Deep Hybrid Self-Prior for Full 3D Mesh Generation

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

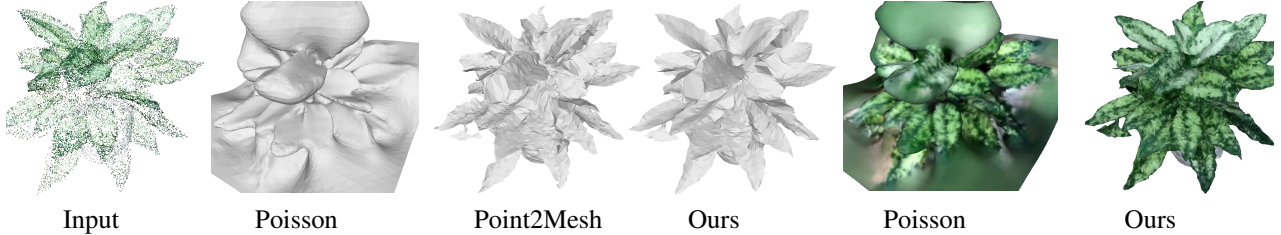| Input | Poisson | Point2Mesh | Ours | Poisson | Ours |

Figure 1. We propose to utilize the deep hybrid 2D-3D self-prior in neural networks to generate the high-quality textured 3D mesh model from the sparse colored point cloud.

Abstract

We present a deep learning pipeline that leverages network self-prior to recover a full 3D model consisting of both a triangular mesh and a texture map from the colored 3D point cloud. Different from previous methods either exploiting 2D self-prior for image editing or 3D self-prior for pure surface reconstruction, we propose to exploit a novel hybrid 2D-3D self-prior in deep neural networks to significantly improve the geometry quality and produce a high-resolution texture map, which is typically missing from the output of commodity-level 3D scanners. In particular, we first generate an initial mesh using a 3D convolutional neural network with 3D self-prior, and then encode both 3D information and color information in the 2D UV atlas, which is further refined by 2D convolutional neural networks with the self-prior. In this way, both 2D and 3D self-priors are utilized for the mesh and texture recovery. Experiments show that, without the need of any additional training data, our method recovers the 3D textured mesh model of high quality from sparse input, and outperforms the state-of-the-art methods in terms of both the geometry and texture quality.

1. Introduction

Textured mesh is one of the most desirable representation for 3D objects, which has been widely used in many applications, such as industrial design and digital entertain-ment because it enables not only the 3D related task like collision detection but also the rendering capability. As a result, the ability to create a full 3D model consisting of both a 3D triangular mesh and a texture map is a long-lasting prob-lem and consistently draws attention. While purely image-based solution exists [25], the 3D scanners with active il-lumination usually provide much more accurate 3D models and are robust against challenging cases, e.g. texture-less regions. Unfortunately, many commodity-level 3D scan-ners, e.g. Artec Eva [1], iReal 2S [3], Einscan Pro [2] etc, only produce colored point clouds as the output, where the object surfaces and texture maps are missing. There are plenty of works that generate a mesh model from a point cloud, but they usually rely on strong assumptions [45, 38], require pre-training on large dataset [8, 41, 18, 11], and do not produce a texture.

In this work, we propose a method for reconstructing a full 3D model, i.e., a textured triangular mesh, from a colored point cloud. This task is highly under-constrained, and thus prior knowledge is extremely important. It is well known that deep learning model is good at learning prior from a large dataset [45, 18, 11], but also at the same time prone to overfitting to the dataset bias. Instead, Ulyanov *et al.* [51] proposed to randomly initialize a convolutional neural network (CNN) to upsample a given image, which used the network structure as a prior without the need of any additional training data. Sharing the similar spirit, we resort to such self-prior naturally encoded in the neural net-work for the full 3D model reconstruction task. As one of the most related prior arts, Hanocka *et al.* [24] proposed to create a mesh, without texture, from a point cloud us-ing a MeshCNN [23] to deform from the convex hull. They

---

*Equal contribution
†Corresponding author
 Project page: https://yqdch.github.io/DHSP3D

found that the graph-based CNN can also learn the self-prior from the input point cloud to reconstruct a 3D mesh with noise suppressed and missing parts filled. Despite significant improvements over previous methods and the capability to handle challenging cases, however, its output quality highly depends on the input noise level and sparsity (See Sec.4.3), and the effect of 3D network self-prior is not phenomenal as its 2D counterpart [51] empirically.

We propose to exploit the hybrid 2D-3D self-prior for full mesh reconstruction. Specifically, we first utilize the 3D MeshCNN [23] to exploit the 3D prior in a similar way as Hanocka *et al.* [24] and generate an initial 3D mesh model. Then we create a UV atlas encoding the 3D location of the points instead of color information, which is then refined by a 2D CNN using the self-prior and used to update the 3D mesh. We find this 2D network is surprisingly more effective, compared to the 3D MeshCNN, in learning self-prior, and can provide valuable regularization in producing high-resolution mesh with delicate details. The 3D-prior and 2D-prior network runs iteratively to refine the 3D mesh model, and extensive experiments show that our model significantly improves the geometry quality.

Besides the triangulated mesh, our method also recovers a high-resolution texture map. While it is not trivial to build such a texture map from colors on the sparse point cloud since the texture maps are usually in much higher resolution than the 3D geometry, e.g., the number of faces, we borrow the help from the 2D self-learned CNN with the belief that the self-prior is stronger and easier to learn on 2D CNN compared to a 3D graph-based convolutional neural network (GCN). Using the same UV atlas generated from the 3D mesh, we train a 2D CNN to recover the color from sparse point, and find appealing texture maps can be generated automatically. The texture map will be iteratively optimized together with the 3D mesh model.

Our contributions can be summarized as follows. First, we propose a deep learning pipeline that reconstructs a full 3D model with both a triangular mesh and a texture map from a sparse colored point cloud by leveraging self-prior from the network. Second, a novel hybrid 2D-3D self-prior is exploited in our pipeline without learning on any extra data for both geometry and texture recovery. Experiments demonstrate that our method outperforms both the traditional and the existing state-of-the-art deep learning based methods, and both 2D prior and 3D prior benefits the full mesh reconstruction.

2. Related work

Traditional Surface Reconstruction methods   There is a long history of reconstructing surfaces from point clouds. Early methods such as Delaunay triangulations [9] and Voronoi diagrams [6] interpolate points by creating a triangular mesh. When there are noises, however, the result-

ing surface is often jagged. As a result, special data pre-processing is usually required to generate a smooth surface.

Mainstream approaches to reconstruct surface are based on implicit function approaches which can be classified into global and local approaches. Global approaches, such as radial basis functions (RBFs) [10], consider all the data at once, and define a scalar function which used for testing if a point is inside or outside the surface. In contrast, local approaches, such as truncated signed distance function (TSDF) [17] and moving least squares (MLS) [7, 36], consider only subsets of nearby points.

The algorithm of (Screened) Poisson surface reconstruction [32, 33] combines the advantages of global and local approaches. It finds an indicator function and uses its gradient field to solve the Poisson equation, and then an iso-surface can be extracted to reconstruct the surface. The reconstructed model is watertight closed and has good surface details, however this approach requires accurate normal orientation, relies on the dense point cloud, and struggles to handle non-watertight cases.

Deep Learning for Full Model Reconstruction   Deep learning provides new opportunities for geometric reconstruction, especially in terms of 3D representation. 3D volumes [15, 21] and point clouds [19, 4, 5] have been prevalent in many early works, which unfortunately are usually restricted to the low resolutions due to the memory constraint. Recently, the implicit representations [40, 45, 29, 14, 18, 11] have been investigated and greatly improve the geometry details, but demand comparatively long inference time due to the sampling and suffer from overfitting on training sets. Liu. *et al.* [38] builds a network to estimate the local connectivity of input points for surface reconstruction, whose performance heavily relies on the quality of inputs. Similar to us, a graph-based CNN is proposed to generate a 3D model directly in a triangulated mesh that is ready to use. A common approach is to deform gradually from an initial shape toward the desired output [23, 53, 13, 31, 46]. with the help of learned guidance.

On the other hand, there has been a lot of efforts [49, 30, 57, 43, 50, 28] on texture generation from 2D images. Nevertheless, very few works focus on learning a texture consistent with the sparse color observed from point clouds, which is still a challenging problem.

