Deep Hybrid Self-Prior for Full 3D Mesh Generation
Supplemental Material

Xingkui Wei[1*] Zhengqing Chen[1*] Yanwei Fu[1†] Zhaopeng Cui[2] Yinda Zhang[3†]
[1] Fudan University [2] Zhejiang University [3] Google

In this supplementary material, we show details about the network architecture, more qualitative results, comparison with marching cube based methods, failure cases, qualitative comparison with sparse inputs, more comparison to baseline methods, and ablation study.

A. Implementation Details

In this section, we introduce detailed network architecture for our 2D- and 3D-prior network. Note that both network is initialized with random weights and trained to reconstruct dense structures from randomly initialized input features with the sparse supervisions. Different values of hyperparameters (optimization steps, std. dev. of initialization and loss weights) have been tried, and the method works for most settings. In practice we choose the hyperpaprameters for the best trade-off between accuracy and efficiency.

A.1. 3D-prior Network

We use a MeshCNN [2] based 3D-Prior network introduced in Point2Mesh [3] to generate an initial mesh and the refined output mesh. We also adopt residual and skip connections in MeshConv [2] layers which compose a residual block. ReLU is used as the active function after each MeshConv layer except for the last layer. The network receives an $n_e \times 2 \times 3$ dimensional initial random vector z as input where n_e is the number of input edges , and the network outputs an edge feature vector ΔE with the same dimension that represents the displacement of two vertices on each side of the edges. In each refinement iteration, the 3D-prior network is optimized for 2000 steps with a learning rate of 1e-3, and the weight of the edge loss is 0.2.

In Figure A, we show the detailed architecture of our 3D-prior network. The whole network includes six residual blocks, two MeshPool layers and two MeshUnpool layers. Each residual block contains three MeshConvs. Input and output channel number of each residual block and the pool proportion of each MeshPool layer are shown in Figure A.

A.2. 2D-prior Network

We follow DIP [8] to design our 2D-prior network. An encoder-decoder architecture with several skip-connections is adopted for all of our experiments with same hyperparameters except for training steps. The number of training steps is 2000 for dense color texture map generation, while for dense XYZ map generation the number is 4000 with a learning rate of 1e-2. LeakyReLU [4] is used as the active function. The downsampling operation in the network is implemented as convolution with strides, and for upsampling we use bilinear upsampling. In each convolution layer, a reflection padding is used instead of zero padding. The input random feature map and the output dense UV map have the same spatial resolution 1024×1024.

In Figure B we provide the details of our 2D-prior network architecture. The whole network contains five downscale convolution blocks, five upscale convolution blocks and five skip connection blocks. The layers and parameters of each block are shown in the right part of Figure B.

A.3. UV Flattening

We use OptCuts[6] to create a UV atlas from the 3D mesh. The UV flattening via OptCuts may not be ideal and would affect the 2D prior network output, but most of artifacts can be fixed by the 3D prior network with strong supervision and regularization, and UV map will be regenerated with improved geometry afterward. Therefore, we find our method doesn't need perfect UV-maps at the beginning. UV flattening is challenging for complex geometry, but we only need the UV space to preserve some local smoothness regardless of other criteria, such as distortion and seam lengths.

B. More Qualitative Results

In this section, we show more qualitative results in Fig. C and Fig. D. In Fig. 4 of main paper, we showed comparison to Poisson [5] and Point2Mesh [3] on a few examples.

