arXiv:1911.13225v2 [cs.CV] 11 Jun 2020

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

Abstract

We propose a differentiable sphere tracing algorithm to bridge the gap between inverse graphics methods and the recently proposed deep learning based implicit signed distance function. Due to the nature of the implicit function, the rendering process requires tremendous function queries, which is particularly problematic when the function is represented as a neural network. We optimize both the forward and backward passes of our rendering layer to make it run efficiently with affordable memory consumption on a commodity graphics card. Our rendering method is fully differentiable such that losses can be directly computed on the rendered 2D observations, and the gradients can be propagated backwards to optimize the 3D geometry. We show that our rendering method can effectively reconstruct accurate 3D shapes from various inputs, such as sparse depth and multi-view images, through inverse optimization. With the geometry based reasoning, our 3D shape prediction methods show excellent generalization capability and robustness against various noises.

1. Introduction

Solving vision problem as an inverse graphics process is one of the most fundamental approaches, where the solution is the visual structure that best explains the given observations. In the realm of 3D geometry understanding, this approach has been used since the very early age [1, 36, 55]. As a critical component to the inverse graphics based 3D geometric reasoning process, an efficient renderer is required to accurately simulate the observations, *e.g.*, depth maps, from an optimizable 3D structure, and also be differentiable to back-propagate the error from the partial observation.

As a natural fit to the deep learning framework, differentiable rendering techniques have drawn great interests recently. Various solutions for different 3D representations, *e.g.*, voxels, point clouds, meshes, have been proposed. However, these 3D representations are all discretized up to a certain resolution, leading to the loss of geometric details and breaking the differentiable properties [24]. Recently, the continuous implicit function has been used to represent the signed distance field [35], which has premium capacity to encode accurate geometry when combined with the deep learning techniques. Given a latent code as the shape representation, the function can produce a signed distance value for any arbitrary point, and thus enable unlimited resolution and better preserved geometric details for rendering purpose. However, a differentiable rendering solution for learning-based continuous signed distance function does not exist yet.

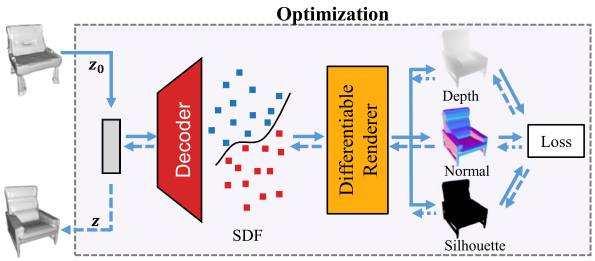


Figure 1. Illustration of our proposed differentiable renderer for continuous signed distance function. Our method enables geometric reasoning with strong generalization capability. With a random shape code z_0 initialized in the learned shape space, we can acquire high-quality 3D shape prediction by performing iterative optimization with various 2D supervisions.

In this paper, we propose a differentiable renderer for continuous implicit signed distance functions (SDF) to facilitate the 3D shape understanding via geometric reasoning in a deep learning framework (Fig. 1). Our method can render an implicit SDF represented by a neural network from a latent code into various 2D observations, *e.g.*, depth images, surface normals, silhouettes, and other properties encoded, from arbitrary camera viewpoints. The rendering process is fully differentiable, such that loss functions

*Work done while Shaohui Liu was an academic guest at ETH Zurich.
†Corresponding author.

can be conveniently defined on the rendered images and the observations, and the gradients can be propagated back to the shape generator. As major applications, our differentiable renderer can be applied to infer the 3D shape from various inputs, *e.g.*, multi-view images and single depth image, through an inverse graphics process. Specifically, given a pre-trained generative model, *e.g.*, DeepSDF [35], we search within the latent code space for the 3D shape that produces the rendered images mostly consistent with the observation. Extensive experiments show that our geometric reasoning based approach exhibits significantly better generalization capability than previous purely learning based approaches, and consistently produce accurate 3D shapes across datasets without finetuning.

Nevertheless, it is challenging to make differentiable rendering work on a learning-based implicit SDF with computationally affordable resources. The main obstacle is that an implicit function provides neither the exact location nor any bound of the surface geometry as in other representations like meshes, voxels, and point clouds.

Inspired by traditional ray-tracing based approaches, we adopt the sphere tracing algorithm [14], which marches along each pixel's ray direction with the queried signed distance until the ray hits the surface, *i.e.*, the signed distance equals to zero (Fig. 2). However, this is not feasible in the neural network based scenario where each query on the ray would require a forward pass and recursive computational graph for back-propagation, which is prohibitive in terms of computation and memory.

To make it work efficiently on a commodity level GPU, we optimize the full life-time of the rendering process for both forward and backward propagations. In the forward pass, *i.e.*, the rendering process, we adopt a coarse-to-fine approach to save computation at initial steps, an aggressive strategy to speed up the ray marching, and a safe convergence criteria to prevent unnecessary queries and maintain resolution. In the backward propagation, we propose a gradient approximation which empirically has negligible impact on the training performance but dramatically reduces the computation and memory consumption. By making the rendering tractable, we show how producing 2D observations with the sphere tracing and interacting with camera extrinsics can be done in differentiable ways.

To sum up, our major contribution is to **enable efficient differentiable rendering on the implicit signed distance function represented as a neural network**. It enables accurate 3D shape prediction via geometric reasoning in deep learning frameworks and exhibits promising generalization capability. The differentiable renderer could also potentially benefit various vision problems thanks to the marriage of implicit SDF and inverse graphics techniques. The code and data are available at https://github.com/B1ueber2y/DIST-Renderer.

2. Related Work

3D Representation for Shape Learning. The study of 3D representations for shape learning is one of the main focuses in 3D deep learning community. Early work quantizes shapes into 3D voxels, where each voxel contains either a binary occupancy status (occupied / not occupied) [53, 6, 47, 40, 13] or a signed distance value [56, 9, 46]. While voxels are the most straightforward extension from the 2D image domain into the 3D geometry domain for neural network operations, they normally require huge memory overhead and result in relatively low resolutions. Meshes are also proposed as a more memory efficient representation for 3D shape learning [48, 12, 23, 21], but the topology of meshes is normally fixed and simple. Many deep learning methods also utilize point clouds as the 3D representation [38, 39]; however, the point-based representation lacks the topology information and thus makes it non-trivial to generate 3D meshes. Very recently, the implicit functions, *e.g.*, continuous SDF and occupancy function, are exploited as 3D representations and show much promising performance in terms of the high-frequency detail modeling and the high resolution [35, 30, 31, 4]. Similar idea has also been used to encode other information such as textures [34, 41] and 4D dynamics [33]. Our work aims to design an efficient and differentiable renderer for the implicit SDF-based representation.

Differentiable Rendering. With the success of deep learning, the differentiable rendering starts to draw more attention as it is essential for end-to-end training, and solutions have been proposed for various 3D representations. Early works focus on 3D triangulated meshes and leverage standard rasterization [29]. Various approaches try to solve the discontinuity issue near triangle boundaries by smoothing the loss function or approximating the gradient [22, 37, 26, 3]. Solutions for point clouds and 3D voxels are also introduced [49, 18, 32] to work jointly with PointNet [38] and 3D convolutional architectures. However, the differentiable rendering for the implicit continuous function representation does not exist yet. Some ray tracing based approaches are related, while they are mostly proposed for explicit representation, such as 3D voxels [28, 32, 44, 19] or meshes [24], but not implicit functions. Liu *et al*. [27] firstly propose to learn from 2D observations over occupancy networks [30]. However, their methods make several approximations and do not benefit from the efficiency of rendering implicit SDF. Most related to our work, Sitzmann *et al*. [45] propose a LSTM-based renderer for an implicit scene representation to generate color images, while their model focuses on simulating the rendering process with an LSTM without clear geometric meaning. This method can only generate low-resolution images due to the expensive memory consumption. Alternatively, our method can directly render 3D geometry represented by an implicit SDF

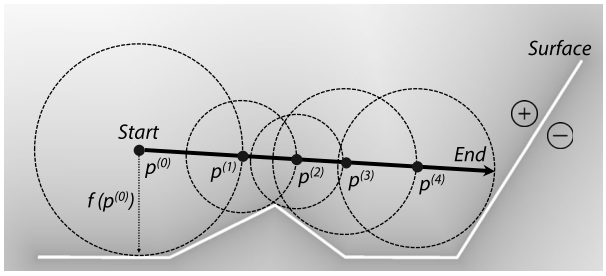


Figure 2. Illustration on the sphere tracing algorithm [14]. A ray is initiated at each pixel and marching along the viewing direction. The front end moves with a step size equals to the signed distance value of the current location. The algorithm converges when the current absolute SDF is smaller than a threshold, which indicates that the surface has been found.

to produce high-resolution images. It can also be applied without training to existing deep learning models.