Deep Network as a prior   Recently, the *deep image prior* (DIP) [51] has shown its strong ability for self-supervised 2D image reconstruction tasks (eg. image super-resolution, denoising, or inpainting). Follow up works [20, 47, 48] shows the capability of deep prior to capture image statistics for multiple low-level 2D vision tasks such as dehazing, transparency separation, deblur, etc. Inspired by [51], our approach aims to transfer 2D deep priors to 3D geom-
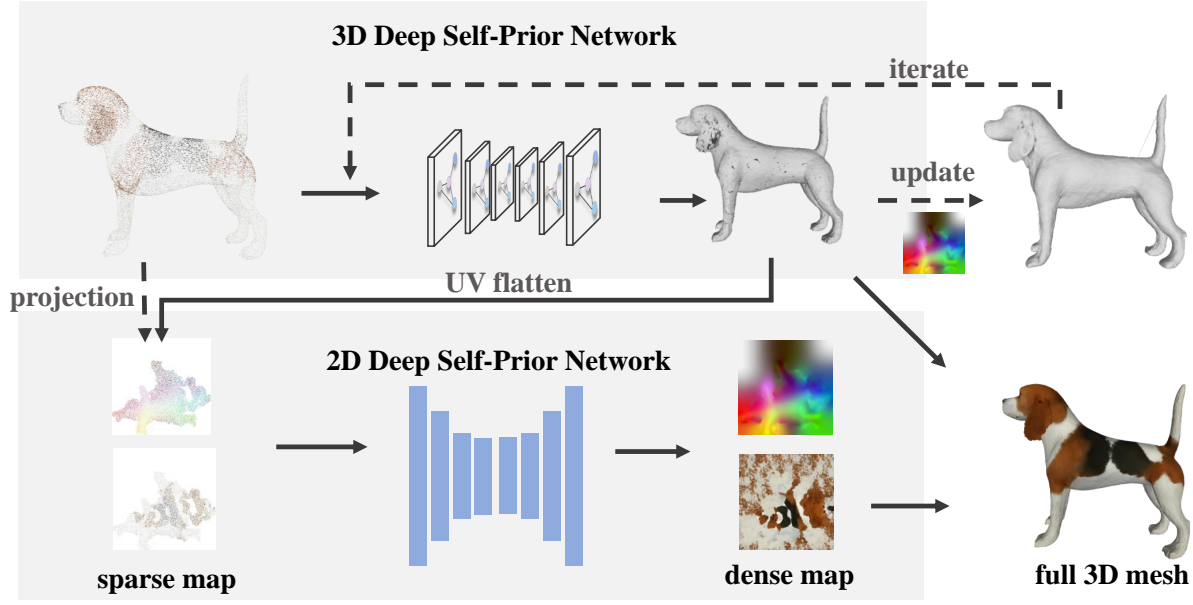
Figure 2. Overview of our model. Our full model contains two building blocks, namely, 3D deep self-prior network, and 2D deep self-prior network, which run iteratively to improve the geometry and texture outputs.

etry space, which enhances the smoothness and robustness of 3D mesh and texture generation.

For 3D surface reconstruction, deep geometric prior is presented in [55], which fits MLPs for different local regions of the point clouds. Point2Mesh [24] reconstructs a surface mesh from an input point cloud by optimizing a CNN-based self-prior deep network [23]. While encoding mesh shape into network parameters, those methods leverage the representational power of the deep network to remove noise. However, the 3D priors of the network from unstructured 3D data is not strong enough. Consequently, the performance is highly relative to the quality of the input point cloud.

3. Full Mesh Generation with Hybrid Prior

Given a colored input point cloud, our goal is to reconstruct the corresponding surface mesh with fine details of both the geometry shape and texture. We design a hybrid-prior network to leverage self-prior in both 3D and 2D space, and an overview of our system is illustrated in Fig. 2. We first run a MeshCNN based 3D self-prior network to reconstruct an initial untextured 3D mesh, then improve the geometry quality and produce a high-resolution texture map in an iterative fashion. Particularly in each iteration, we first build a texture atlas using the current geometry mesh, and then warp the location and color of the input sparse point cloud into texture UV space, which generates a sparse location UV map and a sparse color UV map respectively. Two separate 2D self-prior networks are trained for XYZ re-

finement and RGB generation respectively using the sparse maps as supervision. We use the predicted dense location UV map to update the vertex location in the 3D mesh, which is then fed into the 3D self-prior network for another refinement. This hybrid-prior network runs iteratively until the geometry mesh and texture output is stable.

3.1. 3D Prior Network

Our model starts from a MeshCNN based 3D-Prior network to create an initial mesh from the input point cloud. The main purpose is to build a surface manifold which facilitates the use of 2D prior (Sec. 3.2.2).

Similar to Point2Mesh [24], given an initial point cloud, a convex hull is generated as an initial mesh which will be deformed to the target shape. A graph is built on the edges of the initial convex hull, on which a MeshCNN can be run to produce the displacement of vertices that updates the 3D shape. The feature on each graph node, which corresponds to an edge in the mesh, is assigned as a random vector sampled from a Gaussian distribution, and the MeshCNN is trained to deform the mesh by minimizing the Chamfer distance to the input point cloud. Once converged, the mesh topology is updated by reconstructing a new watertight mesh with more vertices using methods in [27], which is then refined again using MeshCNN.

We apply Chamfer distance loss (Eq. 1) and the edge length regularization (Eq. 2) for the 3D-Prior network. The loss function is defined as $L = \lambda_0 L_{chamfer} + \lambda_1 L_{edge}$, where λ_0 and λ_1 balance the loss term and are empirically

set to 1.0 and 0.2 respectively in our experiment. $L_{chamfer}$ and L_{edge} are calculated with following equations:

$$L_{chamfer} = \sum_{\hat{p}} \min_{q} \|\hat{p} - q\|_2^2 + \sum_{q} \min_{\hat{p}} \|\hat{p} - q\|_2^2, \quad (1)$$

$$L_{edge} = \sum_{p} \sum_{k \in \mathcal{N}(p)} \|p - k\|_2^2, \quad (2)$$

where p is a vertex of the generated mesh; $\hat{p}$ is a point sampled from the generated mesh surface; q is a point in the input point cloud; $\mathcal{N}(p)$ is the one-ring neighboring vertex of p in the generated mesh. Note that the loss function is slightly different from that in Point2Mesh. We find the beam-gap loss in Point2Mesh is computational expensive with limited effect for our system. Meanwhile, adding the regularization term for edge length speeds up the convergence of our network in practice.

3.2. 2D Prior Network

We then refine the mesh via a 2D prior network, The core idea is to create a 2D representation of the 3D mesh, which can be refined effectively with strong self-prior in 2D CNN.

3.2.1 Creating XYZ Map

In this section, we introduce how to create a 2D representation of the 3D mesh which can be refined via 2D CNN. We first create a UV atlas from the initial mesh using OptCuts [37], while theoretically any atlas generation method respecting UV space continuity would also work and more discussions are provided in the supplemental material. Then, for each point q in the input point cloud, we find its nearest $\hat{p}$ on the initial mesh. We then query the triangle face ID and barycentric coordinate within the triangle for $\hat{p}$ to calculate a uv coordinate in the texture atlas, at which the (x, y, z) of q is assigned. Eventually, a sparse three-dimensional XYZ map in the UV space is created to record the 3D locations of all the points in the input point cloud as shown in Fig. 2. We also try to build a distance UV map w.r.t. an anchor point but find it less effective than the XYZ map.

3.2.2 XYZ Map Refinement

We then train a network to produce a dense XYZ map supervised by the sparse one. We adopt a 2D U-Net with skip-connections following DIP [51]. The network takes as input a random noise feature map $z + \epsilon$, and the weights of the network are randomly initialized. z is a $32 \times Height \times Width$ random vector sampled from Gaussian distribution $\mathcal{N}(0, 0.1)$, which is fixed during training. ϵ is a small Gaussian permutation $\mathcal{N}(0, 0.02)$ added to z to prevent the network from overfitting, which changes at every forwarding

pass. The network is supervised by the sparse XYZ map using L_2 loss, and produces the dense XYZ map, whose architecture can be found in the supplementary material.

