

Supplementary Material

Generalizing Curve Skeletonization to Continuous Domains

A Implementation Details	2
B Pseduocode	2
C Derivation for determining intersection within a face of a mesh	3
D Evaluation metric	3
E Curve Shortening using Edge Flip framework	4
E.1. Optimizing the loop	4
E.2. Constraining the Loop	4
F. Curve Shortening using optimization based framework	4
F.1. Optimizing the Loop	4
F.2. Constraining the Loop	4
G Our adaptation of the Practical Cut Locus Identification Method	5
G.1. Our adaptation on intrinsic meshes	5
G.2. Our adaptation on point clouds	5
H Additional results	5
H.1. Complete Results	5
H.2. Results on Complex Meshes	6
H.3. Results on Meshes with Holes	7
H.4. Results on Noisy Meshes	7
H.5. Results on Mesh Resolution	7
H.6. Timing Analysis of CSCD-M	7
H.7. Additional Qualitative Results of CSCD-PC	10
I. Downstream Applications	11
I.1 . Shape Classification	11
I.2 . Shape Segmentation	11
I.3 . Identifying Handles, Tunnels, and Constrictions	11
J. Ablations	11
J.1 . Choice of Geodesic Distance vs Euclidean Distance	11
J.2 . Number of Local Separators calculated in Stage 1	11
K Discussions	11
K.1. On the derivation of LS from CSCD	12
K.2. On the multiscale version of CSCD-M	12
L Limitations and Future Work	12
M Terminology and Definitions	19
N Related Works	19

A. Implementation Details

The code for meshes is implemented in C++, using the IGL [19] and geometry-central [40] package for geometry processing. The code for point clouds is implemented in Python, using PyIGL [19], Open3D [61] and PotPourri3D [40] for geometry processing. KDTree from scikit-learn [34] is used to implement MLS.

B. Pseduocode

We provide in the following section, the pseudocode for our algorithm. The entire framework can be divided into 2 broad stages as depicted in Fig. 3. Stage 1 is to find a set of local separators on the 3D object. Given an input 3D object, we sample a point on the surface. We then calculate the geodesic distance from the source to all other points and identify the target cut locus for the source. We then trace an approximate path, which is then optimized using curve shortening. In Stage 2, we first score the separators, and then select an optimal set of separators from overlapping separators. These steps are followed by assigning neighbouring regions to local separators, calculating the centroid of each region and connecting centroids of neighbouring regions to form the final curve skeleton.

Algorithm 2 Obtain curve skeleton from local separator

Input: A 3D mesh, the number of local separators to construct N

Output: A curve skeleton \mathcal{C} (graph with interior-mesh nodes)

- 1: $LS \leftarrow \{\}$
 - 2: $P_V \leftarrow \mathbb{U}_V$ ▷ Uniform sampling over all mesh vertices
 - 3: **for** $t \leftarrow 0, 1, \dots, N - 1$ **do**
 - 4: $s \sim P_V$ ▷ Sample a source vertex
 - 5: $LS_t \leftarrow \text{local_separator}(s)$ ▷ Alg. 3
 - 6: Update P_V via Sec. 3.1.7
 - 7: Compute weight w_t via Sec. 3.2.1
 - 8: $LS \leftarrow LS \cup \{(LS_t, w_t)\}$
 - 9: **end for**
 - 10: Prune bad local separators (Sec. 3.2.2)
 - 11: $LS \leftarrow \text{set_packing}(LS)$ ▷ Sec. 3.2.3
 - 12: Assign nearest separator to each mesh vertex
 - 13: Compute centroids for each region
 - 14: Connect neighboring regions and remove cliques to form \mathcal{C}
-

Algorithm 3 Obtain a local separator

Input: A mesh with a source point s

Output: A set of points in \mathbb{R}^3 as the local separator

- 1: Compute distance field D (e.g. via the Heat Method [9])
 - 2: Identify cut loci using the algorithm of [30] (see Sec. 3.1.2)
 - 3: Select target cut locus t by the locality constraint (min-Euclid or min-geodesic; Sec. 3.1.3)
 - 4: Determine incoming directions v_1, v_2 (Sec. 3.1.4)
 - 5: Trace two paths from t to s using Alg. 4, yielding $\text{Path1}, \text{Path2}$
 - 6: Connect Path1 and Path2 into a closed loop loop
 - 7: Optimize loop using intrinsic edge-flips ([39]) or the procedure in [60]
-

Algorithm 4 Trace direction to source s

Input: A mesh with distance field D , a target cut locus t , source s , and an incoming direction v_i

Output: A vertex path from v_i to s

```
1:  $Path \leftarrow [v_i]$ 
2: while  $\text{last}(Path) \neq s$  do
3:    $\min D \leftarrow +\infty$ 
4:    $bestV \leftarrow \text{null}$ 
5:   for all  $n \in \text{adj}(\text{last}(Path))$  do
6:     if  $D[n] < \min D$  then
7:        $\min D \leftarrow D[n]$ 
8:        $bestV \leftarrow n$ 
9:     end if
10:  end for
11:  Append  $bestV$  to  $Path$ 
12: end while
```

C. Derivation for determining intersection within a face of a mesh

We derive below the equation that needs to be evaluated to check whether two barycentric vectors are intersecting within the face of a mesh. This is used to check for the overlap of local separators in CSCD-M.

Let the first vector be given by two barycentric points (u_1, v_1, w_1) , (u_2, v_2, w_2) , and the second vector by given by another set of two barycentric points (u_3, v_3, w_3) , (u_4, v_4, w_4) .

Now, if we parameterize the lines using t_1 and t_2 , we have:

$$L_1(t_1) = (u_1 + t_1(u_2 - u_1), v_1 + t_1(v_2 - v_1), w_1 + t_1(w_2 - w_1)) \quad (2)$$

$$L_2(t_2) = (u_3 + t_2(u_4 - u_3), v_3 + t_2(v_4 - v_3), w_3 + t_2(w_4 - w_3)) \quad (3)$$

At the intersection, $L_1(t_1) = L_2(t_2)$. For barycentric coordinates, we have $u + v + w = 1$ for all $u, v, w \in \mathbb{R}$. Therefore, we can pick u, v and set $w = 1 - u - v$. Now, the above relation boils down to two linear equations with two unknowns t_1 and t_2 :

$$u_1 + t_1(u_2 - u_1) = u_3 + t_2(u_4 - u_3) \quad (4)$$

$$v_1 + t_1(v_2 - v_1) = v_3 + t_2(v_4 - v_3) \quad (5)$$

$$(6)$$

, which has exactly one solution,

$$t_1 = \frac{u_1(v_4 - v_3) + u_3(v_1 - v_4) + u_4(v_3 - v_1)}{(u_1 - u_2)(v_4 - v_3) - (u_4 - u_3)(v_1 - v_2)} \quad (7)$$

$$t_2 = \frac{u_1(v_2 - v_3) + u_2(v_3 - v_1) + u_3(v_1 - v_2)}{(u_1 - u_2)(v_4 - v_3) - (u_4 - u_3)(v_1 - v_2)} \quad (8)$$

,

where the denominator is the same and is zero if and only if the two lines are parallel.

If $0 \leq t_1 \leq 1$, then the intersection is on the line segment between the specified points of the first line. Similarly for t_2 .

The point lies within the face if $0 \leq u \leq 1, 0 \leq v \leq 1$ and $0 \leq w \leq 1 \iff 0 \leq u + v \leq 1$, which is always the case here since the point lies at the edge of the face.

D. Evaluation metric

Due to the unclear definition of curve skeletons for 3D objects, there is no single agreed-upon evaluation metric for quantifying the quality of the skeleton. Here, we choose to evaluate the reconstruction quality of the mesh. Each node of the skeleton corresponds to a sphere of a radius r , which we use to perform convolutional surfaces [42] to reconstruct the original mesh. This metric is well-suited since it requires that the nodes not be centered (due to the isotropic nature of the radius), but also capture the finer details of the mesh.

