A Implementation of Medial Skeletonization Flow

Reported by: Lucien Li

Teammate: Pengze Li

1 INTRODUCTION

In this article, we delve into an in-depth exploration of the paper, *Mean Curvature Skeletons*, authored by Andrea Tagliasacchi and colleagues, which introduces the Medial Skeletonization Flow (MSF) to extract the "skeleton" of a mesh (Tagliasacchi et al., 2012). We begin by providing a comprehensive overview of their work, shedding light on the key concepts and methodologies employed. Following this, we introduce our own refined implementation, elaborating on the enhancements and modifications we have applied to optimize the algorithm's performance. Lastly, we showcase a series of rigorous experiments and evaluations to compare our pipeline to Andrea's MSF demo and the actual improvement brought by our innovation. Finally, we give some potential improvements to our MSF pipeline.

2 PAPER SUMMARY

Mesh skeletonization, as the name suggests, is the abstraction of a mesh into a corresponding skeleton, just like the skeleton in the human body, as Figure 1 shows. In this article, we set the target for skeletonization as a watertight manifold triangle mesh.

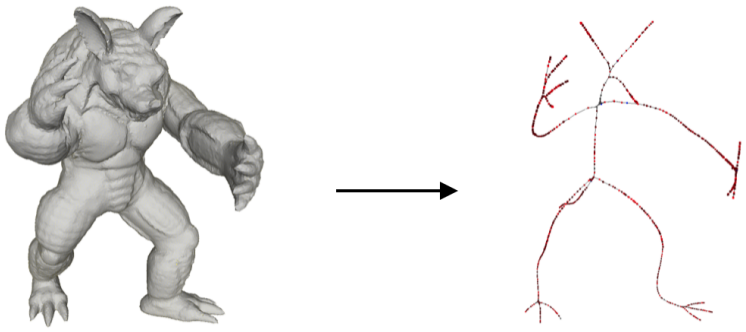

Figure 1. An example of mesh skeletonization

Tagliasacchi et al. (2012)'s paper *Mean Curvature Skeletons* builds on the work of Au et al. (2008)'s *Skeleton Extraction by Mesh Contraction*, which introduces a mesh skeletonization approach by solving a l_2 regularised discrete Laplacian equation. Tagliasacchi et al. (2012) improve this algorithm by adding a medial regularization.

2.1 Skeletonization by Mean Curvature Flow (MCF)

Au et al. (2008)'s method uses MCF to gradually contract a mesh into a skeleton by iteration. The whole process is guided by the mean curvature estimated by a Laplace–Beltrami operator. In each iteration, the main task is to solve:

$$\begin{pmatrix} \mathbf{W}_L\mathbf{L} \\ \mathbf{W}_H \end{pmatrix} \mathbf{V}' = \begin{pmatrix} 0 \\ \mathbf{W}_H\mathbf{V} \end{pmatrix} \tag{1}$$

, where $\mathbf{L}$ is the matrix form of Laplace–Beltrami operator, $\mathbf{V}', \mathbf{V}$ are vertices of a mesh after and before one iteration, and $\mathbf{W}_L, \mathbf{W}_H$ are weight parameters. As Figure 2 shows, the mesh shrinks gradually, and finally they compute a curve using the contracted mesh to be the final skeleton of it.

2.2 Medial Skeletonization Flow (MSF)

Tagliasacchi et al. (2012) optimized Au et al. (2008)'s algorithm by involving the approximation of medial axis by the Voronoi diagram. The equation to solve in each iteration will be different from Eqn 1.:

$$\begin{pmatrix} \mathbf{W}_L\mathbf{L} \\ \mathbf{W}_H \\ \mathbf{W}_M \end{pmatrix} \mathbf{V}' = \begin{pmatrix} 0 \\ \mathbf{W}_H\mathbf{V} \\ \mathbf{W}_M\mu(\mathbf{V}) \end{pmatrix} \tag{2}$$

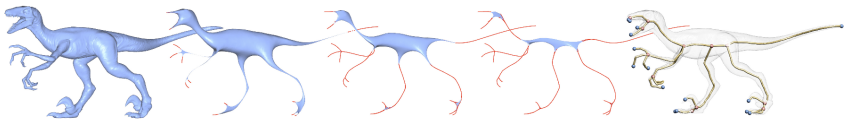

Figure 2. Figure 1 in the Au et al. (2008), shows the effect of their algorithm.