*Equal contribution
†Corresponding author

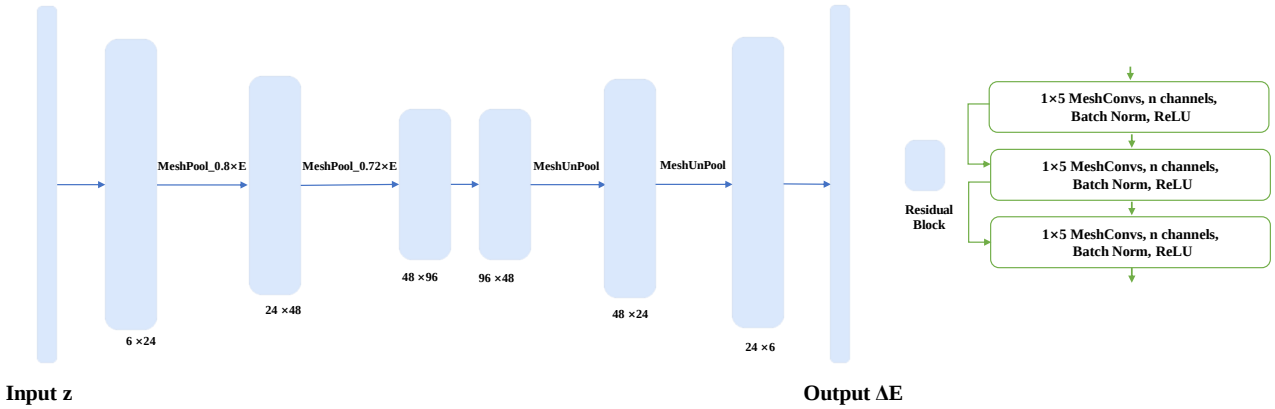


Figure A. Architecture of 3D-prior network. In general, it consists of six residual blocks. The detailed structure and channels of the residual blocks are shown in the left.

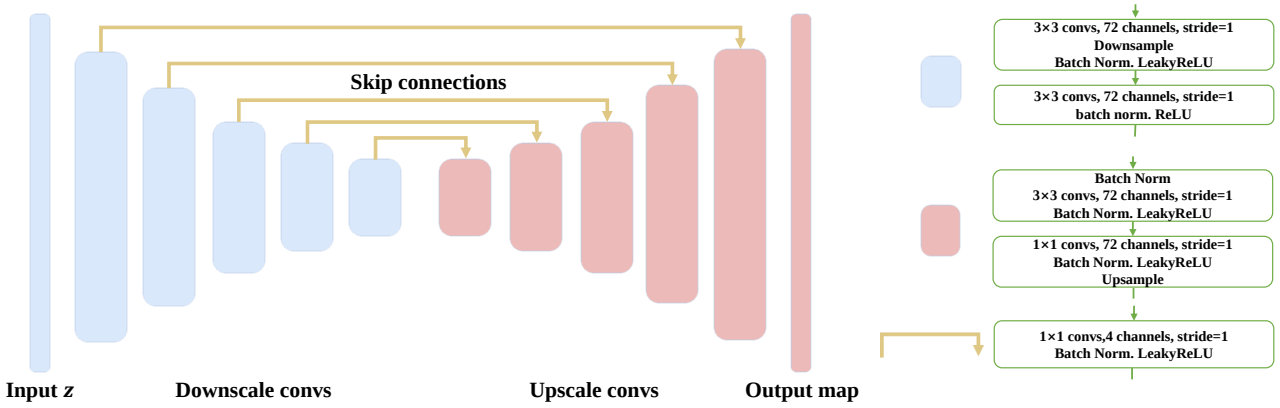


Figure B. Architecture of 2D-prior network. The layers of each component are shown in the right.

Here we add comparison to more previous works in Fig. C. For each example in Fig. D, we show the input colored point cloud and the results from Screen Poisson Surface Reconstruction [5] (Poisson), Point2Mesh [3] (P2M), and our model. We show the mesh results with and without texture. We also show comparison on the surface reconstruction task on the samples with complicate structures in Fig. E.

Overall, our model produces shapes and textures that recovers more details and maintains better visual quality. Screen Poisson surface reconstruction tends to make large geometric errors when the points are sparse or the surface normal cannot be estimated accurately, such as the wing of the aeroplane and the chair legs. The texture map is usually blurry compared to our result generated with hybrid priors.

	F-score↑	CD↓
MC-APSS	93.4	0.0650
MC-RIMLS	96.6	0.0597
P2M-S	97.3	0.0589
Ours	**97.7**	**0.0526**

Table A. Comparison between our method and marching cube based methods on synthetic data. The best results are noted by **Bold**. CD is short for Chamfer Distance.