Note that the 2D U-Net predicts (x, y, z) at each integer pixel location on UV atlas, while uv coordinates of the input points are usually floating points. To obtain accurate supervision from the sparse "ground truth" XYZ map, we use the differentiable bilinear sampling to obtain the value from the predicted dense XYZ map on the sub-pixel locations that have ground truth.

With the dense XYZ map from the 2D-prior network, we update the 3D mesh directly by updating the vertex locations to the value in the predicted dense XYZ map. Fig. 3 shows an example. Compared to the initial mesh (Fig.3 (a)), the 2D-prior refined mesh (Fig.3 (b)) is smoother with a few spikes, and folded regions are fixed.

3.3. Iterative Refinement with 2D and 3D Priors

The refinement via the XYZ map significantly improves the majority part of the 3D geometry and enforces the smoothness. However, some obvious artifacts show up, which typically happens on the locations mapped to boundary of valid regions in UV atlas. A possible reason is that the 2D-prior network is weak at completing region without any supervision signal, e.g. the invalid region on UV atlas, and errors are propagated to the few pixels on the boundary of the valid region, introducing flipped faces and outlier vertexes. We try to expand the UV atlas valid region for a few pixels as commonly adopted by many previous methods but find this not effective since the supervision is too sparse. To fix the problem, we send the updated mesh back to the 3D-prior network for another refinement, and find the 3D-prior network is very effective in keeping benefits from the 2D-prior network and removing the artifacts (Fig.3 (c)) via stronger supervision and regularization. The two networks can run iteratively to improve surface quality gradually.

3.4. Texture Reconstruction

While most of the previous works only on surface reconstruction, our model can also produce a high-resolution texture, which particularly complements commodity 3D scanners since texture map and color images are usually not provided. Thanks to the 2D-prior network, we encode the (r, g, b) of the input point cloud into a sparse UV map instead of the location (x, y, z), and run the 2D-prior network to reconstruct a dense texture map supervised by the sparse signal from the input point cloud. Note that it is also possible to directly produce color for each point in the MeshCNN framework, but the resolution, i.e. color from roughly 10K points, is far from enough to deliver visual appealing rendering quality, e.g., 480K pixels for even VGA resolution.

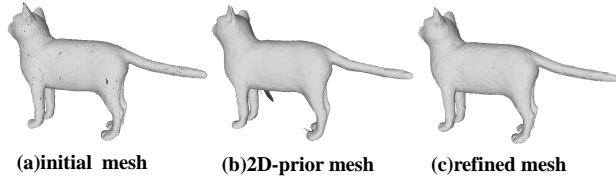(a)initial mesh   (b)2D-prior mesh   (c)refined mesh

Figure 3. Iterative Refinement with 2D-3D prior networks.

3.5. Implementation Details

The GPU memory required for MeshCNN optimizations is linearly increased as the mesh resolution increases. Following Point2Mesh [24], we cut the mesh into parts to guarantee that each part has less than 6000 faces. There are overlapping regions between different parts, and the final vertex position output is the average over all overlaps.

The whole architecture is implemented in Pytorch, and optimized using Adam Optimizer [35]. For 3D-prior network, the initial mesh contains 2000 vertices. In each refinement iteration, the 3D-prior network is optimized for 2000 steps with a learning rate of 1e-3, and the 2D-prior network is optimized for 4000 steps. The mesh is refined for 3 iterations as the performance usually saturates. The overall generation process takes roughly 2.5 hours, where 3D prior initialization, 2D prior refinement, 3D prior refinement take 90, 20, 40 minutes respectively. The 2D-prior network takes much less time compared to 3D-prior network. The runtime of the 2D-prior network is relatively constant w.r.t the number faces, while run-time of the 3D-prior network grows roughly linearly w.r.t the face number. Meanwhile, our initialization stage is faster than Point2Mesh which takes more than 3 hours.

4. Experiments

In this section, we evaluate our method for full 3D mesh model generation. We first show overall performance of our system, and then evaluate the quality of geometry and texture separately. We highlight the effectiveness of our 2D network for both geometry and texture reconstruction.

4.1. Data and Evaluation Metrics

We collected 14 ground-truth meshes for evaluation, including animals, persons, and planes from 3D model repository turbosquid[1] and free3D[2]. All models are 3D meshes with high-quality textures, and the amount of vertices in the original mesh varies from 8000 to 50000. For fair comparison with Point2Mesh [24], we did our best to collect a similar testing set as [24], which contains 4 meshes from Thingi10K [56], 18 meshes from COSEG [54], and 10 meshes from ShapeNet [12] of various object categories.[3]

---

[1] https://www.turbosquid.com
[2] https://free3d.com
[3] Note that the exact testing set of Point2Mesh is not publicly released.

	F-score↑	CD↓	EMD↓	NC↑
Poisson	95.6	0.0630	0.1285	0.908
P2M	97.2	0.0601	0.1068	0.941
MPC-IER	93.2	0.0712	0.1044	0.933
Points2Surf	90.4	0.0974	0.1191	0.903
Ours	**97.7**	**0.0526**	**0.0969**	**0.956**

Table 1. Comparison between our method and other surface reconstruction methods on synthetic data. **Bold**: Best. CD: Chamfer Distance. NC: Normal Consistency.

We synthetically generate input point clouds by uniformly sampling the mesh surface with color. Each point cloud has 25000 colored points, except for the rabbit which has 10000 points because its structure is relatively simple.

To evaluate the accuracy of the reconstructed model, we use the F-score as [24]. We also show quantitative comparison on Chamfer Distance, Normal Consistency and Earth Mover's Distance(EMD). For calculation, we sample 500K points on the surface of the ground truth and the predicted mesh respectively, and empirically find these are sufficiently dense to calculate stable metrics. For F-score, we set the distance threshold at 0.1 with the span of the longest dimension of each model scaled to 100.

Dataset	Method	F-score↑	CD↓	EMD↓	NC↑
Thingi10k	P2M	93.4	0.0428	0.1220	0.841
	Ours	**96.1**	**0.0375**	**0.1126**	**0.884**
COSEG	P2M	92.8	0.0422	0.1327	0.925
	Ours	**96.2**	**0.0349**	**0.1083**	**0.954**
ShapeNet	P2M	91.4	0.0533	0.1417	0.895
	Ours	**95.7**	**0.0405**	**0.1164**	**0.936**

Table 2. Comparison with Point2Mesh. **Bold**: Best. CD: Chamfer Distance. NC: Normal Consistency.

4.2. Full 3D Mesh Generation

We first show the quality of textured mesh generated by our model. Fig. 4 shows the input colored point cloud and the generated mesh visualized with and without the generated texture. Overall, our method successfully recovers thin geometry, e.g. the dog's ear and the bird's wings, and texture with sharp boundaries and details. As a comparison, we show the full model generated by Poisson reconstruction [33] where the surface normal is calculated using implementations in MeshLab [16] for well-known accuracy and robustness, and the texture is generated by linearly blending the color from the input point cloud [34]. From Fig. 4, our method clearly outperforms others on the quality of both geometry and texture.

We visualize the sparse XYZ map and the texture map and the dense predictions from our method in Fig. 6. We increase the size of the point (4 pixels for each point) for vi-

