DOI:10.1111/cgf.14082
Eurographics Symposium on Geometry Processing 2020
Q. Huang and A. Jacobson
(Guest Editors)

Volume 39 (2020), Number 5

DFR: Differentiable Function Rendering for Learning 3D Generation from Images

Yunjie Wu[1] and Zhengxing Sun[1]

[1]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, P R China

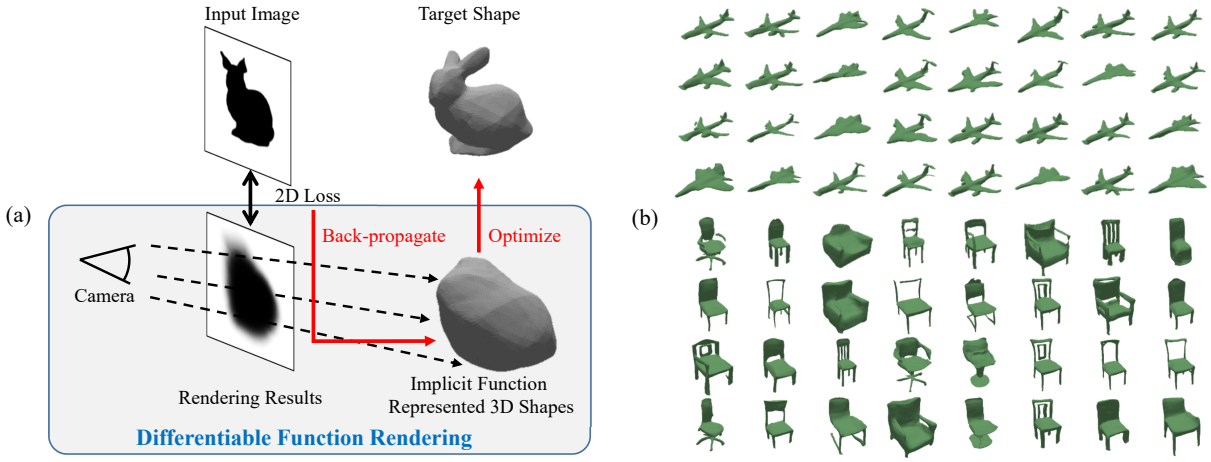

Figure 1: (a) DFR enables 2D-supervised 3D modeling tasks. (b) 3D shapes generated by our GAN model, which is trained with 2D images.

Abstract

Learning-based 3D generation is a popular research field in computer graphics. Recently, some works adapted implicit function defined by a neural network to represent 3D objects and have become the current state-of-the-art. However, training the network requires precise ground truth 3D data and heavy pre-processing, which is unrealistic. To tackle this problem, we propose the DFR, a differentiable process for rendering implicit function representation of 3D objects into 2D images. Briefly, our method is to simulate the physical imaging process by casting multiple rays through the image plane to the function space, aggregating all information along with each ray, and performing a differentiable shading according to every ray's state. Some strategies are also proposed to optimize the rendering pipeline, making it efficient both in time and memory to support training a network. With DFR, we can perform many 3D modeling tasks with only 2D supervision. We conduct several experiments for various applications. The quantitative and qualitative evaluations both demonstrate the effectiveness of our method.

Keywords: Differentiable Rendering; Learning-based 3D generation; Implicit Function Representation for 3D objects

CCS Concepts

• **Computing methodologies** → **Shape inference; Ray tracing; Volumetric models;**

1. Introduction

Modeling and generating 3D shapes is a key problem in computer graphics. Deep-learning based methods have been popular in recent years. Various representations are employed in deep-learning-based 3D generation including voxel [CXG*16, WWX*17, HTM17], point cloud [FSG17] and mesh [GFK*18].

Recently, some methods [CZ19, MON*19, MPJ*19, PFS*19] adapted continuous implicit function to represent 3D shapes and achieved impressive results. The main idea of the function representation is shown in Figure 2(a). It represents a 3D shape with a function, which is defined by a neural network. During training, these methods rely on precise 3D data to decide whether a 3D point

