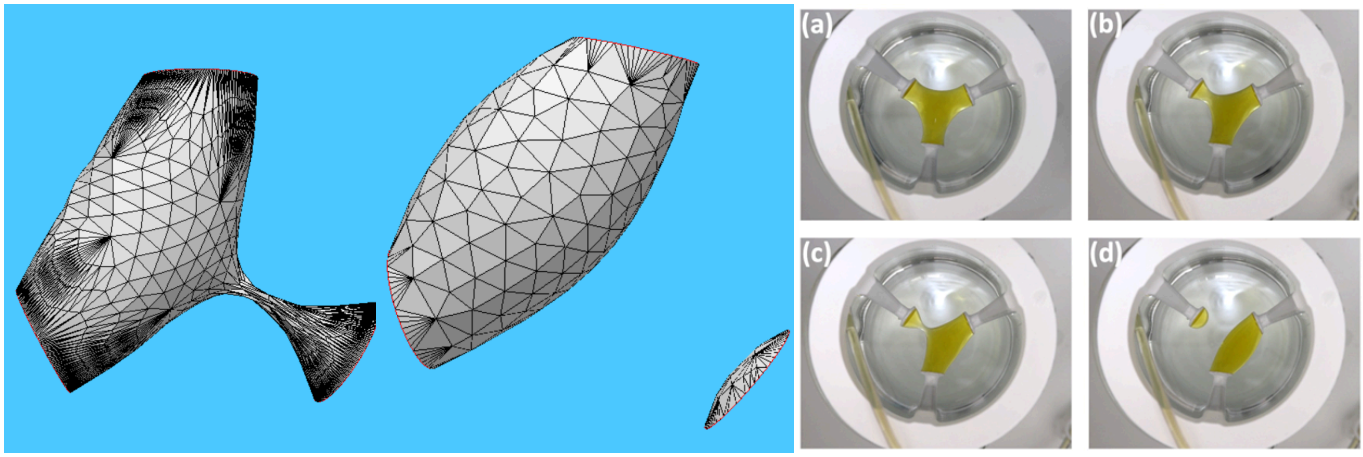
For each model, we compare the inversion runtime. In addition, we've added the relative mean area error for each model, displayed as percent on top of each bar. As can be seen, the symmetric matrix gained by remeshing is inverted faster. It's interesting to note that there's the greatest runtime difference when the original mesh is highly non-symmetric, as in the case of "Iphigena bad". Note that due to highly efficient algorithms, a symmetric matrix is not guaranteed to invert significantly faster than a non-symmetric matrix in all cases, as can be seen in the "Cat" model.

Liquid Simulation Results

Applying the remeshing algorithm from the paper **significantly improves** the regularity of meshes within Surface Evolver when integrated into a basic optimization scheme. This is shown in the figure below. On the left is the original mesh, and on the right is the result when remeshing during optimization.



