Softmesh: Learning Probabilistic Mesh Connectivity via Image Supervision

Eric-Tuan Lê[1] Niloy J. Mitra[1,2] Iasonas Kokkinos[1,3]

[1]University College London [2]Adobe Research [3]Snap

Abstract

In this work we introduce Softmesh, a fully differentiable pipeline to transform a 3D point cloud into a probabilistic mesh representation that allows us to directly render 2D images. We use this pipeline to learn point connectivity from only 2D rendering supervision, reducing the supervision requirements for mesh-based representations.

We evaluate our approach in a set of rendering tasks, including silhouette, normal, and depth rendering on both rigid and non-rigid objects. We introduce transfer learning approaches to handle the diversity of the task requirements, and also explore the potential of learning across categories. We demonstrate that Softmesh achieves competitive performance even against methods trained with full mesh supervision.

1. Introduction

In this work we aim at making point clouds a first-class representation for deep geometry processing. The popularity of point clouds is due on the one hand to their minimal format which has facilitated successful and simple MLP-based deep architectures [26, 35, 22], and their ubiquitous nature, being the natural output of SfM pipelines [30, 31] or depth sensors [8]. At the same time, point clouds are unstructured, namely come as sets. This impedes the connection of 3D point clouds with the abundant supervision available for 2D images; a mesh structure is needed to connect points into a surface, which in turn can be rendered into a two-dimensional image. Unfortunately, meshes are not directly measured, but rather estimated through heuristics [7], estimated as an indexed faceset from scans [11], or time-demanding optimization algorithms [37, 18, 9, 17] that build a triangulated graph on top of a set of points. This obstructs the application of deep learning algorithms, making it hard, or even impossible to train end-to-end. We argue that tackling this problem would allow deep learning approaches for 3D point-cloud processing to leverage on the vast amounts of 2D supervision from real images.

We investigate how to convert a point cloud into an image through a lightweight, mesh-construction operation that can be seamlessly integrated with neural rendering pipelines. Our main goal is to enable this with *only image supervision.* To this end, we propose *Softmesh* that learns a distribution over meshes, by training a deep neural network to deliver the probabilities over different (candidate) mesh faces. We bundle our method together with an appropriately adapted differentiable rendering method [21] and show that our network allows us to successfully render multiple surface properties, including normals, depth and silhouettes *without* any 3D connectivity supervision. The different output spaces – e.g., discrete for silhouette, unit-norm R^3 vectors for normals – are easily accommodated for by appropriate losses on the rendering outputs.

Figure 1. Our approach takes as input a 3D point cloud, estimates the probability of point grouping into faces, and delivers a soft mesh representation that allows us to render 2D images in an end-to-end differentiable manner. We thereby learn a probabilistic mesh estimate from 2D image supervision. (Middle) Softmesh output with probabilistic faces from an input of 512 points. Low probability faces are shown in transparent and high probability faces in red (Left & Right) Sectional views of the interior of the mesh.

We show that our method can be applied on a wide variety of objects, both rigid or non-rigid objects. In particular, we experiment with 3D human scans from the MPI-FAUST dataset [2], as well as the 16 object categories from the ShapeNet-Part dataset [5]. Our proposed approach significantly improves on classical approaches while being on par with other neural network approaches trained with stronger 3D supervision. We further explore the ability of our net-

work to transfer knowledge across different rendering tasks, and also across different shape categories.

In summary, our key contributions are:

- Softmesh, a fully differentiable pipeline that can learn a distribution over non-manifold meshes from 3D point clouds;
- a pipeline that can be trained end-to-end using only 2D-image supervision for multiple rendering tasks and object categories; and
- an extensive evaluation demonstrating improvements over classical methods while achieving similar performance to 3D-supervised method.

2. Related Works

Deep point cloud processing The now seminal PointNet [26] introduced a novel network of MLPs to do both classification and regression tasks directly on unstructured point clouds. The work has resulted in a cascade of followup works [26, 27, 23, 33, 19, 13]. Others have contributed to working directly on point clouds in the form of per-point attribute estimation [13], point upsampling [28], estimating surface geodesic distance directly on point clouds [15], or rendering of an image using a learnt rendering function [1] to name few applications. In the context of generative models, PointFlow [35] learns to model a distribution using continuous normalizing flow. There has been as well several works using transformers and attention mechanism for point cloud processing [36, 6, 32] but without explicitly building a graph from the input point cloud. While these works support analysis/synthesis direclty using point clouds, they do not allow to create a final (rendered) image as output.

Classical Mesh reconstruction One of the earliest papers exploring the problem of surface reconstruction from pointclouds is the work by Hoppe et al. [16]. Subsequently, several methods have been developed using Fourier-based reconstruction, wavelet basis, radial basis functions [3] to approximate the underlying signed/unsigned distance field. The most celebrated result in this context is the Poisson reconstruction formulation where surface reconstruction from signed point clouds is cast as a Poisson problem [17]. A final mesh can then be obtained by running marching cubes [10] on the implicit representation. For a detailed discussion, please refer to the survey [16]. These classical methods produce a single mesh as a reconstruction of the input point cloud.

Deep mesh processing There has been fewer works aiming at obtaining a mesh from a raw scan. One of the earliest attempts was the Scan2Mesh [11] work that resampled a pointset and produced candidate triangles as a 'reconstruction' of the raw pointsets. More recent methods have been introduced to learn point connectivity using ground truth mesh information for geodesic distance supervision [20, 29]. Recent efforts have also focused on 'shrink-wrapping' a mesh to a point cloud [12, 14]. Alternately, deep geometric priors [34] overfits to local parameterization charts to point cloud patches to produce a dense reconstruction of the input point cloud. Starting from raw point clouds, these works attempt to produce a single mesh, manifold or otherwise, as a final underlying surface. We take a different view.