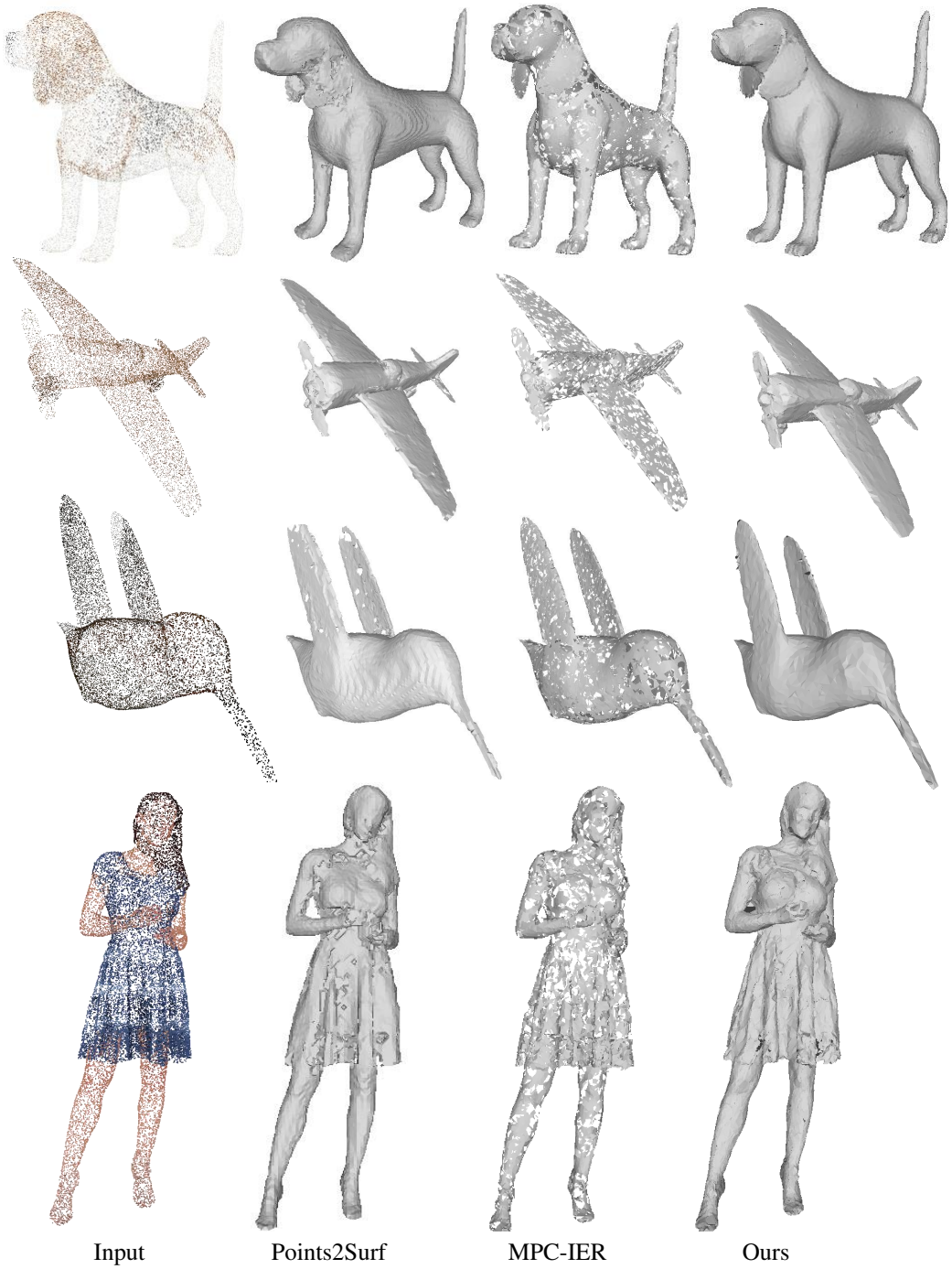
