Supplementary Materials for "DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing"

Shaohui Liu[1,3] Yinda Zhang[2] Songyou Peng[1,6] Boxin Shi[4,7]
Marc Pollefeys[1,5,6] Zhaopeng Cui[1]
[1]ETH Zurich [2]Google [3]Tsinghua University [4]Peking University [5]Microsoft
[6]Max Planck ETH Center for Learing Systems [7]Peng Cheng Laboratory

In this supplementary material, we provide detailed analysis of the proposed renderer, implementation details, and more qualitative results.

A. More Analysis on the Design of Differentiable Sphere Tracing

A.1. Benefits of Aggressive Marching

We use an aggressive strategy (Sec. 3.2 in main submission) to speed up the sphere tracing. Instead of marching with the step size as the SDF of the current location, we march α times of it, where α is larger than 1 and set as 1.5 by default. This brings two benefits - faster convergence and stable training.

Faster Convergence As shown in Fig. 1, the sphere tracing algorithm can become unexpectedly slow when the angle between the camera ray and the surface is relatively small. From an initial point with a ray distance d towards the surface, the marching step k needs to satisfy the equation below to reach convergence:

$$|d(1-\alpha sin\theta)^k| < \epsilon, \tag{1}$$

where α equals to 1.0 in the conventional sphere tracing algorithm. When $|1-\alpha sin\theta| < 1$, we can easily derive the minimum marching step needed,

$$k > k_{min} = \frac{log\epsilon - logd}{log|1-\alpha sin\theta|}. \tag{2}$$

By taking an aggressive strategy with α greater than 1 (we set it to 1.5 by default), the convergence can be speeded up under the ill-posed conditions. For example, suppose $d = 1.0$, $\epsilon = 5 \times 10^{-5}$, the minimum number of convergence steps decreases from 52 to 33 when θ equals to 10 degrees.

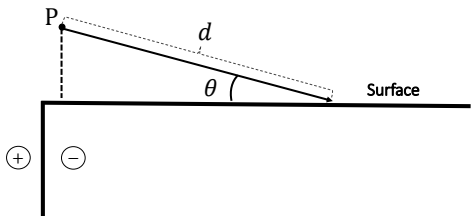

Figure 1. Illustration of ill-posed conditions for sphere tracing algorithm. When θ is relatively small, the queried SDF is much lower than the actual ray distance d towards the surface, making the sphere tracing process slower than expected.

Stable Training Besides speeding up the overall marching, the aggressive marching also allows more samplings from the locations behind the surface, which adds supervision at the interior of the shape and stabilize the training. As shown in Fig. 2, in traditional ray marching, the front end of the ray approaches the surface from the camera side (*i.e.* $SDF > 0$) and less likely to trespass the surface. In contrast, our marching is more likely to pass through the surface (and for sure under the ground truth SDF when the ray direction is orthogonal to the surface since the marching step is larger than SDF). This will not add much computational overhead when working conjointly with the convergence criteria, but achieves ray convergence from both sides the surface. This gives the training more supervision with both positive and negative SDF, compared to positive only using regular marching. The aggressive marching also naturally samples more points near the surface, which are important for network to learn surface details. Coincidentally, DeepSDF [4] also mentioned the importance of sampling more points near surface. While they need to rely on extra ground truth normal and depth to perform the sampling, our method is fully automatic and sample adaptively according to the SDF field.

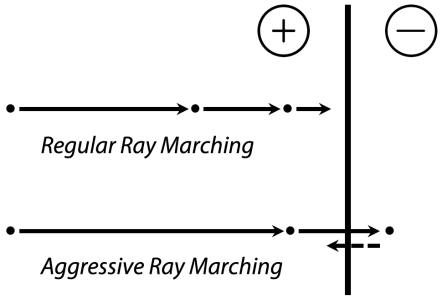

Figure 2. Compared to the regular ray marching algorithm, the aggressive ray marching strategy can march inside the surface and bounce back-and-forth between the inside and outside areas. This gives more samples on the negative side of implicit signed distance function and benefits the optimization.