Acknowledging the space of possible underlying surfaces that a discrete point cloud sampling can represent [25], we venture to directly estimate a distribution over face probabilities that can be inferred from a raw point cloud. This, in turn, allows us to link point clouds, through our Softmesh, to a differentiable rendering approach [21] that operates on triangle mesh representation for image rendering tasks.

3. Method

Our goal is to establish an end-to-end trainable pipeline able to handle a wide variety of rendering tasks directly from point clouds, based on a new representation that we call *Softmesh*. This soft representation is learnt via points affinity that represents how likely an edge connecting two points is. Triangle faces can then be obtained by combining relevant edges and finally allowing rendering operations. Thanks to the end-to-end trainable pipeline, we can back-propagate from the loss through the whole process effectively allowing us to learn (mesh) connectivity with only 2D image supervision. We now turn to a more in-depth presentation of the different components.

3.1. Learning point-to-point connectivity

We aim to find a set of soft edges connecting the different points from a point cloud. This is the first step before getting the set of probabilistic faces that could then be rendered. In particular, we use a position and semantic encoder to map points in a new feature space where point affinity can be measured easily from the two point features of an edge.

3.1.1 Position and Semantic Encoder

To embed the position of each input point $p \in \mathbb{P}$ in the source point set $\mathbb{P}$, we use the embedding initially introduced in the NeRF [24] pipeline. This embedding relies on the idea that mapping inputs to a higher dimensional space using high frequency functions helps networks to better fit to high frequency variations. The function γ maps each coordinate - normalized to [-1,1] - in $\mathbb{R}$ to a feature in $\mathbb{R}^{2L}$

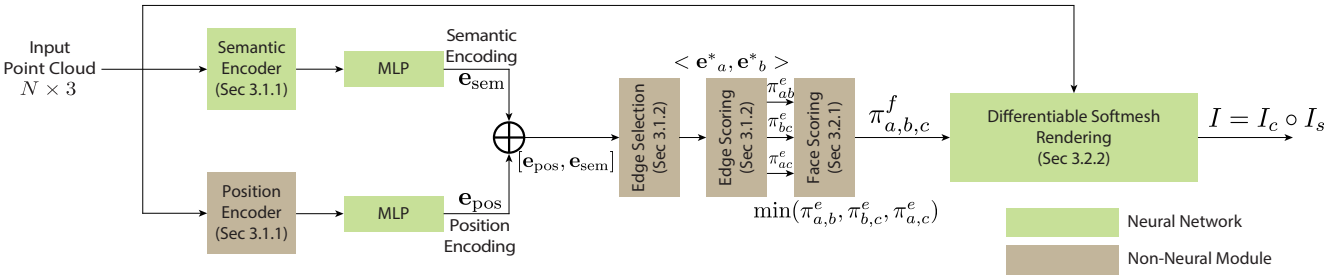

Figure 2. Pipeline of the proposed Softmesh pipeline. Two semantic and position encoders act in parallel to extract an encoding for each point. The Edge Selection module then shortlists possible edges that are scored by a subsequent Edge Scoring module. Edge scores are combined into the Face scoring module. The predicted faces, with associated probabilities, are finally rendered by our customized differentiable soft rasterizer. The whole setup is trained end-to-end with only 2D image level supervision.

as:

$$\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p), ..., \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)) \quad (1)$$

To learn a proper encoding for any given task, we process the output of the γ function with a simple MLP with learnable parameters.

To additionally provide semantic context to the final point encoding, our pipeline relies on a Deep LPN [23] network that extracts semantic context for each point. This semantic embedding complements the previous position embedding by adding some global and local contextual information for each point. A second MLP with the same property as for the position encoding is applied to the output of the Deep LPN network.

The two encodings, position $\mathbf{e}_{\text{pos}}$ and semantic $\mathbf{e}_{\text{sem}}$ based, are then concatenated together as a single vector $\mathbf{e}$.

$$\mathbf{e} = [\mathbf{e}_{\text{pos}}, \mathbf{e}_{\text{sem}}] \quad (2)$$

3.1.2 Edge Selection and Scoring

To avoid dense connectivity, an Edge Selection module is designed to shortlist a set of edges to consider for each point. The Edge Selection module selects the edges based on the k-nearest neighbors in the spatial xyz-space for each point. The shortlisted set of E edges is then scored by the Edge Scoring module. To score an edge (a, b) based on the point encodings $\mathbf{e}_a$ and $\mathbf{e}_b$, we use the cosine similarity:

$$\pi^e_{ab} =< \mathbf{e}^*_a, \mathbf{e}^*_b >$$

with normalised encodings: $\mathbf{e}^* = \mathbf{e}/\|\mathbf{e}\|$.

3.2. Point Cloud Rendering

Our aim is to establish triplets of points that are likely to belong to the surface of an underlying object. This is meant to favor connections between points that for instance are nearby in the geodesic sense, and triplets that form roughly equilateral triangles, while being tangential to the object's surface. Clearly, these properties are largely context-dependent, implying an understanding of the underlying surface geometry in their definition.

3.2.1 Face scoring

In order to render a point cloud we need to form surface faces connecting triplets of points in the case of triangular meshes. As described in Section 3.1, our Edge Weight module has been designed to compute the score of any given edge but not triplet of points. Thus, to score each face composed of the triplet (a, b, c), we consider the minimum score of all three edges $\{(a, b), (b, c), (a, c)\}$ in the point triplet:

$$\pi^f_{a,b,c} = \min(\pi^e_{a,b}, \pi^e_{b,c}, \pi^e_{a,c}) \quad (3)$$

where π^f denotes face score and π^e edge score. This captures the constraint that all three edges need to be highly probable in order to form a strong face candidate.