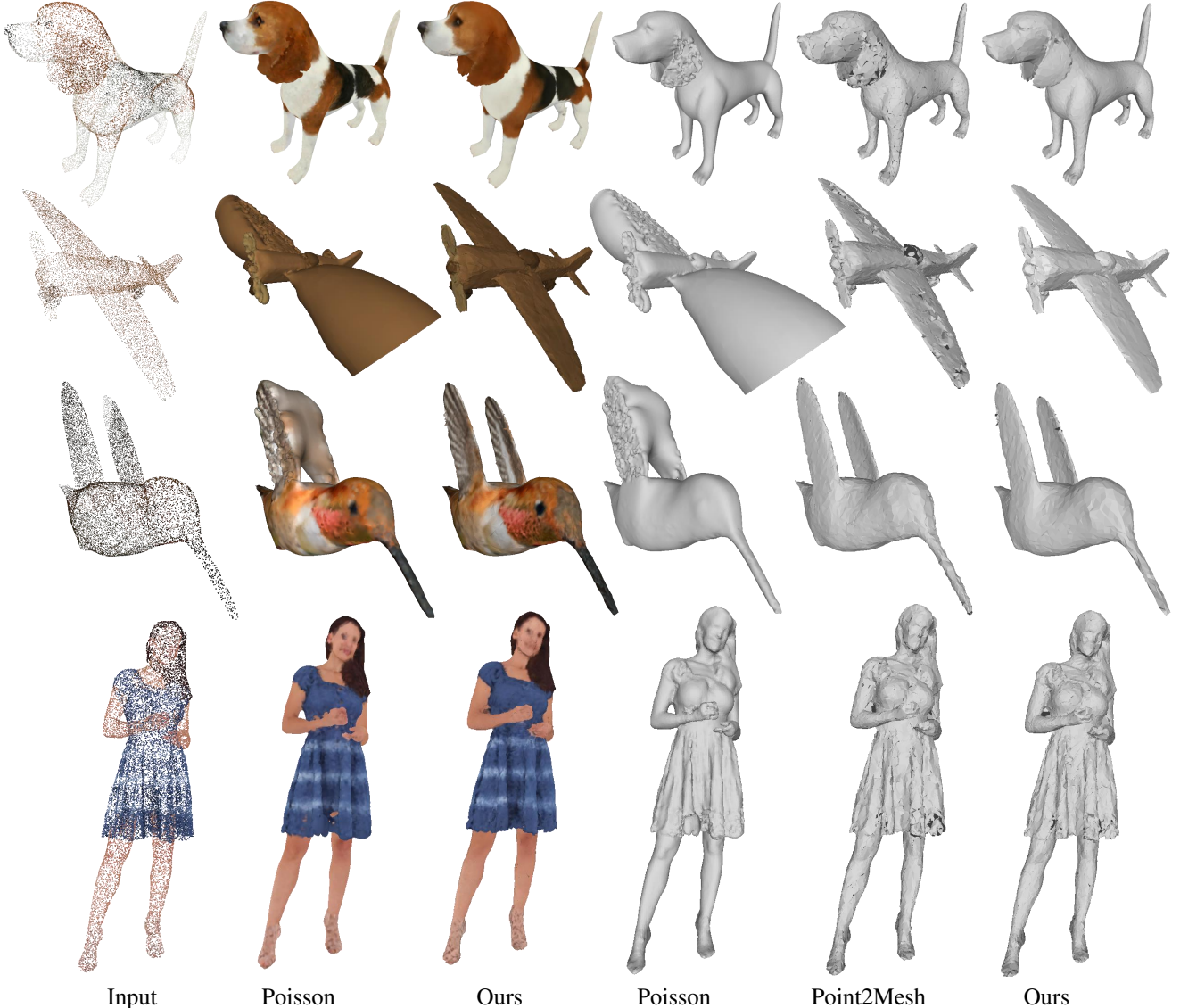| Input | Poisson | Ours | Poisson | Point2Mesh | Ours |

Figure 4. Comparison on synthetic data. We show examples with a full spectrum of difficulties for both geometry and texture, e.g. dog and airplane have more complex geometry, and bird has challenging texture. Please refer to our supplementary material for the comparison with more methods.

sualization since the original input is too sparse to see. Even with such sparse input, the network still manage to produce dense output and improve the details over iterations.

4.3. Surface Reconstruction

In this section, we evaluate the quantitative accuracy of the surface reconstruction, and the results are shown in Tab. 1. In addition to Poisson surface reconstruction [33] and Point2Mesh [24], we also compare to Points2Surf [18] and MPC-IER [38] for surface reconstruction. Note that they both require additional 3D data for training while we do not but purely rely on self-prior. Our method achieves the best score among all the methods, indicating our geometry is more accurate compared to other methods. Our method also

shows consistently better performance than Point2Mesh on Thingi10K, COSEG and ShapeNet as shown in Tab. 2. Note that for surface reconstruction, Point2Mesh can be considered as an ablation of our method without the 2D self-prior. Therefore, our improvements are mostly benefited from the use of the proposed hybrid 2D-3D prior. We also show the performance w.r.t iteration in the supplementary material.

The qualitative results are shown in Fig. 4. We can see that Poisson reconstruction generates incorrect meshes for thin structures, e.g., wings of the bird and plane. The results of Point2Mesh are relatively noisy, e.g., the dog's ear and the plane's wings, due to its weak 3D self-prior. In contrast, our method has the best performance with smooth meshes and correct thin structure.
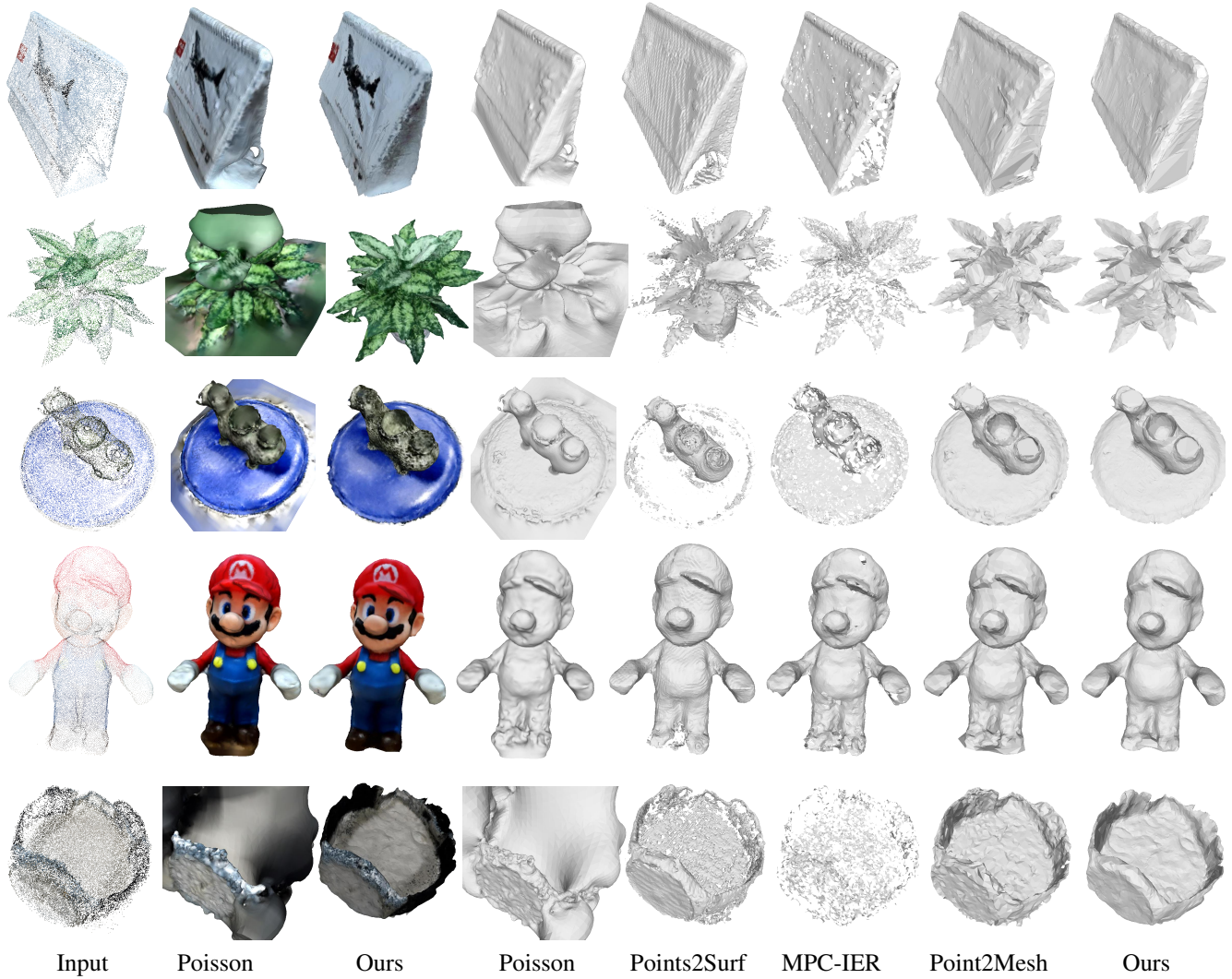
Input   Poisson   Ours   Poisson   Points2Surf   MPC-IER   Point2Mesh   Ours

Figure 5. Comparison between our method and other surface reconstruction methods with real scans.

Sparse maps   Step 1000   Step 2000   Step 4000

Figure 6. Sparse maps and generated dense maps at different training steps.