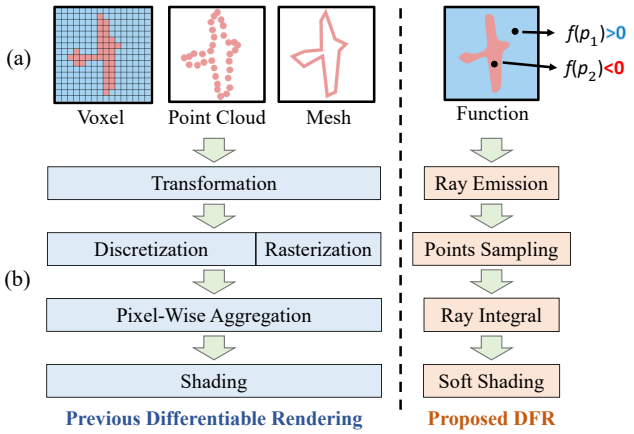

Figure 2: (a) Illustration of explicit and implicit 3D representations. (b) Comparison between previous differentiable rendering and our proposed DFR.

is inside. However, 3D shape data is usually difficult to obtain. In contrast, 2D images are easier to collect. So it is appealing if we can train the 3D generation network with 2D images.

The key to this task is differentiable rendering, which enables us to propagate the loss signal from 2D images to the 3D field. Many works have designed the differentiable rendering process for explicit 3D representation [GCC*17, YYY*16, ID18, GCM*18, KUH18, LADL18, LLCL19, LB14, PBCC18]. These works transform 3D shapes for the view's variety, then perform discretization or rasterization process to map the 3D elements (*e.g.*, 3D voxel grids, points, or meshes) into image's pixels. However, implementing a differentiable rendering for an implicit function is difficult because there are no explicit 3D elements(*e.g.* point cloud or mesh) providing the object's surface information for rendering.

A traditional solution to this problem is the ray-marching based rendering algorithm. It launches rays into a function space and determines the pixel's color by the corresponding ray. For an implicit function, the process is performed iteratively since the surface can not be directly solved. The ray marches forward, and one point is queried at each step. The process will continue until the queried point is inside the object, indicating the ray has intersected with a potential surface. Some adaptive marching strategy, such as sphere-tracing, can accelerate the process. However, as we need to perform the rendering in every batch during training, the iterative process will cost too much time.

In this paper, we propose an efficient differentiable function rendering for implicit 3D representation, referred to as "DFR". The main components are shown in Figure 2(b). Different from iterative ray marching, our method performs not only ray-level but also point-level parallelization, which takes much less time consumption. More specifically, our concern is to represent individual 3D shapes instead of scenes, so we restrict the function's domain to a normalized space. Then we can simultaneously sample multiple points on each ray's bounded part to approximately cover it and query function values of these points for rendering. To keep the rendering differentiable, the intermediate results in the computational graph need to be saved. However, it may cost unbearable memory burden as each point's query involves a large number of tensor operations in the network. Hence, we propose an efficient ray integral process via point-picking strategy, which formulates each ray's state by only picked points instead of all. So the intermediate results of unselected sampling points can be omitted. This process keeps the effectiveness of rendering while significantly reduces memory consumption. Besides, we raise a differentiable shading function to replace the traditional non-differentiable one, enabling the gradient signal flow from pixels to rays and further to the networks.

With the proposed DFR, it is able to train an implicit-represented 3D generation network with only 2D images. Benefiting from the implicit function's representation power, the trained network can produce various-topologies 3D shapes with pleasing visual appearance. Our contribution can be summarized as follows:

- We propose an efficient, gradient-favored rendering process for implicit function-based 3D representation, which is defined by a neural network. The method enables the integration of rendering for function space into deep learning.
- The DFR allows us to train 3D generation networks with only 2D supervision for many applications, including single-image 3D reconstruction, 3D adversarial generation learning, and image-fusion based 3D modeling.
- We conduct experiments on various 3D generation tasks. The quantitative and qualitative evaluations show our method outperforms other 2D-supervised methods, demonstrating our method's effectiveness.

2. Related Work

2.1. Representation of 3D Objects

The representation of 3D objects is a fundamental factor in 3D shape learning. There are various widely used 3D representations, including voxel, point cloud, and mesh. The mostly used 3D voxel is a regular and convolution-favored representation. A standard method to generate the 3D voxel is employing the 3D deconvolution layer, which transforms a feature vector into a 3D grid voxel [CXG*16, YYY*16, WZX*16]. The Point cloud is another popular representation, representing a 3D shape's surface with a set of 3D points. Some methods use a fully-connected layer to produce the 3D coordinates of all points [FSG17, ID18]. The 3D voxel and point cloud takes up lots of memory and leads to a limit of resolution. What is more, the surface normals of the 3D shapes are lost in these two representations. The mesh representation defines a 3D shape as a set of points and faces, which is more efficient than the previous two. However, most mesh-based methods can not produce topology-various shapes [WZL*18, GCM*18, KUH18], as the topology of faces is not differentiable in neural networks. Recently, some methods introduced the implicit representation of the 3D surface and achieved better performance [MON*19, PFS*19, MPJ*19, CZ19] than previous representation. However, these methods rely on precise 3D data for training. On the contrary, our work aims to design a differentiable rendering to allow learning such implicit shape representation without 3D supervision.