3D Shape Prediction. 3D shape prediction from 2D observations is one of the fundamental vision problems. Early works mainly focus on multi-view reconstruction using multi-view stereo methods [42, 15, 43]. These purely geometry-based methods suffer from degraded performance on texture-less regions without prior knowledge [7]. With progress of deep learning, 3D shapes can be recovered under different settings. The simplest setting is to recover 3D shape from a single image [6, 11, 52, 20]. These systems rely heavily on priors, and are prone to weak generalization. Deep learning based multi-view shape prediction methods [54, 16, 17, 50, 51] further involve geometric constraints across views in the deep learning framework, which shows better generalization. Another thread of works [9, 8] take a single depth image as input, and the problem is usually referred as shape completion. Given the shape prior encoded in the neural network [35], our rendering method can effectively predict accurate 3D object shape from a random initial shape code with various inputs, such as depth and multi-view images, through geometric optimization.

3. Differentiable Sphere Tracing

In this section, we introduce our differentiable rendering method for the implicit signed distance function represented as a neural network, such as DeepSDF [35]. In DeepSDF, a network takes a latent code and a 3D location as input, and produces the corresponding signed distance value. Even though such a network can deliver high quality geometry, the explicit surface cannot be directly obtained and requires dense sampling in the 3D space.

Our method is inspired by Sphere Tracing [14] designed for rendering SDF volumes, where rays are shot from the camera pinhole along the direction of each pixel to search for the surface level set according to the signed distance value. However, it is prohibitive to apply this method directly on the implicit signed distance function represented as a neural network, since each tracing step needs a feed-forward neural network and the whole algorithm requires unaffordable computational and memory resources. To make this idea work in deep learning framework for inverse graphics, we optimize both the forward and backward propagations for efficient training and test-time optimization. The sphere traced results, *i.e.*, the distance along the ray, can be converted into many desired outputs, *e.g.*, depth, surface normal, silhouette, and hence losses can be conveniently applied in an end-to-end manner.

Algorithm 1 Naive sphere tracing algorithm for a camera ray $L : \mathbf{c} + d\tilde{\mathbf{v}}$ over a signed distance fields $f : \mathbb{N}^3 \rightarrow \mathbb{R}$.

1: Initialize $n = 0$, $d^{(0)} = 0$, $\mathbf{p}^{(0)} = \mathbf{c}$.
2: **while** not converged **do**:
3: Take the corresponding SDF value $b^{(n)} = f(\mathbf{p}^{(n)})$ of the location $\mathbf{p}^{(n)}$ and make update: $d^{(n+1)} = d^{(n)} + b^{(n)}$.
4: $\mathbf{p}^{(n+1)} = \mathbf{c} + d^{(n+1)}\tilde{\mathbf{v}}$, $n = n + 1$.
5: Check convergence.
6: **end while**

3.1. Preliminaries - Sphere Tracing

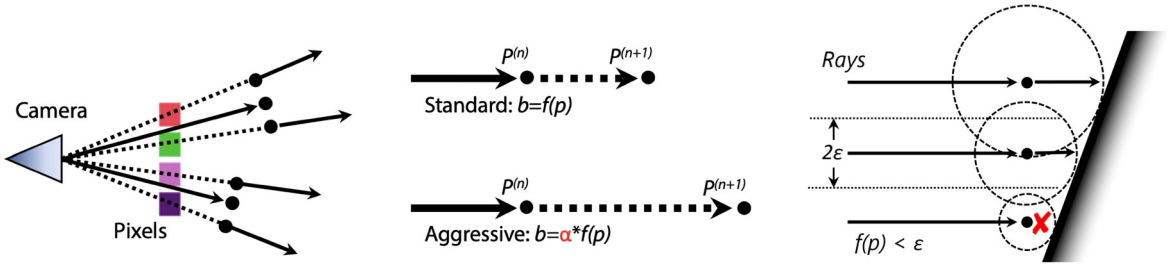
To be self-contained, we first briefly introduce the traditional sphere tracing algorithm [14]. Sphere tracing is a conventional method specifically designed to render depth from volumetric signed distance fields. For each pixel on the image plane, as shown in Figure 2, a ray (L) is shot from the camera center ($\mathbf{c}$) and marches along the direction ($\tilde{\mathbf{v}}$) with a step size that is equal to the queried signed distance value (b). The ray marches iteratively until it hits or gets sufficiently close to the surface (*i.e.* abs(SDF) $<$ threshold). A more detailed algorithm can be found in Algorithm 1.

3.2. Efficient Forward Propagation

Directly applying sphere tracing to an implicit SDF function represented by a neural network is prohibitively computational expensive, because each query of f requires a forward pass of a neural network with considerable capacity. Naive parallelization is not sufficient since essentially millions of network queries are required for a single rendering with VGA resolution (640×480). Therefore, we need to cut off unnecessary marching steps and safely speed up the marching process.

Initialization. Because all the 3D shapes represented by DeepSDF are bounded within the unit sphere, we initialize $\mathbf{p}^{(0)}$ to be the intersection between the camera ray and the unit sphere for each pixel. Pixels with the camera rays that do not intersect with the unit sphere are set as background (*i.e.*, infinite depth).

Coarse-to-fine Strategy. At the beginning of sphere tracing, rays for different pixels are fairly close to each other, which indicates that they will likely march in a similar way.



(a) Coarse-to-fine Strategy (b) Aggressive Marching (c) Convergence Criteria

Figure 3. Strategies for our efficient forward propagation. (a) 1D illustration of our coarse-to-fine strategy, and for 2D cases, one ray will be spitted into 4 rays; (b) Comparison of standard marching and our aggressive marching; (c) We stop the marching once the SDF value is smaller than ϵ, where 2ϵ is the estimated minimal distance between the corresponding 3D points of two neighboring pixels.

To leverage this nice property, we propose a coarse-to-fine sphere tracing strategy as shown in Fig. 3 (a). We start the sphere tracing from an image with $\frac{1}{4}$ of its original resolution, and split each ray into four after every three marching steps, which is equivalent to doubling the resolution. After six steps, each pixel in the full resolution has a corresponding ray, which keeps marching until convergence.

Aggressive Marching. After the ray marching begins, we apply an aggressive strategy (Fig. 3 (b)) to speed up the marching process by updating the ray with α times of the queried signed distance value, where $\alpha = 1.5$ in our implementation. This aggressive sampling has several benefits. First, it makes the ray march faster towards the surface, especially when it is far from surface. Second, it accelerates the convergence for the ill-posed condition, where the angle between the surface normal and the ray direction is small. Third, the ray can pass through the surface such that space in the back (*i.e.*, SDF < 0) could be sampled. This is crucially important to apply supervision on both sides of the surface during optimization.