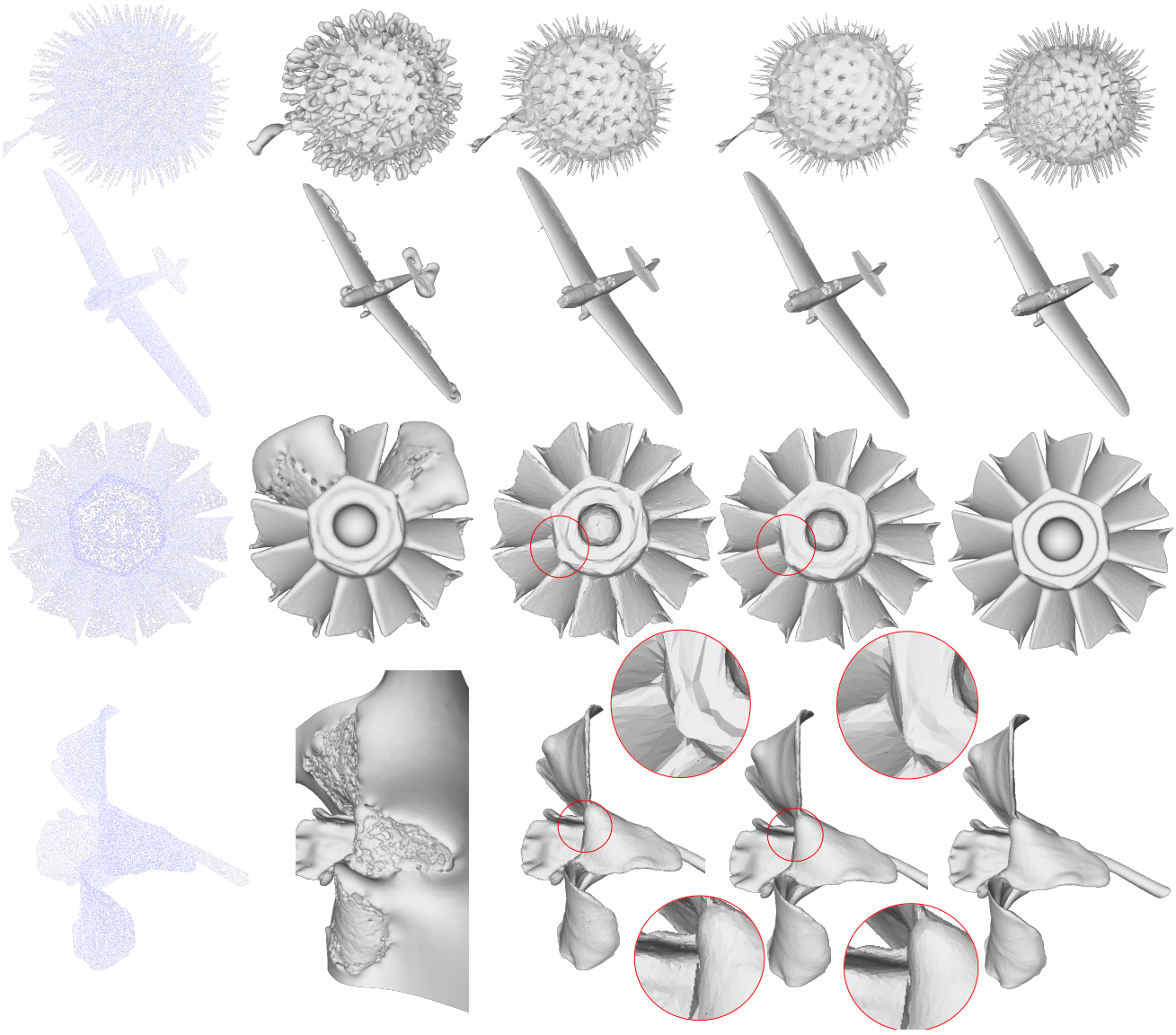