, where $\mu(\mathbf{V})$ represents the coordinate of the corresponding Voronoi poles (Amenta et al., 2001) of vertices in $\mathbf{V}$. As Figure 3(b) shows, the Voronoi poles supply a good approximation of the medial axis, thus constraining the final skeleton close to the real medial axis. The critical steps in MSF are:

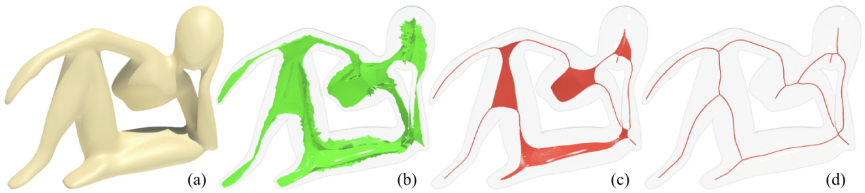

Figure 3. Figure 1 in the Tagliasacchi et al. (2012), (a)the origin watertight mesh, (b)the Voronoi poles as medial axis approximation, (c)meso-skeleton, (d)final skeleton

2.2.1 Compute Voronoi poles

Before the iteration starts, they compute the Voronoi diagram of the vertices in the mesh. Then for each vertex v_i there will be several corresponding Voronoi vertices w_i^j in the diagram, Voronoi pole for the vertex will be the Voronoi vertex which have minimum $\langle \mathbf{n}(v_i), w_i^j - v_i \rangle$, where $\mathbf{n}(v_i)$ is the normal vector of v_i.

2.2.2 Perform mesh contraction

Just solve Eqn 2., but $\mathbf{L}$ and weights need to be updated before this step in each iteration.

2.2.3 Update connectivity

This step is be done after mesh contraction in each iteration, which involves collapsing and splitting. Collapsing is to collapse edges that shorter than a threshold length, and splitting is to triangle faces that have an angle larger than a threshold angle. This step could speeds up the solving of Eqn 2. and avoids a post-processing step.

Algorithm 1 shows whole pipeline of Tagliasacchi et al. (2012)'s MSF.

Algorithm 1 Pseudocode of Tagliasacchi et al. (2012)'s MSF

```
function COMPUTE_SKELETON(mesh M_in)
    M ← Resample(M_in)
    vor ← Voronoi_poles(M)
    while has_volume(M) do
        UPDATE_LAPLACIAN_AND_WEIGHTS(M, vor)
        PERFORM_MESH_CONTRACTION(M, vor)
        UPDATE_CONNECTIVITY(M)
    end while
    COLLAPSE_SHORTEST_EDGES(M)
    return M
end function
```

3 IMPLEMENTATION

We implement a Python version of MSF with some improvements. We use Trimesh to maintain mesh data, and Scipy, Numpy for sparse linear algebra and spatial (Voronoi diagram) calculating. The whole process of our MSF can be seen

in the Algorithm 2. Overall, we replace resample process with reconstruct before iteration, add mesh smoothing and updating length threshold for collapsing in the iteration. Codes are in this repository.

Algorithm 2 Pseudocode of our improved MSF

```
function COMPUTE_SKELETON(mesh M_in, float scale)
    M_rec ← Reconstruct(M_in)
    vor ← Voronoi_poles(M_rec)
    M ← M_in
    while has_volume(M) do
        L ←UPDATE_LAPLACIAN(M)
        M ← PERFORM_MESH_CONTRACTION(M, L, vor)
        M ← LAPLACIAN_SMOOTHING(M)
        M ← OPTIMIZE_MESH_AND_UPDATE_WEIGHTS(M, vor)
        M ← FIX_DEGENERATE_VERTICES(M, scale)
        scale ← UPDATE_SCALE(scale)
    end while
    return M
end function
```

3.1 Preprocessing

In our implementation, we do mesh reconstruction and computing Voronoi poles before iteration.