Dynamic Synchronized Inference. A naive parallelization for speeding up sphere tracing is to batch rays together and synchronously update the front end positions. However, depending on the 3D shape, some rays may converge earlier than others, thus leading to wasted computation. We maintain a dynamic unfinished mask indicating which rays require further marching to prevent unnecessary computation.

Convergence Criteria. Even with aggressive marching, the ray movement can be extremely slow when close to the surface since f is close to zero. We define a convergence criteria to stop the marching when the accuracy is sufficiently good and the gain is marginal (Fig. 3(c)). To fully maintain details supported by the 2D rendering resolution, it is sufficiently safe to stop when the sampled signed distance value does not confuse one pixel with its neighbors. For an object with a smallest distance of $100mm$ captured by a camera with $60mm$ focal length, $32mm$ sensor width, and a resolution of 512×512, the approximate minimal distance between the corresponding 3D points of two neighboring pixels is $10^{-4}m$ $(0.1mm)$. In practice, we set the convergence threshold ϵ as 5×10^{-5} for most of our experiments.

3.3. Rendering 2D Observations

After all rays converge, we can compute the distance along each ray as the following:

$$d = \alpha \sum_{n=0}^{N-1} f(\mathbf{p}^{(n)}) + (1-\alpha) f(\mathbf{p}^{(N-1)}) = d' + e, \quad (1)$$

where $e = (1-\alpha) f(\mathbf{p}^{(N-1)})$ is the residual term on the last query. In the following part we will show how this computed ray distance is converted into 2D observations.

Depth and Surface Normal. Suppose that we find the 3D surface point $\mathbf{p} = \mathbf{c} + d\tilde{\mathbf{v}}$ for a pixel (x, y) in the image, we can directly get the depth for each pixel as the following:

$$z_c = \frac{d}{\sqrt{\tilde{x}^2 + \tilde{y}^2 + 1}}, \quad (2)$$

where $(\tilde{x}, \tilde{y}, 1)^\top = K^{-1}(x, y, 1)^\top$ is the normalized homogeneous coordinate.

The surface normal of the point $\mathbf{p}(x, y, z)$ can be computed as the normalized gradient of the function f. Since f is an implicit signed distance function, we take the approximation of the gradient by sampling neighboring locations:

$$\mathbf{n} = \frac{1}{2\delta} \begin{bmatrix} f(x+\delta, y, z) - f(x-\delta, y, z) \\ f(x, y+\delta, z) - f(x, y-\delta, z) \\ f(x, y, z+\delta) - f(x, y, z-\delta) \end{bmatrix}, \tilde{\mathbf{n}} = \frac{\mathbf{n}}{|\mathbf{n}|}. \quad (3)$$

Silhouette. The silhouette is a commonly used supervision for 3D shape prediction. To make the rendering of silhouettes differentiable, we get the minimum absolute signed distance value for each pixel along its ray and subtract it by the convergence threshold ϵ. This produces a tight approximation of the silhouette, where pixels with positive values belong to the background, and vice versa. Note that directly checking if ray marching stops at infinity can also generate the silhouette but it is not differentiable.

Color and Semantics. Recently, it has been shown that texture can also be represented as an implicit function parameterized with a neural network [34]. Not only color, other

Method	size	#step	#query	time
Naive sphere tracing	512^2	50	N/A	N/A
+ practical grad.	512^2	50	6.06M	1.6h
+ parallel	512^2	50	6.06M	3.39s
+ dynamic	512^2	50	1.99M	1.23s
+ aggressive	512^2	50	1.43M	1.08s
+ coarse-to-fine	512^2	50	**887K**	**0.99s**
+ coarse-to-fine	512^2	100	**898K**	**1.24s**

Table 1. Ablation studies on the cost-efficient feedforward design of our method. The average time for each optimization step was tested on a single NVIDIA GTX-1080Ti over the architecture of DeepSDF [35]. Note that the number of initialized rays is quadratic to the image size, and the numbers are reported for the resolution of 512×512.

spatially varying properties, like semantics, material, etc, can all be potentially learned by implicit functions. These information can be rendered jointly with the implicit SDF to produce corresponding 2D observations, and some examples are depicted in Fig. 8.

3.4. Approximated Gradient Back-Propagation

DeepSDF [35] uses the conditional implicit function to represent a 3D shape as $f_\theta(\mathbf{p}, \mathbf{z})$, where θ is the network parameters, and $\mathbf{z}$ is the latent code representing a certain shape. As a result, each queried point $\mathbf{p}$ in the sphere tracing process is determined by θ and the shape code $\mathbf{z}$, which requires to unroll the network for multiple times and costs huge memory for back-propagation with respect to $\mathbf{z}$:

$$\begin{aligned}\frac{\partial d'}{\partial \mathbf{z}}|_{\mathbf{z}_0} &= \alpha \sum_{i=0}^{N-1} \frac{\partial f_\theta(\mathbf{p}^{(i)}(\mathbf{z}), \mathbf{z})}{\partial \mathbf{z}}|_{\mathbf{z}_0} \\ &= \alpha \sum_{i=0}^{N-1} \left(\frac{\partial f_\theta(\mathbf{p}^{(i)}(\mathbf{z_0}), \mathbf{z})}{\partial \mathbf{z}} + \frac{\partial f_\theta(\mathbf{p}^{(i)}(\mathbf{z}), \mathbf{z_0})}{\partial \mathbf{p}^{(i)}(\mathbf{z})} \frac{\partial \mathbf{p}^{(i)}(\mathbf{z_0})}{\partial \mathbf{z}}\right).\end{aligned} \tag{4}$$

Practically, we ignore the gradients from the residual term e in Equation (1). In order to make back-propagation feasible, we define a loss for K samples with the minimum absolute SDF value on the ray to encourage more signals near the surface. For each sample, we calculate the gradient with only the first term in Equation (4) as the high-order gradients empirically have less impact on the optimization process. In this way, our differentiable renderer is particularly useful to bridge the gap between this strong shape prior and some partial observations. Given a certain observation, we can search for the code that minimizes the difference between the rendering from our network and the observation. This allows a number of applications which will be introduced in the next section.

4. Experiments and Results

In this section, we first verify the efficacy of our differentiable sphere tracing algorithm, and then show that 3D shape understanding can be achieved through geometry based reasoning by our method.

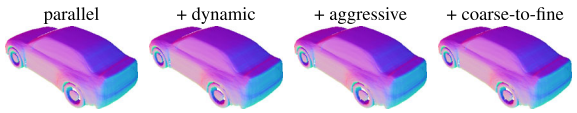


Figure 4. The surface normal rendered with different speed up strategies turned on. Note that adding up these components does not deteriorate the rendering quality.

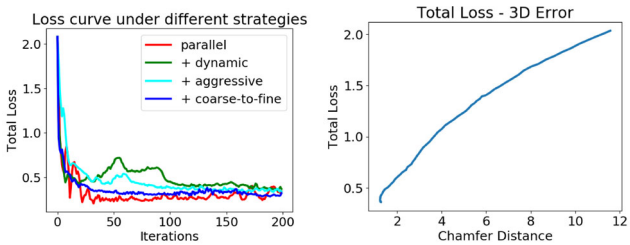


Figure 5. Loss curves for 3D prediction from partial depth. Our accelerated rendering does not impair the back-propagation. The loss on the depth image is tightly correlated with the Chamfer distance on 3D shapes, which indicates effective back-propagation.