Individual face candidates can be efficiently scored based on Equation 3, but forming a proper distribution on faces would require enumerating all possible faces, which has a $O(N^3)$ memory- and time-complexity. Instead, we rely on the pre-selection from the Edge Selection module that shortlists for each point a set of E edges. A face will be kept only if all of its three edges have been shortlisted. To reduce complexity even further, we select a subset of F faces such that this set of faces covers most of the initial surface. This process helps to remove similar and co-planar faces located in the same local neighborhood. All remaining faces are then scored by Equation 3. This results in a lightweight approximation with $O(NE^2)$ time complexity and $O(NF)$ memory complexity that associates each of the N points with at most $E \times (E-1)$ soft faces. For sake of simplicity, we will now refer to each face with its index j and not the index of its three vertices (a,b,c) anymore.

3.2.2 Differentiable Softmesh Rendering

Rendering allows us to turn a discrete function, defined on the 3D mesh vertices, into a continuous 2D image generated by observing the mesh through a camera. Being one of the core components in any graphics pipeline, it has been revisited by many works in deep generative models with the aim of turning rendering into a differentiable module that

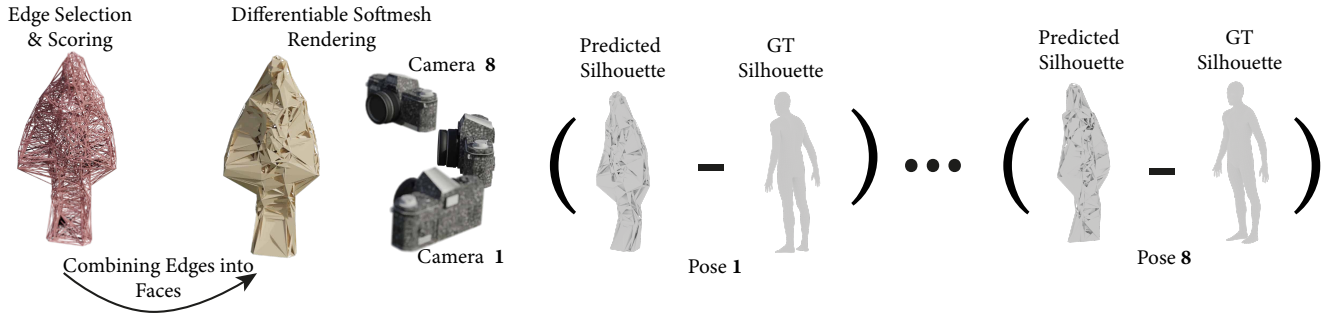

Figure 3. Rendering the point cloud. First, potential edges are selected and scored based on the cosine similarity of (learned) features. Then, edges are combined into soft faces. The soft mesh is then rendered from multiple views and compared to the ground truth.

facilitates end-to-end training of 3D-based generative models. These works have replaced several of the inherently discrete operations involved in rendering, such as ray tracing, with soft, probabilistic counterparts. The multitude of approaches to this problem indicates that this is far from an obvious task.

Our work introduces one more challenge that goes beyond the ones addressed in these works: the mesh is not provided in advance, but rather delivered by our pipeline in the form of a set of triangles that come with different levels of confidence. We adopt SoftRas [21] a recent state-of-the-art differentiable rendering pipeline and modify it so as to operate with soft meshes.

In particular the probabilistic nature of our meshes results in two major differences: firstly, the contribution of a face depends not only on its position but now on its probability as well and secondly, the number of faces hit by a ray can be much higher, meaning that a soft accumulation of many bits of incoherent information can result in an undifferentiated average.

At the core of SoftRas [21] rendering pipeline, the association of a pixel i with the mesh face j is computed as a soft value. Denoting by d_{ij} the euclidean distance between the pixel and the face edges, we determine the affinity of triangle j to pixel i through the following expression:

$$\mathcal{D}_j^i = \sigma\left(\delta_{ij} \cdot \frac{d_{ij}^2}{s}\right), \quad \delta_{ij} = \begin{cases} 1, & i \in f_j \\ -1, & i \notin f_j \end{cases} \tag{4}$$

where σ is the sigmoid function, s is a learnable sharpness parameter, and δ results in an increase of the influence the closer ray i hits the center of the face ($i \in f_j$) and a decrease as soon as the ray does not intersect the face ($i \notin f_j$). Above a certain distance τ, the affinity is set to 0. We define the set $\mathcal{R}_i$ as the set of all faces with a non-zero affinity with pixel i:

$$\mathcal{R}_i = \{j \quad | \quad \mathcal{D}_j^i \neq 0\} \tag{5}$$

Hard rendering would pick the closest face where $\delta_{ij} = 1$ but this would not allow for back-propagating through the rendering operation. Instead we allow multiple faces to influence the value of a single pixel through two modifications. Firstly, Equation 4 allows a face to 'bleed' outside its strict, deterministic projection to the image plane - this underlines soft rasterization. Secondly, a single ray can hit multiple faces of an object. In the standard case of a differentiable renderer [21] these faces are known in advance and limited. In our case, there is a multitude of options as our pipeline predicts a distribution over faces.