3.1.1 Mesh reconstruction

We perform mesh reconstruction for it can remove useless details on the mesh which are unnecessary for computing the skeleton. In Figure 4., we can see after reconstruction the folds on the surface of the mesh disappear, but the main structure of his body is still well maintained which is more important for skeletonization. We use SVD and uniform-

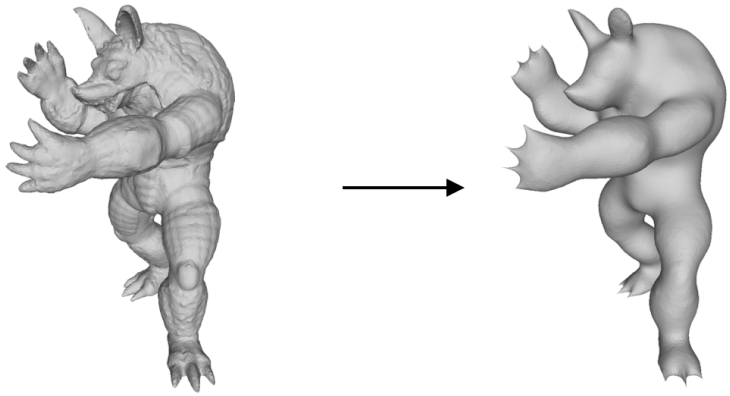

Figure 4. Effect of mesh reconstruction

Laplace-Beltrami based reconstruction method, which is in this part of codes. The mesh in Figure 4. use eigenvectors corresponding to the first 400 small eigenvalues of the uniform-Laplace-Beltrami matrix to reconstruct. However, a over-high degree of reconstruction may cause the mesh to lose too much information and affect the performance of the skeletonization, and also take much longer time to compute, so we set parameters to control whether perform reconstruction or not and how much eigenvectors to use if reconstruct.

3.1.2 Compute Voronoi poles

We also recommend using reconstructed mesh to compute Voronoi poles. Since Voronoi poles is an approximation for the medial axis, so mesh reconstruction can also improve its performance, as Figure 5. shows. We can see the Voronoi poles generated from reconstructed mesh are closer to the actual medial axis. The right part in Figure 5. uses the first 200 small eigenvalues of the uniform-Laplace-Beltrami matrix to reconstruct. Also, we set parameters to control the reconstruction in this part. The codes calculating Voronoi poles can be seen here.

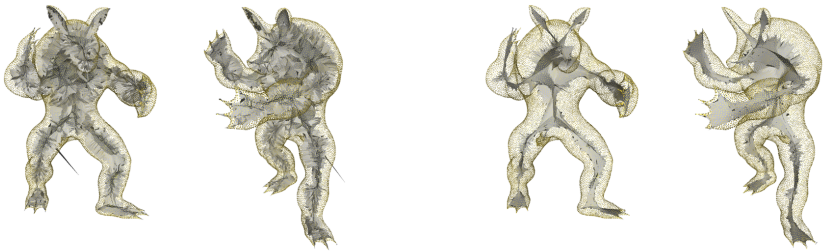

Figure 5. Left: compute Voronoi poles directly; Right: compute Voronoi poles with reconstructed mesh

3.2 Iterative mesh contraction

After preprocessing, we can perform mesh contraction now to skeletonize the mesh. We set 5 steps in the iteration: Computing Laplacian operator (Sec. 3.2.1), performing mesh contraction(Sec. 3.2.2), mesh smoothing (Sec. 3.2.3), optimizing mesh (Sec. 3.2.4), fixing degenerate vertices (Sec. 3.2.5) and updating threshold (Sec. 3.2.6). There are new steps compared to Tagliasacchi et al. (2012)'s algorithm, and also other steps have our own small innovations.