4.1. Rendering Efficiency and Quality

Run-time Efficiency. In this section, we evaluate the runtime efficiency promoted by each design in our differentiable sphere tracing algorithm. The number of queries and runtime for both forward and backward passes at a resolution of 512×512 on a single NVIDIA GTX-1080Ti are reported in Tab. 1, and the corresponding rendered surface normal are shown in Fig. 4. We can see that the proposed back-propagation prunes the graph and reduces the memory usage significantly, making the rendering tractable with a standard graphics card. The dynamic synchronized inference, aggressive marching and coarse-to-fine strategy all speed up rendering. With all these designs, we can render an image with only 887K query steps within 0.99s when the maximum tracing step is set to 50. The number of query steps only increases slightly when the maximum step is set to 100, indicating that most of the pixels converge safely within 50 steps. Note that related works usually render at a much lower resolution [45].

Back-Propagation Effectiveness. We conduct sanity checks to verify the effectiveness of the back-propagation with our approximated gradient. We take a pre-trained DeepSDF [35] model and run geometry based optimization to recover the 3D shape and camera extrinsics separately using our differentiable renderer. We first assume camera pose is known and optimize the latent code for 3D shape w.r.t the given ground truth depth map, surface normal and silhouette. As can be seen in Fig. 5 (left), the loss drops quickly, and using acceleration strategies does not hurt the optimization. Fig. 5 (right) shows the total loss on the 2D image plane is highly correlated with the Chamfer distance on the

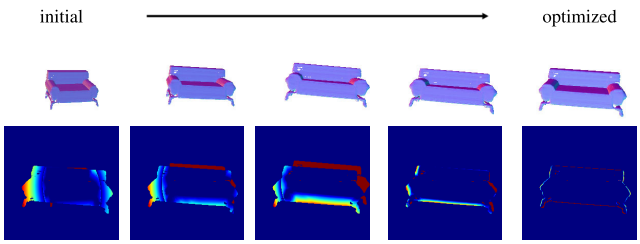


Figure 6. Illustration of the optimization process over the camera extrinsic parameters. Our differentiable renderer is able to propagate the error from the image plane to the camera. Top row: rendered surface normal. Bottom row: error map on the silhouette.

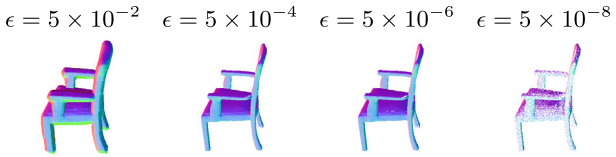


Figure 7. Effects on choices of different convergence thresholds. Under the same marching step, a very large threshold can incur dilation around boundaries while a small threshold may lead to erosion. We pick 5×10^{-5} for all of our experiments.

predicted 3D shape, indicating that the gradients originated from the 2D observation are successfully back-propagated to the shape. We then assume a known shape (fixed latent code) and optimize the camera pose using a depth image and a binary silhouette. Fig. 6 shows that a random initial camera pose can be effectively optimized toward the ground truth pose by minimizing the gradients on 2D observations.

Convergence Criteria. The convergence criteria, *i.e.*, the threshold on signed distance to stop the ray tracing, has a direct impact on the rendering quality. Fig. 7 shows the rendering result under different thresholds. As can be seen, rendering with a large threshold will dilate the shape, which lost boundary details. Using a small threshold, on the other hand, may produces incomplete geometry. This parameter can be tuned according to applications, but in practice we found our threshold is effective in producing complete shape with details up to the image resolution.

Rendering Other Properties. Not only the signed distance function for 3D shape, implicit functions can also encode other spatially variant information. As an example, we train a network to predict both signed distance and color for each 3D location, and this grants us the capability of rendering color images. In Fig. 8, we show that with a 512-dim latent code learned from textured meshes as the ground truth, color images can be rendered in arbitrary resolution, camera viewpoints, and illumination. Note that the latent code size is significantly smaller than the mesh (vertices+triangles+texture map), and thus can be potentially used for model compression. Other per-vertex properties, such as semantic segmentation and material, can also be rendered in the same differentiable way.

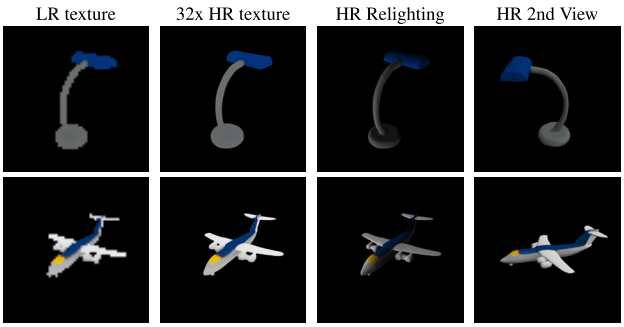


Figure 8. Our method can render information encoded in the implict function other than depth. With a pre-trained network encoding textured meshes, we can render high resolution color images under various resolution, camera viewpoints, and illumination.

4.2. 3D Shape Prediction

Our differentiable implicit SDF renderer builds up the connection between 3D shape and 2D observations and enables geometry based reasoning. In this section, we show results of 3D shape prediction from a single depth image, or multi-view color images using DeepSDF as the shape generator. On a high-level, we take a pre-trained DeepSDF and fixed the decoder parameters. When given 2D observations, we define proper loss functions and propagate the gradient back to the latent code, as introduced in Section 3.4, to generate 3D shape. This method does not require any additional training and only need to run optimization at test time, which is intuitively less vulnerable to overfitting or domain gap issues in pure learning based approach. In this section, we specifically focus on evaluating the generalization capability while maintaining high shape quality.

4.2.1 3D Shape Prediction from Single Depth Image

With the development of commodity range sensors, the dense or sparse depth images can be easily acquired, and several methods have been proposed to solve the problem of 3D shape prediction from a single depth image. DeepSDF [35] has shown state-of-the-art performance for this task, however requires an offline pre-processing to lift the input 2D depth map into 3D space in order to sample the SDF values with the assistance of the surface normal. Our differentiable render makes 3D shape prediction from a depth image more convenient by directly rendering the depth image given a latent code and comparing it with the given depth. Moreover, with the silhouette calculated from the depth map or provided from the rendering, our renderer can also leverage it as an additional supervision. Formally, we obtain the complete 3D shape by solving the following optimization:

$$\arg\min_{\mathbf{z}} \mathcal{L}_d(\mathcal{R}_d(f(\mathbf{z})), I_d) + \mathcal{L}_s(\mathcal{R}_s(f(\mathbf{z})), I_s), \qquad (5)$$

where $f(\mathbf{z})$ is the pre-trained neural network encoding shape priors, $\mathcal{R}_d$ and $\mathcal{R}_s$ represent the rendering function

	dense	50%	10%	100pts	50pts	20pts
sofa						
DeepSDF	5.37	5.56	5.50	5.93	6.03	7.63
Ours	**4.12**	5.75	5.49	5.72	5.57	6.95
Ours (mask)	**4.12**	**3.98**	**4.31**	**3.98**	**4.30**	**4.94**
plane						
DeepSDF	3.71	3.73	4.29	4.44	4.40	5.39
Ours	**2.18**	4.08	4.81	4.44	4.51	5.30
Ours (mask)	**2.18**	**2.08**	**2.62**	**2.26**	**2.55**	**3.60**
table						
DeepSDF	12.93	12.78	11.67	12.87	13.76	15.77
Ours	**5.37**	12.05	11.42	11.70	13.76	15.83
Ours (mask)	**5.37**	**5.15**	**5.16**	**5.26**	**6.33**	**7.62**

Table 2. Quantitative comparison between our geometric optimization with DeepSDF [35] for shape completion over partial dense and sparse depth observation on ShapeNet dataset [2]. We report the median Chamfer Distance on the first 200 instances of the dataset of [6]. We give DeepSDF [35] the groundtruth normal otherwise they could not be applied on the sparse depth.

for the depth and silhouette respectively, $\mathcal{L}_d$ is the L_1 loss of depth observation, and $\mathcal{L}_s$ is the loss defined based on the differentiably rendered silhouette. In our experiment, the initial latent shape $\mathbf{z}_0$ is chosen as the mean shape.