2.2. Learning-based 3D Shape Generation

With the development of deep learning, it has become popular to perform the learning-based 3D shape prediction with a trained neural network. A direct way to learn such a network is to employ lots of existed 3D shapes for training [CXG*16, FSG17, GFK*18, WZL*18, WWX*17, CZ19, MON*19, MPJ*19, PFS*19, MPJ*19]. Another way is to take 2D images as training data and estimate the 3D manifold from 2D samples [KUH18, YYY*16, LLCL19, KTEM18, KH19]. However, these works' results struggle in resolution (voxel), loss of surface topology (point cloud), or topology's diversity (deformation-based mesh).

Introducing the implicit shape representation can improve the quality of results [MON*19, PFS*19, MPJ*19, CZ19]. Because of the implicit characteristics, it is hard to design an encoder for it. Park *et al*. [PFS*19] propose an auto-decoder model to replace the traditional auto-encoder and omit the encoder part. Some works [MON*19, CZ19] rely on encoders for voxel or point cloud to build up the auto-encoder. They successfully train an auto-encoder to generate implicit 3D shapes. However, **it seems unable to cooperate with another popular generative model: GAN** without the encoder (discriminator). In this paper, we show that we can train a GAN model for implicit 3D representation by introducing DFR and a 2D image discriminator.

2.3. Differentiable Rendering

Rendering is the process that aims to project a 3D shape into a 2D image. To be integrated into gradient-based neural networks, the process has to be differentiable. Based on a perspective transformation process, Yan *et al*. [YYY*16] design the Perspective Transformer to get multi-view 2D masks from 3D voxel. Gwak *et al*. [GCC*17] employ a hit-based ray-tracing method to render 3D voxel for comparison with 2D supervision. Insafutdinov *et al*. [ID18] design a framework in which the input point cloud is transformed, smoothed, and discretized, resulting in a differentiable 2D silhouette. As to the mesh representation, most works [GCM*18, KUH18, LADL18, LLCL19] adopt an approximate process for rasterization, which enables the gradient flow. All the works above focus on the differentiable rendering of explicit 3D representations while ignoring the implicit 3D representation.

We notice there are some concurrent works [SZW19, JJHZ20, LZP*19,LSCL19] that focus on the differentiable rendering for implicit 3D shapes. Sitzmann *et al*. [SZW19] propose a trainable renderer and jointly optimizes it with an implicit 3D network. Their renderer needs to be re-trained for different object categories. Unlike them, we can directly apply our renderer on different 3D objects. Jiang *et al*. [JJHZ20] design a differentiable ray-tracing for a finite Signed Distance Function (SDF) grid. Applying it to a continuous implicit field may cause much computation burden. Instead, our method can perform an efficient rendering for a continuous field. Liu *et al*. [LSCL19] propose to compute 2D loss from an implicit function by projecting some sparse anchor points into 2D planes. It leads to a pixel-level sparsity in rendering. Their method is well suited for pixel-level loss while can hardly support high-level image loss (*e.g*. image-level perception losses) because the rendering is not complete. On the contrary, our work can efficiently render a complete and differentiable result, enabling more applications such as training a 3D GAN model with 2D images.

The most related work to ours is the differentiable renderer proposed by Liu *et al*. [LZP*19]. They optimize the iterative ray marching to render a DeepSDF decoder [PFS*19] efficiently. The main difference between their work and ours is **whether the 3D generator (decoder) part can be optimized with 2D supervision during training**. In order to support the sphere-marching process, they require the 3D decoder to be a completely trained DeepSDF network, which is pre-trained with 3D supervision. So they only optimize the latent shape code instead of the decoder's parameters. It is difficult to apply their method to an untrained 3D decoder, because the decoder may probably break the SDF setting and make the sphere-marching invalid. On the contrary, our method does not rely on the SDF setting of the 3D shape decoder and can optimize an initial decoder from scratch with 2D supervision.