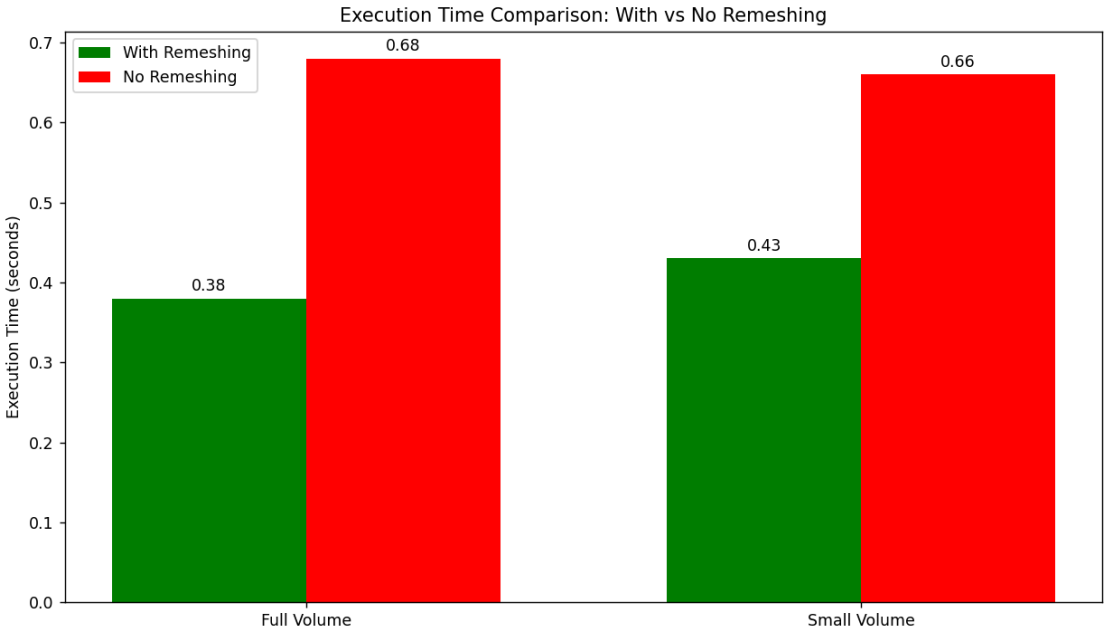
Most importantly, we found that the remeshing is essential in situations in which the volume is insufficient to hold the boundary conditions. Under such conditions, without remeshing, the optimization scheme cannot properly split the mass, thus failing to simulate the physical experiment. However, with the remeshing added, the optimization becomes true to the physics. This is shown below:



Left to right: simulation results without remeshing, simulation results with remeshing, images from the lab showing that the masses actually split.

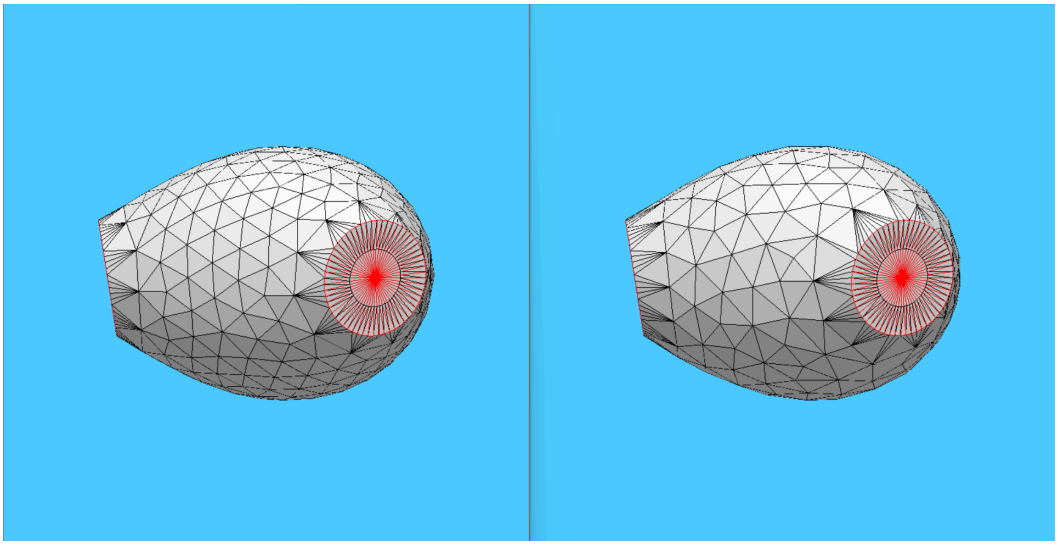
We further observed, that the simulation runs faster when we remesh during convergence, **even** when including the remeshing runtime in the results. Below is a comparison showing that. We note that even though the same amount of gradient steps, Hessian steps, and other optimization-related steps are performed, and on top of that remeshing is added at each iteration, the performance is improved by

remeshing by a factor of 1.5-1.8. These results are consistent over 10 timing iterations. See Running Our Code section to replicate these results.