A.2. Convergence Criteria

It is important to define a proper convergence criteria as shown in Fig. 7 of the main submission. In our differentiable sphere tracing, a ray stops marching if the absolute SDF is smaller than certain threshold ϵ. Essentially, this means that the true intersection on the surface is bounded by a ball with radius of ϵ centered at our current sampling point. There are two guidelines to select this threshold. On one hand, the threshold should not be too large, since the rendering noise is theoretically bounded by this threshold. Large threshold may result in large error in the rendered depth. On the other hand, the threshold must not be too small. The rendering time will be significantly longer if it is too small since more queries would be needed. Moreover, with a fixed maximum number of tracing steps, some pixels may not converge and thus are considered as background, which causes erosion (as shown in Fig. 7 in our main submission). Based on these two observations, we propose to define the threshold as the distance where the ray front ends from neighboring pixels are clearly separable. This is equivalent to finding the radius such that balls centered at the ray front ends of neighboring pixels do not intersect with each other. Fig. 3 demonstrates how to compute the ϵ. For a camera with focal length f, sensor size S, and resolution R, we get the following equation for objects roughly d_{min} away from the camera, according to similar triangles:

$$\frac{S/R \cdot cos(\theta)}{f/cos(\theta)} = \frac{2\epsilon}{d_{min}} \tag{3}$$

This gives:

$$\epsilon = \frac{d_{min} \cdot S \cdot cos^2(\theta)}{2 \cdot f \cdot R} \tag{4}$$

Taking the common set up shown in Fig. 3, where $f = 60mm, d_{min} = 10cm, S = 32mm, R = 512$, we get $\epsilon \approx 0.5 \times 10^{-4}m$ $(0.05mm)$.

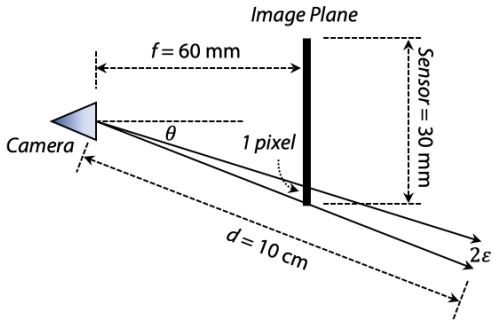

Figure 3. Illustration on the geometric meaning of the threshold ϵ of the convergence criteria.

A.3. Differentiable Rendering of Silhouette

Fig. 4 shows how to render the silhouette in a differentiable way. After running our deep sphere tracing (Fig. 4 (a)), we render the minimal absolute SDF on each pixel, and get the soft silhouette by substracting it by ϵ (Fig. 4 (b)). In this way the binary silhouette (Fig. 4 (c)) can be easily acquired by checking whether the rendered silhouette is positive (background) or not (foreground).

Because our rendered silhouette is fully differentiable with respect to implicit signed distance functions and camera extrinsic parameters, we can define differentiable loss term over the rendered silhouette S_r and the ground-truth binary silhouette S_{gt}. The silhouette loss $\mathcal{L}_s$ can be formulated as below:

$$\mathcal{L}_s = S_{gt} max(0, S_r) + (1 - S_{gt}) max(0, -S_r). \tag{5}$$

This formulation is able to get the silhouette error differentiably back-propagate to the optimized parameters. Note that by using the minimum absolute query, we utilize the nice individual property for signed distance functions, where the nearest surface with respect to the camera ray is optimized (as shown in Fig. 5). This strategy makes the shape optimization smooth and effective. Intuitively, combining the rendered silhouette with the distance transform on the 2D image plane can further improve the efficacy of the term, which is left for future work.

As discussed in the main paper, we check whether the ray intersects with the unit sphere to generate an initialization mask, where the ray without intersection with the unit sphere is directly set to background. To make the rendered silhouette on those background pixels differentiable, we set the soft silhouette value on each of those pixels to be the distance from the origin to the corresponding camera ray minus 1.0. This design shares similar spirits with the previous design on differentiable rendering of the silhouette. Because the distance from the origin to each of those camera rays is always greater than 1.0, we can consistently check whether the rendered silhouette is negative to determine its corresponding binary silhouette.

