arXiv:2012.01203v2 [cs.CV] 6 May 2021

Learning Delaunay Surface Elements for Mesh Reconstruction

Marie-Julie Rakotosaona
LIX, Ecole Polytechnique, IP Paris
mrakotos@lix.polytechnique.fr

Paul Guerrero
Adobe Research
guerrero@adobe.com

Noam Aigerman
Adobe Research
aigerman@adobe.com

Niloy Mitra
UCL, Adobe Research
n.mitra@ucl.ac.uk

Maks Ovsjanikov
LIX, Ecole Polytechnique, IP Paris
maks@lix.polytechnique.fr

Abstract

We present a method for reconstructing triangle meshes from point clouds. Existing learning-based methods for mesh reconstruction mostly generate triangles individually, making it hard to create manifold meshes. We leverage the properties of 2D Delaunay triangulations to construct a mesh from manifold surface elements. Our method first estimates local geodesic neighborhoods around each point. We then perform a 2D projection of these neighborhoods using a learned logarithmic map. A Delaunay triangulation in this 2D domain is guaranteed to produce a manifold patch, which we call a Delaunay surface element. We synchronize the local 2D projections of neighboring elements to maximize the manifoldness of the reconstructed mesh. Our results show that we achieve better overall manifoldness of our reconstructed meshes than current methods to reconstruct meshes with arbitrary topology. Our code, data and pretrained models can be found online: https://github.com/mrakotosaon/dse-meshing

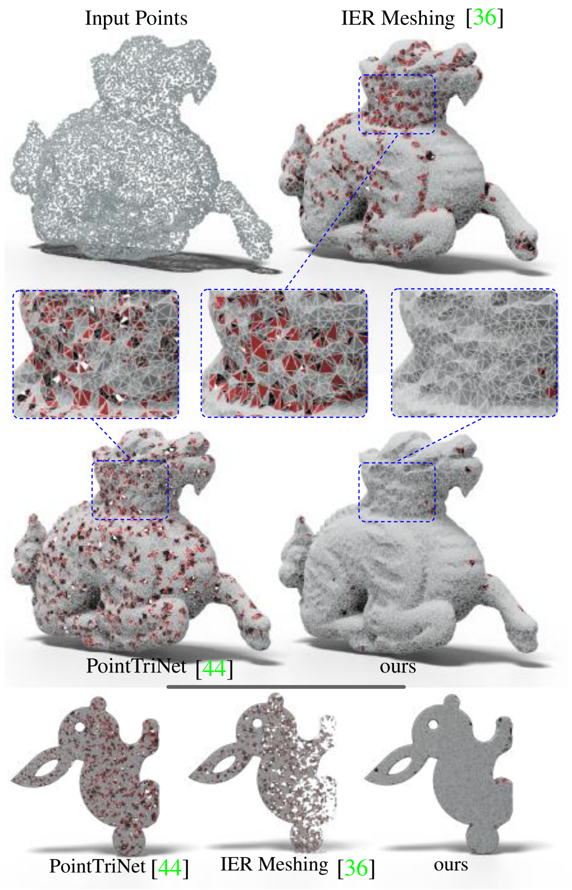

Figure 1. We present a method for mesh reconstruction from point clouds. We combine Delaunay triangulations with learned local parameterizations to obtain a higher-quality mesh than the current state-of-the-art. Bad (non-manifold) triangles are shown in red. Our method is robust to uniformly (top) and non-uniformly (bottom) sampled points.

1. Introduction

Surface reconstruction from a given set of points (e.g., a scan), has a long history in computational geometry and computer vision [5, 40]. A version of the problem requires triangulating a given point cloud to produce a watertight and manifold surface. A key challenge is to handle different sampling conditions while producing well-shaped triangles and preserving the underlying shape features.

A good surface reconstruction algorithm should satisfy the following requirements: (i) produce a connected, manifold and watertight triangulation; (ii) require no case-specific parameter tuning; (iii) preserve sharp features; (iv) handle point sets with non-uniform distribution; and

(v) generalize to handle a variety of shapes.

A widely-used pipeline for surface reconstruction consists in first computing an implicit surface representation [31] and then extracting a triangulation using a volumetric method such as Marching Cubes [37]. Methods in this category often require additional information (e.g., oriented normals), while, crucially, the resulting triangulations may not preserve the original point set and can oversmooth sharp features. On the other hand, methods from computational geometry, e.g., alpha shapes [20], ball pivoting [6], etc., can respect the original point set, come with theoretical guarantees and produce triangulations with desirable properties (e.g., good angle distribution). These approaches, however, typically require careful parameter selection and rely on dense, uniformly sampled point sets.

More recently, learning-based approaches have been developed to extract a triangulation without case-specific parameter selection. Most of such techniques focus on robustly predicting a signed distance field or simply an occupancy map, from which a mesh is subsequently extracted using volumetric triangulation [12, 23, 42]. Only two recent methods [44, 36] produce a triangulation while respecting the original point set, but they ignore the quality of the triangles or have trouble reconstructing sharp features.

We present a method that combines the advantages of classical methods with learning-based data priors. Our method is based on blending together *Delaunay surface elements*, which are defined by a 2D Delaunay triangulation of a local neighborhood in the point set after projecting it to a planar 2D domain. For this, we propose an approach that predicts a local projection via learned logarithmic maps and uses them to propose likely triangles using local Delaunay triangulations. Figure 1 shows an example reconstructions using our method. We evaluate our method on a benchmark of diverse surface point sets, and provide a comparison with both classical and learning-based methods to show the advantages of the proposed approach. Through these extensive experiments, we demonstrate that our method generalizes across diverse test sets, is more robust than classical approaches, and produces higher-quality triangulations than recent learning-based methods.

2. Related Works

Computing a triangulation of a given point set is one of the most fundamental problems in computational geometry, computer vision, and related disciplines. We review methods most closely related to ours and refer to recent surveys [32, 46, 5, 40] for a more in-depth discussion.

A commonly-used pipeline for surface reconstruction [29, 13] consists of computing the implicit surface representation using, e.g., a signed distance function. A mesh can then be extracted with standard methods such as Poisson surface reconstruction [31] combined with Marching Cubes [37] or Dual Contouring [30]. Such approaches work well in the presence of oriented normals and dense/uniform point sets, but do not necessarily preserve the given points in the final mesh and lead to over-smoothing or loss of details (see [5] for a detailed discussion).

We were inspired by classical methods based on Delaunay triangulations [8, 34, 9, 25, 17], alpha shapes [20] or ball pivoting [6]. Such approaches can be shown to recover the shape mesh topology [2] under certain sampling conditions (an excellent overview of such approaches is provided in [16]). Unlike implicit-based methods, approaches in this category, e.g., [6, 1, 10] typically preserve the input point set. However, they can often fail to produce satisfactory results for coarsely sampled shapes or in the presence of complex geometric features. Another more robust, but computationally more expensive, approach capable of feature preservation was introduced in [18], based on iterative optimisation using optimal transport.

2.1. Learning for surface reconstruction

To address the challenges mentioned above, recent methods have aimed to learn surface reconstruction priors from data. The majority of existing learning-based methods in this area use a volumetric shape representation. For example, meshes can be computed by predicting voxel grid occupancy [24, 38] or via a differentiable variant of the marching cubes [35], or more recently using generative models for explicit or implicit surface prediction [12, 23, 42, 39]. While these methods can produce accurate results they solve a different problem to ours and do not compute a mesh over the given point set. Instead, we focus on directly meshing a set of input points, which provides better control over the final shape and avoid over-smoothing, often associated with implicit surface-based techniques.

Other methods have also aimed to compute a surface by deforming a simple template while updating its connectivity [47, 41], fitting parameterized [26, 48] or mesh-aware patches [3], performing local (e.g., convex) shape decomposition. Majority of these schemes are restricted to particular shape topology or category and again do not necessarily guarantee point set preservation.

2.2. Learning mesh connectivity

More directly, our work fits within the line of recent efforts aimed explicitly at learning the mesh connectivity for a given shape geometry. An early approach, Scan2Mesh [14] developed a graph-based formulation to generate triangles in a mesh. However, the method uses a costly volumetric representation, does not aim to produce manifold meshes, and specializes on particular shape categories.

