PointTriNet: Learned Triangulation of 3D Point Sets – Supplementary Material

Nicholas Sharp and Maks Ovsjanikov

A Loss functions

These loss functions are introduced in the main document. Here we give additional details and implementation notes.

A.1 Overlap loss

This loss is used to penalize triangles which overlap in space. The key ingredient is a spatial kernel, the definition of which is reproduced here as

$$g_t(x) := p_t \max(0, 1 - \frac{d_n(x)}{d_e(x)}), \tag{1}$$

for any point $x \in \mathbb{R}^3$, where p_t is the triangle probability, $d_n(x)$ is the distance in the normal direction from the triangle, $d_e(x)$ is the smallest signed perpendicular distance to the triangle's edges, and we let $g_t(x) := 0$ for points where $d_e(x) \leq 0$.

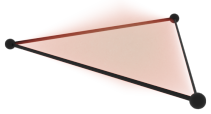

Fig. 1: A volumetric rendering of the overlap kernel from Equation 1.

More precisely, suppose a triangle has vertices (p_0, p_1, p_2) and unit normal n. Then $g_t(x)$ could be computed, for instance, via the expressions

$$d_n(x) := |(x - p_0) \cdot n|, \tag{2}$$

where $\cdot$ denotes the dot product, and

$$d_e(x) := \min \begin{cases} \min\limits_{i \in 0,1,2} (x - p_i) \cdot (\frac{p_{i+1} - p_i}{||p_{i+1} - p_i||} \times n) \\ 0 \end{cases} \tag{3}$$

where the index in p_{i+1} is taken modulo 3, and the outer $\min(\ldots, 0)$ automatically ensures that $g_t(x)$ takes a value of 0 for points which are outside of the triangle when projected in to the triangle plane.

A.2 Watertight loss

In a watertight and manifold mesh, all edges have exactly two incident triangles. Accordingly, our watertightness loss penalizes edges that have a number of incident triangles other than two (that is, just one or more than two incident triangles). Because we work with probabilistic surfaces, the loss is the expected fraction of such non-watertight edges (reproduced here)

$$\mathcal{L}_{\mathrm{W}} := \sum_{ij \in \mathcal{E}} p_{ij}(1 - p_{ij}^{\mathrm{water}}) \tag{4}$$

where $\mathcal{E}$ is the set of edges, p_{ij} denotes the probability that ij appears in the triangulation, and p_{ij}^{water} denotes the probability that ij is watertight. As noted in the main text, this loss only captures connectivity across edges, and thus does not explicitly discourage vertex-nonmanifold triangulations. However, we observe that practice, there are very few configurations which are watertight but not vertex-manifold (essentially just an "hourglass" configuration). Since such configurations are very particular and rare, this watertightness loss seems to be sufficient to encourage vertex-manifold triangulations in practice.

To evaluate this quantity discretely, we decompose it as a sum over halfedges h, where the term *halfedge* refers to a side of a triangle (in a manifold mesh, each of these sides forms *half* of an edge). Each triangle has three halfedges. As always, we will make the approximation that all triangle probabilities are independent.

Recall that p_t denotes the probability associated with triangle t. For any halfedge h, let $\tau(h)$ be the triangle containing h, and $e(h)$ be the edge on which h is incident. For any edge g, let $\mathcal{N}_g$ be the set of halfedges incident on g, and we will say that these halfedges are *neighbors*. For a halfedge h, the probability that there is exactly one other neighbor halfedge incident on the same edge is given by

$$p_h^{1\ \mathrm{other}} := \sum_{\substack{k \in \mathcal{N}_{e(h)} \\ k \neq h}} \left(p_{\tau(k)} \prod_{\substack{j \in \mathcal{N}_{e(h)} \\ j \neq h,k}} 1 - p_{\tau(j)} \right). \tag{5}$$

In this expression, the outer sum is over all halfedges k which could be neighbors of h, and for each it computes the probability that k is present and is the only neighbor of h.

The full loss is then computed as

$$\mathcal{L}_{\mathrm{W}} \leftarrow \frac{\sum_h p_{\tau(h)} * (1 - p_h^{1\ \mathrm{other}})}{\sum_h p_{\tau(h)}} \tag{6}$$

where we normalize the expectation as a fraction of halfedges, because the probabilistic surface does not have an obvious number of edges.

B Comparison to Scan2Mesh

This work is the first to directly consider the triangulation of point sets via machine learning. The most related learning-based approach is the recent Scan2Mesh (see main document for citation). There are significant differences between the problem considered in this work and the problem considered in Scan2Mesh. Most importantly, Scan2Mesh does not attempt to triangulate arbitrary input vertex sets, or scale beyond a few hundred elements. Additionally, Scan2Mesh uses volumetric signed-distance data as input, and is not designed to operate on unstructured point cloud input as considered here. Nonetheless, in the interest of comparison, we construct a network inspired the architecture in Scan2Mesh and apply it to our task.