(a) Sphere Tracing (b) $\texttt{min}(\texttt{abs}(SDF))$ (c) $\texttt{min}(\texttt{abs}(SDF)) < \epsilon$

Figure 4. The differentiable rendering of silhouette. We take minimum absolute SDF value along each ray and determine the silhouette by checking whether the minimum absolute value is less than the threshold ϵ or not. We also consider the rendered soft silhouette as the minimum absolute queries substracted by ϵ, which is fully differentiable and feasible for optimization.

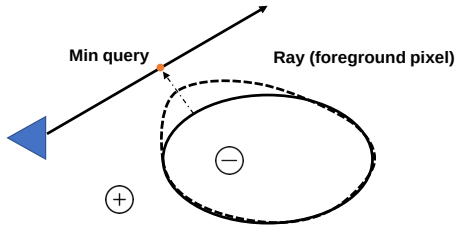

Figure 5. Differentiable error propagation along the boundary of the foreground. We can make use of the nice property of signed distance fields to optimize the nearest surface.

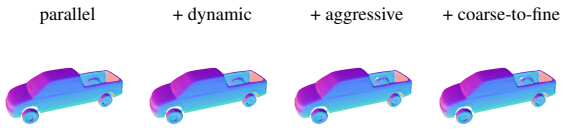

Figure 6. Illustration of the small artifacts induced by the aggressive strategy. Small holes could occur on thin surface areas. Better viewed when zoomed in.

A.4. Drawbacks

Most of our acceleration strategy does not affect the rendering quality, except the aggressive marching. Fig. 6 shows the drawback of the aggressive tracing strategy. This mostly happens for the case when the geometry is super thin such that a single step of marching may trespass two surfaces. As a result, the ray front end is still considered out of the shape and will keep marching till infinite, which causes artifacts shown in the back part of the truck.

Another potential drawback is the well-known aliasing effect, since for each pixel there is only one ray shot from the pixel center. Many well-known anti-aliasing strategies can be directly applied to our renderer to mitigate this issue.

B. Implementation Details

B.1. Network Architecture

We follow the same network architecture as DeepSDF [4], which consists of 9 fully connected layers. Each hidden layer has a dimension of 512. For the texture re-rendering applications, we concatenate the shape code and texture code together and feed the concatenated code to the texture network that employs the same architecture as the geometry network. Both the shape code and texture code has a dimension of 256. Note that the network architecture of DeepSDF [4] is much heavier than the backbone used in [5] (4 layers with 256 dimensions for each). However, with our proposed advanced sphere tracing strategies, we can produce high-resolution images within limited time consumption overhead.

B.2. Implementation of Dynamic Synchronized Inference

Here we present some details on the implementation of dynamic synchronized inference with the off-the-shelf deep learning framework. We maintain a binary flag over each camera ray during the sphere tracing process. For each step, we concatenate all unfinished camera rays together and perform feedforward in a batch-wise manner and map them back to the original image resolution. Then, we check whether the tracing on each ray converges or gets out of the unit sphere and set the corresponding binary flags to zero. Note that because the operation is performed on a concatenated tensor in the computational graph, trivially computing the minimum absolute SDF value for each ray results in unaffordable memory consumption. To address this issue, we introduce an implementation trick that the location of each query is saved globally, and minimization is performed over a detached tensor graph. After this operation, we get the minimum K queries for each ray and feedforward those queries again with the gradients attached. This strategy enables our renderer to produce much high resolution images (2048×2048) on a single GTX-1080Ti.

B.3. Depth / Normal / Silhouette / Color Loss

Our method renders 2D observations in a differentiable manner and computes the loss on the image plane. The depth loss and normal loss are computed over the foreground region determined by the rendered silhouette. For the multi-view shape reconstruction application, we compute the photometric error over the commonly visible pixels in two views.The visibility can be determined by computing the difference Δ_d between the reprojected depth from the source view and the directly rendered depth of the target view. In our experiments, the pixels with $\Delta_d^2 < 0.001$ are considered as visible. L_1 loss is used for depth and photometric error computation, and negative dot product is computed to measure the loss of surface normals [2]. For the rendered silhouette (substracting minimum absolute query with ϵ), we require that the value should be negative over the foreground and positive over the background and use the loss term defined in Eq. (5).