We describe briefly the evaluation process given the reconstructed mesh obtained from applying convolutional surfaces: We compute the Euclidean distance between points on the real object against a location on the reconstructed object. This differs from the standard Chamfer distance, which typically compares two point clouds. In this case, the evaluation compares a set of points – the vertices of the original mesh – against a mesh representing the reconstructed object. Notably, the closest point on the reconstructed mesh may not always be a vertex; it could instead lie on a face.

$$\epsilon_i = \min_{\hat{\mathbf{p}} \in \hat{\mathcal{M}}} \|\mathbf{p}_i - \hat{\mathbf{p}}\|^2 \quad (9)$$

where ϵ_i is the error for vertex i , \mathbf{p}_i is the position of vertex i , and $\hat{\mathbf{p}}$ is a point on the reconstructed mesh $\hat{\mathcal{M}}$.

E. Curve Shortening using Edge Flip framework

E.1. Optimizing the loop

For CSCD-M, we use the curve shortening framework described in [39] as it works out-of-the-box for intrinsic triangulation schemes and is fairly robust. In intrinsic triangulation schemes, the diagonal edges of a quadrilateral, formed by two adjacent triangles, may be flipped. The geodesic curve shortening algorithm utilizes this detail and works by iteratively flipping edges in a wedge, until the interior angle of the two edges connecting two non-adjacent vertices on the path is greater than π . Therefore, one can begin with an approximate path given by a series of vertices and iteratively smoothened it to the final geodesic path by flipping edges connecting the vertices. In the case of loops, there is no fixed start and end vertices, which causes the loop to become geodesic loops.

E.2. Constraining the Loop

To prevent the optimized loop from moving too far from the initial approximate loop, we apply a locality constraint. This is done through restricting the edge flipping procedure to operate only upon certain vertices that are within the locality constraint. This would ensure that the curve shortens and smoothen within the locality but doesn't slide away.

F. Curve Shortening using optimization based framework

F.1. Optimizing the Loop

We employ the method described in [59]. The method proposes it as an optimization problem and solves it using projected gradient descent. We discuss below the method:

Let us take a loop \mathbb{L} with n points $p_1, p_2, p_3, \dots, p_n$ such that there is connectivity between consecutive points and between p_n, p_1 . Assuming proper initialization, it has been shown that the loop can be converted into a geodesic loop by (i.e, geodesic minimization is equivalent to) minimizing the following loss [59]:

$$\mathcal{L} = \sum_{i=1}^n H(\|x_i - x_{i+1}\|_2) \quad (10)$$

where x_i is the Euclidean position of point p_i , and H is a convex kernel function. The kernel function $H(s) = e^{s^2} - 1$ has been shown to work very effectively, leading to quick convergence rates [59], so we use the same kernel.

In order to optimize the loop \mathbb{L} , the gradient of \mathcal{L} is projected onto the surface of the object as:

$$\bar{\nabla}_i \mathcal{L} = \nabla_i \mathcal{L} - \nabla_i \mathcal{L} \cdot N_i$$

where N_i is the normal to point p_i . After each step of the optimization, the points are reprojected onto the surface of the point cloud using MLS [14]. The optimization runs until we reach a sufficiently small step size or a sufficiently small gradient norm.

F.2. Constraining the Loop

We implement the Euclidean distance constraint as a penalty to the original objective (10):

$$\tilde{\mathcal{L}} = \mathcal{L} + \sum_{i=1}^n \lambda_i \max(0, \|x_i - x_s\|_2 - r)^2 \quad (11)$$

where $r = \|x_p - x_s\|_2$, i.e., the Euclidean distance between the source point and the most distant point, and λ_i are hyperparameters for weighing the penalty function. One could apply interior point methods to simultaneously fit λ_i and the positions x_i , but we find that simply setting $\lambda_i = 10$ works pretty well. This constraint prevents the loop (separator) from moving too far from the initial region or collapsing onto a single point.

For CSCD-M, we also experimented with the optimization based framework, but could not get it work as efficiently as mentioned in [60]. The optimization based framework requires the position in \mathbb{R}^3 , but provides greater flexibility than the edge flip method since does not depend upon the triangulation scheme.

G. Our adaptation of the Practical Cut Locus Identification Method

G.1. Our adaptation on intrinsic meshes

We adapt the practical cut locus mentioned in Ref. [30] for intrinsic mesh triangulation. The key ingredients in the algorithm are the computation of the geodesic distance, the computation of the gradients of the distance field and the connectivity/neighbourhood for the ORG tree. We adapt all of the three steps to operate upon the intrinsic vertices and edges.

1. Firstly, we calculate the geodesic distance using Heat method on the vertices of the intrinsic triangulation. This provides improved robustness to poor triangulations.
2. Secondly, we work purely within the tangent spaces of each vertex and each face. This requires us to perform parallel transport of gradients onto the same frames of reference whenever we need to do some comparison, such as when we want to identify angle between neighbouring gradients. This affects the final smoothing step too, where we work with barycentric surface points on the faces of the mesh, and transport and interpolate the gradients appropriately.
3. Finally, when constructing the ORG tree and pruning the ORG tree, we operate purely upon the intrinsic triangulations.

We find that by working with the intrinsic mesh triangulations, our implementation results in a more robust algorithm. In Fig. 13 for the mesh TID:44395, we see that our method results in a more stable and consistent cut loci given identical starting points (red), while the original method results in considerably more number of false negatives. For certain meshes, the original method completely fails to work, or perform the homology step of the algorithm [30]. Of special importance to CSCD-M is that with our adaptation the target cut locus is the correct one that goes around the geometric feature.

G.2. Our adaptation on point clouds

We also adapt the practical cut locus algorithm from Ref. [30] to operate on point clouds. With our adaptation, we have effectively developed a new method for cut locus computation for point clouds.

Our adaptation resembles the original algorithm. First, we construct a neighbourhood using the k nearest neighbours algorithm. This neighbourhood is then pruned by removing edges that connect points between two previously unconnected regions of the object. We then use the heat method to compute the geodesic distance. We construct the ORG tree associated to the heat distance. The gradients of the distance field are obtained by applying the pre-calculated gradient operator to each point. We apply the remaining process similar to the original algorithm with some changes to account for the irregular structure here. For instance, when calculating the angle deviation the gradient within the neighbourhood of the point, we have to be careful in correctly restricting the search space to the immediate neighbourhood of the vertex.

We illustrate the performance our method on a set of fairly complex objects in Fig. 14. Our results, though noisy, seem to indicate that the main structure of the cut loci is captured effectively. The noise can be controlled by tuning the free parameters such as the laplacian threshold and the angle deviation threshold of the gradients. For details on those parameters, see Ref. [30].

H. Additional results

H.1. Complete Results

We present the complete quantitative results for our methods and additional methods on a larger set of objects in Tab. 5. From the table, we can see that our method performs comparably to both LS [2] and MSLS [3]; however, MSLS outperforms us consistently. MCF [46] performs the worst, but surprisingly outperforms both MSLS and ours on some tubular-shaped objects. As mentioned earlier, there is a large scope for improvement, including developing multiscale approaches. This will help alleviate the primary limitation of our method which is the number of local separators calculated – by having a multi-scale approach we could scale up the number of separators significantly.



Figure 13. Comparison of our estimated cut loci (right) and the original code (left) [30]. Furthermore, the original implementation fails to run the homology step of the algorithm. Our adaptation produces a more robust output that selectively chooses the points on the sides away from the source (red) point, while the original code has significant false negatives.