3.2.1 Computing Laplacian operator

In the paper *Mean Curvature Skeletons* by Tagliasacchi et al. (2012), they propose using cotangent-weighted Laplace-Beltrami operator. However, we find that a mean-value-weighted (Amenta et al., 2001) Laplace-Beltrami operator has better performance than cotangent one. Recall that Laplace-Beltrami operator is to pull each vertex v_i towards a specific vector Δv_i:

$$\Delta v_i = \sum_{v_j \in \mathcal{N}_1(v_i)} w_j (v_j - v_i) \tag{3}$$

, where $\mathcal{N}_1(v_i)$ are the one-ring neighbours of v_i and w_j is the weight. In Figure 6., the weight mean-value-weighted Laplace-Beltrami operator for v_0 will be:

$$w_i = \frac{\tan\frac{\alpha_{i-1}}{2} + \tan\frac{\alpha_i}{2}}{||v_i - v_0||_2} \tag{4}$$

. The code for the mean-value-weighted Laplace-Beltrami operator can be seen here.

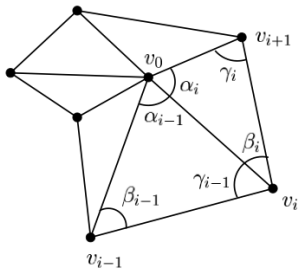

Figure 6. Figure 1 in Amenta et al. (2001)

3.2.2 Performing mesh contraction

Basically, this step is solving Eqn 2. We use Scipy sparse linear algebra library to do solve the equation. All the matrices are in sparse representation, and we use the LSQR solver to solve the linear system. However, the linear system in Eqn 2 can easily be ill-posed, which will affect the convergence speed and performance of the LSQR solver. We make two adjustments to manage this situation: the first one is to set current coordinates of the vertices as the initial guess of LSQR. The second one is corresponding to the paper Tagliasacchi et al. (2012). During the iteration, the mesh will gradually contract to the skeleton, but some vertices may reach the skeleton axis, which is the final position of it, so we should detect them and then "fix" them during iteration, which is just the step in Sec. 3.2.5. Tagliasacchi et al. (2012) propose to of such vertices set the weight in $\mathbf{W_H}$ to a significant large number (like 10^9) and the weight in $\mathbf{W_H}$ and $\mathbf{W_M}$ to 0. However, the significant large number will cause the condition number of the linear system to be extreme large. We perform experiment on a mesh with 3000 vertices and fix 3 arbitrary vertices on it. We estimate the condition number of

matrix $\begin{pmatrix} \mathbf{W}_L\mathbf{L} \\ \mathbf{W}_H \\ \mathbf{W}_M \end{pmatrix}$ by $cond(A) = ||A||_2||A^\dagger||_2$ where $A^\dagger$ is the pseudo-inverse and it is approximately 4×10^{10}. It is an extreme large condition number and it indeed make the performance of LSQR worse. So we make a adjustment, assume v_i is fixed after one iteration, then make Δv_i in Eqn 3 equals to zero vector and set its weight in $\mathbf{W_H}$ to zero. As we also set the initial guess of LSQR to the current coordinates of vertices, v_i will not eager to move and the condition number also decreases from 4×10^{10} to 1×10^5.

3.2.3 Mesh smoothing

During the implementation, we found that only implementing the steps in Algorithm 1, the mesh contraction will converge at the status of meso-skeleton as the left part of Figure 7. Since the system converges in a middle status, we

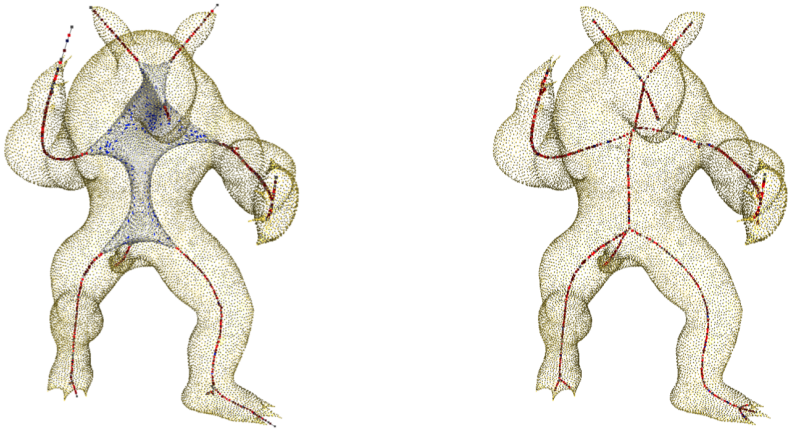

Figure 7. Left:A converged meso-skeleton; Right: Completely converged to a skeleton after adding mesh smoothing