Putting these aspects together, we determine the weight by which each face j influences pixel i according to the following distribution:

$$w_j^i = \alpha \pi_j^f + \beta \mathcal{D}_j^i \exp(-z_j^i/\gamma) \quad \text{with} \quad j \in \mathcal{R}_i \tag{6}$$

where we accommodate the score of face j, π_j^f and combine it with the affinity function $\mathcal{D}_j^i$, defined in Equation 4, and the opposite distance $-z_j^i$ of face j to pixel i - allowing closer faces to have higher influence. Two learnable parameters α and β are used to combine the face score and its depth position. This avoids accumulating information from faces located on the other side of an object.

Based on these weights, the value of pixel i is not determined by a single, hard-assigned face, but rather through an accumulation function $\mathcal{A}_S$ over all faces hit by the ray:

$$I_c^i = \mathcal{A}_S(\{C_j^i\}) = \frac{\sum_{j \in \mathcal{R}_i} \exp(w_j^i) C_j^i}{\sum_{j \in \mathcal{R}_i} \exp(w_j^i)}, \tag{7}$$

where C_j^i is the value of the rendered attribute at face j, and I_c^i is the value induced on pixel i.

Beyond introducing face probabilities, one further deviation from common soft rendering pipelines consists in learning a figure-ground mask I_s^i in tandem with the color I_c^i value function. This has the same functionality as alpha-matting, allowing us to disentangle learning of appearance rendering from the figure-ground segmentation task. The rendered image is obtained by the element-wise product between those two maps: $I = I_c \circ I_s$.

Due to the fact that Equation 7 accumulates the contribution of each face, statistics can be biased by a large set

of faces even with a low probability. Hence, in particular for the figure-ground generation we propose a max-pooling alpha map aggregation function to predict the alpha channel of our rendered image:

$$I_s^i = \mathcal{A}_O(\{p_j^b\}_{|j\in\mathcal{R}_i}) = \max_{j\in\mathcal{R}_i}\{p_j^b\} \tag{8}$$

with $p_j^b = \sigma(\pi_j^f) = 1/(1+e^{-\nu\pi_j^f})$ and ν a learnable parameter. p_j^b represents the probability of face j to be located within the ground truth silhouette boundary and not the background on all views. However, p_j^b does not prioritize any specific face j in $\mathcal{R}_i$ when rendering a given attribute (i.e. normals, depth) on pixel i if all faces only span over the ground truth silhouette.

This ensures that many low-probability faces do not accumulate to a large value and delivers sharper segmentation masks.

3.2.3 Computing face probabilities

As explained in Sec 3.2.1, a global score can be computed for any face by Equation 3. However, even with a high global score, a face hidden within an object will never be rendered and should be assigned a low probability. Thus the probability of a face should be based on its ability to influence output pixels on rendered images which is defined by Equation 6.

However, such an equation is view dependent which should not be the case for face probabilities. Thus we compute the probability of a face $\mathbf{P}_j$ as the maximum normalised weight $w_j^{v,i}$ of face j across a set $\mathbf{V}$ of different views and all pixels i.

$$\mathbf{P}_j = \min\left\{\max_{v\in\mathbf{V}}\max_{i|j\in\mathcal{R}_i}\frac{\exp(w_j^{v,i})}{\sum_k\exp(w_k^{v,i})}, p_j^b\right\} \tag{9}$$

Of course, a face lying outside of the silhouette boundary will be visible under certain views and thus should be scaled down to the probability p_j^b.

3.2.4 Training the renderer

Once our probabilistic mesh representation is bundled together with a differentiable renderer, we have a fully-differentiable pipeline for converting a point cloud into a 2D image. As such, we can learn about mesh connectivity by training our network in an end-to-end manner based on 2D images generated from ground-truth meshes.

The global loss function is defined as a weighted sum of intermediate losses, adapted to the tasks at hand. In particular for continuous outputs we use ℓ_1 losses $\mathcal{L}_1^{norm}$ and $\mathcal{L}_1^{depth}$ for the normals and depth, respectively; we add a cross-entropy loss $\mathcal{L}_{cross}^{silh}$ for the silhouette task which can be seen as binary classification; finally we use an $\ell_{0.5}$ sparsity loss on the face probabilities assigned to each pixel i $\mathcal{L}_{reg}^i = \left(\sum_j p_j^{1/2}\right)^2$ to minimize the number of faces with non-zero probabilities. The combination weights for these losses are determined by cross-validation on a small subset of the training set.

$$\mathcal{L} = w_1^{norm}\cdot\mathcal{L}_1^{norm} + w_1^{depth}\cdot\mathcal{L}_1^{depth} + w_{cross}^{silh}\cdot\mathcal{L}_{cross}^{silh} + w_{reg}\cdot\sum_i\mathcal{L}_{reg}^i \tag{10}$$

The ℓ_1 losses are applied only within the object boundary as the rendered attributes (normals, depth) are learnt separately from the silhouette as explained in Section 3.2.2. For the depth, we have for example:

$$\mathcal{L}_1^{depth} = \sum_i |I_{depth}^i \circ \overline{I_s^i} - \overline{I_{depth}^i}| \tag{11}$$

with $\overline{I_s^i}$ and $\overline{I_{depth}^i}$ the ground truth silhouette and the ground truth depth map respectively at pixel i.

4. Evaluation

We extensively evaluate Softmesh on three rendering tasks, (i) silhouette, (ii) normals and (iii) depth from a 3D point cloud from any given camera position. We use two datasets in our benchmarks to assess our approach on both rigid and non-rigid objects. We further analyze the effect of noise level and input point cloud resolution on the quality of the output rendering.

4.1. Datasets

We evaluate Softmesh on two datasets, one with non-rigid human bodies and the second with rigid objects from humanmade models.

(i) **MPI-FAUST [2]:** The training data is composed of 100 watertight meshes with 6 890 vertices each, corresponding to 10 human bodies, each in 10 different poses. All the shapes in the training set come with vertex level registration. The test data is composed of 200 scans but in different poses.