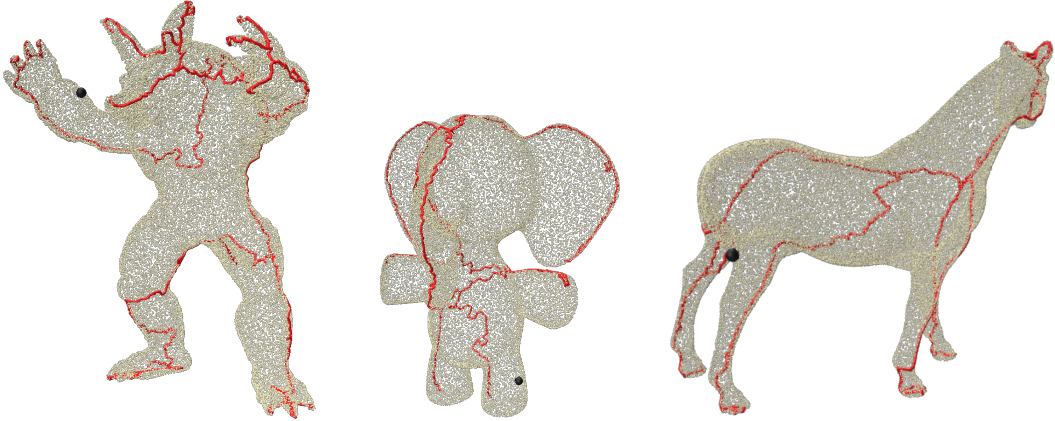


Figure 14. Our adaptation for cut locus identification on point clouds on various point clouds armadillo, elephant, horse. The black point is the source point, and the red curve is the cut locus of the point.

H.2. Results on Complex Meshes

In Fig. 15, we also show qualitative results of our method vs LS [2] on a set of complex meshes. We find that our method performs similar to LS when representing the mesh. Here it is set to a fixed value of $N = 4096$. In future works, one can explore the automation of this choice, or a better way to choose N .

Table 5. Convolutional Surfaces reconstruction error ($\times 10^{-3}$) for objects using skeletons from CSCD-M (Ours), MCF [46], LS [2] and MSLS [3]. MCF performs the worst of all the methods. CSCD-M and the two variants of LS are comparable. Though MSLS results in slightly better skeletons.

Object	LS	MSLS	MCF	CSCD-M (Ours)
armadillo	11.10	08.27	08.09	08.84
Copper-key	06.13	04.60	6.02	04.70
rocker-arm	26.30	24.40	25.30	24.90
fertility	17.50	12.20	15.40	13.60
gorilla	6.13	06.71	06.90	07.00
neptune	04.16	04.82	10.62	03.80
TID:37358	4.852	04.78	11.18	05.08
TID:44395	09.56	10.70	16.80	10.07
TID:39878	03.20	03.92	15.83	05.03
TID:40987	09.29	09.81	11.36	08.32
TID:80516	10.40	10.11	09.92	10.49
TID:82324	03.73	04.17	04.17	04.49
TID:133568	04.75	05.49	05.51	04.99
TID:133079	05.27	06.72	05.27	07.03
Carcinoplax-Suruguensis	03.69	03.58	05.47	05.36
Average	08.40	8.02	10.52	08.25

H.3. Results on Meshes with Holes

Our current realization doesn’t handle all the cases with holes, as mentioned in Sec 2. Nevertheless our framework can in principle handle all these cases. One particular case (namely, with only a single boundary) and its solution is mentioned in Sec. 2. In that case, we talk about connecting points in the boundary where the gradient seems to point in opposite directions, such that they cancel each other. We, however, note that this would miss few local separators for cases with more than a single boundary. Any two points on two separate boundaries and the path connecting between them could act as local separator too.

A particular case where our current realization cannot work is when there is a longitudinal hole in a torus such that it may be completely flattened onto a 2D sheet. This is because now the (open) path connecting two points in the two boundaries would act as a local separator.

In Fig. 17, we show illustrative results for meshes with small holes and how local separators are able to wrap around it. The torus has three holes – two small holes back-to-back on the right side, and a break on the left side. Notice that our curve skeleton does not close because the torus itself completely breaks at one point. This is topologically valid since the nature of the input object has itself changed. We still outperform both LS and MSLS though. LS completely fails to run on this object with holes and MSLS while producing a curve skeleton does so more poorly compared to our method, changing the topology of the mesh itself.

H.4. Results on Noisy Meshes

We test CSCD-M on a small subset of noisy meshes to evaluate its robustness against noise in the vertex positions. We create the noisy objects by adding standard normal noise proportional to the average edge length τ of the mesh.

H.5. Results on Mesh Resolution

We present results with low mesh resolution in Fig. 19. Both LS and MSLS lead to distorted skeletons especially around the torso of the *armadillo*. Our method reliably generates the skeleton – we believe this is due to our ability to capitalize on the dual areas of the vertices which allows us to handle unequal sized faces, thereby creating a better estimate for the 3D positions.

H.6. Timing Analysis of CSCD-M

We present in Tab. 6, the inference times for LS, MSLS and CSCD-M on a set of objects. All the inference-time experiments were run on a high-end laptop with a Intel Core i7 processor. In order to make a fair comparison, all methods were restricted