3. Differentiable Function Rendering

In this section, we first describe the implicit function-based 3D representation we employed. Then we introduce our differentiable function rendering (DFR). The overview of our method is presented in Figure 3.

3.1. Implicit Function-based 3D Representation

Similar to existing works [CZ19, MON*19, MPJ*19, PFS*19], we employ an implicit function to represent the 3D shape. The function is defined by a network f. It takes a 3D location coordinate p as input and outputs a real value, which indicates whether the location is inside or outside the 3D shape by the sign: positive value means outside, and vice versa. Optionally, the network can take an extra condition c, enabling the representation of various 3D objects with the same f and different c.

$$f_c(p) \rightarrow \mathbb{R}, \|p\| < 1 \tag{1}$$

In our work, we assume each shape is normalized, leading the function's space bounded in a unit sphere.

3.2. Ray Emission

Ray emission is an initial stage of our rendering. Given a camera position, we emit rays from the camera through each pixel's center in the image. After the computation of rays' direction, we perform a hit-test between all rays and the unit sphere of function space. If a ray does not intersect with the unit sphere, the corresponding pixel will be colored with background color, and the ray will not enter the following process.

3.3. Ray-wise Point Sampling

Rendering 2D observations from the implicit function requires deciding each ray's state by evaluating sampled points. A typical method is to perform an iterative raymarching and test the front ends of all beams in each step. It can be optimized in a ray-level parallelization, but the time consumption is still highly relevant to the number of steps. In our setting, as the unit sphere bounds the

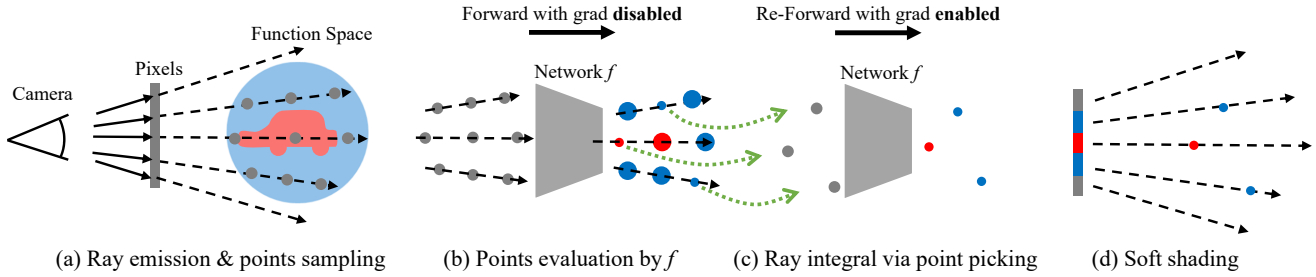

Figure 3: Differentiable Function Rendering. Given an implicit function defined by a neural network f and camera parameters, the rendering process includes: (a) We first emit rays from the camera through pixels into the function space. Then we perform points sampling on bounded parts of each ray. (b) All points are evaluated by network f in a point-level parallelization without keeping the computational graphs. The sign and abs value are marked by color and size. (c) We compare all sampled points and pick one point per ray. The picked points are re-forwarded via f again with intermediate tensors saved in the computational graph. (d) We propose a soft shading function to replace the normal shading, keeping the pipeline differentiable. With DFR, loss signal in pixels can flow into each ray, and further to network f.

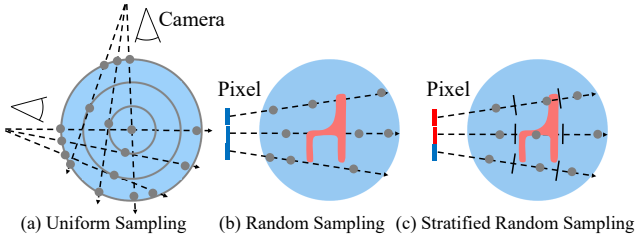

Figure 4: (a) **The uniform sampling** restricts the possible sampling distribution on a set of concentric spheres, which can not cover the whole function space. (b) **The random sampling**'s high instability may lead to an error in rendering results. (c) The proposed **stratified random sampling** can cover the whole function space and reduce erroneous coloring.

function space, it is unnecessary to follow this framework. We can split the bounded part of rays into segments and sample some points at each of those segments. The main advantage is we can decide rays' states in a point-level parallelization, which costs much less time.