We test our method and DeepSDF [35] on 200 models on plane, sofa and table category respectively from ShapeNet Core [2]. Specifically, for each model, we use the first camera in the dataset of Choy *et al*. [6] to generate dense depth images for testing. The comparison between DeepSDF and our method is listed in Tab. 2. We can see that our method with only depth supervision performs better than DeepSDF [35] when dense depth image is given. This is probably because that DeepSDF samples the 3D space with pre-defined rule (at fixed distances along the normal direction), which may not necessarily sample correct location especially near object boundary or thin structures. In contrast, our differentiable sphere tracing algorithm samples the space adaptively with the current estimation of shape.

Robustness against sparsity. The depth from laser scanners can be very sparse, so we also study the robustness of our method and DeepSDF against sparse depth. The results are shown in Tab. 2. Specifically, we randomly sample different percentages or fixed numbers of points from the original dense depth for testing. To make a competitive baseline, we provide DeepSDF ground truth normals to sample SDF, since it cannot be reliably estimated from sparse depth. From the table, we can see that even with very sparse depth observations, our method still recovers accurate shapes and gets consistently better performance than DeepSDF with additional normal information. When the silhouette is available, our method achieves significantly better performance and robustness against the sparsity, indicating that our rendering method can back-propagate gradients effectively from the silhouette loss.

Figure 9. Illustration of the optimization process under multi-view setup. Our differentiable renderer is able to successfully recover 3D geometry from a random code with only the photometric loss.

Method	car	plane
PMO (original)	0.661	1.129
PMO (rand init)	1.187	6.124
Ours (rand init)	0.919	1.595

Table 3. Quantitative results on 3D shape prediction from multi-view images under the metric of Chamfer Distance (only in the direction of gt→pred for fair comparison). We randomly picked 50 instances from the PMO test set to perform the evaluation. 10000 points are sampled from meshes for evaluation.

4.2.2 3D Shape Prediction from Multiple Images

Our differentiable renderer can also enable geometry based reasoning for shape prediction from multi-view color images by leveraging cross-view photometric consistency.

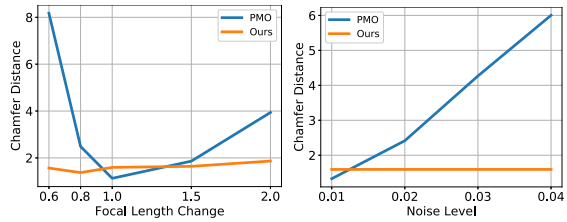
Specifically, we first initialize the latent code with a random vector and render a depth image for each of the input views. We then warp each color image to other input views using the rendered depth image and the known camera pose. The difference between the warped and input images are then defined as the photometric loss, and the shape can be predicted by minimizing this loss. To sum up, the optimization problem is formulated as follows,

$$\arg\min_{\mathbf{z}} \sum_{i=0}^{N-1} \sum_{j \in \mathcal{N}_i} \|I_i - I_{j\to i}(\mathcal{R}_d^i(f(\mathbf{z}))\|, \qquad (6)$$

where $\mathcal{R}_d^i$ represents the rendered depth image at view i, $\mathcal{N}_i$ are the neighboring images of I_i, and $I_{j\to i}$ is the warped image from view j to view i using the rendered depth. Note that no mask is required under the multi-view setup. Fig. 9 shows an example of the optimization process of our method. As can be seen, the shape is gradually improved while the loss is being optimized.

We take PMO [25] as a competitive baseline, since they also perform deep learning based geometric reasoning via optimization over a pre-trained decoder, but use the triangular mesh representation. Their model first predicts an initial mesh from a selected input view and improve the quality via cross-view photo-consistency. Both the synthetic and real datasets provided in [25] are used for evaluation.

In Tab. 3, we show quantitative comparison to PMO on their synthetic test set. It can be seen that our method achieves comparable results with PMO [25] from only random initializations. Note that while PMO uses both the encoder and decoder trained on the PMO training set, our DeepSDF decoder was neither trained nor finetuned on it.



(a) (b)

Figure 10. Robustness of geometric reasoning via multi-view photometric optimization. (a) Performance w.r.t changes on camera focal length. (b) Performance w.r.t noise in the initialization code. Our model is robust against focal length change and not affected by noise in the latent code since we start from random initialization. In contrast, PMO is very sensitive to both factors, and the performance drops significantly when the testing images are different from the training set.

Besides, if the shape code for PMO, instead of being predicted from their trained image encoder, is also initialized randomly, their performance decreases dramatically, which indicates that with our rendering method, our geometric reasoning becomes more effective. Our method can be further improved with good initialization.

Generalization Capability To further evaluate the generalization capability, we compare to PMO on some unseen data and initialization. We first evaluate both methods on a testing set generated using different camera focal lengths, and the quantitative comparison is in Fig. 10 (a). It clearly shows that our method generalizes well to the new images, while PMO suffers from overfitting or domain gap. To further test the effectiveness of the geometric reasoning, we also directly add random noise to the initial latent code. The performance of PMO again drops significantly, while our method is not affected since the initialization is randomized (Fig. 10 (b)). Some qualitative results are shown in Fig. 11. Our method produces accurate shapes with detailed surfaces. In contrast, PMO suffers from two main issues: 1) the low resolution mesh is not capable of maintaining geometric details; 2) their geometric reasoning struggles with the initialization from image encoder.

We further show comparison on real data in Fig. 12. Following PMO, since the provided initial similarity transformation is not accurate in some cases, we also optimize over the similarity transformation in addition to the shape code. As can be seen, both methods perform worse on this challenging dataset. In comparison, our method produces shapes with higher quality and correct structures, while PMO only produce very rough shapes. Overall, our method shows better generalization capability and robustness against domain change.

5. Conclusion

We propose a differentiable sphere tracing algorithm to render 2D observations such as depth maps, normals, silhouettes, from implicit signed distance functions parameterized as a neural network. This enables geometric reasoning in 3D shape prediction from both single and multiple views in conjunction with the high capacity 3D neural representation. Extensive experiments show that our geometry based optimization algorithm produces 3D shapes that are more accurate than SOTA, generalizes well to new datasets, and is robust to imperfect or partial observations. Promising directions to explore using our renderer include self-supervised learning, recovering other properties jointly with geometry, and neural image rendering.

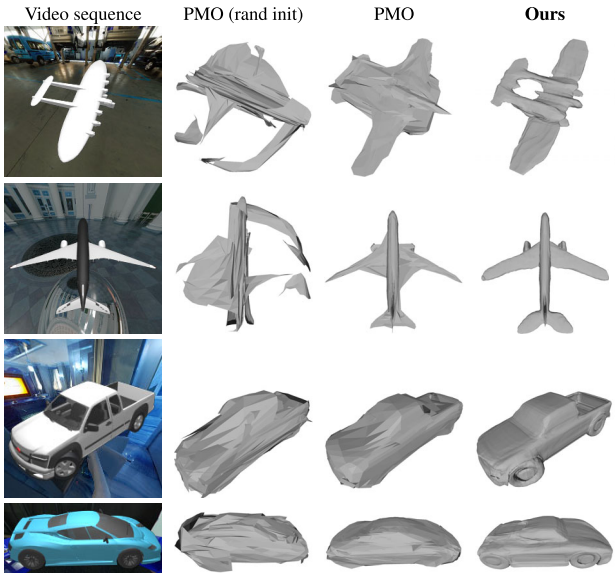


Figure 11. Comparison on 3D shape prediction from multi-view images on the PMO test set. Our method maintains good surface details, while PMO suffers from the mesh representation and may not effectively optimize the shape.