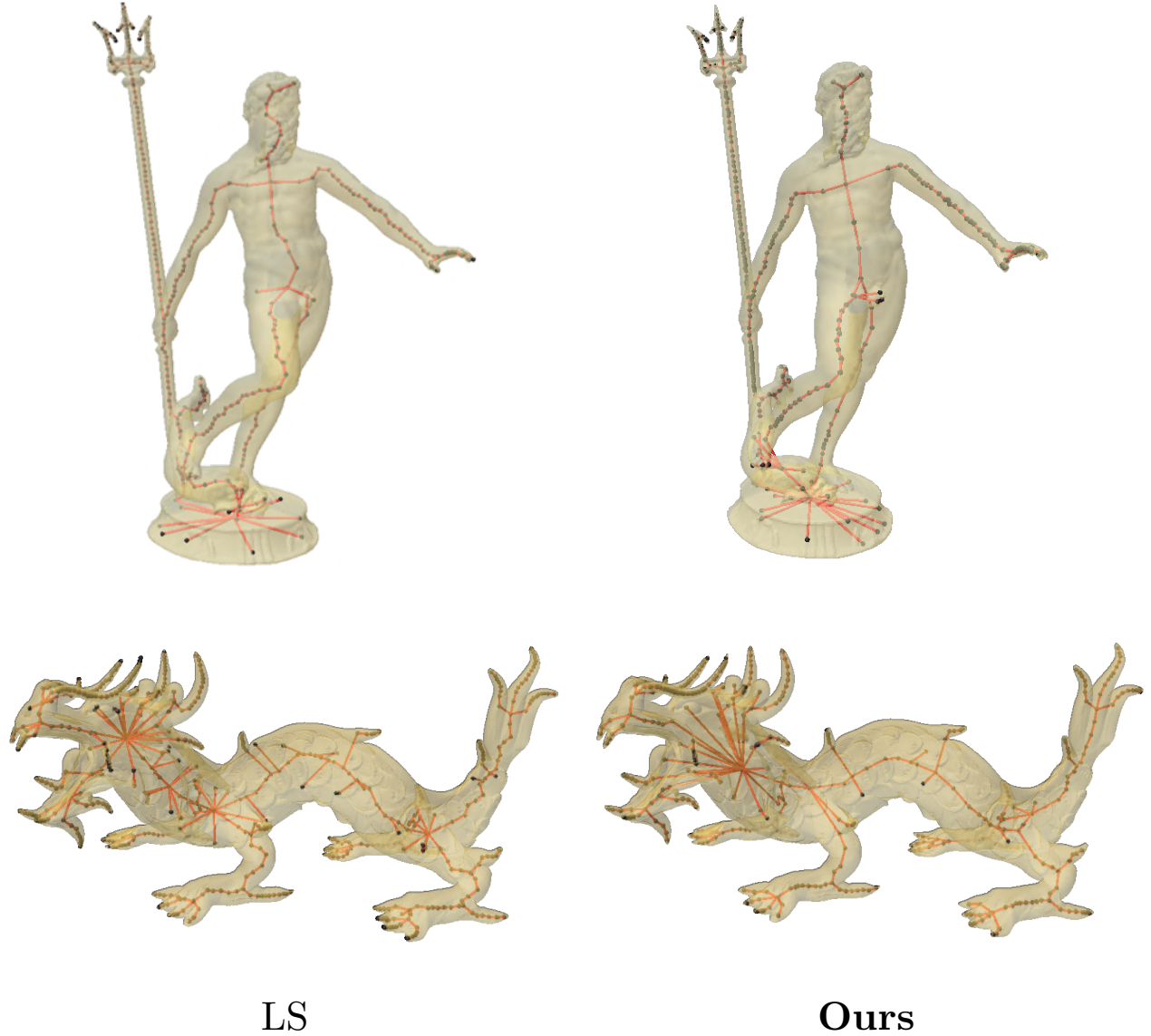


Figure 15. CSCD-M vs LS on complex shapes: Our method results in skeletons that are comparable to LS. Specifically on the `xyzrgb-dragon` object, our skeleton appears to be smoother and contains fewer noisy branches in the body of the dragon.

to use only a single thread of the CPU.

Why is this in the appendix?

There are a few reasons as to why we decided to put this analysis within the appendix:

1. LS, MSLS and our method use different heuristics for sampling and selecting local separators, so the comparison is not direct. One could argue that LS could, in principle, be sped up by reducing the number of local separators sampled, but that parameter is not available in their codebase; therefore we have to stick to the pre-defined heuristics chosen by the original authors.
2. Our method uses a fixed $N = 3000$, while the number of separators can vary for both LS and MSLS. We have chosen to implement this simple strategy, but we still observe comparable performance to LS (both qualitatively and quantitatively). We suspect this is due to the differences in the heuristics for sampling, where we use geodesic distance to sample farther away points. Our choice to fix N can go both ways – on one hand, we could increase N to gain better performance at the cost of inference times, while on the other, we are also overestimating N for simpler objects such as `fertility`, where $N = 1000$ also suffices. In some cases, however, we have taken $N = 4000$ on the account of the meshes being more

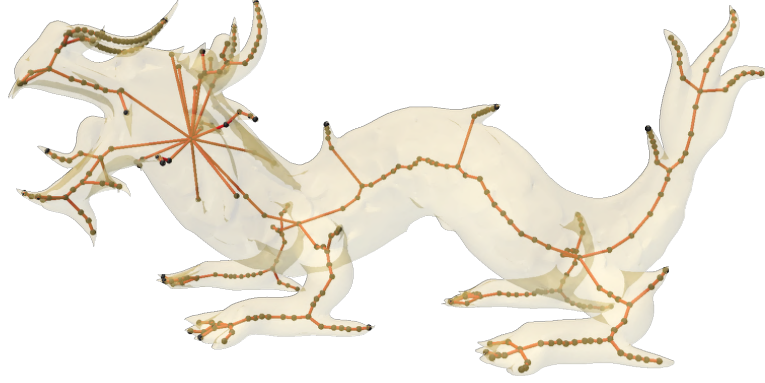


Figure 16. Smoothened xyzrgb-dragon for CSCD-M. We see that the spurious branches have reduced after smoothing the bod

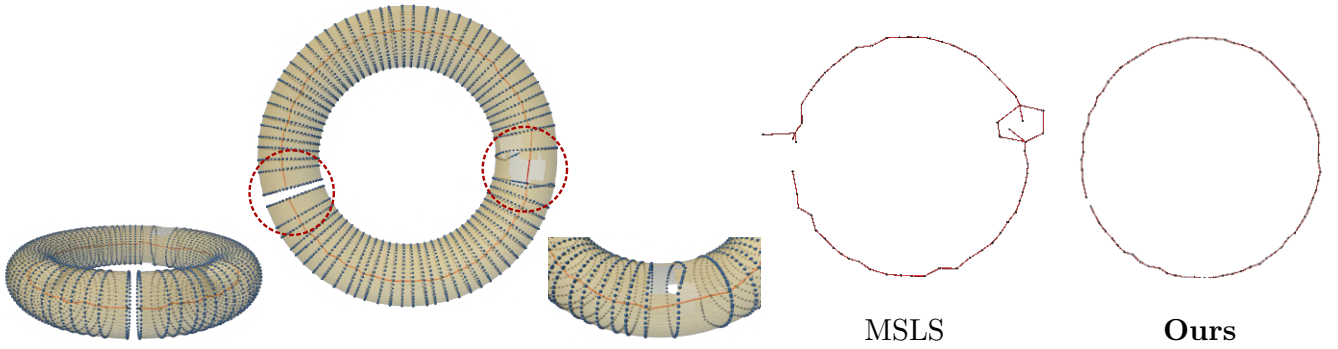


Figure 17. CSCD-M and MSLS on a torus with holes. Left panel, **CSCD-M local separators on the torus** – highlighting how the separators go around the holes. Right panel, **comparison of the curve skeleton** obtained by MSLS and Ours (CSCD-M).

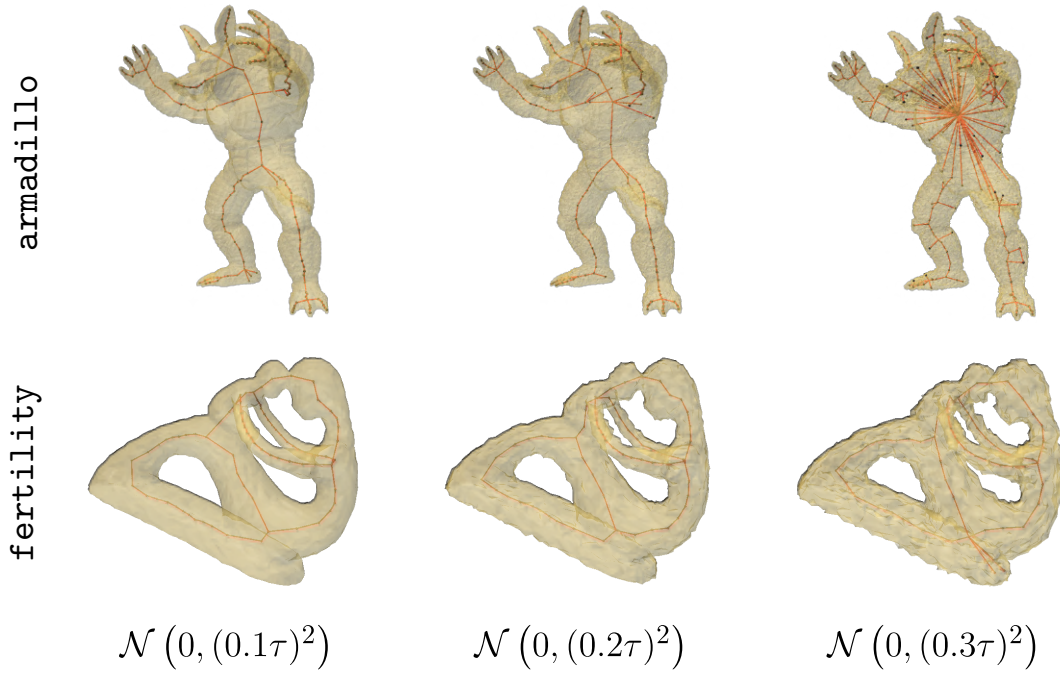


Figure 18. **Results of CSCD-M on noisy meshes armadillo and fertility:** Here we show the curve skeleton generated at progressively noisier vertex positions. $\mathcal{N}(0, (c\tau)^2)$ indicates the amount of gaussian noise added to each vertex's position, where τ is the average edge length of the mesh and c is a constant that controls the amount of noise.