Some common sampling strategies, such as uniform sampling or random sampling, can be employed. However, we reveal that both sampling strategies are not suitable for our task. With uniform sampling, rendering results are stable. However, the potential sampling space can not cover the full function's space. As shown in Figure 4(a), with a certain focal length and camera distance (It is a common assumption in most works [KH19, KUH18, LLCL19, YYY*16]), no matter how the viewpoint changes, the sampling points of all rays would distribute on a set of concentric spheres. The effect of this problem is illustrated in Section 4.2.2. As to the randomly sampling, although the sampled points' distribution can cover the function's space, the high uncertainty may interfere with the rendering results. For example, it may lead to erroneous coloring in actual foreground pixels when every randomly sampled point is located outside the object by chance, as shown in Figure 4(b). Here we introduce a stratified random sampling strategy, which performs jittering in uniformly sampled positions. Let In_i and Out_i denote the two intersections of i-th Ray and the unit sphere. We first split the uniform segments between In_i and Out_i into N parts. Then an extra stochastic offset is added to each k-th part's starting point. With the proposed sampling strategy, the distribution of sampling points can cover the full function's domain, and the rendering result is relatively stable. Under this setting, the k-th sampling point's location of i-th ray is

$$s_i^k = In_i + \left(\frac{(k-1)}{N} + \xi\right) * (Out_i - In_i) \quad ,\xi \in U\left(0, \frac{1}{N}\right) \tag{2}$$

3.4. Point-level Parallelized Evaluation

With the process above, we now obtain all sampled points' 3D locations. We need to evaluate their values to get each pixel's state. One direct way is to batch and forward them through f in parallel. However, it is not feasible because computational graphs have to be kept for being differentiable. The heavy memory consumption prevents us from performing the parallelized evaluation.

We raise a key observation here: although we sample multiple points, the rendering results are actually decided by a few critical points (*e.g.*, the points lie on the implicit surface) instead of all points. Based on this observation, we can first evaluate all points without keeping computational graphs, as shown in Figure 3. With the evaluated results, we can select the critical points and re-forward them to achieve a differentiable rendering pipeline (see details in Section 3.5).

This point-level parallelized evaluation strategy enables us to achieve all points' value efficiently under affordable memory consumption. We can easily implement this process in practice by disabling the autograd module in common frameworks (*e.g.*, Pytorch [PGM*19], or TensorFlow [ABC*16]) and forwarding all sampled points through f parallelly (It may not fit in a single batch with high resolution and sampling number N. We can split them into a few batches and combine the results).

3.5. Ray Integral

After obtaining every point's value on a ray, we need to integrate the information and determine the ray's state for further rendering pipeline. Let F_i denote the set of all N sampled points' function values on Ray_i (These values are achieved with no gradients):

$$F_i = \left\{ f_c\left(s_i^1\right), \dots f_c\left(s_i^N\right) \right\} \tag{3}$$

We can tell whether the collision happens by only checking the minimal value among the N values. That is because: (1) Positive minimum indicates all values are positive, which means no collision happens; (2) Negative minimum indicates that there exists at least one point inside the object. We first get the minimal value $\min(F_i)$ and the corresponding point's location s_i^m on Ray_i.

$$\min(F_i) = f_c\left(s_i^m\right) \tag{4}$$

Then we classify all rays into hit ray and unhit ray by $f_c(s_i^m)$'s sign. For a hit ray, we further search the first negative point s_i^n in sampling order, which is approximately the intersection point of the ray and implicit surface. Although we compare all points together, only these two points are important for yielding the rendering results. The minimal point decides whether collision happens, and the intersection point tells where it happens. Especially, we can obtain some approximate geometry information of implicit surface from s_i^n, *e.g.*, the surface normal. By now, we can render the 2D pixels, but the pipeline is still non-differentiable, as described in Section 3.4. To enable the gradient, we define a state of a ray by reforwarding a point with the autograd turned on. A hit ray's state T_i is defined by $f_c(s_i^n)$ (approximate intersection point), and an unhit ray's state is defined by $f_c(s_i^m)$ (most possible to become inside):

$$T_i = \begin{cases} f_c(s_i^m), f_c(s_i^m) > 0 \\ f_c(s_i^n), f_c(s_i^m) \le 0 \end{cases} \tag{5}$$

The computational graph is maintained during this time's forward pass. So far, we have obtained a differentiable state of each ray.