Most closely related to ours are two very recent approaches aimed directly to address the point set triangulation problem. The first method PointTriNet [44] works

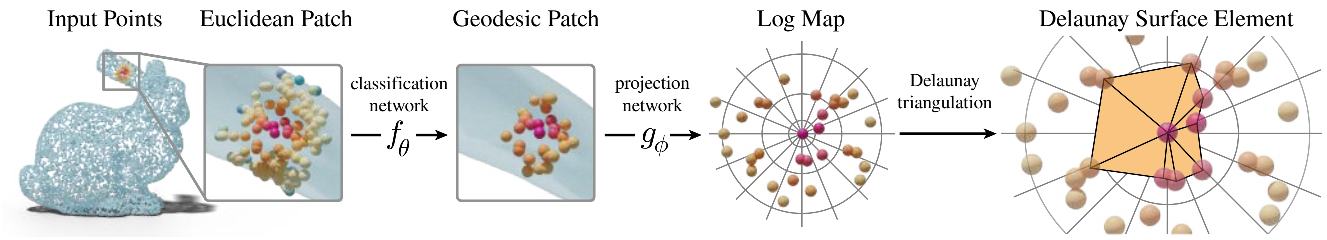

Figure 2. Overview of Delaunay Surface Element (DSE) generation. For any point p_i in an input point cloud, we select the k-nearest neighbors and extract the subset of points that are in the geodesic neighborhood of p_i, using a learned classification network. A projection network then estimates a log map projection of the points into a 2D embedding, where we can apply Delaunay Triangulation to get a DSE.

on point clouds and, similarly to ours, uses a local patch-based network for predicting connectivity. However, this technique processes triangles independently and only promotes watertight and manifold structure through soft penalties. The second method was presented in [36], and estimates local connectivity by predicting the ratio between geodesic and Euclidean distances. This is a powerful signal, which is then fed into a non-learning based selection procedure, which aims to finally output a coherent mesh.

In contrast to both of these approaches [44, 36], we formulate the meshing problem as learning of (local) Delaunay triangulations. Starting from the restricted Voronoi diagram based formulation proposed in [10] we use data-driven priors to directly learn local projections to create local Delaunay patches. As a result, locally our network *guarantees* the coherence of the computed mesh. As we demonstrate below, learning Delaunay surface elements, both leads to better shaped triangles (i.e., more desirable angle distribution) and improves the overall manifold and watertight nature of the computed triangle mesh.

3. Method

We assume to be given an point set $P \in \mathbb{R}^{N\times 3}$ sampled from a surface $\hat{S}$. Our goal is to create a mesh $M = (P', T)$ that approximates $\hat{S}$, by choosing a new triangulation T that triangulates a subset $P' \subset P$ of the input point cloud. It is easy to obtain a high-quality triangulation for any set of points that lies in *2D*, via Delaunay triangulation [15]. However, when the set of points lies in 3D, finding a triangulation is a much harder problem. A simple solution is to locally project points to an estimated tangent plane of the surface, resulting in local 2D embeddings where we can apply a Delaunay triangulation. However, this is problematic near complex geometry, such as edges or thin structures and is sensitive to an imperfect estimation of the tangent plane. *Logarithmic maps* [19, 28], or *log maps* for short, provide a systematic solution to this problem by providing local geodesic charts of the ground truth surface that are good local parameterizations of complex geometry.

The core idea of our method is therefore to combine Delaunay triangulations and learned log maps to create small triangulated patches that we call *Delaunay Surface Elements* (DSEs). Each DSE approximates a small part of the surface and is guaranteed to have a manifold triangulation. Since neighboring log maps may disagree, especially in regions of high curvature, we align them locally with non-rigid transformations of the 2D parameterizations of each DSE. DSEs enable us to maintain the good properties of Delaunay Triangulations, like manifoldness and high-quality triangles, within a data-driven approach, that learns to extract local geodesic patches and parameterize them with a the log map, thereby increasing robustness and reconstruction accuracy. Our approach proceeds in four steps (the first two steps are illustrated in Figure 2):

1. For each point $p_i \in P$, a network estimates a geodesic ball, by extracting a 3D patch $P^i \in \mathbb{R}^{k\times 3}$ made up of its k-geodesically-closest points.
2. For each 3D point patch P^i, a second network approximates the log map parameterization, to get a 2D embedding of the patch, denoted $U^i \in \mathbb{R}^{k\times 2}$.
3. We improve the consistency of neighboring patches by aligning their 2D embeddings, giving us improved patch embeddings $\hat{U}^i$, which we then use to compute the Delaunay Surface Elements.
4. The Delaunay Surface Elements *vote* for candidate triangles, which are then aggregated iteratively into a mesh.

3.1. Constructing Local Embeddings

The first two steps in our method are aimed at creating a patch P^i around each point p_i and a local 2D embedding U_i of the points inside the patch. These two ingredients will later be used to compute a 2D Delaunay triangulation that defines a Delaunay Surface Element.

Geodesic patch construction Given the point p_i and its K nearest neighbors Q^i, we train a network to find a subset of k points from these neighbors that are *geodesically* closest to p_i on the ground truth surface. In our experiments, we set $K = 120$ and $k = 30$. More details on the choice of k and K are provided in Section H of the supplementary. The network is trained to model a function $c_j := f_\theta([q^i_j, d^i_j] \mid Q^i)$ that classifies each point q^i_j in Q^i as being one of the k geodesically closest points if $c_j = 1$ or not if $c_j = 0$. We concatenate the Euclidean distance d^i_j to the center point as additional input. The network is parameterized by θ, conditioned on the point set Q^i, and models a function from 3D position to classification value. We train this network with an L2 loss $\|c_j - \sigma(\hat{c}_j)\|^2$, where $\hat{c}_j$ is the ground classification and σ is the sigmoid function. To obtain a fixed number of k points, we select the top-k points based on their predicted labels c^i_j, giving the (geodesic) patch P^i.

Log map estimation We train a second network to compute the log map coordinates of each point in P^i, denoted as $U^i \in \mathbb{R}^{k\times 2}$. The network is trained to model a function $u^i_j := g_\phi([p^i_j, d^i_j] \mid P^i)$, where ϕ denotes the network parameters and p_j are the 3D coordinates of a point in P^i. These coordinates are concatenated with the Euclidean distance d^i_j to the center point. The network outputs the log map coordinates u^i_j in U^i, consisting of the Euclidean coordinates of the log map with an origin at the center point of the patch. Like the classification network, this network is conditioned on the input point set P^i. We use the sum of two losses: a loss that penalizes the difference to the ground truth coordinates and one that penalizes only the radial component of the coordinates, i.e. the geodesic distance to the center. Since log map coordinates are defined only up to a 2D rotation around the central point, we use the Kabsh algorithm [7] to find a 2D rotation and/or reflection that optimally aligns the predicted log map and the ground truth log map before computing the loss: $\|RU^i - \hat{U}^i\|^2_2$, where $\hat{U}^i$ is the ground truth and R is the optimal rigid transformation computed by the Kabsh algorithm. Note that the Kabsh algorithm is differentiable. Our second loss measures the error in the radial component: $\sum_j (\|u^i_j\|_2 - \|\hat{u}^i_j\|_2)^2$. This loss measures how well the network can recover geodesic distances regardless of the orientation in the patch.

Network architecture When approximating log maps with a network, continuity is an important property. If the estimated mapping from 3D space to the 2D log map parameterization is not continuous, the resulting Delaunay triangulation may have flipped or intersecting triangles. We base our architecture on FoldingNet [49] that produces continuous mappings from an input to an output domain. Unlike the original implementation, however, which maps from 2D to 3D, we want to map from 3D to 2D. Our experiments have shown that this network architecture leads to more continuous results than a PointNet-based architecture. We have also found that it improves the performance of our classification network, where we also adopt an architecture based on FoldingNet. Since we train our network on individual patches, we can train on relatively small datasets, where each shape provides a large number of patches as training samples. More details on the architecture are provided in Section E of the supplementary.

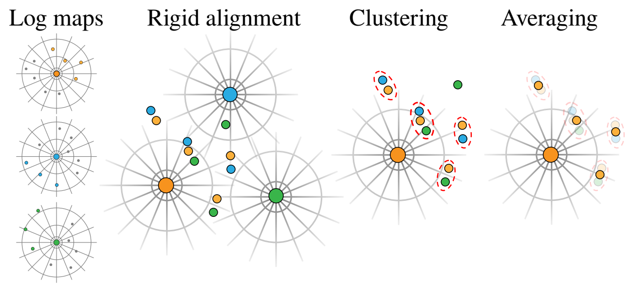

Figure 3. Log map alignment. To improve the consistency of log maps, we align corresponding points in neighboring log maps with rigid transformations. The resulting sets of corresponding points are then clustered to remove outliers and averaged, giving us 2D point embeddings that are more consistent with their neighbors.

3.2. Combining Delaunay Surface Elements

At this point, we have a local 2D parameterization for each patch. We could use these local parameterizations to construct a triangulation of the patch by Delaunay-triangulating it. However, each patch may be rather inconsistent with neighboring patches, in the sense that if two patches P^i, P^j share three points a, b, c, the Delaunay triangulation of U^i may produce the triangle (a, b, c) while the triangulation of U^j may not, since the points are laid out differently in each of the two parameterization. An example is shown in Figure 4, right. Hence, the final pair of steps is aimed at improving the consistency between the different patch parameterizations of neighboring DSEs before combining all DSEs into the final mesh M.

Log map alignment In 2D, Delaunay triangulation are guaranteed to produce a manifold triangulation. However, we produce independent 2D parametrizations for each DSE. Large differences in the parameterization of neighboring DSEs may make their triangulations incompatible (i.e., the union of their triangles may be non-manifold). In this step, we locally align the log maps to one another to ensure better consistency, without requiring the construction of a global parameterization. Namely, a point $p_k \in P$ from the original point cloud has an image in the log maps of each patch that contains that point. We denote this set of all log map images of point p_k as R^k. We say U^i, U^j are neighbor patches if they both have a point in the same R^k. Denote the image of p_k in the log map of each of the two patches as $U^i(p_k)$, $U^j(p_k)$, respectively.