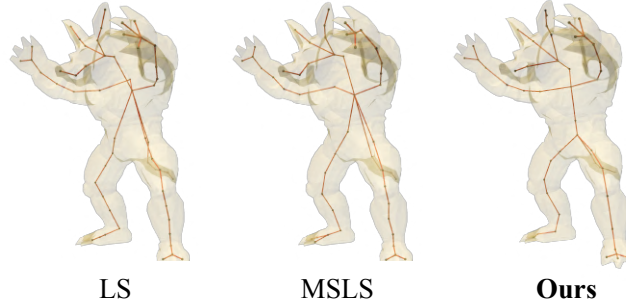


Figure 19. **Results on a low mesh resolution:** Both LS and MSLS lead to distorted skeletons at reduced mesh resolution while our method still obtains well-centered skeletons. Here the mesh resolution was reduced by $2\times$ the original mesh through mesh decimation.

Table 6. Runtime Analysis of CSCD-M (Ours), LS [2] and MSLS [3] (in seconds). Our results are calculated $N = 3000$, which on average provided similar results to LS. * However, three cases specifically use $N = 4000$ on the account of the mesh being more complex. We report both the average times and the median times since LS seems to have large outliers. [†] For LS, we additionally remove gorilla since it takes too long to compute.

Object	$ V $	$ F $	LS	MSLS	CSCD-M (Ours)
TID:44395	2948	5900	6.781	2.137	37.652
fertility	4494	9000	9.385	2.793	41.434
TID:37358	9780	19560	6.225	3.477	75.91
Copper_key	10000	20036	13.58	5.485	75.483
rocker-arm	10044	20088	141.032	6.538	75.764
TID:32770	20125	40246	1889.703	13.201	225.1745*
gorilla	48762	97520	$\geq 2400^\dagger$	32.458	532.163
armadillo	49990	99976	994.747	33.698	495.326
garuda-vishnu	49972	100084	292.425	30.133	609.304*
neptune	50000	100008	886.682	31.529	611.6585*
Average			471.1733	16.1449	277.9869
Median			141.032	9.8695	150.54225

complicated.

- As mentioned in the limitations, our realizations CSCD-M and CSCD-PC are simply meant to be a starting point. Our choices in the framework were made due to its simplicity and directness. There are many places (such as the cut locus identification procedure) where one can easily speed up the algorithm. We also believe that our particular codebase can be considerably improved and sped up through better coding practices – we had to add additional checks and balances due to bugs present in the packages used. One could fix those bugs, or implement them from scratch and yield better speed.

For the above reasons, we didn’t feel it was fair to call this a strong comparison. However, these, still, are inference times of the methods that yield comparable performance (all are within 3% of each other).

H.7. Additional Qualitative Results of CSCD-PC

In Fig. 20, we show additional qualitative results of CSCD-PC against another popular point cloud skeletonization technique ROSA [45]. Compared to ROSA, our method yields more detailed curve skeletons. Additionally, not shown here, CSCD-PC is faster than ROSA on these objects, but that could simply be an outcome of different programming languages used – Python vs MATLAB.

I. Downstream Applications

I.1. Shape Classification

We present the details and the results for the shape classification on the Princeton Shape Dataset in Tab. 7. We perform the shape classification by using the shape diameter function and comparison function that calculates the 1-D Wasserstein distance between two distributions. With a sufficiently high resolution for the bins of the distribution, 1D wasserstein distance would be robust to the boundaries of the bins.

Table 7. **Results on Shape Classification on PSD:** We calculate the Shape Diameter Function (SDF) from the obtained skeletons and use that to classify the a subset of classes from the Princeton Shape Dataset. We find that the SDF obtained from our method works the most reliably at classifying the objects.

Metric	LS	MSLS	CSCD-M
Accuracy	0.63	0.74	0.79
F1 Score	0.59	0.76	0.80

I.2. Shape Segmentation

We perform unsupervised shape segmentation of various objects in the Princeton Shape Dataset. We show that the segmentation remains consistent over different poses demonstrating the robustness of the skeletonization algorithm to pose changes.

I.3. Identifying Handles, Tunnels, and Constrictions

By removing the strong locality constraint during curve shortening, the optimized loop slides toward bottleneck regions. Sampling multiple source points allows us to detect global constrictions, while overlapping loops are pruned. Scoring is then based on loop length. Our framework also automatically identifies handles, pruned in the skeletonization task, tunnels and other constrictions (see Fig. 22).

J. Ablations

J.1. Choice of Geodesic Distance vs Euclidean Distance

We find that while the choice of the geodesic distance results in un-intuitive local separators, the final curve skeletons seem to resemble the one obtained from the Euclidean distance constraints. This could be simply due to the large number of local separators calculated – eventually leading to similar local separators being calculated for both constraints. However there exist many cases (such as one in Fig. 23) where the geodesic based method may fail completely but the Euclidean would work.

In Fig. 23, we illustrate an example where the two methods perform similarly for a wide range of objects. However in the case of *gorilla*, the Geodesic constraint fails to give a good curve skeleton with missing features and off-axis nodes. This is most likely due to the large abdomen width of the object due to which tracing a path around it is generally longer then going around the nearest arm or leg. However with Euclidean constraints, we are not limited to the surface of the manifold and we can choose the cut loci at the back of the abdomen. See the inset from Sec. 3.1.3 for a visual example of this phenomenon.

J.2. Number of Local Separators calculated in Stage 1

Generally speaking, the number of local separators required is proportional to the complexity of the object in terms of the geometrical features of the mesh. Simpler tubular-like structures like *fertility* require significantly fewer separators compared to objects like *armadillo*. In Fig. 24 we plot the number of local separators calculated vs the reconstruction error of the object.

K. Discussions

We present below a short discussion on some topics that are related to the paper. These discuss some open directions for future work.

K.1. On the derivation of LS from CSCD

While we would not exactly obtain LS, we would derive a graph version of CSCD that effectively replicates the results from LS.

1. For the geodesic distance, we use a simple graph based distance.
2. The target cut locus is then chosen similar to the current method, i.e., based on the minimum Euclidean distance.
3. To optimize the curve, we can follow an iterative unfolding scheme, where the path between two vertices is iteratively shortened using Dijkstra’s shortest path algorithm.
4. With the optimized local separators, overlap can simply be checked by determining if two local separators share a vertex.
5. Next, we assign the nearest vertices to each local separator, thereby creating the quotient graph.
6. Finally, based on the quotient graph, the curve skeleton is constructed and post-processed.

K.2. On the multiscale version of CSCD-M

CSCD-M has immense potential for a multi-scale approach. This is because we have the ability to sample points on the face of a low-poly mesh, and operate on these face surface points through barycentric interpolations. In this way, we could, in principle, sample as densely or sparsely as desired – effectively independent of the polygon count of the mesh. However, the quality of the mesh would influence the accuracy of the quantities being calculated; for example, low-poly meshes could hinder the processes like geodesic distance calculation or edge flip operations. In these cases, one could explore using higher-order corrections to account for these issues. Therefore, unlike other multi-scale approaches, we could get away with directly working upon reduced poly-meshes and introducing higher order corrections, which could make it extremely quick.

L. Limitations and Future Work

Our work has a few limitations:

1. CSCD requires a defined metric (for the calculation of geodesic distances) on the representation, which may hinder adaptation to certain formats such as NeRFs. However, most common representations — meshes, point clouds [38], and digital surfaces [7] — provide such metrics.
2. Our realizations are intended as starting points, with primary results on meshes and limited tests on point clouds. Future work could enhance individual modules for greater speed, robustness, and performance, and extend the framework to other representations.