want to try to add some perturbations to the system so that the iterations could continue efficiently. A highly effective method is mesh smoothing, and another one is updating threshold in Sec 3.2.6. We use a implicit Laplacian mesh smoothing (Desbrun et al., 1999) after mesh contraction in each iteration. After this adjustment, our MSF iteration can converge to a skeleton nicely.

3.2.4 Optimizing mesh

This step has the same function as the update connectivity 2.2.3 in Tagliasacchi et al. (2012)'s MSF. This step contains two part: edge collapsing and face splitting. Edge collapsing will collapse edges shorter than a threshold t_c, which can be estimated according to diagonal length of the mesh's bounding box. The two vertices belonging to the edge that will be collapsed will be merged into their midpoints. If there is an angle on a triangular face greater than a certain threshold θ_s, then its opposite side will be split. There are some points need to be noticed. For edge collapsing, if the two vertices on the edge to be collapsed do not have exactly two common one-ring neighbours, then the edge could not be collapsed, as Figure 8. shows. For face splitting, it the edge to be split is too short, then we should not split it, preventing infinite

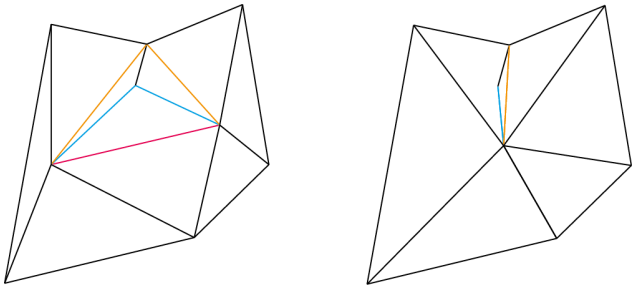

Figure 8. [link] The red edge could not be collapsed, or the triangle between the orange and cyan edge is not manifold anymore.

splitting. The parameter t_c and θ_s can be set here.

3.2.5 Fixing degenerate vertices

As mentioned in Sec. 3.2.2, this step is used to fix points that already reach the final position during the MSF iteration. If a vertex have two one-ring neighbours satisfy: the edge between them is shorter than a threshold t_f and cannot be collapsed, then the vertex can be marked as fixed. t_f can be estimated by $\frac{t_c}{10}$. The code of this step can be seen here.

3.2.6 Updating threshold

As mentioned in Sec. 3.2.3, this step can help the system escape from being caught in a status of meso-skeleton. In particular, this step adjust t_c in each iteration. In our implementation, t_c will increase from t_{min} at the step length of Δt during the iteration. After reaching the set upper limit t_{max}, it will recover to t_{min}. This step can also accelerate the convergence. However, this step is not so important as Mesh smoothing in Sec. 3.2.3. The parameter $t_c, t_{min}, t_{max}, \Delta t$ and whether using this step can be adjusted in the configure file.

4 EXPERIMENTS AND EVALUATION

In this section, we perform experiment on testing how can the variation of different parameters influence the effect of skeletonization. Then, we compare the quality of skeletonization between our implementation and Tagliasacchi et al. (2012)'s demo software. All the experiments on our implementation in this article are performed on a MacBook Pro with Apple M2 pro.

4.1 Experiments on parameters

4.1.1 $\mathbf{W}_L, \mathbf{W}_H, \mathbf{W}_M$

$\mathbf{W}_L, \mathbf{W}_H$ and $\mathbf{W}_M$ are all square matrices with diagonal element w_L, w_H and w_M. The absolute values of w_L, w_H and w_M is not so important (just keep them in appropriate values), but their relative values matter. From Figure 9, we can see when w_H/w_M is fixed, the larger w_L/w_H is, the severe the contraction in the first iteration is, and the worse the skeleton is. If w_H is overlarge, the vertices will be close to their original position in the first iteration, some area with median curvature may can't degenerate to skeleton. If w_M is overlarge, the vertices will be close to their Voronoi pole in the first iteration, some area with low curvature may evolve to skeleton.