B.4. More Details and Hyperparameters

Our framework is implemented in PyTorch and code will be made publicly available. For all experiments, we use the Adam optimizer with the initial learning rate 1e-2. To compute the Chamfer Distance for evaluation, we sample 30k points on depth completion and 10k points on multi-view shape reconstruction respectively. We follow the common practice to report the distance scaled by 1000. In the multi-view scenario, the 3D shapes from the DeepSDF decoder may consist of structures inside some objects but the ground-truth meshes only have the structure on the surface. Therefore, we report the Chamfer Distance only in the direction of gt→pred for fair comparison. For the sampling stretegy when rendering depth, empirically, $K = 1$ already works reasonably well. We use $K = 3$ for shape completion and $K = 1$ for the multi-view shape reconstruction. The maximum marching step we use is 100. Empirically, a value choice ranging from 1.2 to 1.8 can produce an effective α. Small α results in slow convergence while large α can lead to small holes in thin areas.

For shape completion, we initialize the latent code to be zero (which denotes the mean shape) and perform optimization for 100 iterations. The loss weights of the depth loss and silhouette loss are set to 10.0 and 1.0 respectively. We also follow [4] to add an ℓ_2 regularizer over the shape code during optimization and the loss weight of the regularization term is set to 1.0.

For multi-view shape reconstruction under the PMO [3] synthetic test set, for each iteration we sample 8 views uniformly distributed 360° around the object, warp each of them and calculate the photometric loss to the closest next view based on the estimated depth image. We downsample input images to its half size 112×112 in order to back-propagate the photometric and regularization losses from all 8 views together. Since there are 72 views provided in the dataset for each object, we run 9 iterations for each epoch and 20 epochs in total.

As for the real-world multi-view dataset, in contrast to PMO [3] which uses over 100 images of an object, we only picked between 20 and 30 views and further downsample them from 480×640 to 96×128. 6 views are selected during each iteration. Since the provided initial similarity transformation is not accurate in some cases, we also optimize over this similarity transformation in addition to the shape code. We find out that it is usually enough to acquire an accurate similarity transformation after only 1 or 2 epochs. The weight of the photometric loss is set to 5.0 for both synthetic and real-world experiments.

C. Rendering Demos

We attach a video demo in the supplementary material to show that our method can render high-resolution depth, surface normals, silhouette and RGB image with various lighting conditions and camera viewpoints. Also, more qualitative results on multi-view reconstruction are included in the video. We also show a larger version on the texture re-rendering demo (Fig. 8 in the main paper) in Fig. 7.

D. Additional Experimental Results

D.1. Qualitative Comparison for Shape Completion

We show qualitative comparison on shape completion under different sparsity of input depth in Fig. 8, Fig. 9 and Fig. 10. The visual quality of our optimized mesh clearly outperforms the baseline method DeepSDF [4]. By employing perspective camera model and using online computed error (compared to the fixed sampling strategy in DeepSDF [4]) to perform inverse optimization on the rendered 2D observations, our method generates less holes in the predicted mesh when the input depth is sparse, particularly from the original view where the input depth is captured. When the silhouette information is available, our method can work reasonably well when the input depth is extremely sparse. Note that we give DeepSDF [4] the surface normal predicted by the dense ground-truth depth to make its sampling feasible. On the contrary, we do not use the surface normal information in the optimization of our method, which potentially can further improve our performance. Our method can generate reasonable occluded part with the assistance of the pretrained shape model prior. However, our method also could fail when the view of the camera cannot give sufficient information and the depth completion problem becomes extremely ill-posed.