We will refer to this comparison approach as *Scan2Mesh-like*, to distinguish it from the original work. In the style of the two-phase triangle prediction network in Scan2Mesh, we first form the k-nearest-neighbor graph among input vertices with $k = 16$, and apply a message-passing graph network. This network is identical to the architecture used in Scan2Mesh, except that the inputs are simply the vertex positions, and the output is an edge probability in $[0, 1]$. We then form all possible triangles among the resulting edges, and assign each an initial probability as the product of the three edge values. For the second phase, we construct another graph network among the dual graph of these triangles, and compute per-triangle input features as in Scan2Mesh. This graph network predicts a new probability for each triangle, which is multiplied by the initial triangle probability to generate an output value. Scan2Mesh is trained in part using cross-entropy losses against simplified target meshes; to apply Scan2Mesh-like to our task, we instead apply our unsupervised loss functions, and train with the dataset and methodology described in Section 4 until convergence.

We emphasize that Scan2Mesh-like has many differences from Scan2Mesh, and should not be considered an implementation of that work, merely a similar method inspired by Scan2Mesh. Differences include:

- Scan2Mesh as presented performs dense $|V|\times|V|$ edge prediction. Scan2Mesh-like predicts edges on just the k nearest-neighbors, to enable scaling to vertex sets of size $|V| = 1000$ at the cost of likely missing some edges and triangles.
- Scan2Mesh is trained using a multi-stage, supervised loss function, while Scan2Mesh-like is trained using the probabilistic loss functions and point dataset presented in this work.
- The triangulation networks in Scan2Mesh access data on a volumetric grid, while Scan2Mesh-like operates solely on points.

Table 1 gives the result of evaluating the trained Scan2Mesh-like on our dataset, and Figure 2 visualizes some samples. The task and architecture of Scan2Mesh-like have several differences from Scan2Mesh, but this experiment serves as some basic evidence that the iterative, PointNet-based architecture presented in this work can outperform the dual-graph prediction network from Scan2Mesh.

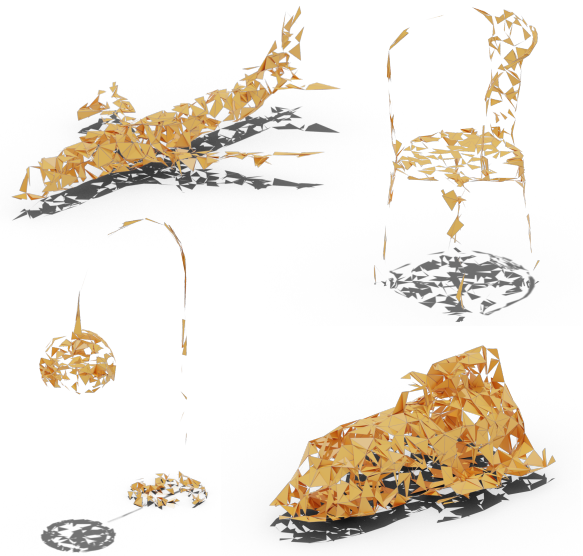

Fig. 2: Meshes resulting from the Scan2Mesh-like network, corresponding to Figure 4 from the main document. Attempting to generate watertight, manifold meshes yields incomplete triangulations. One cause is that in an attempt to scale Scan2Mesh to 1000s of input vertices, we consider only triangles formed among the 16 nearest-neighbor vertices, which omits important triangles (using $k > 16$ is prohibitively expensive). In our method, the proposal network solves this issue by scalably generating good candidates.

Table 1: Evaluation of the Scan2Mesh-like network for the triangulation task on uniformly-sampled ShapeNet.

	Chamfer $\times 100$	watertight	manifold
ours	**0.7417**	**77.0%**	**97.4%**
scan2mesh-like	1.1887	23.3%	96.5%

C Ablation Study

We justify the components of our approach via an ablation study (Table 2). The `no_tris` variant omits the neighboring triangles in the classification network. The `no_proposal` variant omits the proposal network, and instead samples new neighbors heuristically, preferring close neighbors on the correct side of each edge. The `no_overlap` variant omits the overlap loss, setting $\lambda_O = 0$. We find that both the neighboring triangles and the proposal network are critical for high-quality connectivity in the results, removing either significantly degrades the watertightness. The overlap term contributes a small improvement to all metrics.

Table 2: An ablation study over the components of our method on ShapeNet. Both the proposal network and neighboring triangles improve watertightness.