3.6. Soft Shading

A shading process is to assign a color for each pixel of the image. In this work, we mainly concern about the rendering of silhouette, which is widely used as supervision for 3D generation tasks [KUH18, LLCL19, YYY*16]. Notice other types of observations, for example, a normal map, can also be achieved by evaluating more points in s_i^n's neighborhood and calculating the difference as the approximation of the surface's normal. The details of the normal map's rendering are contained in the supplementary material.

A direct way to generate a silhouette by ray's state is a binary function:

$$I_i = \begin{cases} 1, T_i > 0 \\ 0, T_i \le 0 \end{cases} \tag{6}$$

where I_i denotes the intensity value of i-th pixel. However, with such a shading function, I_i's gradient w.r.t. ray's state T_i is always 0, preventing the gradient flow from image to the function field. We replace it with a soft, differentiable shading function:

$$I_i = \frac{1}{1 + e^{-k * T_i}} \tag{7}$$

It is a sigmoid function with an extra k parameter. The k denotes a shading sharpness parameter. Larger k leads to a sharper shading effect. With the increase of k, we can get a more sharp boundary between the inside and outside colors. However, a large k may cause the gradient to gather near the zero point, making the training difficult. When k approaches infinity, the shading will degenerate into Equation 6. In our implementation, we set k to 10 empirically. A detailed analysis of the sharpness parameter is provided in Section 4.2.3.

3.7. Gradient Computation

To train a network f, it requires gradient to propagate from a rendered image to sampled points in function space. More specifically, we need the gradient from each pixel's intensity I_i to the picked point s_i^j in the corresponding ray. The I_i is achieved by performing the shading process on the ray's state T_i. So, with the chain rule, we can obtain that:

$$\frac{\partial \mathrm{I}\, i}{\partial f_c\left(s_i^j\right)} = \frac{\partial \mathrm{I}\, i}{\partial T_i} \frac{\partial T_i}{\partial f_c\left(s_i^j\right)}, \quad j = n \text{ or } m \tag{8}$$

According to the Equation 7, we can obtain that:

$$\frac{\partial \mathrm{I}_i}{\partial T_i} = k \frac{1}{1 + e^{-kT_i}} \left(1 - \frac{1}{1 + e^{-kT_i}}\right) = k\,\mathrm{I}_i\,(1 - \mathrm{I}_i) \tag{9}$$

According to the definition of T_i, we can omit the $\frac{\partial T_i}{\partial f_c(s_i^j)}$ term in Equation 8. Hence we get that, for Ray_i, its pixel's gradient w.r.t. picked point s_i^j is $k\,\mathrm{I}_i\,(1 - \mathrm{I}_i)$. A detailed deduction of the formula is provided in the supplementary material.

4. Experiments

In this section, we first demonstrate the efficiency and effectiveness of our DFR pipeline. Then, we integrate our DFR into various 3D generation applications with only 2D supervision. We make the core part of our code public in the link `https://github.com/JiejiangWu/DFR`.

4.1. Efficiency and Effectiveness of Rendering

To verify the proposed DFR's efficiency and effectiveness, we conduct some evaluations. First, we perform the rendering process on an implicit shape represented by a trained network f. The network's architecture is shown in Figure 7(a). The sampling number is set to 32. We report efficiency comparison in Table 1 and show corresponding rendered results in Figure 5. In the naive version, we perform no parallelization and evaluate each sampling point iteratively.

Setting	Res	Diff	Time	GPU memory
Naive	64	No	3.87h	4.3GB
+Ray para	64	No	1.6s	4.3GB
+Point para	64	No	0.09s	4.3GB
+Point para	128	No	\	>11GB
+ Ray Integral	64	No	0.12s	2.0GB
+ Ray Integral	128	No	0.17s	3.4GB
+ Soft Shading	128	**Yes**	**0.17s**	**3.4GB**

Table 1: The efficiency comparison of different settings. From left to right: rendering resolution, whether differentiable, the runtime, and the GPU memory consumption. Test on a single GTX-1080Ti with Pytorch 1.1.0 [PGM*19].

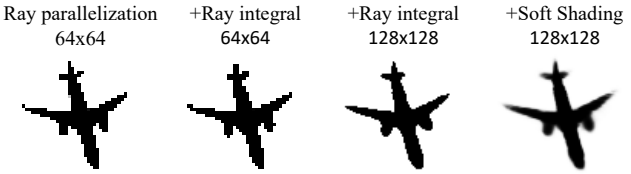

Figure 5: Rendered silhouettes under different settings. With much less time and memory consumption, our method's rendered silhouette maintains correct.