D.2. Qualitative Comparison for Multi-view Shape Prediction

We firstly show more qualitative comparisons on PMO test dataset in Fig. 11. It can be noticed that our method produces much more visually satisfactory 3D shapes with only random initializations. In contrast, even if PMO [3] uses a much better initialization from the encoder pretrained on their PMO training set, their 3D shapes have low resolutions because of the limited number of vertices. Moreover, if the random initialized codes are applied, PMO fails to generate reasonable 3D shapes in most of the cases. When checking closely the second column in Fig. 11, the results even converge to rather similar shapes, especially for chairs and planes.

We also illustrate more results on the real-world multi-view chair dataset in Fig. 12. Note that similar to PMO, we now also optimize over the similarity transformation including rotation, translation and scale, together with the shape code. It can be noticed easily that our method with random initialization again generates much superior outputs over PMO. We also show a failure case in the last row.

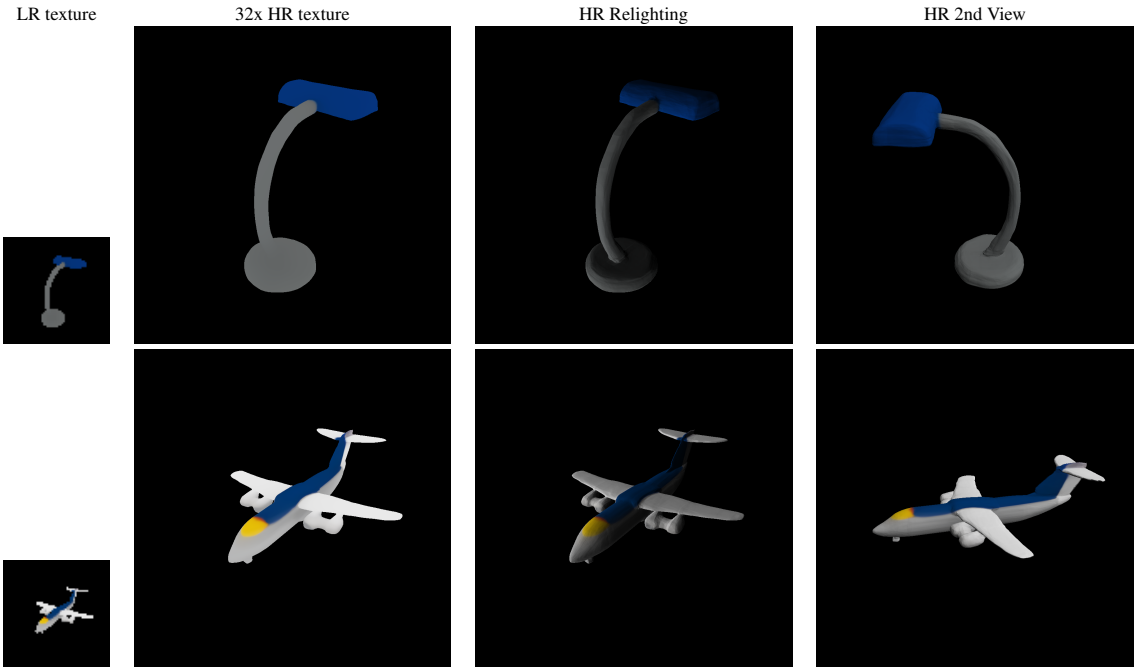

Figure 7. Qualitative results on the applications on texture re-rendering, where we can generate high-resolution outputs under various resolution, camera viewpoints and illumination.

Our method fails when there is insufficient texture on foreground or background as photometric cues. Moreover, our method may fail when the similarity transformation can not be correctly estimated.

References

[1] Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. A large dataset of object scans. *arXiv preprint arXiv:1602.02481*, 2016. 10

[2] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015. 3

[3] Chen-Hsuan Lin, Oliver Wang, Bryan C Russell, Eli Shechtman, Vladimir G Kim, Matthew Fisher, and Simon Lucey. Photometric mesh optimization for video-aligned 3d object reconstruction. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2019. 4

[4] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2019. 1, 3, 4, 6, 7, 8

[5] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Proc. of Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 3