Our approach is illustrated in Figure 3. Considering the

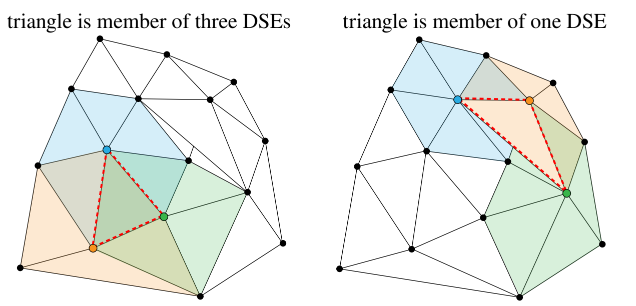
triangle is member of three DSEs
triangle is member of one DSE

Figure 4. Triangle membership count. Delaunay Surface Elements are shown as colored triangles. Triangles that are part of exactly three DSEs, like the dotted red triangle on the left, result in a manifold triangulation. Triangles that are part of less than three DSEs, like the triangle on the right, result in non-manifold triangulations. We use this property to define a triangle confidence when selecting triangles.

patch U^i, we align the neighboring patch U^j to it, by taking all corresponding points and using the Kabsch algorithm to find the rigid motion that best aligns (in the least-squares sense) the points based on their correspondences $U^i(p_k) \leftrightarrow U^j(p_k)$. Repeating this for all neighboring patches aligns them all to U^i. We then define the set R^i_k to be the set of images of the point p_k in the aligned log maps and cluster R^i_k with DBSCAN [21]. The largest cluster corresponds to the largest agreement between neighboring patches on the 2D coordinates u^i_k of point p_k in patch i. We average all 2D coordinates in the cluster to update u^i_k, and weigh the average based on the distance of each point in R^i_k to the center of its patch. Applying this process to all 2D coordinates U^i in each patch, we get a corrected log map $\hat{U}^i$ for each patch, giving us DSEs that are more consistent with the neighboring DSEs.

Delaunay triangulation Given a patch P^i and its 2D parameterization $\hat{U}^i$, we can compute a Delaunay Triangulation on the 2D points u^i_j. If $\hat{U}^i$ approximates the log map, this gives us a manifold triangulation of the 3D patch that locally approximates the ground truth surface $\hat{S}$. We define a Delaunay Surface Element $D := (P^i, T^i)$ as the set of Delaunay triangles T^i corresponding to the Voronoi cell centered at p_i. These triangles form an umbrella with p_i as its central point. We restrict our triangulation to triangles that include the central point, as triangulations are increasingly inconsistent with neighboring DSEs as the distance from the central point increases.

Triangle selection Combining the triangles of all DSEs yields a set of candidate triangles that we use in a final triangle selection step to obtain a near-manifold mesh. We base our selection criteria on our DSEs by observing that a triangulation is manifold exactly if all triangles are part of three DSEs (see Figure 4). Therefore, we divide our triangles into three confidence bins. Triangles that appear in three different DSEs will be considered the most likely to appear in a manifold triangulation. And triangles that appear only once are considered least likely. Finally, we use the triangle selection process proposed in [36] to produce a triangulation based on our priority queue.

Table 1. Quantitative results on the FAMOUSTHINGI testset. We compare the percentage of non-watertight edges (NW), the Chamfer distance (CD) and normal reconstruction error in degrees (NR).

Method	NW (%)	CD $*1^{e-2}$	NR
ball pivoting	25.7	0.524	6.59
PointTriNet [44]	17.2	0.337	6.24
RVE [10]	9.2	0.344	15.71
IER meshing [36]	5.3	0.343	6.30
α-shapes 3%	2.5	0.939	28.50
α-shapes 5%	1.7	1.064	17.69
Ours	**0.4**	**0.326**	**5.23**

4. Results

We evaluate our method by comparing the quality and accuracy of our reconstructed meshes to the current state-of-the-art.

Dataset Since our networks are trained on individual patches, our method is able to train successfully from a small training set of shapes. Each shape provides a large set of patches as training samples. We create a dataset with a total of 91 shapes chosen from Thingi10k [50] and the PCPNet [27] dataset, that we call FAMOUSTHINGI, since the PCPNet dataset contains several shapes that are well-known in the graphics and vision literature. Each shape is sampled uniformly with 10k points. We compute ground truth log maps at each point using the recent method by Sharp et al. [45]. The training set contains 56 of these shapes and the remaining shapes are used for evaluation. Example shapes and more details are given in Section D of the supplementary.

4.1. Comparison to Baselines

We compare our method to recent state of the art learning based methods for point-set triangulation, as well as to more classical methods.

Ball pivoting [6] and α-shapes [20]. These two classic techniques use the concept of rolling a ball on the surface to deduce connectivity at points of contact. For ball-pivoting, the ball radius is automatically guessed as the bounding box diagonal divided by the square root of the vertex count. For

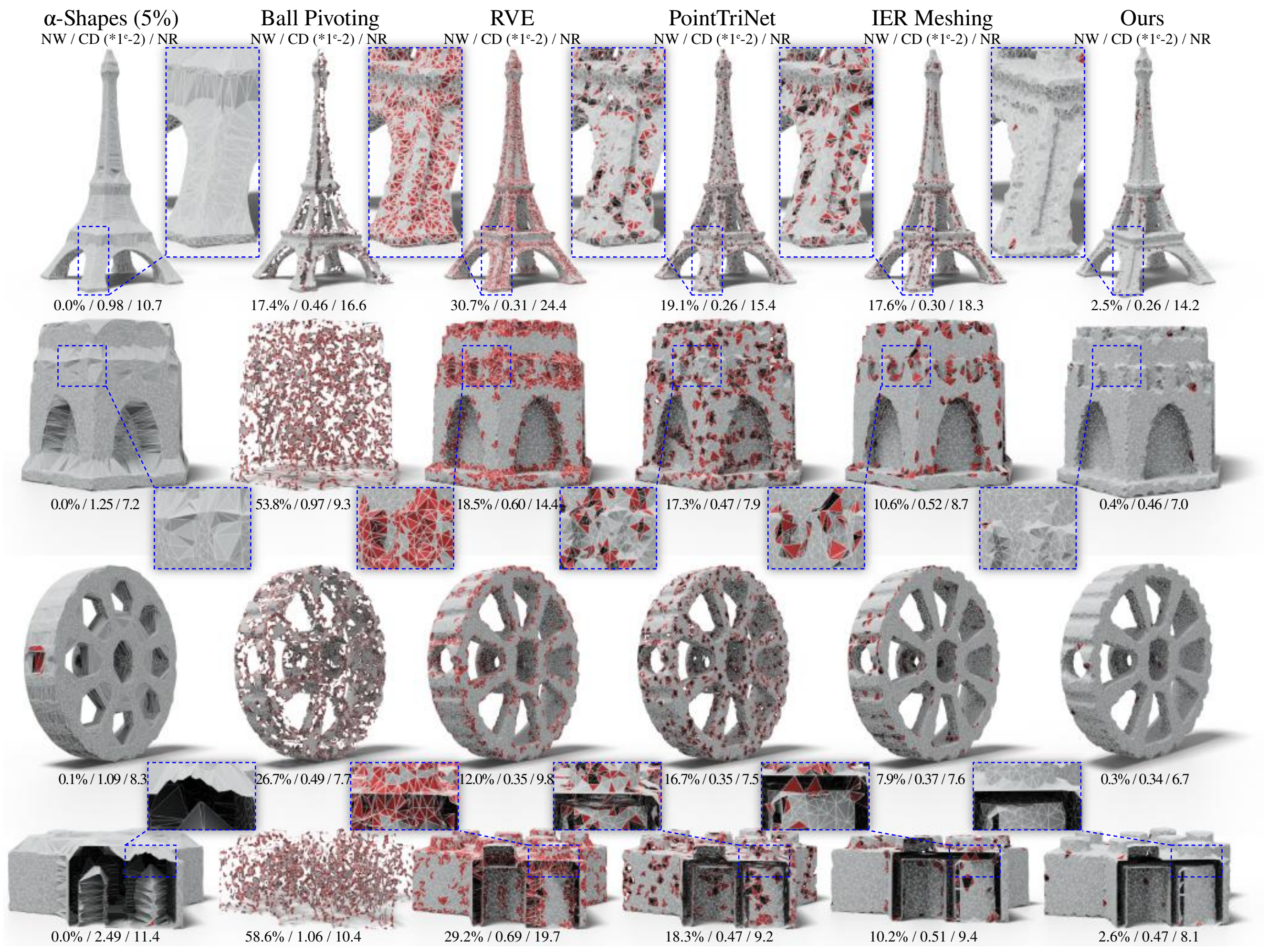

Figure 5. Qualitative comparison. We compare four meshes reconstructed by our method to the results of five current methods. Non-manifold triangles are marked in red and we show both the percentage of non-watertight edges (NW) and the Chamfer distance multiplied by 100 (CD) below each shape. Note that classical non-data-driven methods struggle to separate thin surfaces and data-driven methods have significantly more non-manifold triangles.