(ii) **ShapeNet-Part [5]:** ShapeNet-Part is a subset of 13 998 objects from the bigger ShapeNet dataset. It is composed of a set of humanmade models organized across 16 different categories.

4.2. Data Processing

To generate the input point clouds, we randomly sample points on the surface of each object until the given resolution is reached. To make the task more complex, very low resolution point clouds are sampled for each dataset. If not

Table 1. Quantitative comparisons across our Softmesh pipeline, other baseline methods (rows 1 to 6), and ablation on transfer across tasks (rows 7 to 13) on the MPI-FAUST [2] dataset sampled at resolution 512. Performance is evaluated in terms of ℓ_1 difference to the ground truth images for all three tasks (silhouette, normals and depth). mIoU is also measured for the Silhouette task. Check marks indicate which image renderings are available to Softmesh during training.

Id	Method			Silhouette		Normals	Depth
				ℓ_1 ($\times 10^{-2}$) ↓	mIoU (%) ↑	ℓ_1 ($\times 10^{-2}$) ↓	ℓ_1 ($\times 10^{-2}$) ↓
1	KNN10			**1.196**	**91.19**	7.611	**3.706**
2	KNN17			1.508	89.34	7.615	4.446
3	Poisson [17] w/ **N**			1.369	90.08	**7.311**	4.099
4	Poisson [17] w/o **N**			4.513	73.98	13.223	13.112
5	IER Meshing [20]			1.391	89.99	12.005	4.112
6	Softmesh (ours)			1.441	89.84	7.386	4.302
	Silhouette	Normals	Depth	Transferring across tasks			
7	✓			1.389	90.13	9.958	4.337
8		✓		1.670	88.24	7.654	4.945
9			✓	1.606	88.64	7.567	4.779
10	✓	✓		**1.357**	**90.35**	7.339	**4.076**
11		✓	✓	1.604	88.64	7.570	4.766
12	✓		✓	1.407	90.09	**7.314**	4.214
13	✓	✓	✓	1.441	89.84	7.386	4.302

mentioned otherwise, the point clouds used as input to our network are at a resolution of 512 points. To further analyze the effect of resolution, two additional point cloud versions are sampled for each shape at resolutions 1 024 and 2 048. To generate noisy point clouds, each point is (uniform) randomly displaced along the ground truth normal direction, computed using underlying mesh, with a magnitude sampled in the range $[-\nu, \nu]$.

Each mesh is also rendered with different camera poses to provide the 2D supervision signal required at training time. We choose the camera calibration to be the same as our soft rasterizer and rotate the camera around the object at fixed elevation to get 8 equally spaced views. For each object and each view, the silhouette, the rendered normals and rendered depth are saved as ground truth signal.

In all experiments, we consider that the 3D attributes to be rendered is known. For example, to render the normals, we use the ground truth per-point normals that we feed to the renderer. However, we suppose that our network has not access to any of these attributes to learn point connectivity.

For fair comparison, we also estimate 3D point normals directly from the point cloud by jet fitting [4] (using CGAL function) for the Poisson solver. We report Poisson performance both with ground truth normals (Poisson w/ **N**) and with estimated normals (Poisson w/o **N**).

Differently to IER Meshing [20], our pipeline does not require to know the distance ratio geodesic/Euclidean for each edge nor the distance of a face to the ground truth mesh. Only 2D images of the object are required to build and render our soft mesh representation.

4.3. Evaluation metrics

To compare the different methods on the same ground, we use two types of metrics, 2D-based and 3D-based.

2D-based: The output of each method is rendered using the same camera calibration used to generate the ground truth images. For baseline methods, we use the same generic rasterizer as in Section 4.2. For our pipeline, we use the output of our soft-rasterizer but we binarize the silhouette map. All pixels above 0.5 are assigned to 1 and the rest of the pixels set to 0.

We compute and report the ℓ_1 difference between the predicted and the ground truth images obtained from the original mesh. As a second metric, we use the IoU (Intersection over Union) in % between the predicted and ground truth silhouette.

3D-based: To assess the 3D outputs of each method, we measure the distance between the predicted faces and the ground truth mesh. The distance of a face to a (soft) mesh is simplified as the distance of its centroid to the (soft) mesh. The distances are then averaged to get our 3D metric. We report on this metric in the Supplemental material.

4.4. Baselines

In our evaluation, we evaluate our method against three baselines, (i) KNN (with $k = 10$ and $k = 17$), (ii) Poisson [17], w/ or w/o ground truth normals **N** and (iii) IER Meshing [20].

KNN: KNN builds edges between each pair of neighbors and then extracts the faces from the set of edges.

Poisson [17]: Poisson reconstruction, as implemented in CGAL, estimates the implicit function of a shape from an input point cloud and its associated normals. Then, a mesh is generated by extracting an iso-surface from this function.

IER Meshing [20]: For IER Meshing [20], we use the same default parameters as the official implementation from the author.

4.5. Results

We compare our method with the three aforementioned baselines. We also report KNN performance on MPI-FAUST [2] for intermediate k values in Figure 4. We show

Table 2. Performance of KNN, Poisson [17], IER Meshing [20] and Softmesh for silhouette reconstruction on each of the 16 ShapeNet-Part [5] categories sampled at resolution 512. Results are shown as the L1 difference (- the lower the better -) to the ground truth ($\times 10^{-2}$).