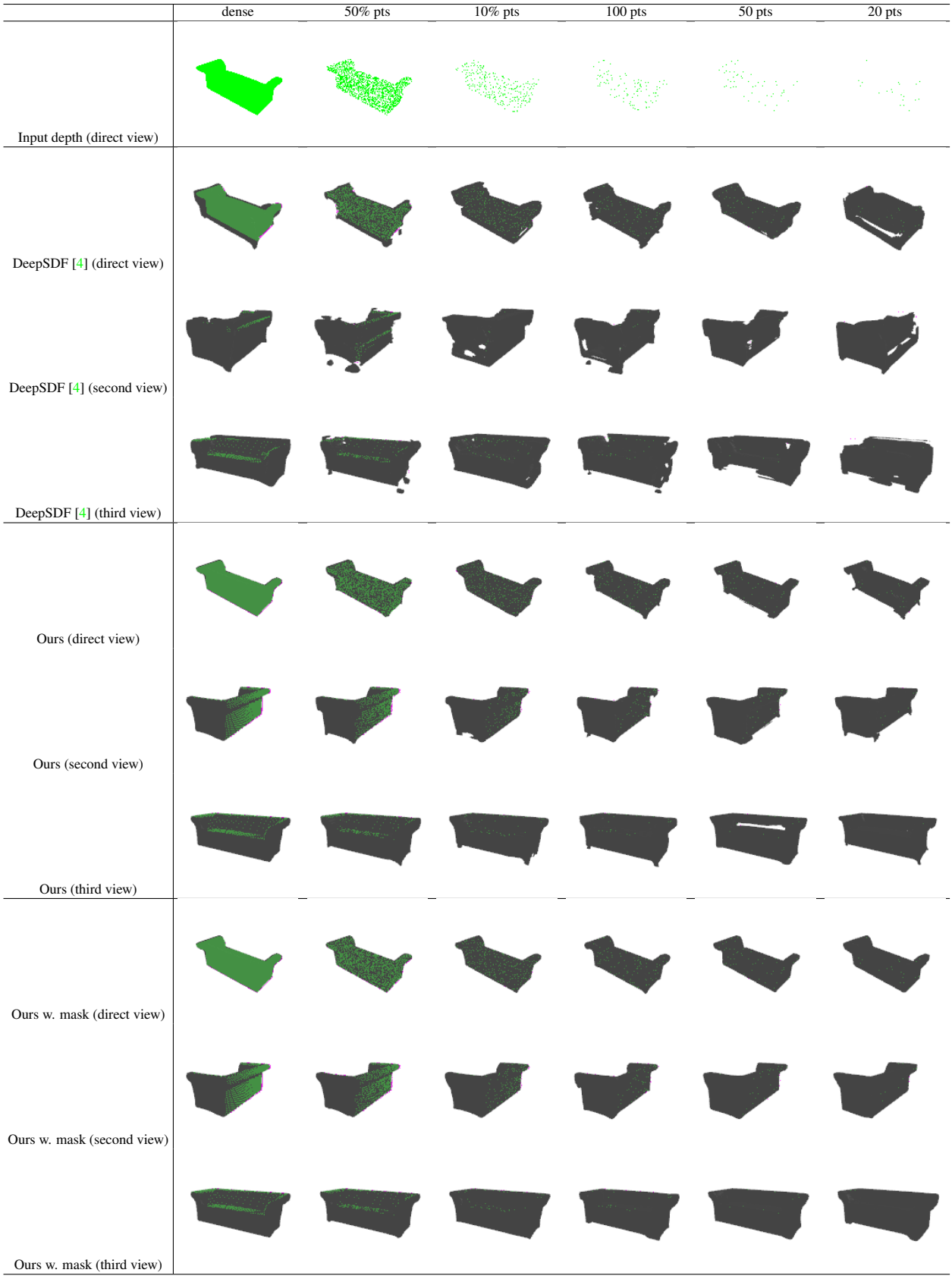

Figure 8. Qualitative comparisons on shape completion under different sparsity of input depth (sofa).

