Meshing Point Clouds with Predicted Intrinsic-Extrinsic Ratio Guidance Supplementary Materials

Minghua Liu, Xiaoshuai Zhang, and Hao Su

University of California, San Diego
{minghua,zxs,haosu}@ucsd.edu

Table of Contents

Appendix A: Pseudo Code of the Remeshing Algorithm

Algorithm 1: Remeshing with Intrinsic-Extrinsic Ratio as Guidance

input : Reference mesh M_R, point cloud P sampled on M_R, IER threshold τ

$M_N.V = P$;
$M_N.F = \emptyset$;
Construct a k-NN graph on P and propose candidate triangle faces;
Calculate the intrinsic-extrinsic ratio for each candidate;
Filter out the incorrect candidates with IER $\geq \tau$;
Sort the remaining candidates with respect to their distance to M_R and the length of their longest edges;
for *each remaining candidate triangle* f_i **do**
  if $\exists$ $f_j \in M_N.F$ *intersects with* f_i ***or*** $\exists$ *edge* $\in M_N$ *has more than two incident faces after adding* f_i **then**
    continue;
  end
  else
    $M_N.F = M_N.F \cup \{f_i\}$;
  end
end
return M_N;

Appendix B: More figures of the Reconstructed Meshes

Fig. 1: Reconstructed meshes of the ShapeNet test set. In each pair, the above one is the result of our method, and the below one is the result of the traditional ball-pivoting algorithm.

Appendix C: Confusion Matrix of the Candidate Classification

Table 1 shows the confusion matrix of the candidate classification on the ShapeNet test set. Specifically, category 0 indicates the incorrect triangles filtered out by the IER. Both category 1 and 2 are the correct candidates. The candidates of category 1 are closer to the ground truth surface than candidates of category 2. We find that the overall performance of the candidate classification is satisfactory.

Table 1: Confusion matrix of the candidate classification on the ShapeNet test set.

	category 0	category 1	category 2
category 0	89.8%	3.9 %	6.3%
category 1	1.4%	96.5%	2.1%
category 2	20.2%	8.8%	71.0%

Appendix D: Results of Different Sampling Density

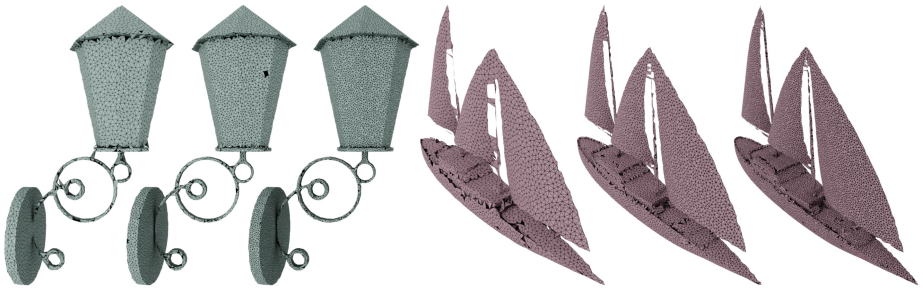

Fig. 2: Qualitative results of different sampling densities. Training on point clouds with 12,800 points (middle), our method can transfer to point clouds with 6,400 (left) and 25,600 points (right).

Table 2: Quantitative results on point clouds with different densities (F-score with two thresholds, Chamfer distance, and normal consistency score). The results are averaged across the eight categories.

#points	F-score(μ) ↑	F-score(2μ) ↑	CD ($\times 100$) ↓	normal ↑
6,400	0.814	0.916	0.091	0.949
12,800	0.872	0.959	0.071	0.962
25,600	0.907	0.983	0.062	0.969

To evaluate the transferability of our method across different sampling density. We trained our models on point clouds with $12,800$ points and test it on point clouds with $6,400$ and $25,600$ points respectively. Fig. 2 shows the qualitative results, from which we find that our method can generalize to different density

distribution, and as the resolution of the point clouds increases, the details become more accurate. The quantitative results shown in Table 2 further confirm our arguments.

Appendix E: Results on Virtual Scans

In order to examine the robustness of our method with regard to different distributions of the input point clouds, we test our method on virtual scans of the ShapeNet models. Specifically, we utilize the VCG Lib to mimic the Kinect sensors and randomly select 10 camera poses to scan the models. The scanned point clouds of different views are fused and downsampled to 12,800 points (Poisson-disk sampling) before feeding into our method.

Fig. 3 shows the point clouds and the reconstructed meshes. Although the scanned points clouds are unevenly distributed, our method still reconstructs high-quality meshes, which demonstrates the generalizability of our method. Please note that due to the limited number of views, the scanned point clouds may not cover the full area of the shape and the reconstructed meshes are thus also incomplete.