Input   Poisson   P2M   Ours

Figure 7. Comparison between our method and other surface reconstruction methods with noisy input.

Robustness Against Noise   It is well-known that the quality of the surface reconstruction highly depends on the input point cloud quality. We test the robustness of our method by manually adding Gaussian noise with standard deviation
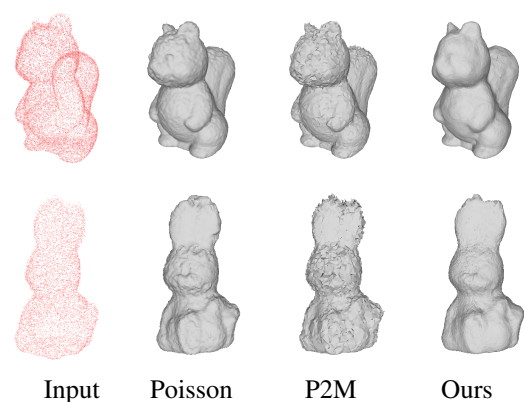
as 2% of the original coordinate value on the input point cloud, and the results are shown in Fig. 7. Both Poisson and Point2Mesh are easily get affected by the noise in
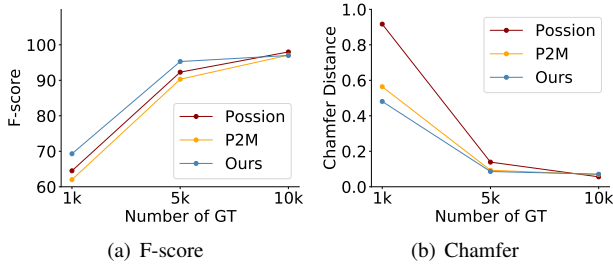
(a) F-score   (b) Chamfer

Figure 8. Comparison with Screen Poisson reconstruction
and Point2Mesh with different sparseness. (a) F-score met-
ric. Higher is better. (b) Chamfer metric. Lower is better.

the input, and the surface quality is inferior compared to
ours. Especially, more noise is reflected in the results from
Point2Mesh, which indicates the self-prior is not strong
enough in 3D GCN. Comparatively, our method leverag-
ing hybrid 2D-3D self-prior still produce reasonable surface
quality by removing noise in the XYZ map. More results
w.r.t different noise levels and robustness on texture gener-
ation are provided in the supplementary material.

Robustness Against Sparsity   We also test the robustness
of our model on sparse input point clouds. We run Poisson,
Point2Mesh, and our method by taking input point clouds
with different number of points, and show the performance
in Fig. 8. As expected, the performance of all the meth-
ods drops when the input point number is decreasing, but
our performance drops slower than the others, which again
shows that the 2D-3D self-prior is strong to interpolate rel-
atively large missing regions. More qualitative results are
provided in the supplementary material.

4.4. Texture Reconstruction

In this section, we evaluate our texture reconstruction.
It is not easy to find previous work with code for this spe-
cific task under our setting, so we compare to Kazhdan *et
al.* [34] which is implemented with Poisson surface recon-
struction in Meshlab. In Fig. 9, we show the texture on
the reconstructed mesh and highlight some regions. Our
method significantly outperforms the baseline method es-
pecially on edges, where the baseline texture tends to be
blurry and ours is usually sharp. We also build a MeshCNN
baseline method which directly predicts color for each point
in the MeshCNN [23] framework and provided the compar-
isons in supplementary material.

For the quantitative measure, we render the generated
colored mesh uniformly in 16 different views as [39] and
evaluate the Naturalness Image Quality Evaluator (NIQE)
score [42] of the rendered 4096×4096 images. NIQE is a
no-reference metric that solely considers perceptual quality.
The NIQE results of MeshCNN baseline, Poisson and our
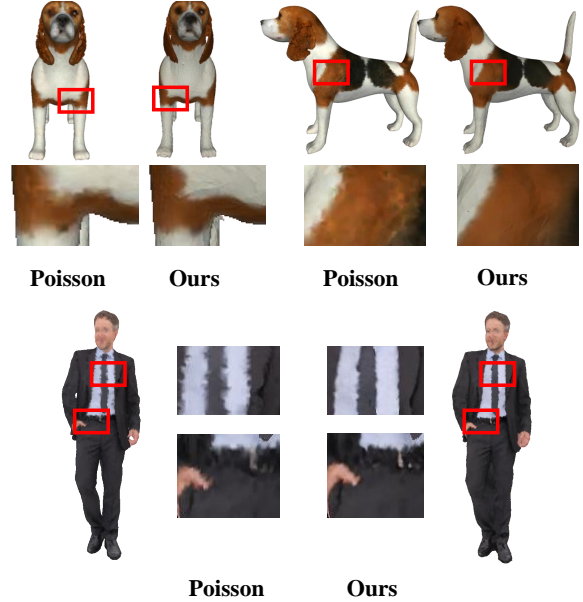method are 20.39, 20.65 and 19.35 respectively (lower is

Poisson   **Ours**   **Poisson**   **Ours**

Poisson   **Ours**

Figure 9. Comparison between our method and the Pois-
son surface reconstruction on texture quality. Our method
produces texture with better quality especially on edges.

better), which indicates ours shows better visual quality.

4.5. Generalization to Real Scans

We also test how our method performs on scans collected
from the commodity 3D scanner. We collect 3D textured
point clouds of five objects using Huawei 3D Live Maker,
each of which contains roughly 30000 points. Note that the
noise and sparsity of the points are not ideally uniform, and
thus these data are more challenging for full mesh recon-
struction. Fig. 5 shows results on these scans. Compare
to other methods, we produce overall better geometry for
smooth surface, sharp boundary, thin structure, and sparse
areas. Our texture is also sharper than Kazhdan *et al.* [34],
e.g., the airplane on calendar, plant, and mario face.

5. Conclusion

We propose a method to reconstruct textured mesh from
a colored point cloud by leveraging self-prior in deep neural
networks. A novel hybrid 2D-3D self-prior is exploited in
an iterative way. Based on an initial mesh generation using
a 3D convolutional neural network with 3D self-prior, the
2D UV atlas is generated and used to encode both 3D infor-
mation and color information that can be further refined by
2D CNNs with the self-prior. Experiments demonstrate the
advantages of the proposed method over SOTA methods.

References

[1] 3d object scanner artec eva. `https://www.artec3d.com/portable-3d-scanners/artec-eva`.

[2] Einscan pro. `https://www.einscan.com/handheld-3d-scanner/einscan-pro-hd/`.

[3] ireal 2s color 3d scanner. `https://www.3d-scantech.com/product/ireal-2s-color-3d-scanner`.

[4] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Representation learning and adversarial generation of 3d point clouds. *arXiv preprint arXiv:1707.02392*, 2(3):4, 2017.

[5] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pages 40–49, 2018.

[6] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 415–421, 1998.

[7] Nina Amenta and Yong Joo Kil. Defining point-set surfaces. *ACM Transactions on Graphics (TOG)*, 23(3):264–270, 2004.

[8] 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.

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

[10] Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim 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*, pages 67–76, 2001.

[11] Rohan Chabra, Jan Eric Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. Deep local shapes: Learning local sdf priors for detailed 3d reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[12] 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], 2015.

[13] ChaoWen, Yinda Zhang, Zhuwen Li, and Yanwei Fu. Pixel2mesh++: Multi-view 3d mesh generation via deformation. In *ICCV*, 2019.

[14] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. Implicit functions in feature space for 3d shape reconstruction and completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6970–6981, 2020.

[15] Christopher Bongsoo Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *ECCV*, 2016.

[16] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference*, volume 2008, pages 129–136. Salerno, 2008.

[17] 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*, pages 303–312, 1996.

[18] Philipp Erler, Paul Guerrero, Stefan Ohrhallinger, Niloy J. Mitra, and Michael Wimmer. Points2surf: Learning implicit surfaces from point clouds. In *ECCV*, 2020.

[19] 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.

[20] Yossi Gandelsman, Assaf Shocher, and Michal Irani. double-dip": Unsupervised image decomposition via coupled deep-image-priors. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 6, page 2, 2019.

[21] Rohit Girdhar, David F. Fouhey, Mikel Rodriguez, and Abhinav Gupta. Learning a predictable and generative vector representation for objects. In *ECCV*, 2016.

[22] 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.

[23] 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.

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

[25] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[26] 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.