α-shapes, we report two different choices of the radius parameter α, as 3% and 5% of the bounding box diagonal.

Restricted Voronoi estimation [10] (RVE) This method is the closest existing baseline to our method. It estimates Voronoi cells restricted to the surface by projecting local patches to local tangent planes. Note that this method requires normal information that we estimate from the input point cloud.

PointTriNet [44] and IER meshing [36] We compare our method to two recent learning based methods for triangulating point clouds. We retrain PointTriNet on our dataset. Intrinsic-Extrinsic Ratio Guidance Meshing (IER meshing), however, needs a larger amount of data to train and overfits on our dataset. Since it is not patch based, it needs a larger variety of shapes to train. We use the pre-trained model provided by the authors, that was trained the larger ShapeNet dataset.

Metrics We compare to these methods using two metrics for the mesh quality and two metrics for the mesh accuracy. As mesh quality measures, we use the percentage of non-watertight edges (NW) and the standard deviation (A_σ) of triangle angles in the mesh. Note that due to the triangle selection step, all the produced edges are manifold (have one or two adjacent triangles) but the edges can be open. An angle of 60 degrees corresponds to equilateral triangles, while skinny triangles have more extreme angles.

As a measure of the surface reconstruction accuracy, we use the Chamfer Distance [4, 22] (CD) between a dense point set P_M sampled on the reconstructed surface and a dense point set $P_{\hat{S}}$ sampled on the ground truth surface:

$$\mathrm{CD}(P_M, P_{\hat{S}}) = \frac{1}{N} \sum_{p_i \in P_M} \min_{q_j \in P_{\hat{S}}} \|p_i - q_j\|_2 + \frac{1}{N} \sum_{q_j \in P_{\hat{S}}} \min_{p_i \in P_M} \|q_j - p_i\|_2$$

We also compare the normal reconstruction error (NR). At each vertex of the mesh we measure the angle difference in degrees between the ground truth normal and the normal obtained from our reconstructed mesh.

Quantitative Comparison In Table 4, we show a quantitative comparison between our method and the baselines. Our method yields lower chamfer distance, and less non-manifold edges, showing we both better-approximate the surface while at the same time outputting a triangulation with far less non-manifold artifacts. Indeed, only the classic technique of α-shapes manages to come close to our degree of manifoldness, at the cost of lower accuracy, due to filling in concave surface regions (see examples in Figure 5).

In Figure 7, we evaluate the quality of the generated triangles by considering the histogram of triangle angles. The standard deviation of each method is given next to its name. Our method yields superior triangle quality to all learning-based methods, and to all classic techniques except for ball-pivoting, which achieves better triangle quality by sacrificing manifoldness to a large degree.

Qualitative Comparison We show qualitative results in Figure 5, on 4 meshes of our FAMOUSTHINGI dataset. Non-manifold triangles are visualized in red, with the percentage of non-manifold triangles, as well as the Chamfer distance error, written beneath each result. The figure gives a very clear visual insight to the numbers from Table 4: the classic techniques work in a non-adaptive way which enables them to produce meshes with mostly-manifold edges, but they cannot handle thin and tight structures, like the scaffolds of the tower. In contrast, the learning-based methods are more local and can handle the concavities in, e.g., the wheel, but fall short on producing manifold triangulations. Our method, combining the robustness of classic Delaunay triangulation, with modern, data-driven learning techniques, manages to produce triangulations that both respect the original fine geometry and have less non-manifoldness.

We show additional results on five shapes of the ShapeNet dataset [11] in Figure 6. Compared to the two data-driven methods PointTriNet and IER Meshing, we improve upon the manifoldness, especially in regions with detailed geometry and high curvature, like the edge of the table, or the backrest of the chair. The results show a similar trend as in our FAMOUSTHINGI dataset. Note that IER meshing is trained on the ShapeNet dataset while PointTriNet and our method are trained on FAMOUSTHINGI dataset, demonstrating the ability of our method to generalize to unseen data. We show quantitative and qualitative results on the ShapeNet dataset in Section B and additional qualitative results in C of the supplementary.

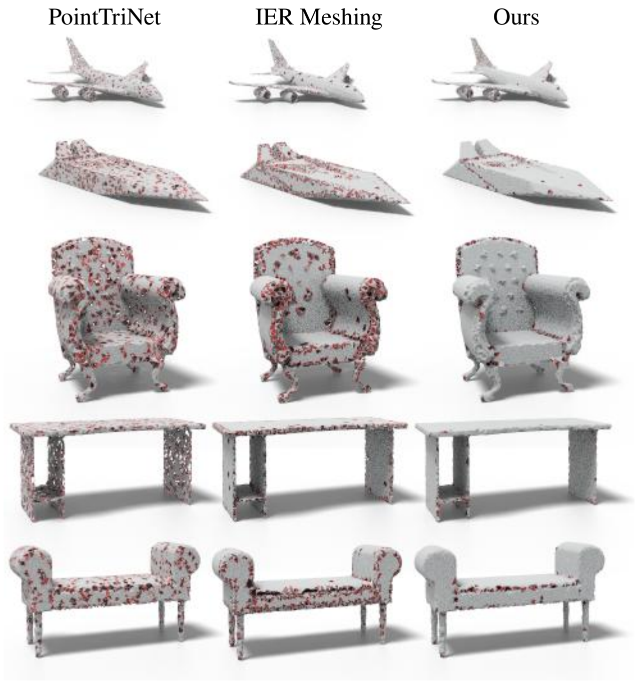

Figure 6. Qualitative comparison on ShapeNet [11]. We compare with the two data-driven methods PointTriNet and IER Meshing on five shapes taken from five different categories of the ShapeNet dataset. Our approach results in more manifold meshes, especially in detailed areas like the backrest of the chair.

Limitations Finding a geometrically complex surface, like on parts of the Eiffel tower in Figure 5, can be difficult. In such cases, the geodesic neighbors or logmap networks may misclassify/misplace some points. Moreover, thin parts of a model are particularly challenging. We can handle these cases better than existing works (Lego piece of Figure 5, or the plane wing in Figure 6. More extreme cases, like the leafs of a plant, would require training on a dataset where these cases are more common.

Non uniform sampling We evaluate our method on non uniformly sampled point clouds. In particular we sample points following a probability gradient along the y-axis (horizontal). We observe in Figure 1 (bottom) that our method performs better than other learning-based baselines. Note that PointTriNet, IER Meshing and our method have *not* been retrained on a non-uniformly sampled dataset. We

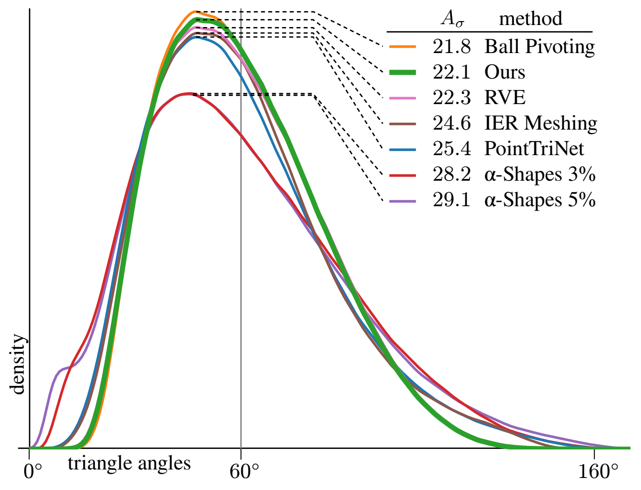

Figure 7. Distribution of triangle angles in the reconstructed meshes. Our method produces better shaped triangles than all other methods except for ball pivoting which sacrifices mesh manifoldness. We show the angle variance next to each method.

Table 2. Ablation study over the components of our method. Log map alignment, triangle selection as well as the the log map parametrization improve manifoldness in the output meshes.

Method	NW (%)	CD $*1^{e-2}$	NR
Ours w/o align, select	22.51	0.326	7.26
Ours w/o select	10.98	0.348	6.86
Ours w/o log maps	1.18	0.334	5.93
Ours w/o align	1.07	**0.325**	**5.19**
Ours	**0.40**	0.326	5.22

provide further evaluation on non uniformly sampled point clouds in Section A of the supplementary.

4.2. Ablation study

We evaluate the impact of each step in our pipeline using an ablation study, shown in Table 2. We remove one component at a time and compute the percentage of non-watertight edges (NW), Chamfer distance (CD) and normal reconstruction error (NR) as described before. We first remove the *align*ment of the logmaps of the delaunay triangulation, which results in a slight degradation manifoldness. Next, we evaluate the efficacy of our triangle *select*ion process by instead creating a mesh from all triangles in our Delaunay Surface Elements. This results in a significant drop in manifoldness of the triangulation, since we do not achieve perfect alignment of our logmaps. Dropping both the alignment and the selection results in a much more significant decrease in manifoldness than just removing the selection – this hints that the alignment is indeed producing more consistent local DSE's. Lastly, we replace the *log map*s with simple 2D projections, to get the local patch parameterization along the approximated normal vector. Please note that we still use the learned geodesic neighborhood. Manifoldness deteriorates as well, showing the necessity of our specific parameterization method. In particular, the 2D projection parametrization performs poorly for complex shapes such as the Eiffel tower (NW: 5.59% (w/o logmaps), 2.48% (Ours)) or Trilego (NW: 5.59% (w/o logmaps) NW: 1.64% (Ours)) shapes. Note that the Chamfer distance is not significantly increased by the removal of any component from our pipeline, as our method's locality prevents strong errors in the surface location by design, due to considering only the learned geodesic neighborhoods of the surface. We provide additional ablation of the learned Logmap component in Section G of the supplementary.