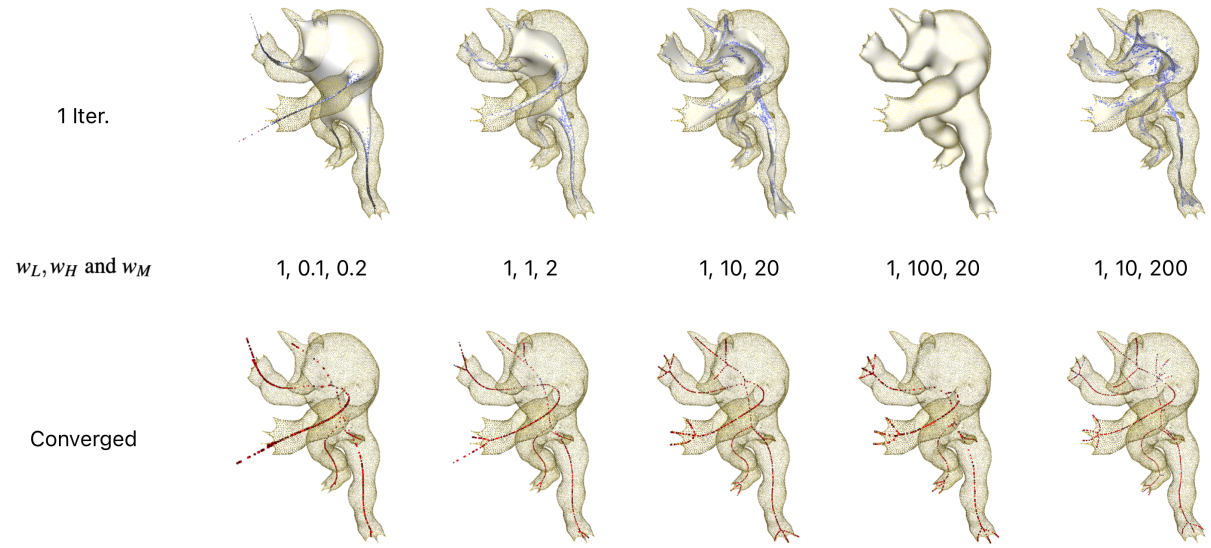

Figure 9. Experiment with different w_L, w_H and w_M.

4.1.2 Mesh reconstruction

We experiment with replacing original mesh with reconstructed mesh as the input of iteration. We can see from Figure 10 that the less eigenvectors we use, the less detailed the reconstructed mesh and final skeleton is. Also, large number of eigenvectors cost much more time to compute, so maybe mesh reconstruction is not so necessary to replace the original mesh, but it really helps improve the quality of Voronoi poles as Figure 5.