[27] Jingwei Huang, Hao Su, and Leonidas Guibas. Robust watertight manifold surface generation method for shapenet models. *arXiv preprint arXiv:1802.01698*, 2018.

[28] Jingwei Huang, Justus Thies, Angela Dai, Abhijit Kundu, Chiyu Jiang, Leonidas J Guibas, Matthias Nießner, and Thomas Funkhouser. Adversarial texture optimization from rgb-d scans. In *CVPR*, 2020.

[29] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas Funkhouser. Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6001–6010, 2020.

[30] Angjoo Kanazawa, Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 371–386, 2018.

[31] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *CVPR*, 2018.

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

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

[34] Misha Kazhdan and Hugues Hoppe. An adaptive multi-grid solver for applications in computer graphics. In *Computer Graphics Forum*, volume 38, pages 138–150. Wiley Online Library, 2019.

[35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[36] David Levin. Mesh-independent surface interpolation. In *Geometric modeling for scientific visualization*, pages 37–49. Springer, 2004.

[37] 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.

[38] Minghua Liu, Xiaoshuai Zhang, and Hao Su. Meshing point clouds with predicted intrinsic-extrinsic ratio guidance. In *ECCV*, 2020.

[39] Ricardo Martin-Brualla, Rohit Pandey, Sofien Bouaziz, Matthew Brown, and Dan B Goldman. Gelato: Generative latent textured objects. In *ECCV*, 2020.

[40] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[41] Zhenxing Mi, Yiming Luo, and Wenbing Tao. Ssrnet: Scalable 3d surface reconstruction network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 970–979, 2020.

[42] Anish Mittal, Rajiv Soundararajan, and Alan C Bovik. Making a "completely blind" image quality analyzer. *IEEE Signal processing letters*, 20(3):209–212, 2012.

[43] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4531–4540, 2019.

[44] 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.

[45] 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.

[46] Jhony K. Pontes, Chen Kong, Sridha Sridharan, Simon Lucey, Anders Eriksson, and Clinton Fookes. Image2mesh: A learning framework for single image 3d reconstruction. Technical Report arXiv:1711.10669 [cs.CV], 2017.

[47] Yuhui Quan, Mingqin Chen, Tongyao Pang, and Hui Ji. Self2self with dropout: Learning self-supervised denoising from single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1890–1898, 2020.

[48] Dongwei Ren, Kai Zhang, Qilong Wang, Qinghua Hu, and Wangmeng Zuo. Neural blind deconvolution using deep priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3341–3350, 2020.

[49] Shunsuke Saito, Lingyu Wei, Liwen Hu, Koki Nagano, and Hao Li. Photorealistic facial texture inference using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5144–5153, 2017.

[50] Yongbin Sun, Ziwei Liu, Yue Wang, and Sanjay E Sarma. Im2avatar: Colorful 3d reconstruction from a single image. *arXiv preprint arXiv:1804.06375*, 2018.

[51] 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.

[52] 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.

[53] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *ECCV*, 2018.

[54] Yunhai Wang, Shmulik Asafi, Oliver Van Kaick, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Active co-analysis of a set of shapes. *ACM Transactions on Graphics (TOG)*, 31(6):1–10, 2012.

[55] 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.

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

[57] Jun-Yan Zhu, Zhoutong Zhang, Chengkai Zhang, Jiajun Wu, Antonio Torralba, Josh Tenenbaum, and Bill Freeman. Visual object networks: Image generation with disentangled 3d representations. In *Advances in neural information processing systems*, pages 118–129, 2018.

Supplementary Material

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 hyperparameters for the best trade-off between accuracy and efficiency.

A.1. 3D-prior Network

We use a MeshCNN [23] based 3D-Prior network introduced in Point2Mesh [24] to generate an initial mesh and the refined output mesh. We also adopt residual and skip connections in MeshConv [23] 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 J, 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 J.

A.2. 2D-prior Network

We follow DIP [51] 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 [26] 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 K we provide the details of our 2D-prior network architecture. The whole network contains five downscale convolution blocks, five upscale convolution blocks

	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 C. 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.

and five skip connection blocks. The layers and parameters of each block are shown in the right part of Figure K.

A.3. UV Flattening

We use OptCuts[37] 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. L and Fig. M. In Fig. 4 of main paper, we showed comparison to Poisson [33] and Point2Mesh [24] on a few examples. Here we add comparison to more previous works in Fig. L. For each example in Fig. M, we show the input colored point cloud and the results from Screen Poisson Surface Reconstruction [33] (Poisson), Point2Mesh [24] (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. N.

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.

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[22] and MC-RIMLS[44]. We also show
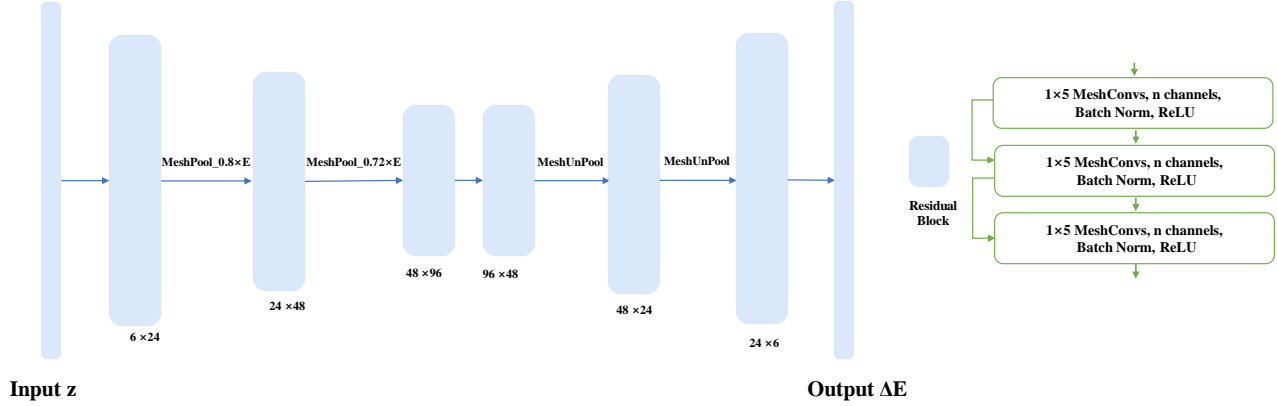
Figure J. Architecture of 3D-prior network. In general, it consists of six residual blocks. The detailed structure and channels of the residual blocks are shown in the left.
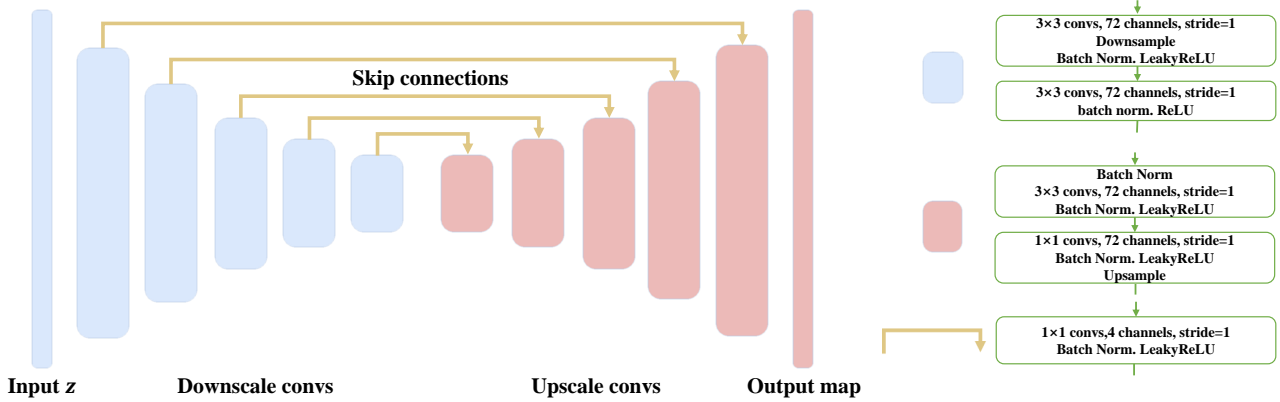

Figure K. Architecture of 2D-prior network. The layers of each component are shown in the right.

the comparison with Point2Mesh [24] with a HC Laplacian smooth [52], namely P2M-S. The Chamfer distance and F-Score are reported in Tab. C, which are worse than our method.