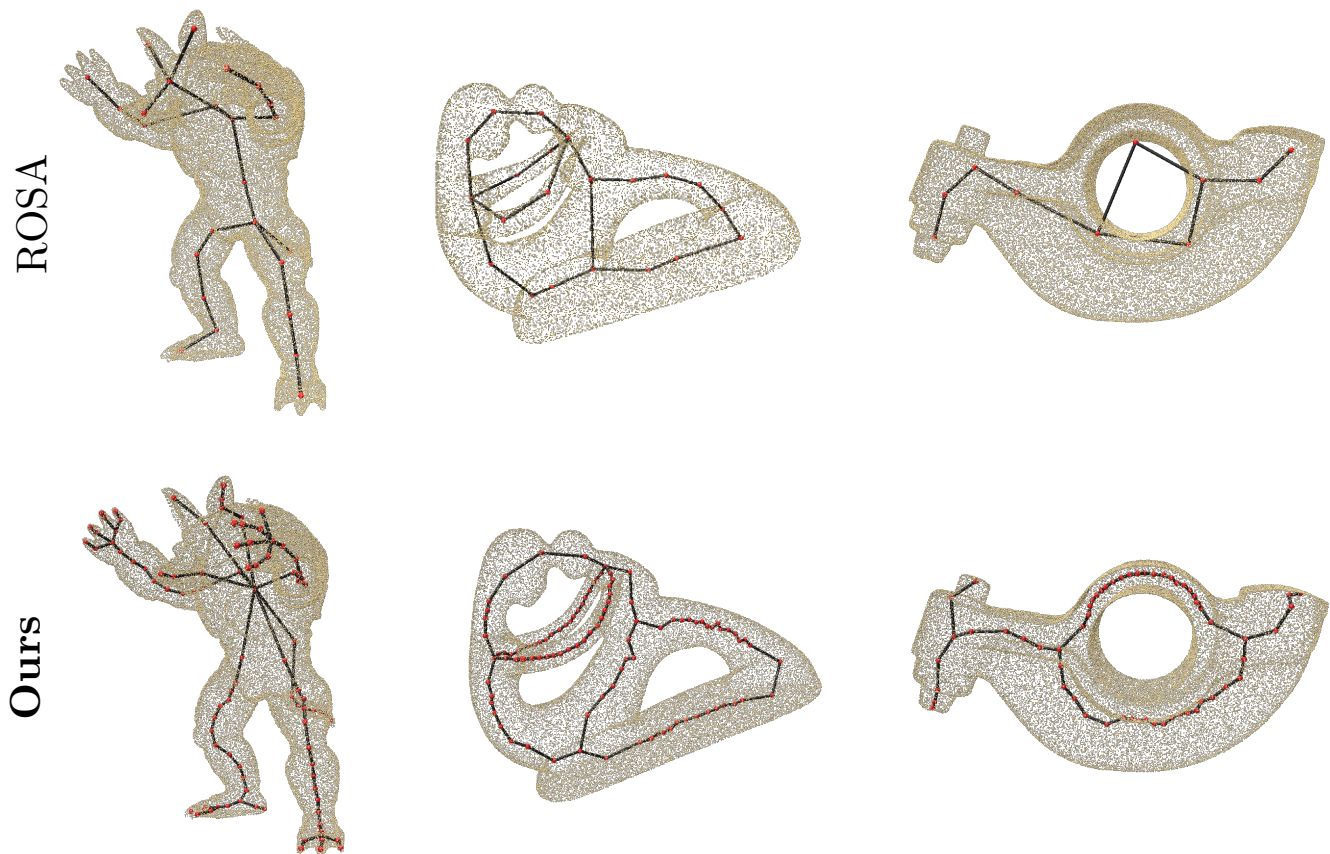


Figure 20. **CSCD-PC vs ROSA:** Comparative results of our method vs ROSA. We are able to capture the details of the object while ROSA [45] struggles to do so. Specifically, we see that ROSA results in fewer nodes and coarser skeletons compared to CSCD-PC.

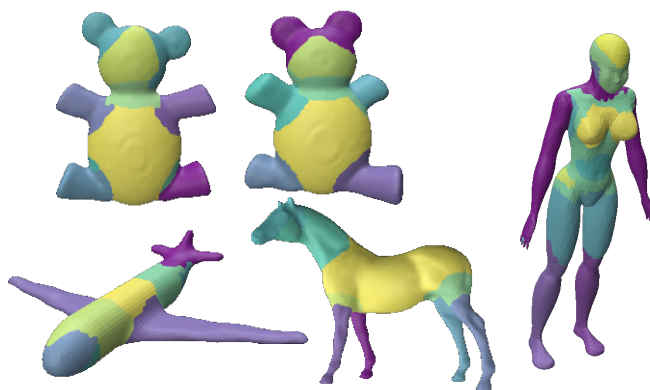


Figure 21. **[Downstream Application: Shape Segmentation]:** Our method works as a strong skeletonization technique for the shape diameter function (SDF) based unsupervised segmentation. The obtained segmentation is robust to pose variations and works very well.

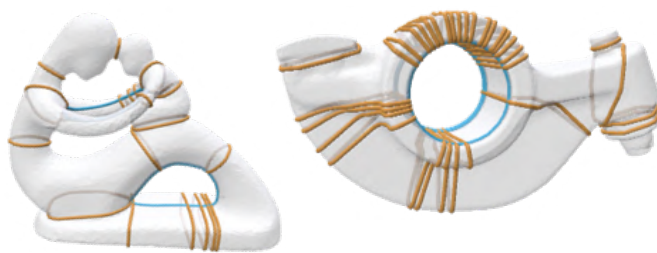
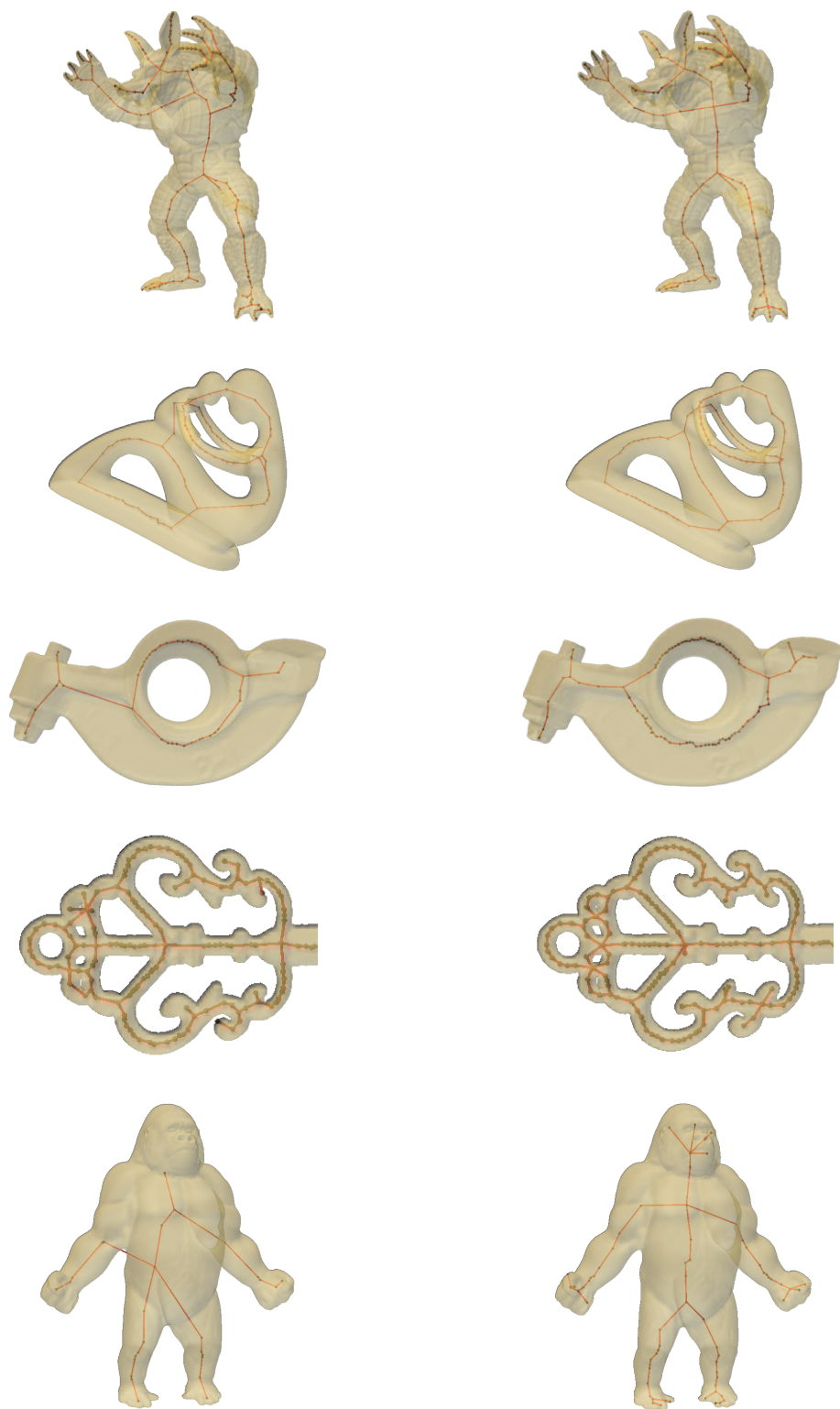