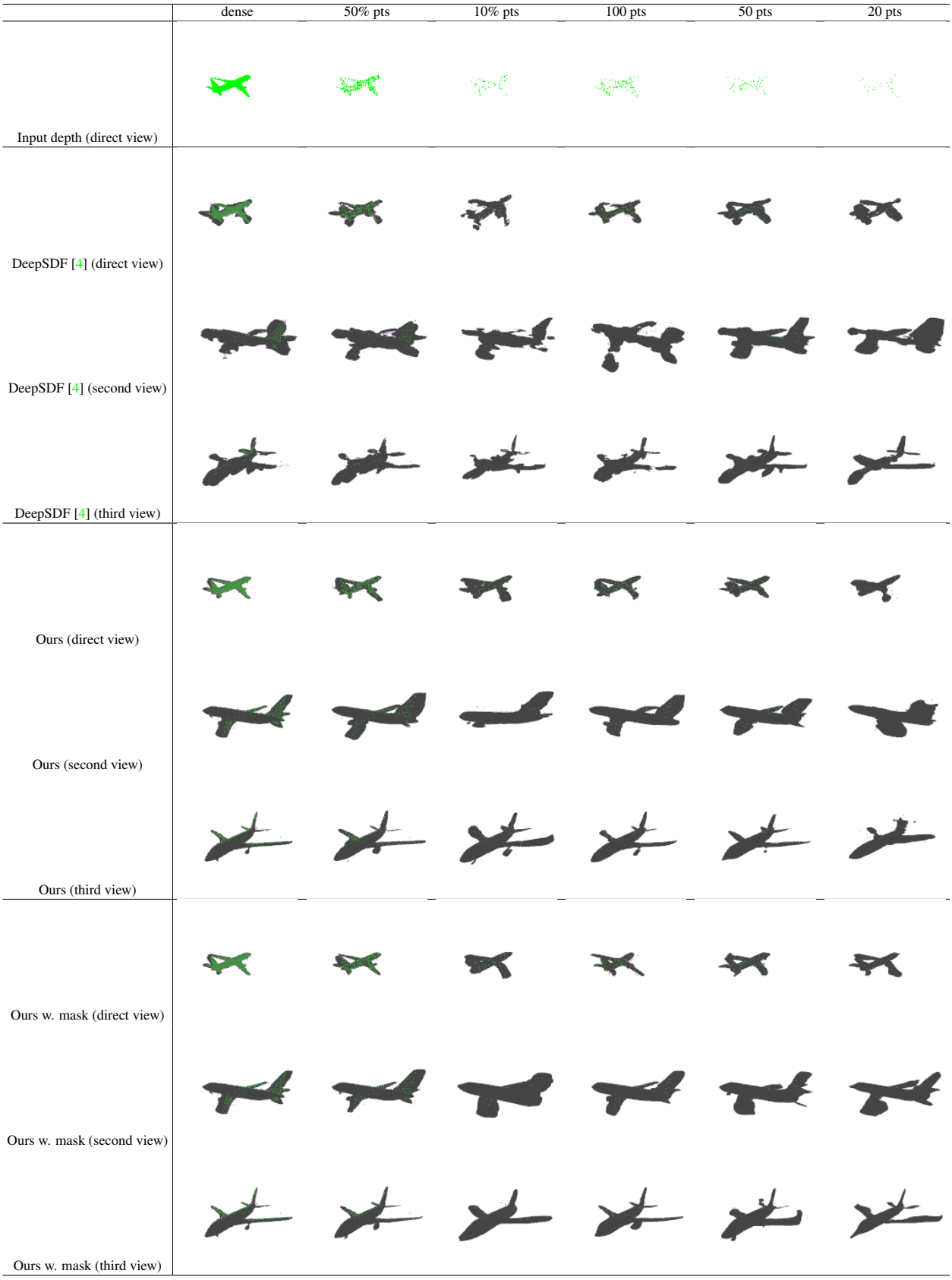

Figure 9. Qualitative comparisons on shape completion under different sparsity of input depth (plane).

Figure 10. Qualitative comparisons on shape completion under different sparsity of input depth (table).