Category	Airplane	Bag	Cap	Car	Chair	Earphone	Guitar	Knife	Lamp	Laptop	Motorbike	Mug	Pistol	Rocket	Skate	Table
Nb. Objects	2299	68	50	817	3362	63	709	357	1404	407	176	168	253	58	137	4683
KNN10	1.268	4.958	4.143	3.411	4.940	4.228	1.578	1.309	2.704	5.667	5.146	5.844	2.363	1.056	1.420	4.777
KNN17	1.049	3.674	2.252	**2.704**	4.442	3.611	0.987	0.761	2.272	2.713	4.680	4.094	1.576	0.679	1.315	4.151
Poisson [17] w/ $\mathbf{N}$	5.353	14.213	34.418	13.943	11.755	7.903	6.774	0.976	16.072	11.091	5.853	9.7832	4.238	1.737	6.562	14.444
Poisson [17] w/o $\mathbf{N}$	23.650	14.758	21.419	27.912	32.932	36.810	48.856	43.500	13.980	12.892	43.371	15.695	30.473	15.844	27.754	25.478
IER Meshing [20]	1.480	3.991	2.839	12.119	3.869	**3.047**	1.353	1.114	2.206	4.395	4.909	4.220	2.329	1.750	1.391	**3.323**
Softmesh (ours)	**1.006**	**3.424**	**2.143**	2.836	**3.495**	3.520	**0.941**	**0.713**	**1.971**	**2.360**	**4.364**	**3.404**	**1.537**	**0.652**	**1.204**	3.427

Table 3. Quantitative comparisons across our Softmesh pipeline, other baseline methods on the full ShapeNet-Part [5] dataset sampled at resolution 512.

Tasks		Silhouette		Normals ↓	Depth ↓
		ℓ_1 ($\times 10^{-2}$) ↓	mIoU (%) ↑	ℓ_1 ($\times 10^{-2}$)	ℓ_1 ($\times 10^{-2}$)
KNN10		3.799	76.71	16.243	11.518
KNN17		3.188	79.335	15.606	10.100
Poisson [17] w/ $\mathbf{N}$		11.725	62.083	28.511	33.677
Poisson [17] w/o $\mathbf{N}$		27.745	36.166	53.079	78.244
2D	3D	IER Meshing [20]			
	✓	3.468	78.46	21.664	10.664
2D	3D	Softmesh (ours)			
✓		**2.683**	**83.40**	**14.827**	**8.006**

on Figure 5 the output meshes of our approach and the baseline methods for different input shapes.

MPI-FAUST dataset: The performance of Softmesh and the three baselines are reported on Table 1. For some very specific (manually selected) values for k (i.e., $k = 10$), KNN achieves better than all the other baselines. However, as shown on Figure 4, only a very tiny range of k values can give this higher performance and this range also depends on the task. IER Meshing [20] has similar issues as it requires the user to define the intrinsic-extrinsic ratio threshold τ to classify a face as correct or incorrect. With the default parameters, IER Meshing [20], even with access to ground truth connectivity during training, is on par with our method. The choice of τ is not suited for all shapes and thus may not classify correctly the potential faces. Poisson [17] is efficient under the assumption that the ground truth normals are known. Otherwise, the performance degrades significantly (+230% for the silhouette loss). Due to the very different poses contained in the test set, learning based methods are challenged on this dataset.

ShapeNet-Part dataset: We report the performance of our approach compared to the baselines on ShapeNet-Part in Table 3 and further detail the performance per category in Table 2. Softmesh performs consistently better than the baselines for each of the three tasks and for most of the ShapeNet-Part categories. KNN is efficient overall especially for genus 0 shapes such as *Cars* but fails on more complex categories (*Chairs*, *Tables*). IER Meshing [20] is also competitive on few classes but not all classes as the hyperparameters it relies on are class-specific. At such a resolution, as shown in Figure 5, Poisson [17] may fail totally to retrieve the object surface which penalizes its overall performance.

Table 4. Quantitative comparisons across our Softmesh pipeline and other baseline methods on the MPI-FAUST [2] dataset of the effect of input resolution and noise scale ν. Here, Poisson [17] is fed with estimated normals from jet fitting [4]

	KNN10	KNN17	Poisson [17]	IER Meshing [20]	Softmesh
Resolution	Varying resolution				
512	**91.19**	89.34	73.98	89.99	89.84
1024	**94.52**	94.34	82.01	93.65	92.16
2048	96.34	**96.60**	85.77	96.04	96.44
Noise scale	Increasing noise level				
$\nu = 0.000$	**91.19**	89.34	73.98	89.99	89.84
$\nu = 0.005$	**90.87**	89.01	71.97	89.13	89.18
$\nu = 0.010$	**90.18**	88.36	71.20	87.40	88.44
$\nu = 0.020$	**88.07**	86.04	64.91	82.78	86.40
$\nu = 0.050$	**79.85**	77.26	40.84	68.85	77.69

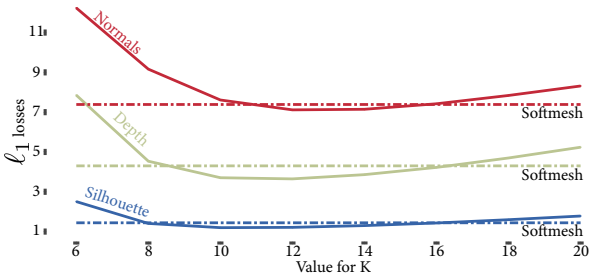

Figure 4. Evolution of KNN performance with different values for K in rendering silhouette, depth and normals on the MPI-FAUST dataset. Choosing the correct value for k requires a complex tuning: KNN can achieve better performance (lower ℓ_1 losses) than Softmesh with some very specific values for k.