Figure 22. **[Downstream Application: Handle/Tunnel/Constricting Loop Detection]:** With minor changes, our local separator identification algorithm works effectively to identify handles/tunnels and constricting loops on the shape. These loops form useful basis for other tasks such as surface cutting, etc.



Geodesic Distance

Euclidean Distance

Figure 23. **Geodesic Constraint vs Euclid Constraint** for calculating the local separators: We see that for many objects the resultant skeleton looks pretty acceptable. However, for certain meshes like `gorilla`, the geodesic constraint would lead to poor results.

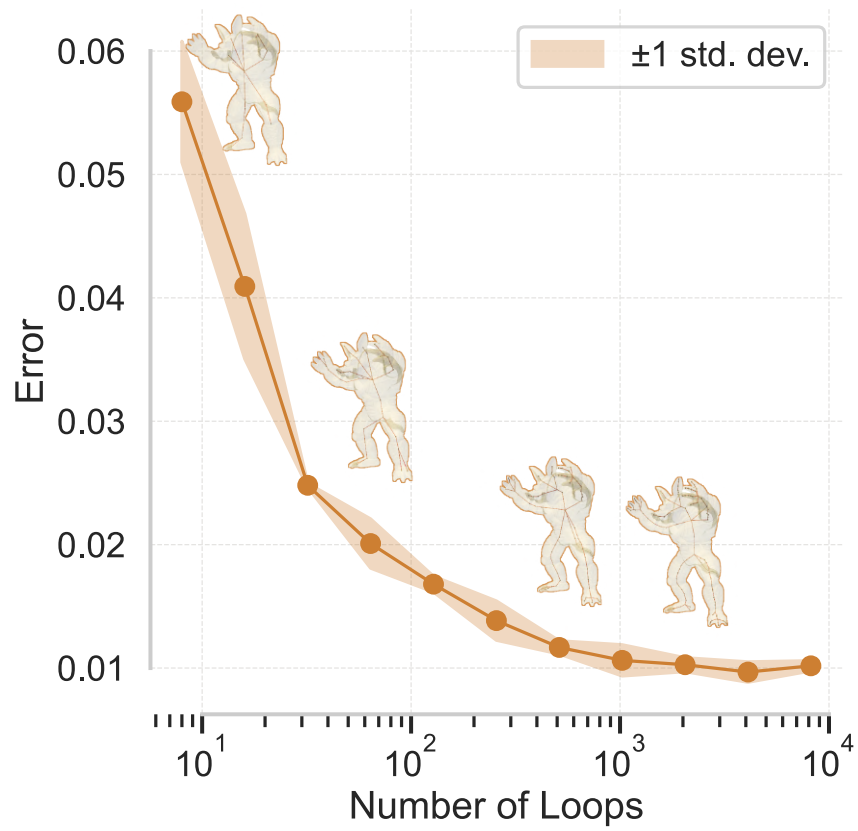


Figure 24. **Number of local separators vs reconstruction Error for armadillo**: Increasing the number of local separators reduces the error drastically, but the decrease in error varies based on the complexity of the object. The reconstruction errors are evaluated for 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096 and 8192 number of local separators.

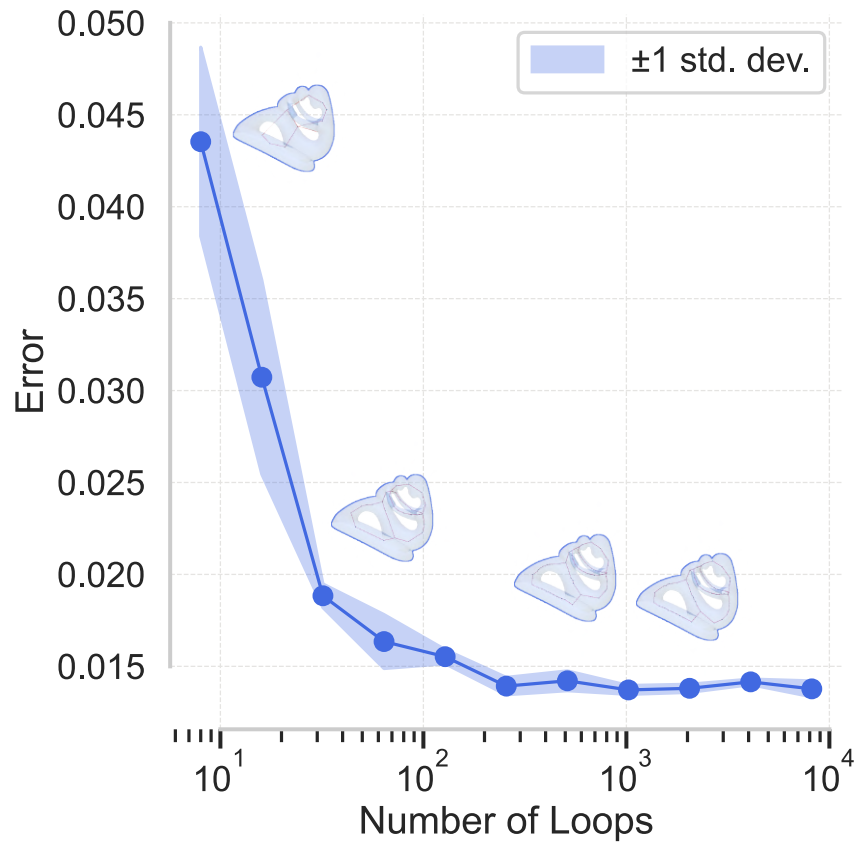


Figure 25. **Number of local separators vs reconstruction Error for fertility**: Increasing the number of local separators reduces the error drastically, but the decrease in error varies based on the complexity of the object. The reconstruction errors are evaluated for 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096 and 8192 number of local separators.