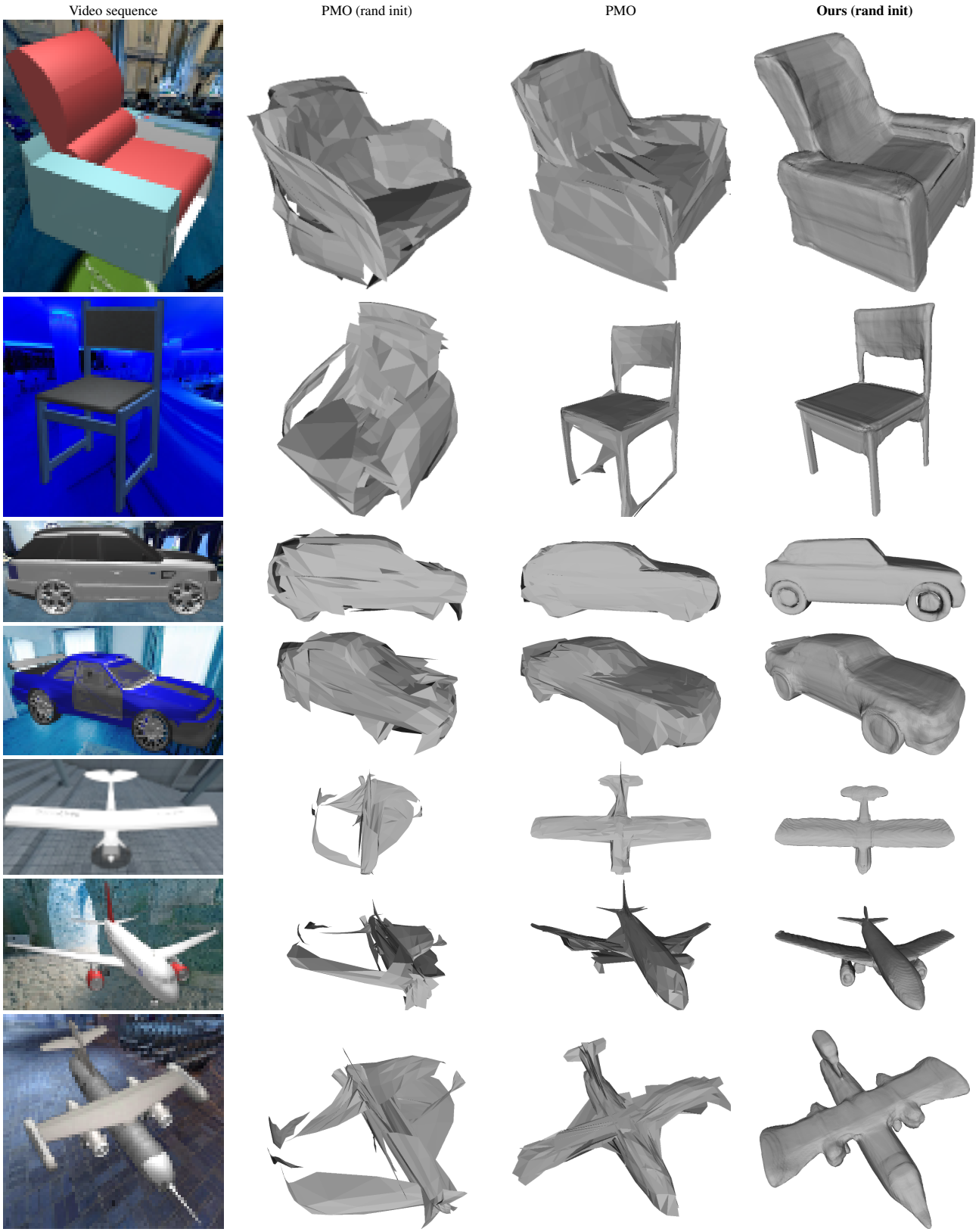

Figure 11. More quatitative comparisons on 3D shape prediction from multi-view images on the PMO test set.

Figure 12. More Comparisons on 3D shape prediction from multi-view images on real-world chair dataset [1]. It is in general challenging for shape prediction on real images. Comparatively, our method produces more reasonable results with correct structure.