5. Conclusion

We presented Delaunay Surface Elements for robust surface reconstruction from points sets. In the process, we combine the best of two worlds: 2D Delaunay triangulation from classical computational geometry which comes with guarantee about mesh quality and manifoldness; and local logmaps learned using networks, followed by synchonization to get local data-driven 2D projection domains to handle non-planar regions. We demonstrated that the method can be trained with very limited training data and produces near-manifold triangulations that respect the original point set and have a higher mesh quality than the state-of-the-art.

In the proposed method, the final mesh extraction is done via a non-differentiable growing approach. In the future, it would be interesting to also learn the triangle selection via a network. This could enable a truly end-to-end optimization and allow us to optimize for context-specific point distributions accounting for data-priors (e.g., sharp edges) and scanner characteristics. Another direction would be to consider weighted Delaunay triangulations that provide additional freedom to local triangulations.

6. Acknowledgements

Parts of this work were supported by an Adobe internship, the KAUST OSR Award No. CRG-2017-3426, the ERC Starting Grant No. 758800 (EXPROTEA) and the ANR AI Chair AIGRETTE.

References

[1] Nina Amenta and Marshall Bern. Surface reconstruction by voronoi filtering. *Discrete & Computational Geometry*, 22(4):481–504, 1999. 2

[2] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. *Proc. SIGGRAPH*, pages 415–421, 1998. 2

[3] Abhishek Badki, Orazio Gallo, Jan Kautz, and Pradeep Sen. Meshlet priors for 3d mesh reconstruction. In *Proceedings of*

the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2849–2858, 2020. 2

[4] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'77, pages 659–663, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc. 6

[5] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*, volume 36, pages 301–329. Wiley Online Library, 2017. 1, 2

[6] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE transactions on visualization and computer graphics*, 5(4):349–359, 1999. 2, 5

[7] KP Horn Berthold and P Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the optical society of America*, 4(4):629–642, 1987. 4

[8] Jean-Daniel Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics (TOG)*, 3(4):266–286, 1984. 2

[9] Jean-Daniel Boissonnat and Steve Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67(5):405–451, 2005. 2

[10] Dobrina Boltcheva and Bruno Lévy. Surface reconstruction by computing restricted voronoi cells in parallel. *Computer-Aided Design*, 90:123–134, 2017. 2, 3, 5, 6

[11] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 7

[12] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019. 2

[13] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, Proc. SIGGRAPH, pages 303–312, 1996. 2

[14] Angela Dai and Matthias Nießner. Scan2mesh: From unstructured range scans to 3d meshes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5574–5583, 2019. 2

[15] Boris Delaunay et al. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934. 3

[16] Tamal K Dey. *Curve and surface reconstruction: algorithms with mathematical analysis*, volume 23. Cambridge University Press, 2006. 2

[17] Tamal K Dey, Joachim Giesen, and James Hudson. Delaunay based shape reconstruction from large data. In *Proceedings IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics (Cat. No. 01EX520)*, pages 19–146. IEEE, 2001. 2

[18] Julie Digne, David Cohen-Steiner, Pierre Alliez, Fernando De Goes, and Mathieu Desbrun. Feature-preserving surface reconstruction and simplification from defect-laden point sets. *Journal of mathematical imaging and vision*, 48(2):369–382, 2014. 2

[19] Manfredo P Do Carmo. *Differential geometry of curves and surfaces: revised and updated second edition*. Courier Dover Publications, 2016. 3

[20] Herbert Edelsbrunner and Ernst P Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics (TOG)*, 13(1):43–72, 1994. 2, 5

[21] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996. 5

[22] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017. 6

[23] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. *arXiv preprint arXiv:1904.06447*, 2019. 2

[24] Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh r-cnn. *arXiv preprint arXiv:1906.02739*, 2019. 2

[25] Meenakshisundaram Gopi, Shankar Krishnan, and Cláudio T Silva. Surface reconstruction based on lower dimensional localized delaunay triangulation. In *Computer Graphics Forum*, volume 19, pages 467–478. Wiley Online Library, 2000. 2, 17

[26] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 216–224, 2018. 2

[27] Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J Mitra. Pcpnet learning local shape properties from raw point clouds. In *Computer Graphics Forum*, volume 37, pages 75–85. Wiley Online Library, 2018. 5

[28] Philipp Herholz and Marc Alexa. Efficient computation of smoothed exponential maps. In *Computer Graphics Forum*, volume 38, pages 79–90. Wiley Online Library, 2019. 3

[29] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. *Surface reconstruction from unorganized points*, volume 26. ACM, 1992. 2

[30] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 339–346, 2002. 2

[31] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006. 2

[32] Alireza Khatamian and Hamid R Arabnia. Survey on 3d surface reconstruction. *Journal of Information Processing Systems*, 12(3), 2016. 2

[33] Arno Knapitsch et al. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM TOG*, 36(4), 2017. 15

[34] Ravikrishna Kolluri, Jonathan Richard Shewchuk, and James F O'Brien. Spectral surface reconstruction from noisy point clouds. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 11–21. ACM, 2004. 2

[35] Yiyi Liao, Simon Donne, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2916–2925, 2018. 2

[36] Minghua Liu, Xiaoshuai Zhang, and Hao Su. Meshing point clouds with predicted intrinsic-extrinsic ratio guidance. *arXiv preprint arXiv:2007.09267*, 2020. 1, 2, 3, 5, 6

[37] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987. 2

[38] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. 2

[39] Charlie Nash, Yaroslav Ganin, SM Eslami, and Peter W Battaglia. Polygen: An autoregressive generative model of 3d meshes. *arXiv preprint arXiv:2002.10880*, 2020. 2

[40] Timothy S Newman and Hong Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30(5):854–879, 2006. 1, 2

[41] Junyi Pan, Xiaoguang Han, Weikai Chen, Jiapeng Tang, and Kui Jia. Deep mesh reconstruction from single rgb images via topology modification networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9964–9973, 2019. 2

[42] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019. 2

[43] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 15

[44] Nicholas Sharp and Maks Ovsjanikov. Pointtrinet: Learned triangulation of 3d point sets. *arXiv preprint arXiv:2005.02138*, 2020. 1, 2, 3, 5, 6

[45] Nicholas Sharp, Yousuf Soliman, and Keenan Crane. The vector heat method. *ACM Trans. Graph.*, 38(3), 2019. 5

[46] Jonathan Shewchuk, Tamal K Dey, and Siu-Wing Cheng. *Delaunay mesh generation*. Chapman and Hall/CRC, 2016. 2

[47] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–67, 2018. 2

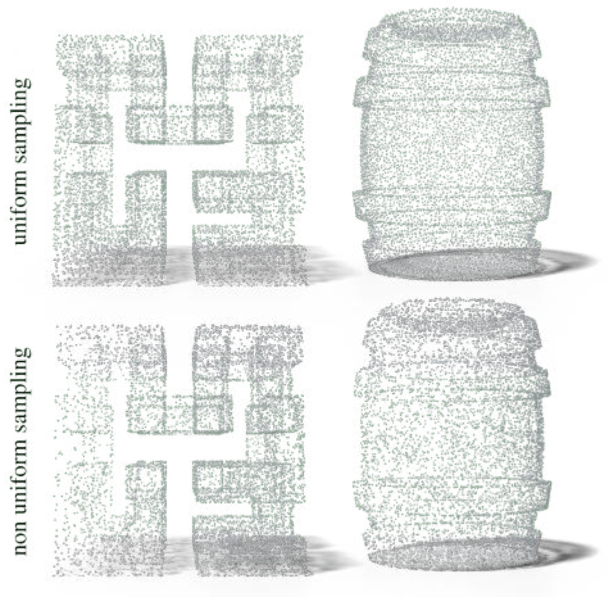

Figure 8. Examples of uniformly sampled point clouds (top) and non uniformly sampled point clouds. The density of points follows a gradient along the y axis (horizontal).