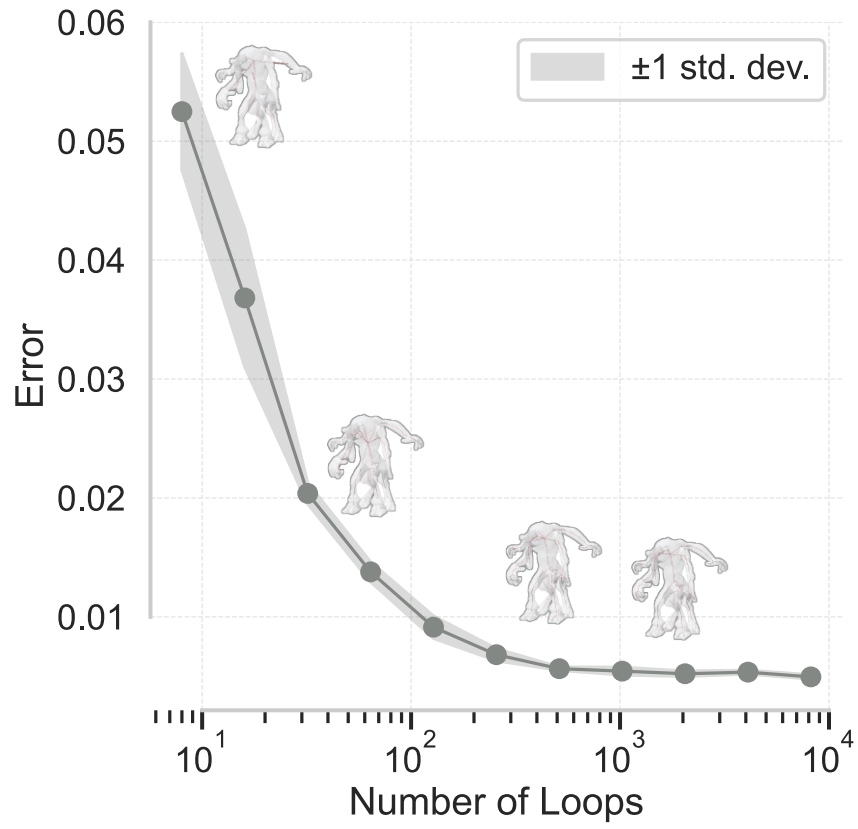


Figure 26. **Number of local separators vs reconstruction Error for TID: 133568**: Increasing the number of local separators reduces the error drastically, but the decrease in error varies based on the complexity of the object. The reconstruction errors are evaluated for 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096 and 8192 number of local separators.

M. Terminology and Definitions

Here we define some of the technical terms used in the paper:

Cut locus / Cut loci Given a compact Riemannian manifold (M, g) and a point $p \in M$, the *cut locus* of p , denoted $\text{Cut}(p)$, is the closure of the set of endpoints of geodesics emanating from p that cease to be globally minimizing beyond those points. Equivalently, a point $q \in M$ lies in $\text{Cut}(p)$ if there exist two or more distinct minimizing geodesics from p to q . The plural *cut loci* refers to such sets for one or more base points.

Geodesic A smooth curve $\gamma: I \rightarrow M$ on a Riemannian manifold (M, g) is called a *geodesic* if it locally extremizes the arc-length functional. Equivalently, it satisfies the geodesic equation

$$\nabla_{\dot{\gamma}} \dot{\gamma} = 0,$$

where ∇ is the Levi-Civita connection associated with g .

Curve shortening The process of evolving a smooth embedded curve $\gamma(s) \subset M$ under the *curve-shortening flow*

$$\frac{\partial \gamma}{\partial t} = k \mathbf{n},$$

where k is the (scalar) curvature of the curve and \mathbf{n} its inward unit normal. This flow decreases the total length of γ as quickly as possible at each instant.

Intrinsic triangulation vs. extrinsic triangulation An *extrinsic triangulation* of a surface $S \subset R^3$ is given by the mesh connectivity and the 3D positions of its vertices, with triangle geometry inherited from the embedding in R^3 . An *intrinsic triangulation*, by contrast, is specified solely by edge-length assignments satisfying the triangle inequalities, encoding the surface’s intrinsic metric independently of any embedding.

Integer coordinate system An *integer coordinate system* on a mesh is an assignment of integer-valued coordinates (e.g. in Z^2 or Z^3) to each vertex, in such a way that all edge-lengths and combinatorial relationships can be computed exactly using integer arithmetic. This avoids floating-point error in geometric algorithms.

Laplacian / Laplace–Beltrami operator On a smooth Riemannian manifold (M, g) , the *Laplace–Beltrami operator* Δ acting on a scalar function f is

$$\Delta f = \div(\nabla f),$$

where ∇ and \div are the gradient and divergence induced by g . On a triangle mesh, the *cotangent Laplacian* at vertex i is discretely given by

$$(\Delta f)_i = \frac{1}{2A_i} \sum_{j \in N(i)} (\cot \alpha_{ij} + \cot \beta_{ij}) (f_j - f_i),$$

where A_i is a local area weight (e.g. the Voronoi area) and α_{ij}, β_{ij} are the angles opposite the edge (i, j) .

Interior Point Method A class of algorithms for solving constrained optimization problems of the form

$$\min_x f(x) \quad \text{s.t.} \quad g_i(x) \leq 0, \quad h_j(x) = 0,$$

by introducing a barrier term (e.g. $-\mu \sum_i \log(-g_i(x))$) for each inequality and solving a sequence of unconstrained problems as the barrier parameter $\mu \rightarrow 0^+$. At each iteration, the algorithm stays strictly in the interior of the feasible region and follows a “central path” to the optimum.

N. Related Works

Our work builds upon several fundamental techniques in 3D shape analysis, which we outline in detail below.

Geodesic distance computation on meshes/point clouds: Critical to our work are methods for computing geodesic distances on 3D shapes. Geodesic distance computation on meshes has been extensively studied [10]. Of interest to us are PDE-based methods, which assume that the discrete representation is based on a continuous underlying manifold. Heat method [9], based on the diffusion process, connects Varadhan’s formula [50] to geodesic distances – thereby reducing the computation to solving a series of linear equations. [9] leads to faster computation with only a small hit to the accuracy.

Curve Shortening on 3D shapes: Also critical to our framework are methods for shortening curves on 3D shapes. These methods are often analogous to geodesic computation on surfaces [26, 31, 39, 55, 56, 58, 60].

Curve Skeletonization for Meshes: The concept of curve skeletons for 3D objects is not well-defined; this lack of a formal definition has led to the multitude of hand-crafted methods [36, 47]. Most of these methods rely on the idea that for tubular shapes, there exists a 1D structure that preserves the shape of the topology. These methods are based on the geometrical features of the objects, local decimation of objects, or regional division of objects. Tierny et al. propose an unified method for constructing and simplifying Reeb graphs on 3D meshes using discrete contours to extract affine-invariant, visually meaningful topological skeletons, however, it is not clear how the method performs on pointclouds and other geometric domains [49]. Other representative works include [1, 5, 8, 11, 27, 33, 35, 46]. CSCD-M is inspired by the LS method[2]. MSLS [3] is the multi-scale variant of LS that allows it to sample more loops efficiently. In principle, constructing a multi-scale variant of CSCD-M should be easy as we can simply work on faces of coarse meshes, but we leave this for a future exploration.

Curve Skeletonization for Point Clouds: Curve skeletonization on point clouds has gained attention recently with increased data availability. Methods like [18] use medians over centroids, while [45] leverages approximate rotational symmetry. However, these approaches often yield lower fidelity and miss fine details. Learning-based methods [25, 57] struggle to generalize to new meshes. Other recent point cloud methods include EPCS [23] and CA++ [12, 53].

While we have not explicitly evaluated our method on volumetric data, voxel representations can be converted to surface meshes as a preprocessing step prior to skeletonization.[4, 28, 54]