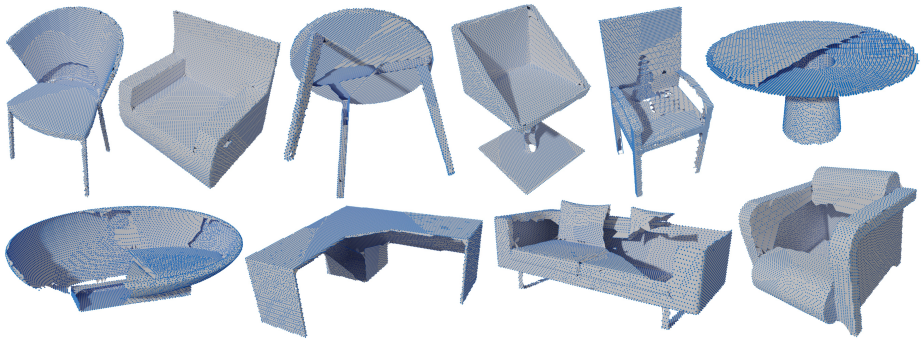

Fig. 3: Input point clouds with 12,800 points and the reconstructed meshes by our method.

Appendix F: Hyper-parameters of the Experiments

Some baseline algorithms require the normals of the point clouds as input. However, meshes in ShapeNet are not perfect manifolds, there are lots of flipped faces and faces that are visible from multiple views. As a result, it's not trivial to determine the consistent directions of the normals (point inward or point outward). We thus employ PCPNet to predict normals for the input point clouds. We then utilize MeshLab to reconstruct the meshes for the three traditional methods. We basically follow the default hyper-parameters of MeshLab. Specifically, the reconstruction depth of PSR is set to be 8, and the grid resolution of Marching

Cube (APSS) is set to be 200. Since ball-pivoting algorithms are sensitive to the radius, we tried the auto-guess mode of the MeshLab and also manually selected 3 radii $1\%, 2\%$, and 3% to choose the best radius. For PSR, we also provide outlier removal as post-processing, which filters out all the vertices that cannot find a point in the input point cloud within a radius of 0.02.

For all the learning-based methods, we used our training set (point clouds with 12,800 points) and followed their released hyperparameters to retrain the model. For DeepSDF, we retrained the network category-by-category. Since both positive and negative signed distance samples are required by the DeepSDF, we assume the normal direction is known for each point in order to sample testing signed distance samples. For AtlasNet, we use the "Autoencoder 25 Squares" model. For Deep Geometric Prior (DGP), "radius" is set to be 0.05, "local-epochs" and "global-epochs" is set to be 125, and "upsamples-per-patch" is set to be 64. Since DGP outputs point clouds of millions of points and reconstructing meshes on such point clouds is time-consuming, we directly use the generated points to calculate the F-score and Chamfer distance, and do not report the normal consistency score.

For the noise experiments, we test on the point clouds of concentric dual spheres. The diameters of the two spheres are 0.933 and 1 respectively. We add a Gaussian noise of a standard deviation of $0.001 \times t$ to point coordinates, where t indicates the level of the noise. For the figure of the main paper, t is set to be 0, 0.8, 1.6, and 3.2 respectively.

Appendix G: Geodesic Distance Regression

In our method, we use the ratio of the geodesic distance and Euclidean distance as guidance to train the network to classify the candidate triangles. Another straightforward idea is that regressing the geodesic distances between pairs of vertices directly with methods such as GeoNet and then use the regressed geodesic distance to classify the candidates. We tried this ablated version to estimate its effectiveness. Specifically, since GeoNet didn't release their source code, we modify our classification network to regress the geodesic distance directly. As we only care about the geodesic distance within a small neighborhood, the training set only contains the pairs between each vertex and its k-nearest neighbors, and the distances are truncated with a threshold of 0.1.

Table 3: Results of the geodesic distance regression. The first row shows the relative error of the predicted distance. The second row shows the accuracy of the candidate classification using the predicted geodesic distance.

category	airplane	cabinet	chair	display	lamp	sofa	table	vessel	average
relative error	137.7%	44.0%	59.8%	47.6%	132.0%	47.5%	49.1%	91.7%	70.7%
accuracy	58.0%	83.2%	61.9%	70.1%	47.4%	71.1%	73.3%	51.6%	65.7%

Table 3 shows the results of our regression version. We find that it's not easy for the network to regress the geodesic distances and the predicted distances

are not accurate. Using the predicted geodesic distance to classify the candidate triangles (into 2 categories) also produces poor results. The accuracy of 65.7% is much lower than the accuracy of our original version of 91.8%. In fact, in our original version, the network does not need to regress the geodesic distance. The labels inferred by the ratio of the geodesic distance and the Euclidean distance only serve as the supervision for the network to learn some local priors to recognize those incorrect triangles, which is a more reasonable and easy task.