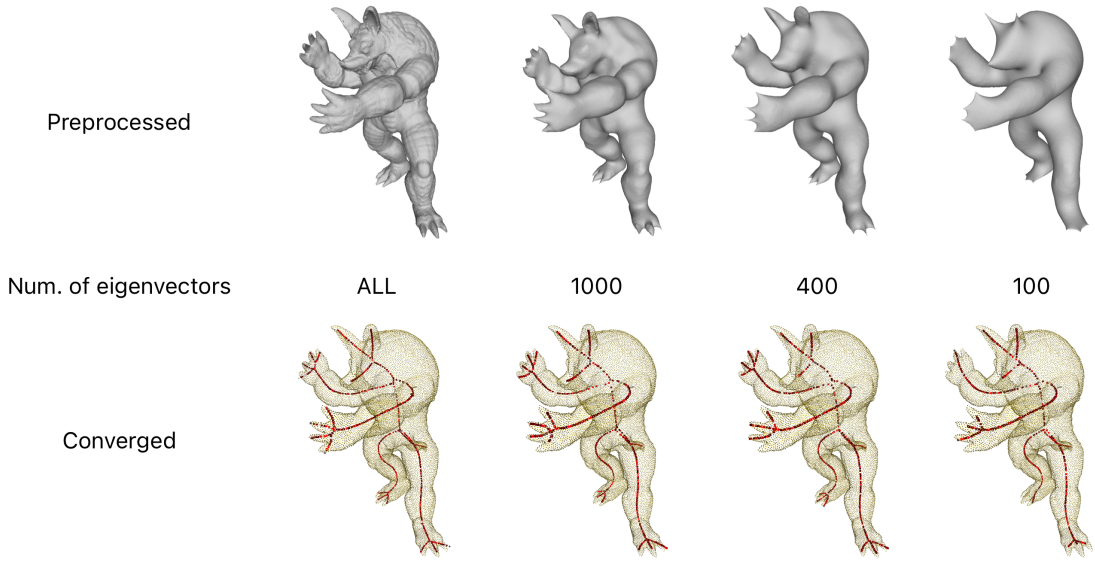

Figure 10. Experiment with different numbers of eigenvectors.

4.1.3 Laplacian smoothing

We experiment with different step length (λ) of Implicit Laplacian Smoothing. Although it is critical for converging to the skeleton, but we can see from Figure 11. that larger λ will cause the skeleton over-shrinking. A small *lambda* is enough for the convergence of skeleton.

4.1.4 Updating threshold

This step does not improve the actual performance of skeletonization too much, but it accelerate the convergence of algorithm. For the default parameter set, updating threshold could reduce $5-10$ iterations to converge.

4.2 Comparing performance

We compare our version of MSF to Tagliasacchi et al. (2012)'s demo software. Maybe due to the software is developed a decade ago, it is very unstable and it can only work with a small range of parameters without crashing. As Figure 12. shows, for the simple mesh (leftmost one), it can work well, so as our implementation. However, for more complex models like the one in the middle and right, it can converge to a not so detailed skeleton, with missing of some fingers, or with the fingers but can hard converge to a skeleton.

5 CONCLUSION

We implement our version of Medial skeletonization flow and get a good performance. However, there can still be some improvements like calculate a curve to represent skeleton, and implement our own data structure to maintain the mesh to accelerate the whole algorithm further.

REFERENCES

Amenta, N., Choi, S., and Kolluri, R. K. (2001). The power crust, unions of balls, and the medial axis transform. *Computational Geometry*, 19(2):127–153. Combinatorial Curves and Surfaces.

Au, O. K.-C., Tai, C.-L., Chu, H.-K., Cohen-Or, D., and Lee, T.-Y. (2008). Skeleton extraction by mesh contraction. *ACM Trans. Graph.*, 27(3):1–10.

Desbrun, M., Meyer, M., Schröder, P., and Barr, A. H. (1999). Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, page 317–324, USA. ACM Press/Addison-Wesley Publishing Co.

Tagliasacchi, A., Alhashim, I., Olson, M., and Zhang, H. (2012). Mean curvature skeletons. *Computer graphics forum*, 31(5):1735–1744.