D. Failure Cases

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

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
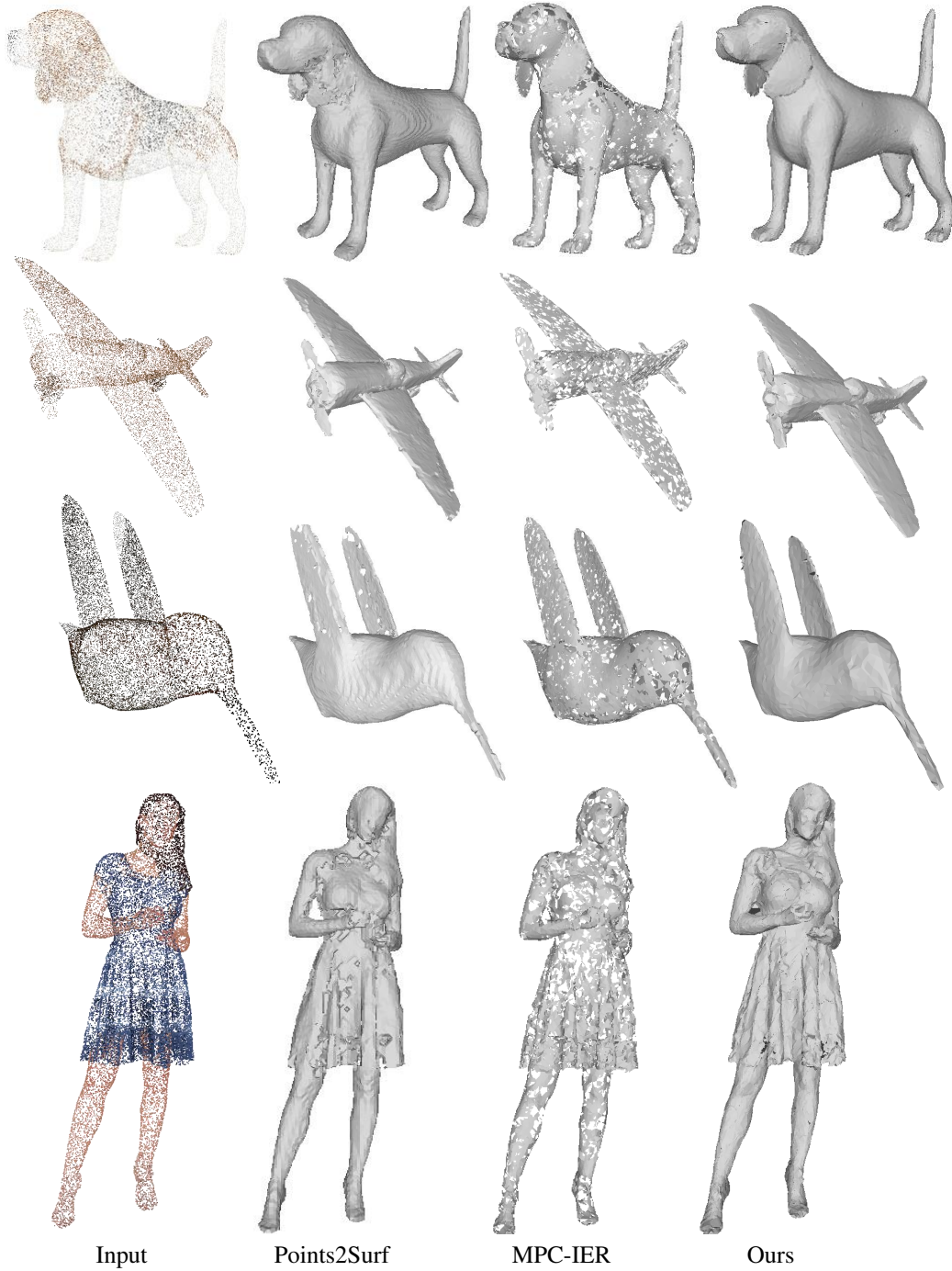
| Input | Points2Surf | MPC-IER | Ours |

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

point cloud. Table D 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.

GT Mesh

Input   **Poisson**   **Ours**   **Poisson**   **P2M**   **Ours**

Figure M. Comparison between our method and other surface reconstruction methods. The groundtruth meshes used to sample input point clouds are shown in the first row.
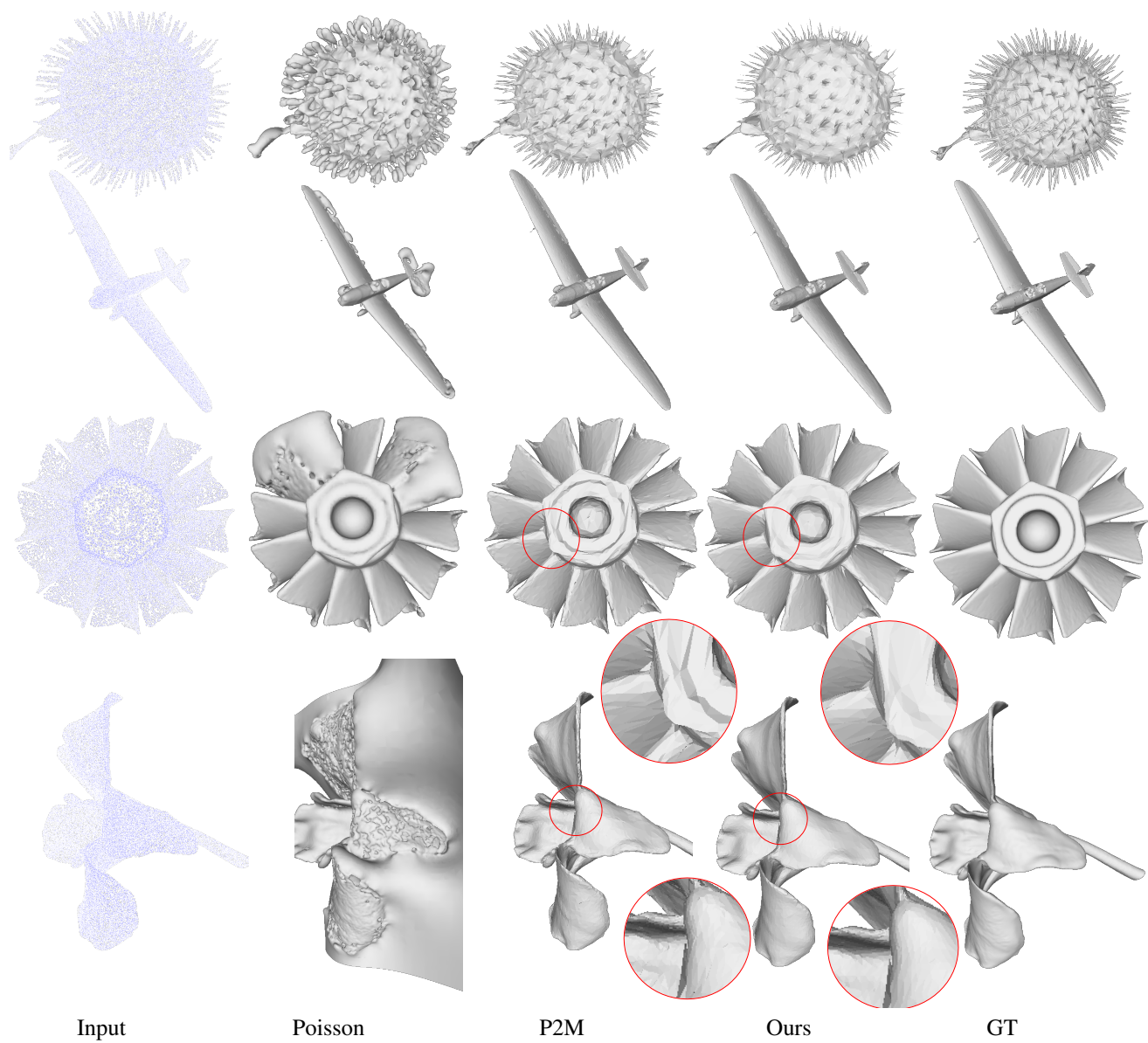
| Input | Poisson | P2M | Ours | GT |

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

| Ours | GT | Ours | GT |

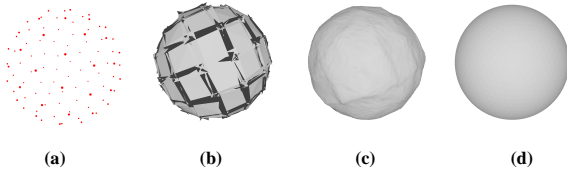Figure O. Failure cases in geometry and texture reconstruction.