Figure C. Comparison between our method and more surface reconstruction methods on synthetic data.

C. Comparison with More Surface Reconstruction Methods

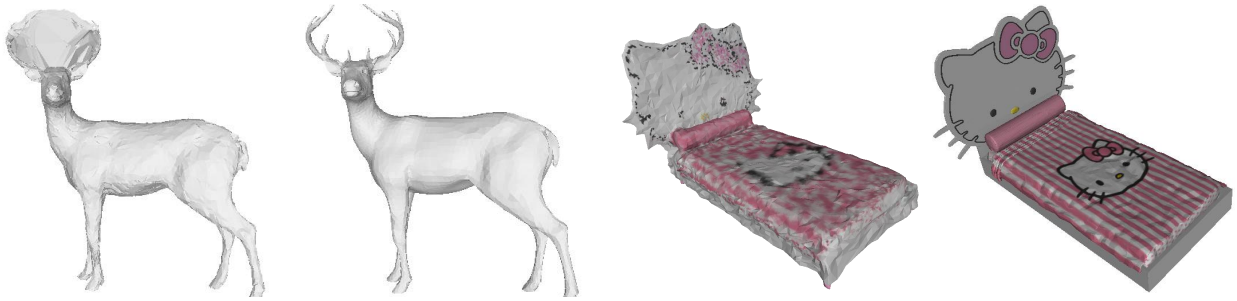
We show the surface reconstruction comparison with two recent marching cube methods implemented in Meshlab, named MC-APSS[1] and MC-RIMLS[7]. We also show the comparison with Point2Mesh [3] with a HC Laplacion smooth [9], namely P2M-S. The Chamfer distance and F-Score are reported in Tab. A, which are worse than our method.

Figure D. Comparison between our method and other surface reconstruction methods. The groundtruth meshes used to sample input point clouds are shown in the first row.



Input Poisson P2M Ours GT

Figure E. Comparison between our method and other methods on surface reconstruction.



Ours GT Ours GT

Figure F. Failure cases in geometry and texture reconstruction.

D. Failure Cases

Fig. F shows some typical failure cases of our method on geometry and texture reconstruction. Some local structures are not separated correctly due to taking convex hull as initialization. Meanwhile, our method is unable to recover high frequency strip texture or tiny patterns with unstructured sparse input colored points.

	2%	5%	10%
Poisson	77.3	53.8	25.9
P2M	70.1	50.4	18.0
Ours	**83.5**	**69.3**	**37.6**

Table B. Comparison on F-score between our method and other surface reconstruction methods with noisy input. The best results are noted by **Bold**.

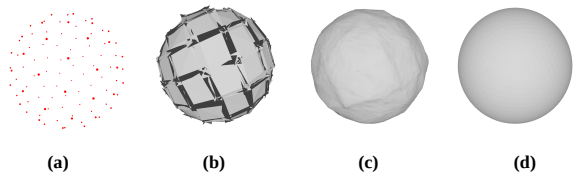


Figure G. Surface reconstruction results of very sparse spherical input. (a) Input point cloud with only 100 points; (b) 3D mesh generated by Point2Mesh; (c) Our result; (d) Ground truth.

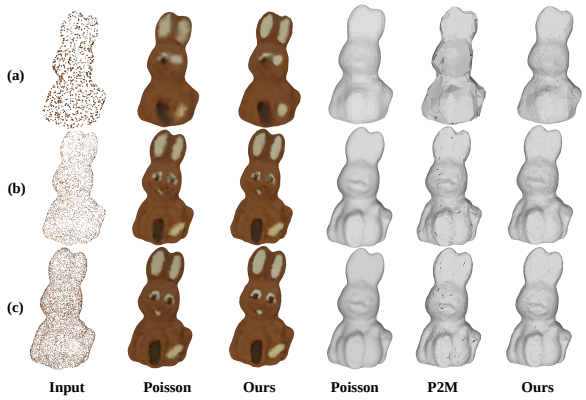


Figure H. Generated textured mesh with different sparseness. (a) 1k input points; (b) 5k input points; (c) 10k input points.