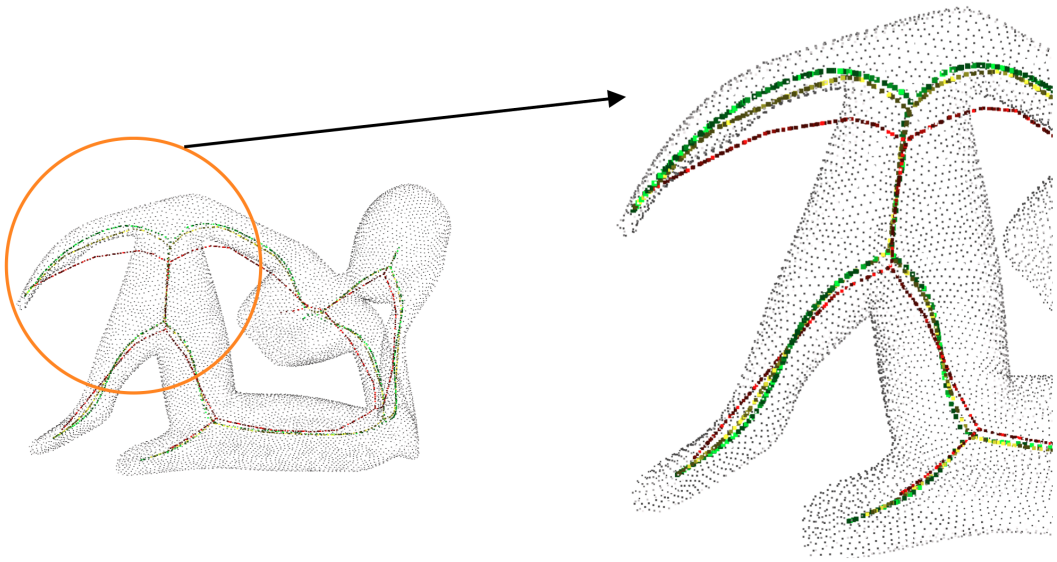

Figure 11. Experiment with different λ of Implicit Laplacian Smoothing. Red skeleton: $\lambda = 0.1$; Yellow skeleton: $\lambda = 0.01$; Green skeleton: $\lambda = 0.001$;

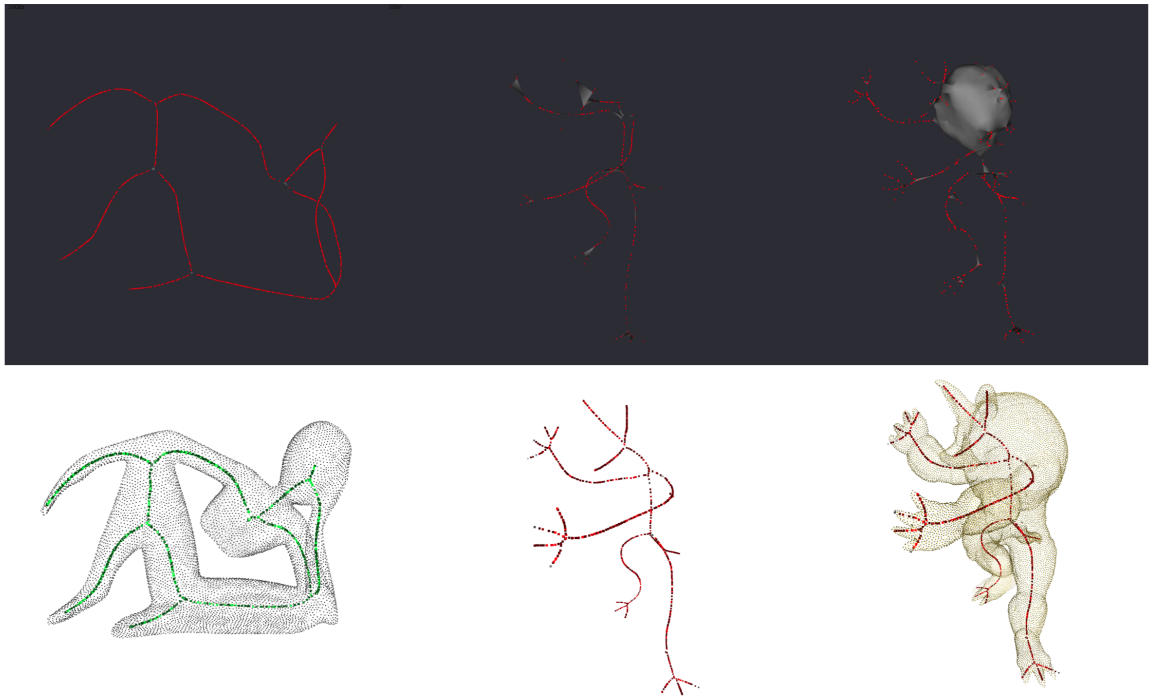

Figure 12. Above: Tagliasacchi et al. (2012)'s demo software; Below: Ours result.