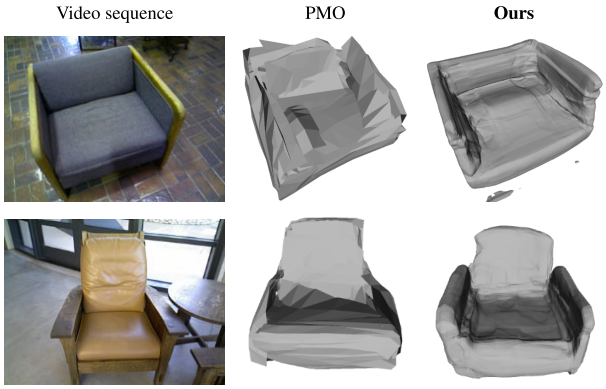


Figure 12. Comparison on 3D shape prediction from multi-view images on real-world dataset [5]. It is in general challenging for shape prediction on real image. Comparatively, our method produces more reasonable results with correct structure.

Acknowledgements

This work is partly supported by National Natural Science Foundation of China under Grant No. 61872012, National Key R&D Program of China (2019YFF0302902), and Beijing Academy of Artificial Intelligence (BAAI).

References

[1] Bruce Guenther Baumgart. Geometric modeling for computer vision. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1974. 1

[2] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 7

[3] Wenzheng Chen, Jun Gao, Huan Ling, Edward J Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *Proc. of Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 2

[4] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[5] Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. A large dataset of object scans. *arXiv preprint arXiv:1602.02481*, 2016. 8, 20

[6] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Proc. of European Conference on Computer Vision (ECCV)*. Springer, 2016. 2, 3, 7

[7] Zhaopeng Cui, Jinwei Gu, Boxin Shi, Ping Tan, and Jan Kautz. Polarimetric multi-view stereo. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2017. 3

[8] Angela Dai and Matthias Nießner. Scan2mesh: From unstructured range scans to 3d meshes. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, pages 5574–5583, 2019. 3

[9] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. Shape completion using 3d-encoder-predictor cnns and shape synthesis. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 3

[10] 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. 13

[11] Rohit Girdhar, David F Fouhey, Mikel Rodriguez, and Abhinav Gupta. Learning a predictable and generative vector representation for objects. In *Proc. of European Conference on Computer Vision (ECCV)*. Springer, 2016. 3

[12] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[13] Christian Häne, Shubham Tulsiani, and Jitendra Malik. Hierarchical surface prediction for 3d object reconstruction. In *Proc. of International Conference on 3D Vision (3DV)*. IEEE, 2017. 2

[14] John C Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10), 1996. 2, 3

[15] Carlos Hernandez, George Vogiatzis, and Roberto Cipolla. Multiview photometric stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(3):548–554, 2008. 3

[16] Po-Han Huang, Kevin Matzen, Johannes Kopf, Narendra Ahuja, and Jia-Bin Huang. Deepmvs: Learning multi-view stereopsis. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, pages 2821–2830, 2018. 3

[17] Sunghoon Im, Hae-Gon Jeon, Stephen Lin, and In So Kweon. Dpsnet: end-to-end deep plane sweep stereo. In *Proc. of International Conference on Learning Representations (ICLR)*, 2019. 3

[18] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. In *Proc. of Advances in Neural Information Processing Systems (NeurIPS)*, 2018. 2

[19] Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2020. 2

[20] Adrian Johnston, Ravi Garg, Gustavo Carneiro, Ian Reid, and Anton van den Hengel. Scaling cnns for high resolution volumetric reconstruction from a single image. In *Proc. of International Conference on Computer Vision (ICCV)*, pages 939–948, 2017. 3

[21] Angjoo Kanazawa, Michael J Black, David W Jacobs, and Jitendra Malik. End-to-end recovery of human shape and pose. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[22] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[23] Chen Kong, Chen-Hsuan Lin, and Simon Lucey. Using locally corresponding cad models for dense 3d reconstructions from a single image. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[24] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. In *Proc. of ACM SIGGRAPH*, page 222. ACM, 2018. 1, 2

[25] 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. 7, 14

[26] Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. Soft rasterizer: Differentiable rendering for unsupervised single-view mesh reconstruction. In *Proc. of International Conference on Computer Vision (ICCV)*, 2019. 2

[27] Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. Learning to infer implicit surfaces without 3d supervision.

In *Proc. of Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 2

[28] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *Proc. of ACM SIGGRAPH*, 2019. 2

[29] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *Proc. of European Conference on Computer Vision (ECCV)*, pages 154–169. Springer, 2014. 2

[30] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[31] Mateusz Michalkiewicz, Jhony K Pontes, Dominic Jack, Mahsa Baktashmotlagh, and Anders Eriksson. Implicit surface representations as layers in neural networks. In *Proc. of International Conference on Computer Vision (ICCV)*, 2019. 2

[32] Thu H Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yongliang Yang. Rendernet: A deep convolutional network for differentiable rendering from 3d shapes. In *Proc. of Advances in Neural Information Processing Systems (NeurIPS)*, 2018. 2

[33] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Occupancy flow: 4d reconstruction by learning particle dynamics. In *Proc. of International Conference on Computer Vision (ICCV)*, 2019. 2

[34] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *Proc. of International Conference on Computer Vision (ICCV)*, 2019. 2, 4

[35] 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, 2, 3, 5, 6, 7, 11, 13, 14, 16, 17, 18

[36] Gustavo Patow and Xavier Pueyo. A survey of inverse rendering problems. In *Computer graphics forum*, volume 22, pages 663–687. Wiley Online Library, 2003. 1

[37] Felix Petersen, Amit H Bermano, Oliver Deussen, and Daniel Cohen-Or. Pix2vex: Image-to-geometry reconstruction using a smooth differentiable renderer. *arXiv preprint arXiv:1903.11149*, 2019. 2

[38] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[39] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proc. of Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 2

[40] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[41] Shunsuke Saito, Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. PIFu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Proc. of International Conference on Computer Vision (ICCV)*, 2019. 2

[42] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2006. 3

[43] Ben Semerjian. A new variational framework for multiview surface reconstruction. In *Proc. of European Conference on Computer Vision (ECCV)*. Springer, 2014. 3

[44] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[45] 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. 2, 5, 13

[46] David Stutz and Andreas Geiger. Learning 3d shape completion under weak supervision. *International Journal of Computer Vision (IJCV)*, pages 1–20, 2018. 2

[47] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *Proc. of International Conference on Computer Vision (ICCV)*, 2017. 2

[48] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proc. of European Conference on Computer Vision (ECCV)*, 2018. 2

[49] Yifan Wang, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *Proc. of ACM SIGGRAPH Asia*, 2019. 2

[50] Yi Wei, Shaohui Liu, Wang Zhao, and Jiwen Lu. Conditional single-view shape generation for multi-view stereo reconstruction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 3

[51] Chao Wen, Yinda Zhang, Zhuwen Li, and Yanwei Fu. Pixel2mesh++: Multi-view 3d mesh generation via deformation. In *Proc. of International Conference on Computer Vision (ICCV)*, 2019. 3

[52] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Proc. of Advances in Neural Information Processing Systems (NeurIPS)*, 2016. 3

[53] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2015. 2

[54] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. Mvsnet: Depth inference for unstructured multi-view stereo. In *Proc. of European Conference on Computer Vision (ECCV)*, pages 767–783, 2018. 3

[55] Yizhou Yu, Paul Debevec, Jitendra Malik, and Tim Hawkins. Inverse global illumination: Recovering reflectance models of real scenes from photographs. volume 99, pages 215–224, 1999. 1

[56] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

Appendix

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. 13, 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:

$$|d(1-\alpha sin\theta)^k| < \epsilon, \tag{7}$$

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,

$$k > k_{min} = \frac{log\epsilon - logd}{log|1-\alpha sin\theta|}. \tag{8}$$

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.