E. Robustness Evaluation

E.1. Robustness against Noise

We test the robustness of our method by manually adding Gaussian noise to the original coordinate value on the input point cloud. Table B shows quantitative results as the standard deviation of noise increased from 2% to 10%. We also conduct experiments that add Gaussian noise with standard deviation as 10% on color. The NIQE results of MeshCNN baseline, Poisson and our method are 22.47, 23.90 and 20.16 (lower is better). Compared to the case without noise (see Sec. 4.4), our method suffers the minimum decline, which indicates comparatively good robustness.

E.2. Robustness against Sparsity

In Fig. G, we show a toy example where only 100 points are sampled from the ball as the input for surface reconstruction in order to understand the advantage of our hybrid 2D-3D prior over the 3D only prior in Point2Mesh. Point2Mesh completely fails since no reasonable prior can be learned by 3D GCN from such an extremely sparse input. In contrast, our method is successful in reconstructing a reasonable ball. We would like to advocate that this is mostly benefit from the strong prior encoded in the XYZ map.

Fig. H shows results with input point clouds of different sparseness on a textured mesh. For Point2Mesh, noisy or large planar surfaces show up quickly when inputs become sparse, while our method still produce smooth surface maintaining roughly correct geometry with certain level of details. With dense input and simple structure, Poisson can generate both good and texture information, while when the input become sparser, it lose more details of texture and geometry compared to our method.

F. Comparison for Texture Generation

Our model produces a high-resolution texture for each mesh, by reconstructing a dense texture map from the sparse UV color map with the 2D-prior network. In this section, we compare our method to a texture generation baseline method which directly predicts color for each point in the MeshCNN [2] framework. On a high-level, this method uses 3D-prior only to recover the texture compared to our method that uses both 2D and 3D prior. The network architecture of the baseline is the same as our 3D-Prior network. Instead of producing the displacement of vertices, the MeshCNN is fed with the predicted mesh shape to build the graph and trained to predict the RGB color of each vertex. The feature on each graph node is random initialized and the loss function is

$$L_{color} = \sum_{\hat{p}} \|C_{\hat{p}} - C_q\|, \tag{1}$$

Figure I. Qualitative comparisons of texture between our method and the baseline method which uses a MeshCNN network to predict vertex color.

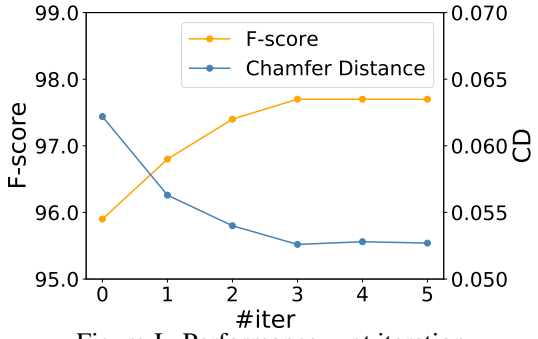


Figure J. Performance w.r.t iteration.

where q is the closest vertex in the input point cloud for each $\hat{p}$, and $C_{\hat{p}}$ and C_q are the color estimated for $\hat{p}$ and observed on q. Note that the $\hat{p}$ is randomly sampled from the predicted mesh as described in Section 3.2 of the paper, and $C_{\hat{p}}$ is calculated by linearly interpolating the predicted color of three vertexes on the corresponding triangle mesh face. Finally, the color on mesh faces is calculated with vertex color via linear interpolation.

The qualitative results are shown in Fig. I. The quality of this baseline results is highly restricted by the vertex number of the predicted triangle mesh. Moreover, it's observed that the baseline method tends to be blurry and lose high frequency information. In contrast, our method always produces texture of high visual quality.

G. Ablation Studies

In this section, we provide more ablation studies of our method.

G.1. Performance w.r.t Iteration

As shown in Fig. J, we report F-score and Chamfer Distance of each iteration to measure the convergence. Typically the numbers stabilize in 3 iterations.