Our experiments to determine on what edge-length to split and collapse showed that the values presented in the paper produced worse results than others in Surface Evolver. We believe this is caused by the edge conditions of the liquid (marked red in the simulation), which are usually finer and more detailed in order to capture the real conditions of the materials, at least when compared to the smooth liquid. These differences create challenging geometries which cannot be altered, and therefore require a more “relaxed” target for the remeshing algorithm. We have seen this experimentally in a few simulations, and found that using 1.5 and 0.5 as criteria work best. One such situation is presented here for visualization:

On the left remeshing with splitting edges over $1.5l$ and collapsing edges under $0.5l$
On the right remeshing with splitting edges over $1.34l$ and collapsing edges under $0.8l$, as in paper



Appendix - running the code

Our code both for the Python project and the simulations in Surface Evolver is available at https://github.com/Amir-Mann/DGPIsotropicRemeshing.

Running our Python remeshing project

The code was tested both on an Ubuntu and Window OS. The commands below are for Unix systems. Adjust the instructions for different operating systems.

1. **Navigate to the Project Directory**: Change to the project directory where `main.py` is located:
   `cd <path_to_your_project_directory>`
2. **Set Up Your Environment**: Ensure you have the required Python environment.
   a. You can create it using the `environment.yml` file included in the project. Run the following command in your terminal:
      `conda env create -f environment.yml`
   b. Then activate the environment:
      `conda activate DGPcourse`
3. **Run the Main Script**: Use the following syntax to run the remeshing script:
   `python3 main.py <model_name> [options]`
   a. Replace `<model_name>` with the path to your model file (e.g., `.obj`, `.json`, or `.off` format). All the ones on which results are presented are in the samples folder.
   b. **Available Options**: You can customize the execution with various options:
      - `--no_sliver`: Don't run sliver triangles checks before edge flipping improves runtime by ~50% but yields worse results.
      - `--foldover <FOLDOVER>`: Set the foldover angle threshold (default: π/9).
      - `--num_iters <NUM_ITERS>`: Specify the number of iterations for remeshing (default: 5).
      - `--L_factor <L_FACTOR>`: Set the factor for computing target edge length (default: 0.9).
      - `--save_stats`: Save statistics before and after remeshing.
      - `--visualize`: Visualize the mesh after remeshing.
      - `--save_to_obj`: Save the remeshed model in `.obj` format.
      - `--verbose_timing`: Print additional timing information.

Help Command: To see all available options and their descriptions, run:
`python3 main.py --help`

Example Command

Here's an example command incorporating options:

```
python3 main.py samples/iphi_bad10k.off --num_iters 10 --no_sliver
--visualize --save_to_obj
```

Running Surface Evolver

All simulations were originally run on Windows, and we are not sure if and what errors will arise from running them on a different operating system.
First of all, install Surface Evolver program to your machine as instructed in the Evolver's page.
Make sure you set Evolver to open files with the `.fe` extension.

In the project, there is a simulation folder with `.fe` files, which are the simulations that were in the demo at our project presentation during the semester and from which the images showing our remeshing quality were taken.
The naming scheme for the files is: `(full/sml)_volume_(no/wt)_remeshing` with:

1. `full` - Enough volume to keep all the boundary conditions.
2. `sml` - Small volume which would create a split in the mass, only connecting two of the boundary conditions.
3. `no` - No remeshing in the optimization process.
4. `wt` - With remeshing in the optimization process, the optimization process itself is unchanged.

The simulation process itself is splitted into user controlled steps which are grouped into s1, s2 and s3 commands in the Evolver's command language for ease of execution, with the same naming convention for the commands in all simulations.

In the simulation folder there is also the timing folder, in which are the same simulations but without visualization and without waiting for user inputs. And also a Python script to time these simulations. The data presented in the timing results is generated from the commands:

```
python time_simulations.py "full_volume_no_remeshing.fe > NUL 2>&1"
"full_volume_wt_remeshing.fe > NUL 2>&1" -r 10

python time_simulations.py "sml_volume_no_remeshing.fe > NUL 2>&1"
"sml_volume_wt_remeshing.fe > NUL 2>&1" -r 10
```