Table 5. Quantitative comparisons for Softmesh on Shapenet-Part [5] at resolution 512 when training on a single class (*Single*) or on all classes (*Multi*). Softmesh consistently performs better in the multi category setup.

Category	Airplane		Chair		Table	
Method	Single	Multi	Single	Multi	Single	Multi
Silhouette	1.112	**1.006**	4.518	**3.495**	4.068	**3.427**
Normals	6.571	**5.809**	18.868	**15.723**	19.270	**16.625**
Depth	3.408	**3.058**	12.736	**10.033**	11.704	**10.046**

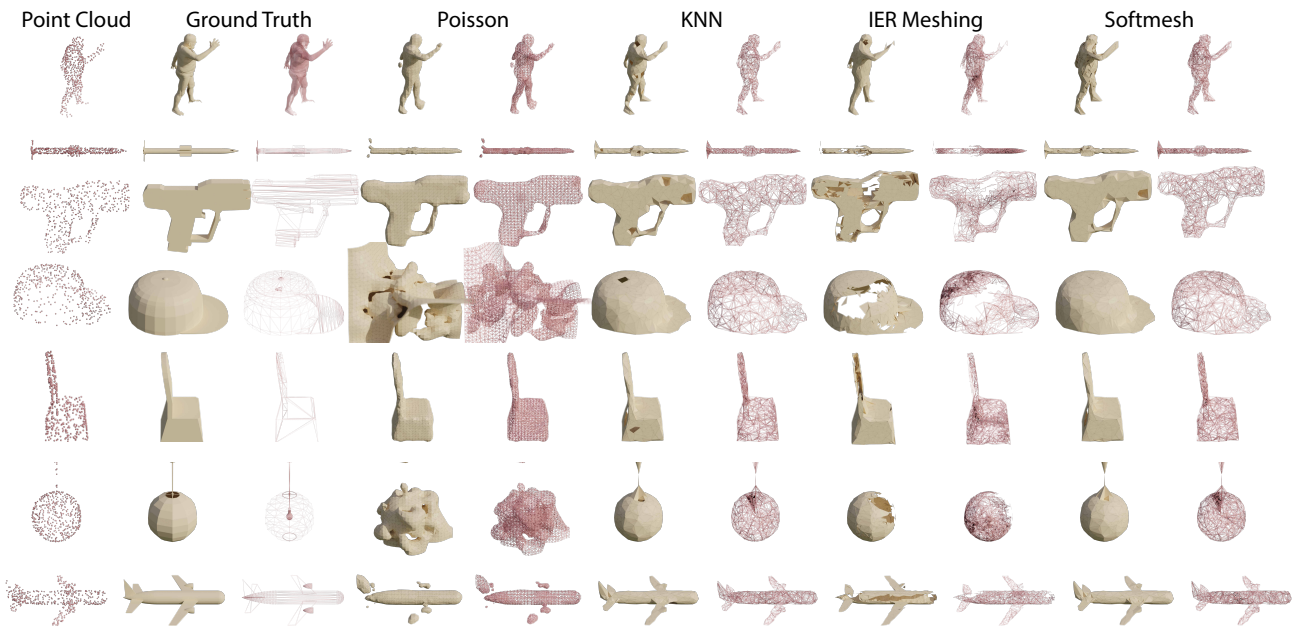

Figure 5. Qualitative comparisons across baseline approaches on both MPI-FAUST [2] and ShapeNet-Part [5] at input resolution 512.

Impact of the noise level: We add noise to all 3D point clouds following the process described in Section 4.2. We vary the scale of the noise ν from 0 to 0.050 and report the silhouette IoU of our method and all baselines in Table 4. We observe that our method is more robust than IER Meshing [20] as a learning based method. IER Meshing [20] is better for noise-free point clouds ($+0.05\%$) but its performance degrades quickly. At noise scale $\nu = 0.005$, our approach becomes better ($+0.05\%$) and the gap increases at very high noise level ($+8.84\%$ at scale $\nu = 0.05$). Poisson [17] is also sensitive to the noise when estimating the normals from the point cloud.

Impact of point cloud resolution: We evaluate how performance changes when increasing the resolution of the input point cloud in Table 4. Increasing the resolution of the input point cloud simplifies the problem as there will be less ambiguous areas in each point cloud. All methods obtain better performance at higher resolution. We notice however that the performance gap between all four methods reduces as the resolution increases. A strong learning pipeline will be even more crucial at low resolution where classical approaches are more challenged.

Transfer across tasks: We report in Table 1 (rows 7-13) the performance of Softmesh when training on a subset of all three rendering tasks. Rows 7-9 show the performance when training the pipeline with a single task. Training with multiple task (rows 10-13) tends to improve the general performance of the pipeline which proves synergy across tasks. As our training disentangles silhouette from other tasks, the network needs to be supervised on silhouette to make sure that the correct silhouette will be used to mask out background pixels.

Transfer across categories: We demonstrate that our approach is able to generalize efficiently across many object categories from ShapeNet-Part [5]. In Table 5, we compare the performance of our approach on three categories, (i) Airplane, (ii) Chair and (iii) Table when training the network on that single category (*Single*) or on all ShapeNet-Part categories at once (*Multi*). We observe the performance consistently improves when trained on multiple categories. By learning the common patterns across multiple categories, the network can efficiently generalize over the dataset to improve its performance on all three rendering tasks.