Figure P. 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.
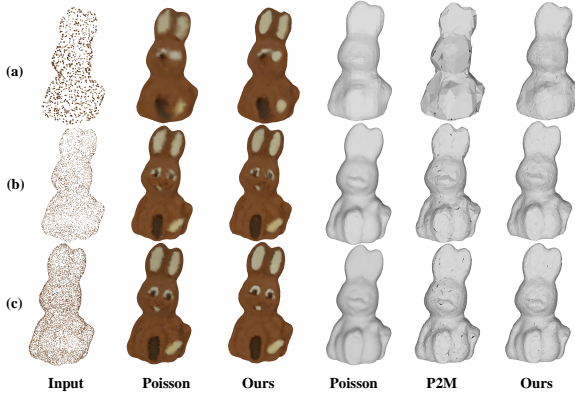

Figure Q. Generated textured mesh with different sparseness. (a) 1k input points; (b) 5k input points; (c) 10k input points.

E.2. Robustness against Sparsity

In Fig. P, 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. Q 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 [23] 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

$$L_{color} = \sum_{\hat{p}} \|C_{\hat{p}} - C_q\|, \qquad (3)$$

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. R. 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. S, 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 T (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 T (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

| Input | Baseline | Ours | Baseline | Ours |

Figure R. Qualitative comparisons of texture between our method and the baseline method which uses a MeshCNN network to predict vertex color.
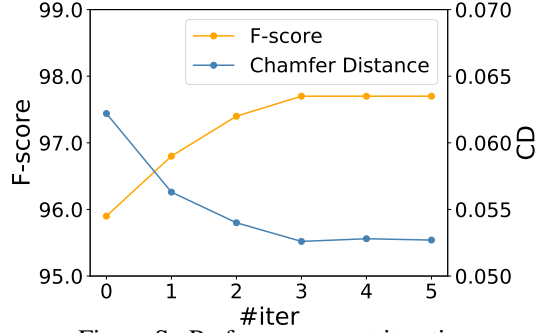
Figure S. Performance w.r.t iteration.

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 U with ϵ sampled from different standard deviations.

As shown in Figure U, 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.
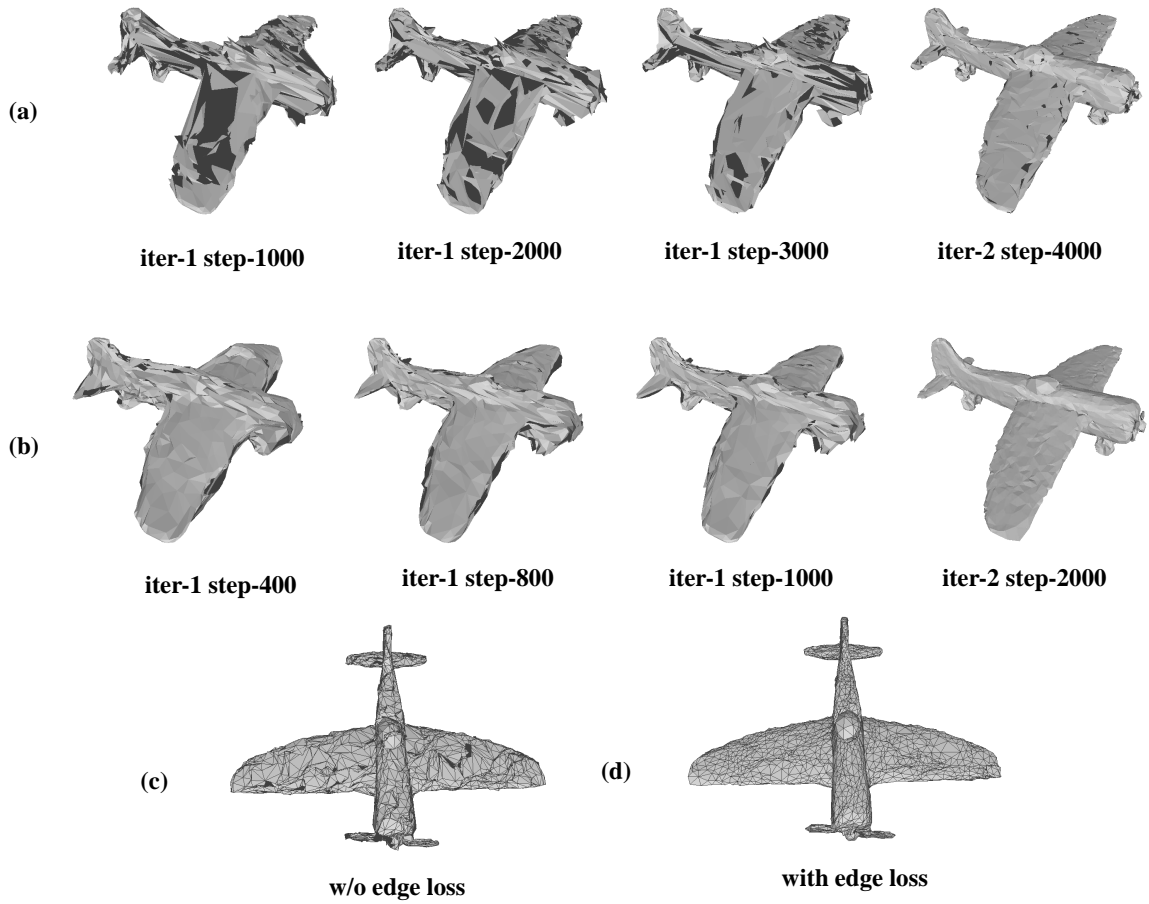
(a)

iter-1 step-1000   iter-1 step-2000   iter-1 step-3000   iter-2 step-4000

(b)

iter-1 step-400   iter-1 step-800   iter-1 step-1000   iter-2 step-2000

(c) w/o edge loss   **(d)** with edge loss

Figure T. 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.

Geometry

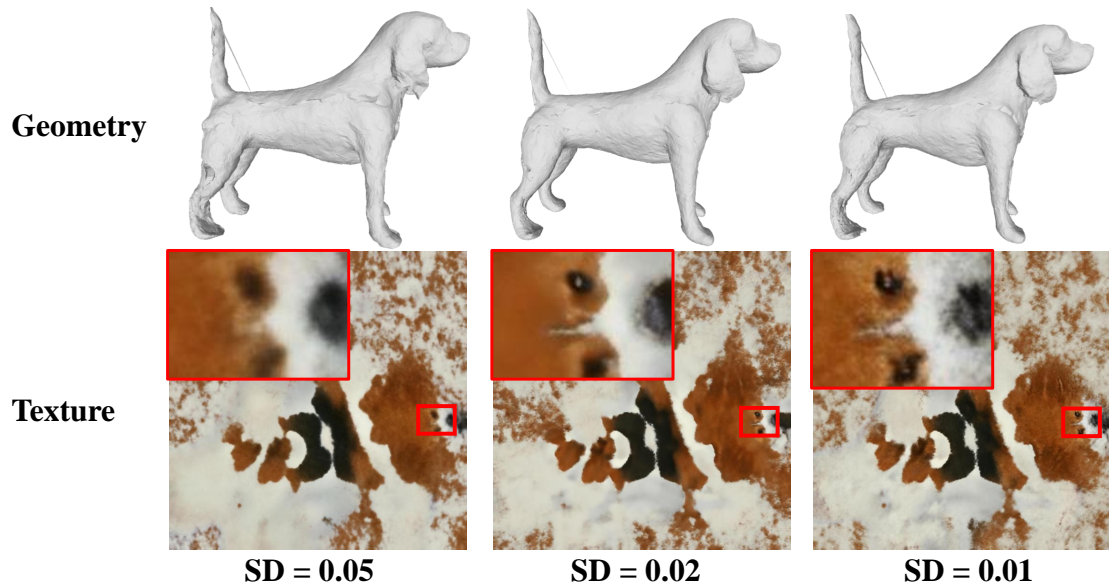**Texture**

SD = 0.05   SD = 0.02   SD = 0.01

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