[48] Francis Williams, Teseo Schneider, Claudio Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. Deep geometric prior for surface reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10130–10139, 2019. 2

[49] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 206–215, 2018. 4, 15

[50] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*, 2016. 5

A. Non-uniform sampling

We provide additional results on a non-uniformly sampled variant of the FAMOUSTHINGI dataset. We sample points following a density gradient along the y-axis (horizontal in the figures), where point density correlates with the y-coordinate. A few examples are shown in Figure 8 (bottom). We did *not* retrain on this dataset variant and evaluate the same model we used for the uniform point clouds. In Table 3 we show that our method remains robust even with this non-uniform sampling, with only a small decrease in performance compared to uniform sampling. IER meshing takes the largest performance hit with over twice as many non-manifold triangles and significantly increased Chamfer distance. Overall our method shows a similar im-

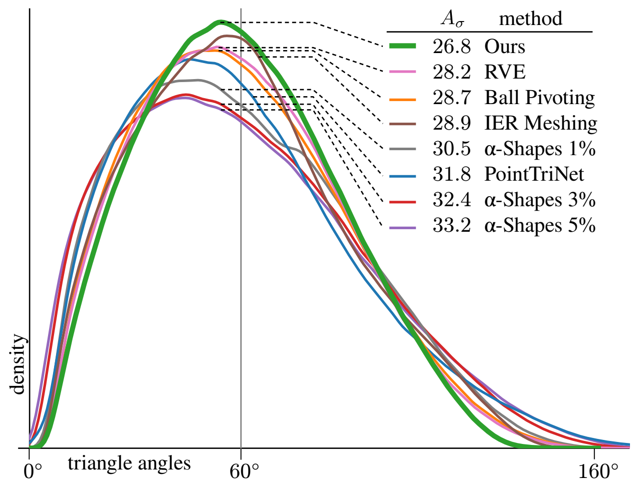

Figure 9. Triangle angles distribution. Our method produces triangles with angles more centered around 60 degrees and fewer very obtuse or very acute angles.

provement over the baselines as in uniform sampling. The angle distribution of triangles produced by our method is compared to all baselines in Figure 9. Our method achieves the best performance with angles more centered around 60 degrees.

We show qualitative comparison in Figure 10. We observe that ball pivoting and IER meshing are particularly impacted by the non uniform sampling while our method achieves the best quality reconstructions.

Table 3. Quantitative comparison the FAMOUSTHINGI testset where points are sampled non-uniformly. We compare the percentage of non-watertight edges (NW), the Chamfer distance (CD), and the normal reconstruction in degrees (NR) to all baselines.

Method	NW (%)	CD $*1^{e-2}$	NR
Ball pivoting	31.5	0.396	6.84
PointTriNet	14.2	0.383	6.59
IER meshing	13.5	0.487	7.00
RVE	11.0	0.396	9.08
α-shapes 1%	3.5	3.228	63.21
α-shapes 3%	2.7	0.971	28.88
α-shapes 5%	1.7	1.061	17.71
Ours	**1.3**	**0.356**	**6.02**

B. Results on ShapeNet

We compare our method to PointTriNet and IER meshing. Both our method and PointTriNet are trained on the FAMOUSTHINGI dataset, showing their generalization performance, while IER meshing was trained on ShapeNet (since IER meshing requires more shapes for training than the other two methods). Even though this gives IER meshing an advantage, we observe in Table 4 that our method still produces shapes with better manifoldness and Chamfer distance than other methods.

Table 4. Quantitative comparison on 100 random shapes from ShapeNet. We compare our three main metrics to learning-based baselines. IER meshing was specifically trained on ShapeNet, while our method trained on a different dataset (FAMOUSTHINGI). Even with this handicap, our method obtains better manifoldness and Chamfer distance.

Method	NW (%)	CD $*1^{e-2}$	NR
PointTriNet	22.33	0.416	10.95
IER meshing	6.96	0.456	**6.54**
Ours	**5.51**	**0.396**	9.44

Additional qualitative results are provided in Figure 11. We observe that our method produces meshes with better manifoldness and preserves details such as the drawer handles (row 2) or the two sides of the plane wings (row 1) more accurately. Finally, our method produces fewer large holes in the reconstructed mesh.

C. More Qualitative Results

We provide additional qualitative results by meshing point clouds of well-known monuments obtained from Famous Paris Buildings. Results are shown in Figure 12. Since these shapes are geometrically more complex than the shapes in FAMOUSTHINGI or ShapeNet, we uniformly sample 50k points from each monument. We do not re-train on this dataset. Our method generalizes well to unseen data and denser point clouds.

We also include a real scan reconstruction in Figure 13. We reconstruct a point cloud with 50k sampled points and compare to other learning-based methods. Since IER meshing can not handle 50k points, we sample 12k points for comaring to IER meshing.

D. Dataset Examples

A few examples of shapes from our FAMOUSTHINGI dataset are shown Figure 14. In Figure 8, we show two examples of uniformly sampled point clouds we use as input to our method, and two non-uniformly sampled point clouds that we use in the experiments described in Section A.

E. Architecture Details

We show the detailed architecture of our geodesic patch *classification network* and the 2D log map *projection network* in Figure 15. The classification network implements a function $c_j := f_\theta([q_j^i, d_j^i] \mid Q^i)$ that classifies if each point q_i in the euclidean patch Q^i is part of the geodesic patch P_i, while the projection network implements a function $u_j^i := g_\phi([p_j^i, d_j^i] \mid P^i)$ that projects points p_j^i in the geodesic patch P_i to their 2D log map coordinates u_j^i. Here d_i is the euclidean distance from a point to its patch center.

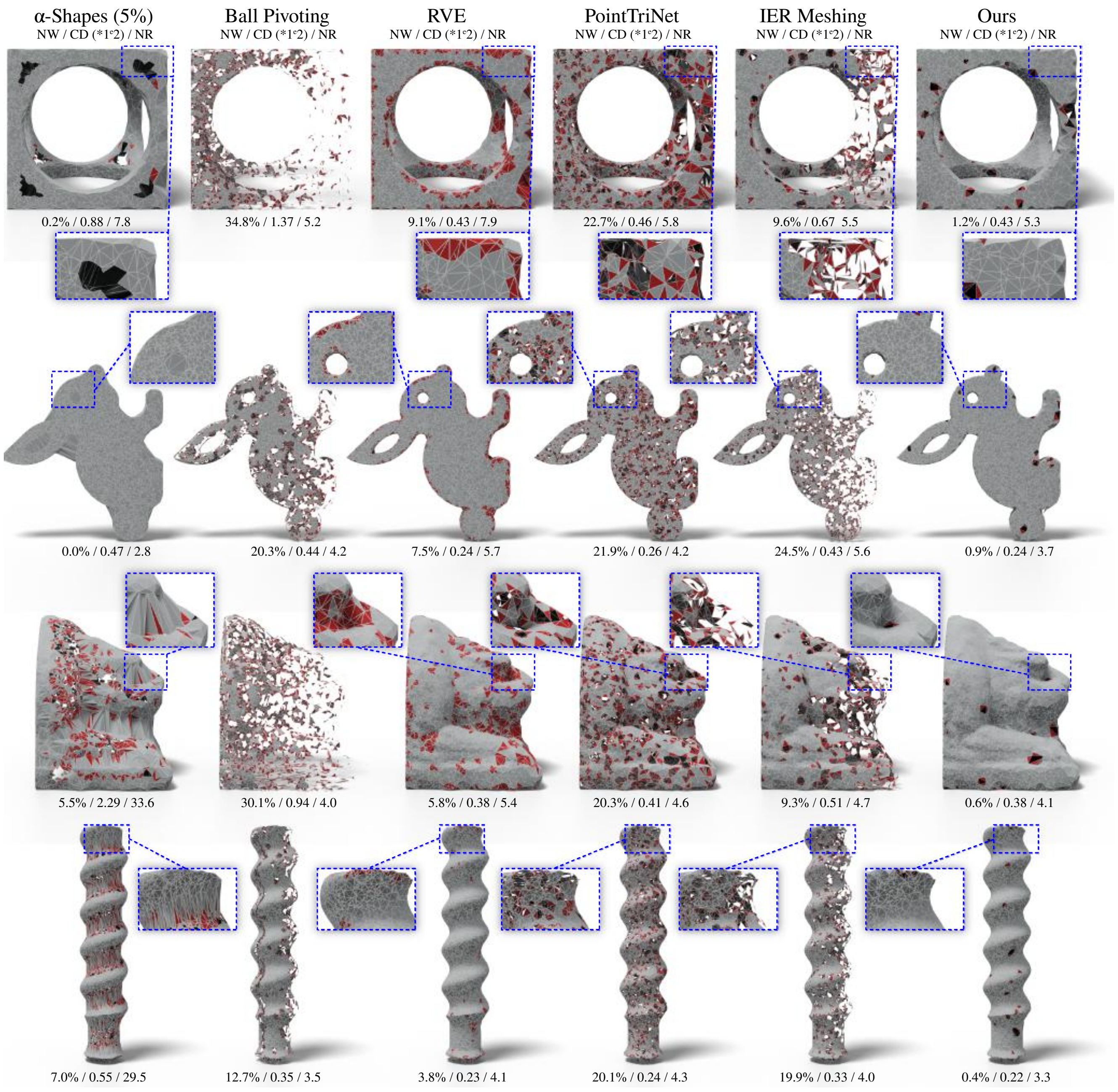

Figure 10. Surface reconstructions from non-uniform point clouds. Non-manifold triangles are marked in red. Shapes are sampled more densely to the left and more coarsely to the right. We can see that methods struggle to reconstruct the coarsely sampled parts of the point cloud. While our method also has slightly more errors in the coarsely sampled regions, the mesh quality drops by a much smaller amount from densely to coarsely sampled regions.