G.2. Effect of Edge Loss in 3D-Prior Network

As mentioned in Section 3.1, we add a loss term to constrain the edge length for the MeshCNN, which speeds up the converging speed. In Figure K (a) (b), we show the output of 3D-prior network w/o or w edge loss under different iteration steps and the final reconstruction mesh w/o or w edge loss in Figure K (c) (d). Under the same iteration, the output mesh from 3D-prior network optimized with edge loss apparently exhibit better geometry, e.g. less holes and folded faces, smoother surface, compared to the case without edge loss . Overall, to achieve similar mesh quality we get in 1000 steps using the edge loss, the network without the edge loss would need at least 4000 steps.

The edge loss can be also calculated very efficiently with known topology thus intrigues negligible computational cost to the optimization. Overall, we can speed up the convergence of MeshCNN for 3-4 times.

G.3. Effect of Gaussian Permutation in 2D-Prior Network

As illustrated in Section 3.2.2 of our main submission, our 2D-Prior network takes as input a random noise feature map $z + \epsilon$, and the ϵ serves as a Gaussian permutation in each training step to prevent the network from overfitting. In this section, we show some qualitative comparison results in Figure L with ϵ sampled from different standard deviations.

As shown in Figure L, a larger permutation makes the result smoother, but also lose high frequency information. On the contrary, the network with smaller permutation tends to be overfitting. In practice, we choose 0.02 as the standard deviation of ϵ for both XYZ map and texture map generation.

References

[1] Gaël Guennebaud, Marcel Germann, and Markus Gross. Dynamic sampling and rendering of algebraic point set surfaces. In *Computer Graphics Forum*, volume 27, pages 653–662. Wiley Online Library, 2008.

[2] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

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

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[5] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013.

[6] Minchen Li, Danny M. Kaufman, Vladimir G. Kim, Justin Solomon, and Alla Sheffer. Optcuts: Joint optimization of surface cuts and parameterization. *ACM Siggraph Asia*, 2018.

[7] A Cengiz Öztireli, Gael Guennebaud, and Markus Gross. Feature preserving point set surfaces based on non-linear kernel regression. In *Computer Graphics Forum*, volume 28, pages 493–501. Wiley Online Library, 2009.

[8] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9446–9454, 2018.

[9] Jörg Vollmer, Robert Mencl, and Heinrich Mueller. Improved laplacian smoothing of noisy surface meshes. In *Computer graphics forum*, volume 18, pages 131–138. Wiley Online Library, 1999.