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. 14, in traditional ray marching, the front end of the ray approaches the surface from the camera side

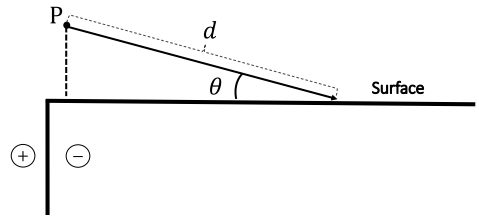


Figure 13. Illustration of ill-posed conditions for sphere tracing algorithm. When θ is relatively small, the queried SDF is much lower than the actual ray distance d towards the surface, making the sphere tracing process slower than expected.

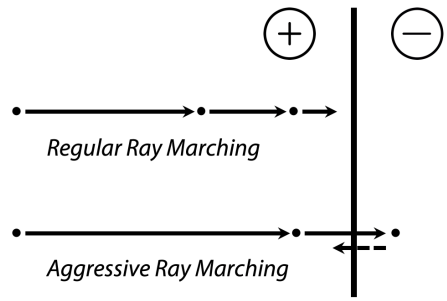


Figure 14. Compared to the regular ray marching algorithm, the aggressive ray marching strategy can march inside the surface and bounce back-and-forth between the inside and outside areas. This gives more samples on the negative side of implicit signed distance function and benefits the optimization.

(*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 [35] 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.

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

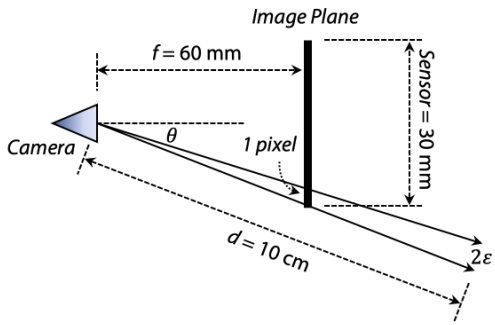


Figure 15. Illustration on the geometric meaning of the threshold ϵ of the convergence criteria.

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. 15 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:

$$\frac{S/R \cdot cos(\theta)}{f/cos(\theta)} = \frac{2\epsilon}{d_{min}} \tag{9}$$

This gives:

$$\epsilon = \frac{d_{min} \cdot S \cdot cos^2(\theta)}{2 \cdot f \cdot R} \tag{10}$$

Taking the common set up shown in Fig. 15, where $f = 60mm, d_{min} = 10cm, S = 32mm, R = 512$, we get $\epsilon \approx 0.5 \times 10^{-4}m$ $(0.05mm)$.

A.3. Differentiable Rendering of Silhouette

Fig. 16 shows how to render the silhouette in a differentiable way. After running our deep sphere tracing (Fig. 16 (a)), we render the minimal absolute SDF on each pixel, and get the soft silhouette by substracting it by ϵ (Fig. 16 (b)). In this way the binary silhouette (Fig. 16 (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:

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

Figure 16. 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.

$$\mathcal{L}_s = S_{gt} max(0, S_r) + (1 - S_{gt}) max(0, -S_r). \tag{11}$$

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. 17). 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.4. Drawbacks

Most of our acceleration strategy does not affect the rendering quality, except the aggressive marching. Fig. 18 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.

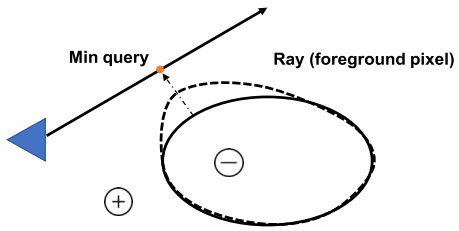


Figure 17. Differentiable error propagation along the boundary of the foreground. We can make use of the nice property of signed distance fields to optimize the nearest surface.

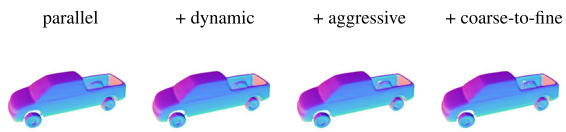


Figure 18. Illustration of the small artifacts induced by the aggressive strategy. Small holes could occur on thin surface areas. Better viewed when zoomed in.

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 [35], 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 [35] is much heavier than the backbone used in [45] (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 [10]. 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. (11).

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$\rightarrow$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 optimiza-

tion 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 [35] 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 [25] 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 [25] 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. 19.

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. 20, Fig. 21 and Fig. 22. The visual quality of our optimized mesh clearly outperforms the baseline method DeepSDF [35]. By employing perspective camera model and using online computed error (compared to the fixed sampling strategy in DeepSDF [35]) 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 [35] 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. 23. It can be noticed that our method produces much more visually satisfactory 3D shapes with only random initializations. In contrast, even if PMO [25] 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. 23, 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. 24. 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. 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.

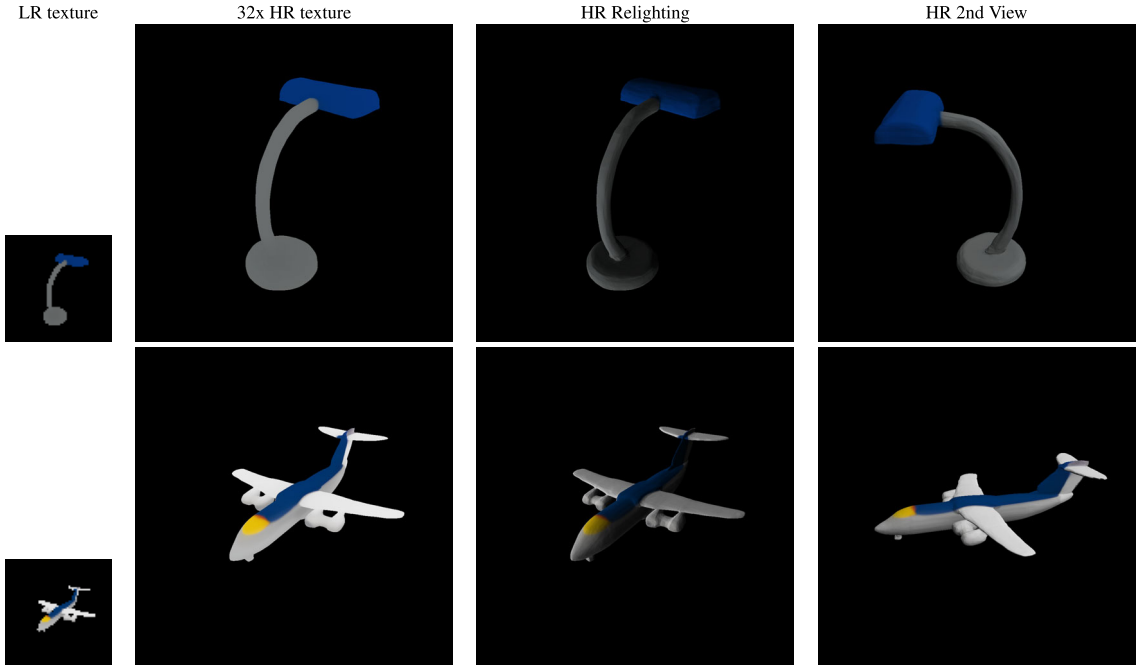


Figure 19. Qualitative results on the applications on texture re-rendering, where we can generate high-resolution outputs under various resolution, camera viewpoints and illumination.

Figure 20. Qualitative comparisons on shape completion under different sparsity of input depth (sofa).