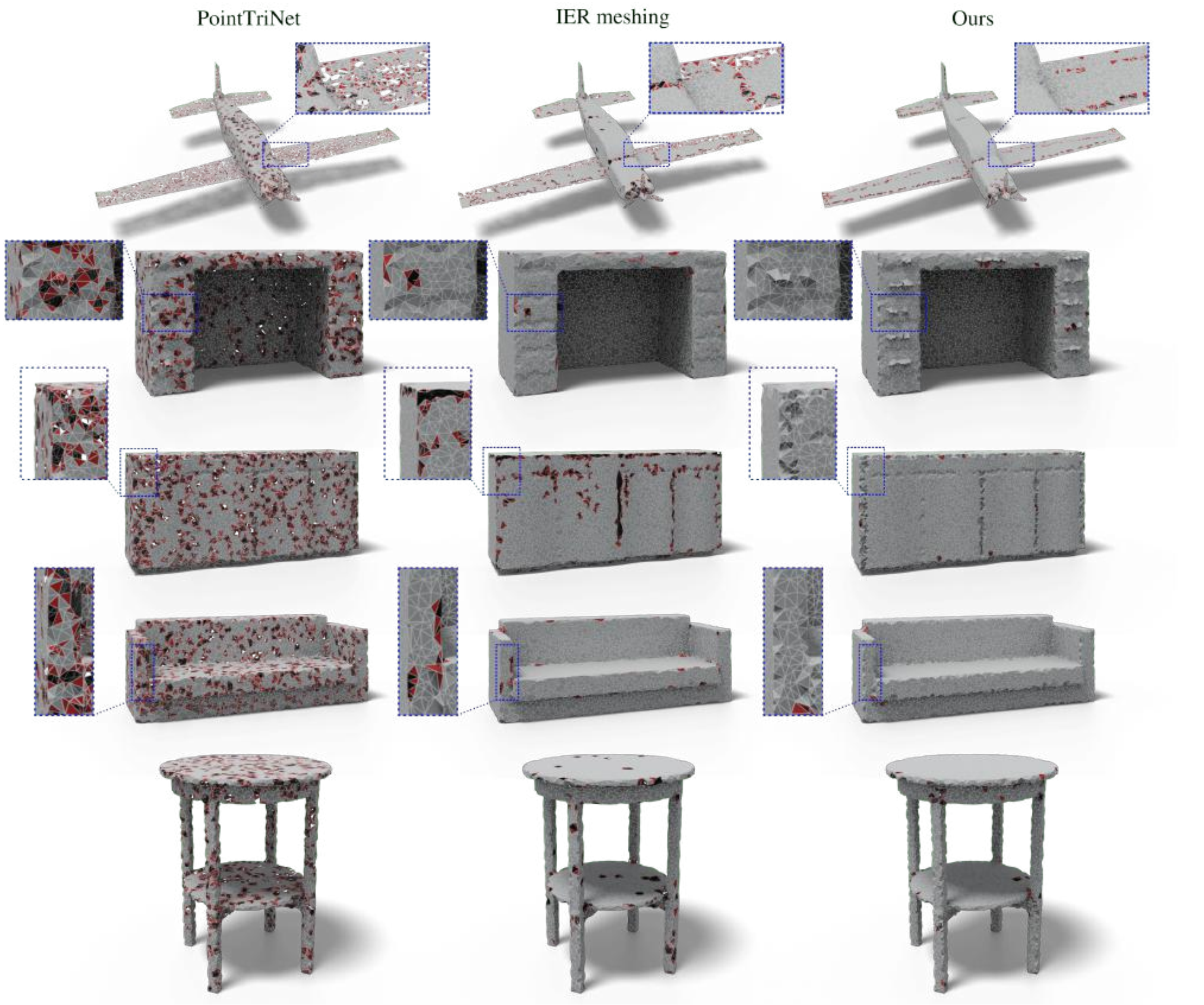

Figure 11. Qualitative results on ShapeNet testset. We do not retrain our method on the ShapeNet dataset while IER meshing *was* trained on this dataset. Even so, our method produces more manifold meshes and preserves details such as the drawer handles (row 2) more accurately. We better separate the two sides (top and bottom) of the plane wings (row 1). Finally, our method presents fewer large holes in the reconstructed mesh.

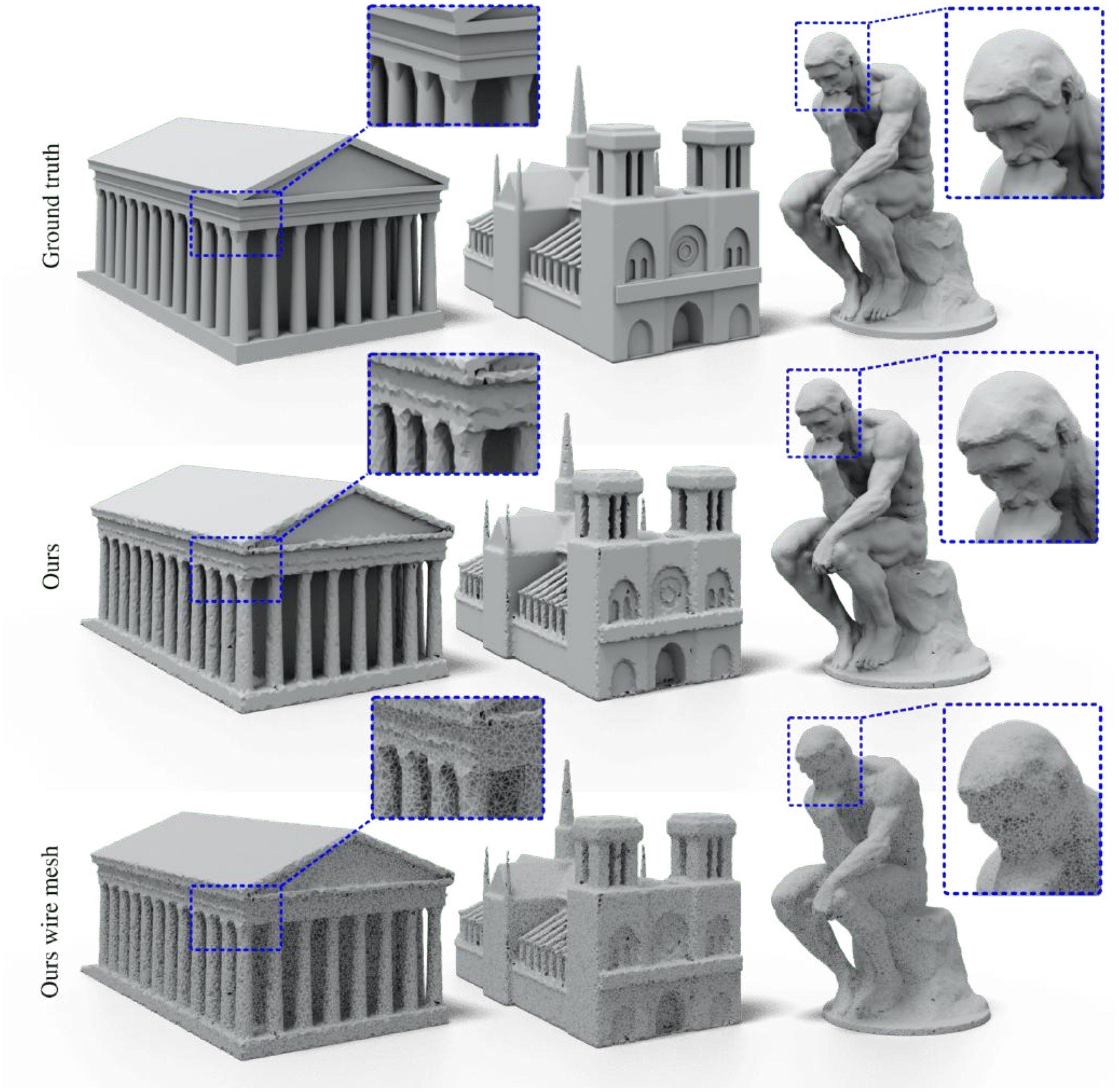

Figure 12. Meshing well-known monuments. We show the ground truth (top), the reconstructed mesh (middle), the reconstructed mesh with non manifold triangles colored in red (bottom). Our method generalizes well this more complex data that is also sampled more densely than our training set.