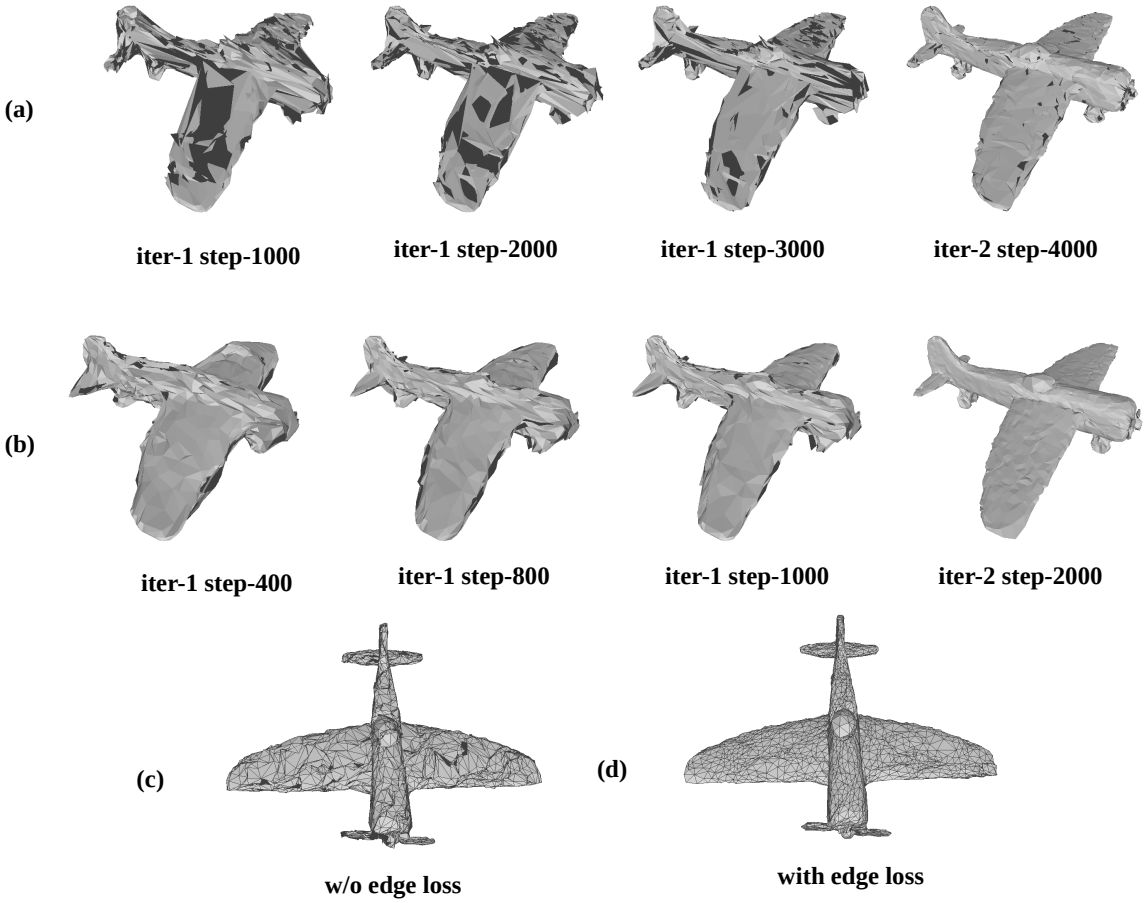


Figure K. Results (a) without edge loss (b) with edge loss under different iteration steps and the final reconstruction mesh (c)without edge loss, (d)with edge loss.

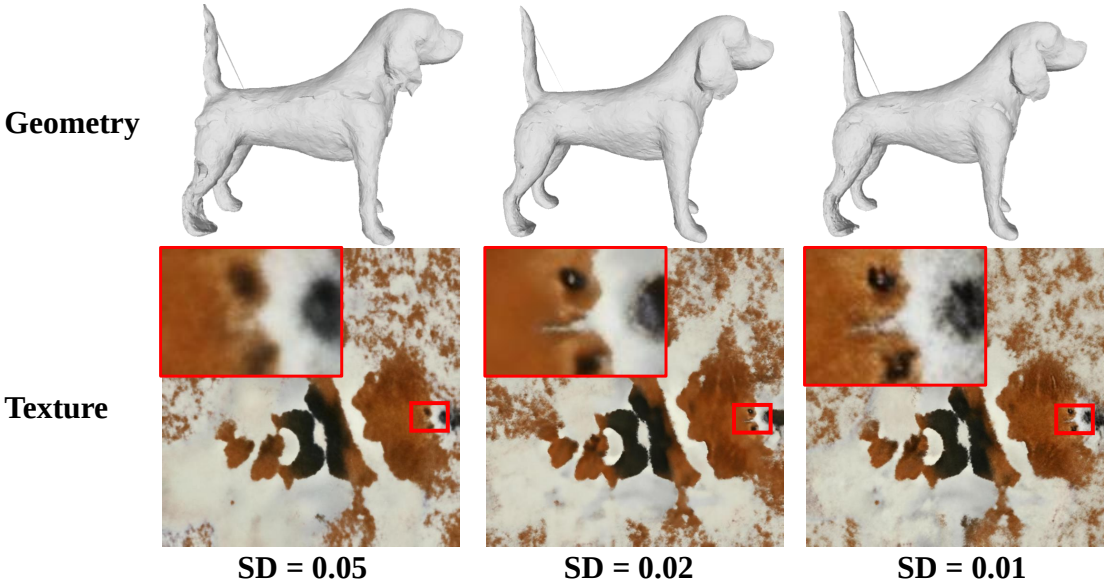


Figure L. Geometry and texture outputs with perturbation ϵ sampled from different standard deviations (SD).