	Chamfer	watertight	manifold
full method	**0.7417**	**77.0%**	**97.4%**
`no_tris`	0.8310	64.6%	95.7%
`no_proposal`	0.7748	66.4%	97.1%
`no_overlap`	0.7489	76.0%	96.8%

D Extended results

In Table 3 below, we give per-class results for the evaluation of our method over uniformly-sampled ShapeNetCore, as presented in Table 1 of the main document. Our method is purely local and geometric, and does not rely on class-specific features; we always train and test on all classes simultaneously. Evaluation statistics are shown per-class here only for the sake of analysis.

table

	Chamfer	watertight	manifold
ours	0.8536	76.3%	97.4%
ballpivot	1.4081	**82.5%**	**100.0%**
alpha3	1.1418	50.5%	61.8%
alpha5	**0.7649**	48.2%	53.3%

chair

	Chamfer	watertight	manifold
ours	**0.7716**	76.9%	97.5%
ballpivot	1.4477	**82.9%**	**100.0%**
alpha3	1.1966	53.4%	65.6%
alpha5	0.9103	54.8%	60.2%

airplane

	Chamfer	watertight	manifold
ours	**0.5475**	75.4%	97.3%
ballpivot	0.6295	**93.9%**	**100.0%**
alpha3	0.6594	48.0%	53.8%
alpha5	1.0095	45.5%	50.4%

car

	Chamfer	watertight	manifold
ours	**1.0552**	73.3%	97.0%
ballpivot	1.5956	**78.7%**	**100.0%**
alpha3	1.3668	52.8%	64.5%
alpha5	1.3501	49.8%	55.1%

sofa

	Chamfer	watertight	manifold
ours	**0.8276**	**77.6%**	97.4%
ballpivot	1.9450	75.8%	**100.0%**
alpha3	1.5633	51.0%	67.6%
alpha5	1.3046	52.9%	58.5%

rifle

	Chamfer	watertight	manifold
ours	**0.4703**	75.7%	97.3%
ballpivot	0.5567	**98.5%**	**100.0%**
alpha3	0.5510	49.2%	54.9%
alpha5	0.7479	44.2%	49.6%

lamp

	Chamfer	watertight	manifold
ours	**0.5009**	79.9%	97.9%
ballpivot	0.8036	**92.0%**	**100.0%**
alpha3	0.6486	43.8%	52.0%
alpha5	0.7780	41.7%	47.1%

watercraft

	Chamfer	watertight	manifold
ours	**0.5996**	76.6%	97.4%
ballpivot	0.7777	**91.5%**	**100.0%**
alpha3	0.7643	49.3%	56.3%
alpha5	1.0656	43.0%	48.3%

bench

	Chamfer	watertight	manifold
ours	**0.8119**	74.2%	97.2%
ballpivot	1.0239	**87.6%**	**100.0%**
alpha3	0.8777	51.7%	59.7%
alpha5	0.8638	48.9%	53.9%

speaker

	Chamfer	watertight	manifold
ours	**0.7806**	**79.4%**	97.6%
ballpivot	2.1313	74.1%	**100.0%**
alpha3	1.5941	48.2%	67.1%
alpha5	0.9852	45.5%	51.8%

Table 3: Extended results from Table 1 of the main document, reported by class over the 10 most common classes. Chamfer values are upscaled $\times 100$ for display.

Triangle quality can have numerical implications for subsequent applications; extremely acute or obtuse triangles may increase approximation error or lead to poor conditioning of optimization problems. In Figure 3, we analyze the triangle quality in the meshes resulting from various reconstruction methods. The results are generally similar, though we observe that ball pivoting has the least prevalence of very acute triangles with angles < 10 degrees. We also recall that widely-used marching cubes reconstruction likewise can yield many acute sliver triangles.

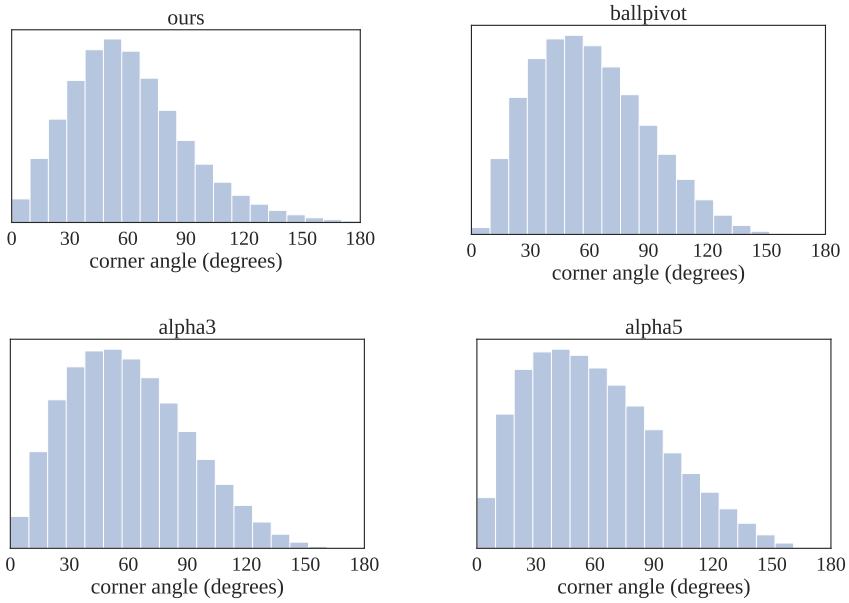

Fig. 3: Histograms of corner angles in all triangles resulting from various reconstruction schemes on sampled ShapeNet, corresponding to Table 1 from the main document.