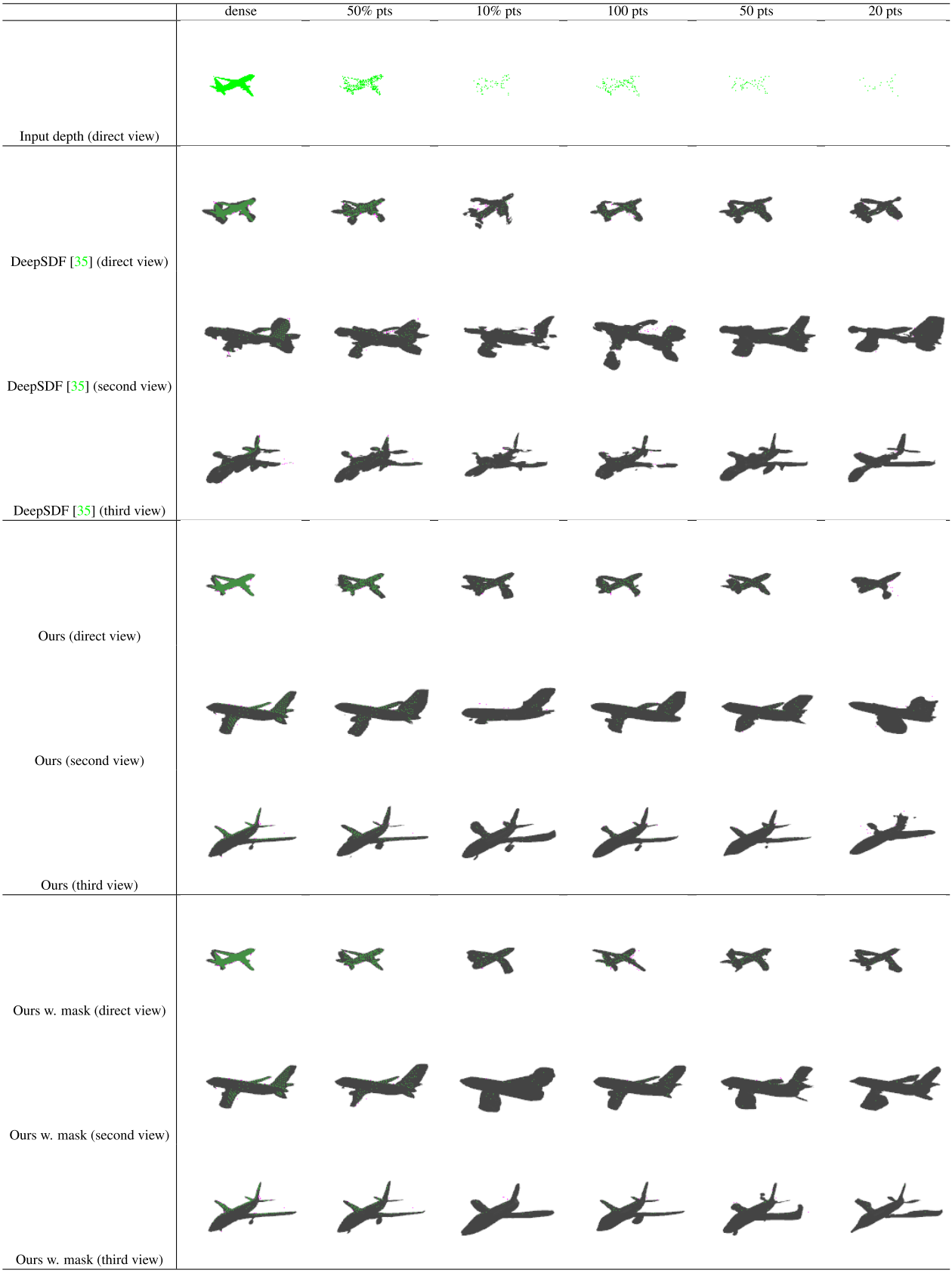


Figure 21. Qualitative comparisons on shape completion under different sparsity of input depth (plane).

Figure 22. Qualitative comparisons on shape completion under different sparsity of input depth (table).

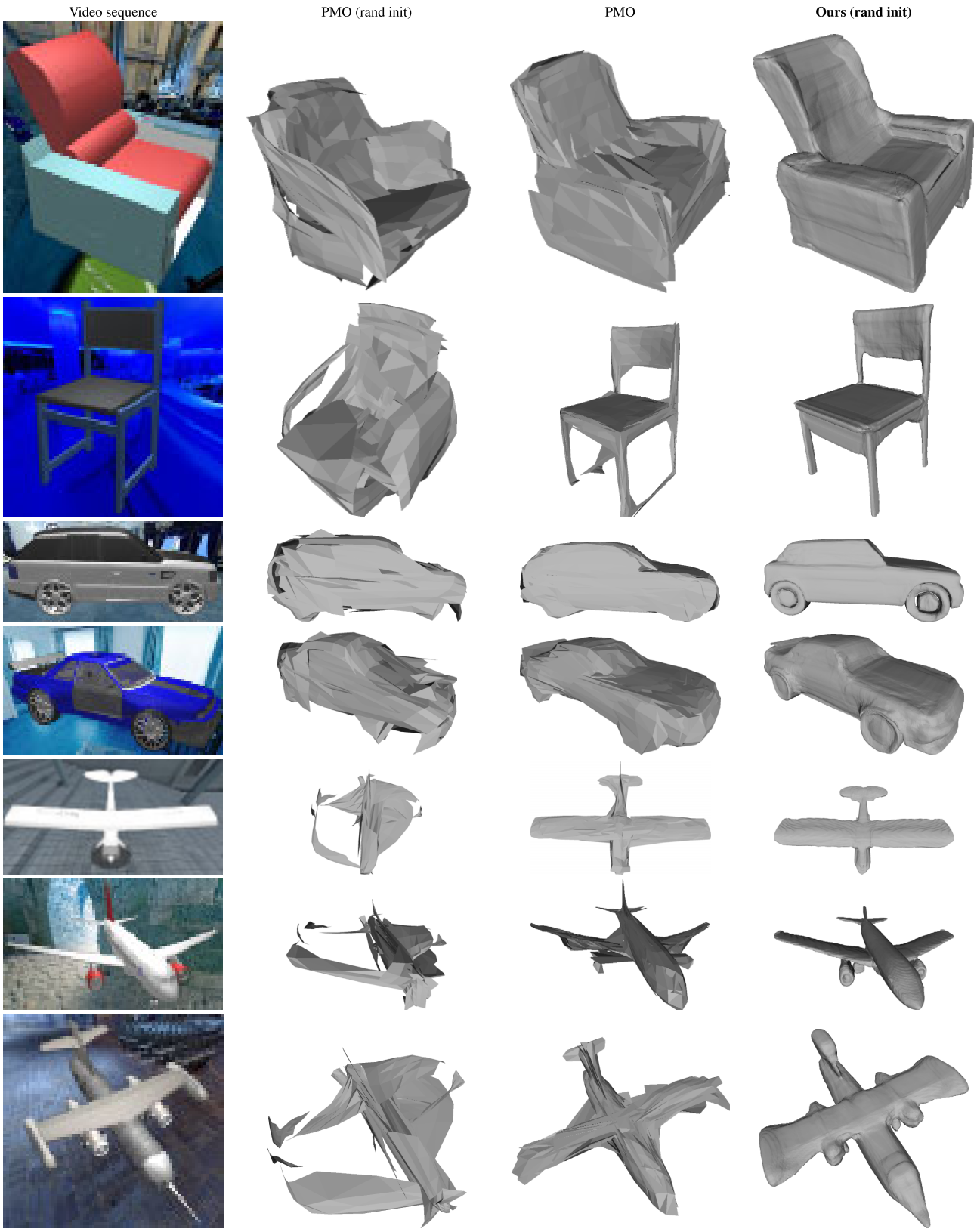


Figure 23. More quatitative comparisons on 3D shape prediction from multi-view images on the PMO test set.

Figure 24. More Comparisons on 3D shape prediction from multi-view images on real-world chair dataset [5]. It is in general challenging for shape prediction on real images. Comparatively, our method produces more reasonable results with correct structure.