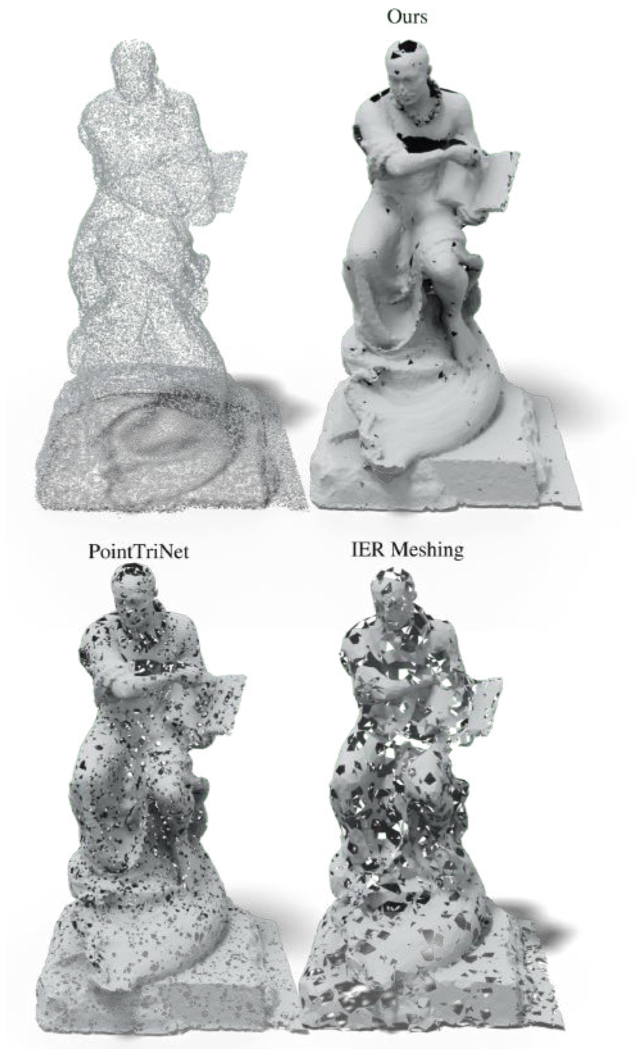

Figure 13. Reconstructing real scans from Tanks and Temples [33]

Both networks use the same architecture based on FoldingNet [49], except for their output dimension. They take as input a 3D point concatenated with the distance to the patch center and proceed to compute a 1024-dimensional global feature vector for the input patch with a PointNet [43]. Each input point is then augmented with this global feature vector and transformed by two blocks of per-point MLPs into a one-dimensional (classification network) or two-dimensional (projection network) per-point output.

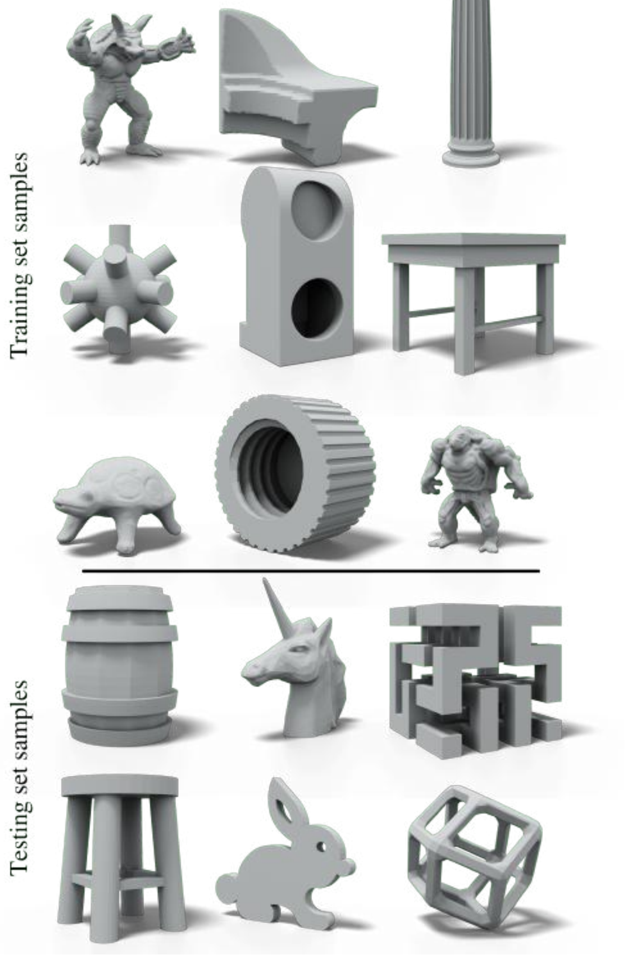

Figure 14. Examples from our dataset. We show ground truth meshes from both the training set and the test set.

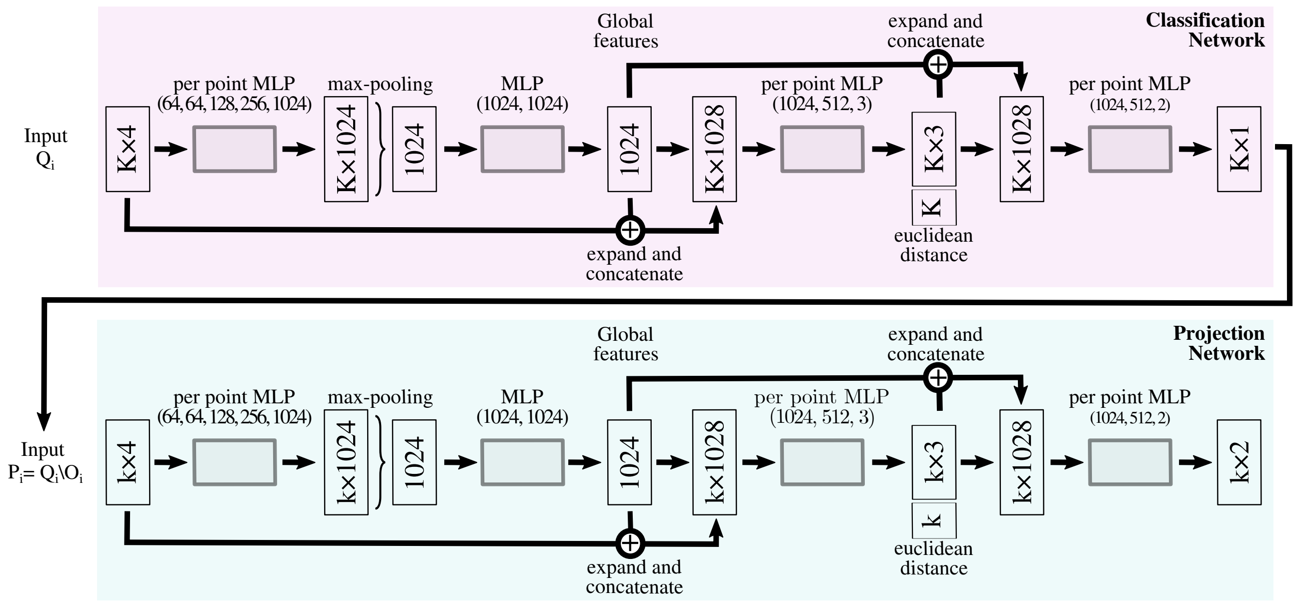

Figure 15. Detailed architecture of our pipeline. We first select a small geodesic patch using the classification network (purple). The projection network (blue) then applies a 2D projection to this patch that approximates a log map.

Table 5. Average runtime estimation per step on 10k point clouds in seconds.

Log map est.	Log map align.	Selection	Total
5.8	24.8	2.1	32.8

Table 6. Quantitative comparison our our learned logmap component to two logmap approximation methods.

	geodesic distance $*1^{e-3}$	2D position $*1^{e-2}$
Projection	1.943	2.627
Rotation	1.943	2.835
Ours	**0.471**	**0.835**

F. Runtime

We measure the average runtime of our method on point clouds of 10k vertices in Table 5, including the runtime for each step of the method.

G. Ablation of the Learned Logmap

We compare the performance of our learned logmap component to two baselines. The first baseline approximates the logmap by projecting neighboring points onto the tangent plane computed from the ground truth normal. The second baseline is the approach proposed in [25], where points are rotated onto the tangent plane. Please note the both of these baselines use ground truth normal information, while our method does not. We evaluate the methods on a subset of 33 manifold shapes from our FAMOUSTHINGI testset and sample 2k patches of k=30 geodesic neighbors per shape. We measure the MSE of the geodesic distance and of the 2D coordinates after patch alignment. Our method produces significantly better logmap estimates compared to other baselines as we show in Table 6.

H. Ablation of Neighbor Counts k and K

We evaluate our method on different values of the geodesic neighbor count k (20, 30, and 50) and different values for the euclidean neighbor count K (80, 120, 160) in Table 7. For each pair of (k, K) values, we train our models for 30 epochs. The choice of the geodesic neighbor count k affects the performance of our method significantly. If k is small, the Delaunay element approximation quality is affected. If k is large, it is more difficult for the logmap estimation network to produce a usable logmap. Changes in the choice of the euclidean neighbor count K lead to less significant performance drops. In our experiment we choose the parameter values $k = 30$ and $K = 120$ which produce the best results for the non-watertightness and normal reconstruction metrics. Please note that for $k = 30$ the difference in Chamfer distance values is negligible.

Table 7. Ablation of different values for the geodesic and euclidean and neighbor counts k and K.

k	K	CD($*1^{e-2}$)	NW(%)	NR
20	80	0.3437	5.569	5.921
20	120	0.3394	4.381	5.845
20	160	0.3496	4.712	6.483
30	80	0.3274	0.509	5.682
30	120	0.3276	**0.485**	**5.661**
30	160	**0.3272**	0.524	5.690
50	80	0.3335	1.822	6.046
50	120	0.3282	0.667	5.856
50	160	0.3286	0.728	5.883