5. Conclusion

We presented Softmesh, a fully differentiable pipeline that learns a distribution over non manifold-meshes from 3D point clouds. Our approach differs from previous work that requires stronger 3D supervision that can be very costly to acquire. Our pipeline consists of two steps: (i) learning a probabilistic score for a selection of potential faces and (ii) rendering the probabilistic mesh with our customized soft rasterizer. Our approach outperforms classical approach while being on par with 3D-supervision learning based approach for both rigid and non-rigid objects. In future work, we would like to explore potential other applications that could benefit from our Softmesh representation. We also plan to explore impact of pose estimation networks to remove the requirement of known camera poses.

References

[1] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. 2020. 4322

[2] Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Piscataway, NJ, USA, June 2014. IEEE. 4321, 4325, 4326, 4327, 4328

[3] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, page 67–76, New York, NY, USA, 2001. Association for Computing Machinery. 4322

[4] F. Cazals and M. Pouget. Estimating differential quantities using polynomial fitting of osculating jets. In *Symposium on Geometry Processing (SGP)*, 2003. 4326, 4327

[5] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 4321, 4325, 4327, 4328

[6] S. Chen, S. Niu, T. Lan, and B. Liu. Pct: Large-scale 3d point cloud representations via graph inception networks with applications to autonomous driving. In *ICIP*, 2019. 4322

[7] Z.-Q. Cheng, Y.-Z. Wang, B. Li, K. Xu, G. Dang, and S.-Y. Jin. A survey of methods for moving least squares surfaces. In *Proceedings of the Fifth Eurographics / IEEE VGTC Conference on Point-Based Graphics*, SPBG'08, page 9–23, Goslar, DEU, 2008. Eurographics Association. 4321

[8] Yaodong Cui, Ren Chen, Wenbo Chu, Long Chen, Daxin Tian, and Dongpu Cao. Deep learning for image and point cloud fusion in autonomous driving: A review. *ArXiv*, abs/2004.05224, 2020. 4321

[9] 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*, SIGGRAPH '96, page 303–312, New York, NY, USA, 1996. Association for Computing Machinery. 4321

[10] 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*, SIGGRAPH '96, page 303–312, New York, NY, USA, 1996. Association for Computing Machinery. 4322

[11] Angela Dai and Matthias Nießner. Scan2mesh: From unstructured range scans to 3d meshes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2019. 4321, 4322

[12] Lin Gao, Jie Yang, Tong Wu, Yu-Jie Yuan, Hongbo Fu, Yu-Kun Lai, and Hao Zhang. Sdm-net. *ACM Transactions on Graphics (TOG)*, 38:1 – 15, 2019. 4322

[13] Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J. Mitra. PCPNet: Learning local shape properties from raw point clouds. *CGF*, 37(2):75–85, 2018. 4322

[14] Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. Point2mesh: A self-prior for deformable meshes. *ACM Trans. Graph.*, 39(4), 2020. 4322

[15] Tong He, Haibin Huang, Li Yi, Yuqian Zhou, Chihao Wu, Jue Wang, and Stefano Soatto. Geonet: Deep geodesic networks for point cloud analysis. *CoRR*, abs/1901.00680, 2019. 4322

[16] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *SIGGRAPH Comput. Graph.*, 26(2):71–78, July 1992. 4322

[17] Michael M. Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32(3):29:1–29:13, 2013. 4321, 4322, 4326, 4327, 4328

[18] Lubor Ladicky, Olivier Saurer, SoHyeon Jeong, Fabio Maninchedda, and Marc Pollefeys. From point clouds to mesh using regression. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 3913–3922. IEEE Computer Society, 2017. 4321

[19] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *The IEEE International Conference on Computer Vision (ICCV)*, 2019. 4322

[20] Minghua Liu, Xiaoshuai Zhang, and Hao Su. Meshing point clouds with predicted intrinsic-extrinsic ratio guidance. *arXiv preprint arXiv:2007.09267*, 2020. 4322, 4326, 4327, 4328

[21] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019. 4321, 4322, 4324

[22] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. *CoRR*, abs/1904.07601, 2019. 4321

[23] Eric-Tuan Lê, Iasonas Kokkinos, and Niloy J. Mitra. Going deeper with lean point networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 4322, 4323

[24] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 4322

[25] M. Pauly, N. J. Mitra, and L. Guibas. Uncertainty and variability in point cloud surface data. In *Symposium on Point-Based Graphics*, pages 77–84, 2004. 4322

[26] 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 (CVPR)*, July 2017. 4321, 4322

[27] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, pages 5099–5108, 2017. 4322

[28] Guocheng Qian, Abdulellah Abualshour, Guohao Li, Ali Thabet, and Bernard Ghanem. Pu-gcn: Point cloud upsampling via graph convolutional network, 2019. 4322

[29] Marie-Julie Rakotosaona, Paul Guerrero, Noam Aigerman, Niloy Mitra, and Maks Ovsjanikov. Learning delaunay surface elements for mesh reconstruction, 2020. 4322

[30] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 4321

[31] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016. 4321

[32] Jiayun Wang, Rudrasis Chakraborty, and Stella X. Yu. Spatial transformer for 3d points. *CoRR*, abs/1906.10887, 2019. 4322

[33] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019. 4322

[34] Francis Williams, Teseo Schneider, Cláudio T. Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. Deep geometric prior for surface reconstruction. *CoRR*, abs/1811.10943, 2018. 4322

[35] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge J. Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. *CoRR*, abs/1906.12320, 2019. 4321, 4322

[36] Jiancheng Yang, Qiang Zhang, Bingbing Ni, Linguo Li, Jinxian Liu, Mengdie Zhou, and Qi Tian. Modeling point clouds with self-attention and gumbel subset sampling, 2019. 4322

[37] Christopher Zach, Thomas Pock, and Horst Bischof. A globally optimal algorithm for robust tv-l^1 range image integration. In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, pages 1–8. IEEE Computer Society, 2007. 4321