To check the effectiveness of differentiability, we conduct a test. Given an input silhouette and a simple network f with three fully-connected layers, we render the f by DFR and optimize the difference between rendered and input silhouettes. We employ a zero iso-surface extraction method [MON*19] to extract the explicit mesh from f for visualization. Figure 6 illustrates the optimization process. It shows that, with the optimization of the silhouette, the extracted shape is also transformed, meaning the loss in 2D pixels can propagate to the 3D implicit function filed via DFR.

4.2. Single-view 3D Reconstruction

The first application of DFR is to integrate it into the learning of single-image 3D reconstruction. With the DFR, we can train a 3D reconstruction network with only 2D supervision.

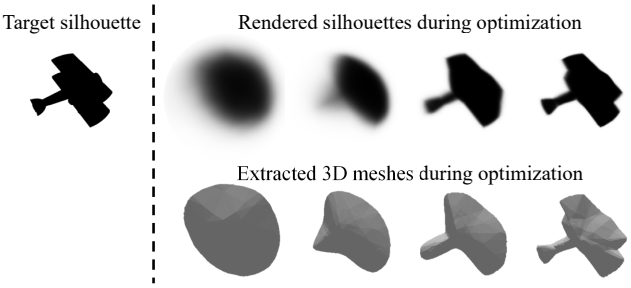

Figure 6: Differentiability test. Samples of silhouettes and extracted 3D meshes during optimization are presented above.

Implementation details. We employ an encoder-decoder network for this task. The encoder is a resnet-18 [HZRS16], which takes the rgb image as input. The decoder is a conditional network f with four fully-connected res-blocks to represent predicted implicit 3D shapes. Its detailed architecture follows the design in [MON*19], shown in Figure 7(a). The condition c is defined as the image's feature. During training, we take a pair of same object's images with known viewpoints and silhouettes as training data. DFR renders the silhouettes under the two conditions from both viewpoints, respectively. The loss is defined as cross-entropy between input silhouette s_i and rendered silhouette $\hat{s}_i$:

$$L_s = \frac{1}{N}\sum_{i=1}^{N}[\hat{s}_i \log s_i + (1-\hat{s}_i)\log(1-s_i)] \tag{10}$$

The resolution of the rendering image is 64. We empirically set the sampling number to 32. The batch size is 12. The ADAM [KB14] with $\beta = 0.9$ and learning rate = 1e-4 is employed. More implementation details are contained in the supplementary material.

Dataset. We use the subset of ShapeNet [CFG*15] and the same train/test splits as Choy *et al.* [CXG*16]. Each 3D shape is rendered from 24 random viewpoints with both RGB and silhouette. During training, we require a random pair of the same object's images.

Comparisons. We select three 2D-supervised methods: differentiable ray consistency (DRC) [TZEM17], neural mesh renderer (NMR) [KUH18], soft rasterizer (SoftR) [LLCL19] and two 3D-supervised methods: AtlasNet (Atlas) [GFK*18] and occupancy net (Onet) [MON*19] for comparison. We use the pre-trained model provided by authors for evaluation.

Metrics. We select two widely used metrics for evaluation: 3D intersection over union (IoU) and the chamfer distance (CD) between meshes. We evaluate the IoU in a 64 voxelization resolution. To compute the chamfer distance, 100k random points are sampled from each mesh's surface.

4.2.1. Qualitative and Quantitative Results

The qualitative results are shown in Figure 8. We can observe that all methods can capture the global and coarse shapes of the objects. The DRC's results are coarse because it is limited by voxel resolution. The NMR's model cannot handle different topologies of 3D shapes (*e.g.*, the chair's arms in row 1, and legs in row 2) and show unsmoothness in results. The SoftR's are better but still fails to reflect different topologies as the meshes are produced by moving vertices. The Atlas's results are various in topology, but the surface is often discontinuous. Benefiting from the implicit function representation, Onet achieves a much better visual effect. Our method can obtain similar visual effects with Onet while we only employ 2D silhouettes supervision. More qualitative results are provided in supplementary material and videos.

The quantitative evaluation is reported in Table 2. Our method achieves a leading result among 2D-supervised methods. We can further find that our method is superior in categories with various topologies, such as chairs. On the contrary, the SoftR achieves a better result in simple-topology categories like cabinet and car.

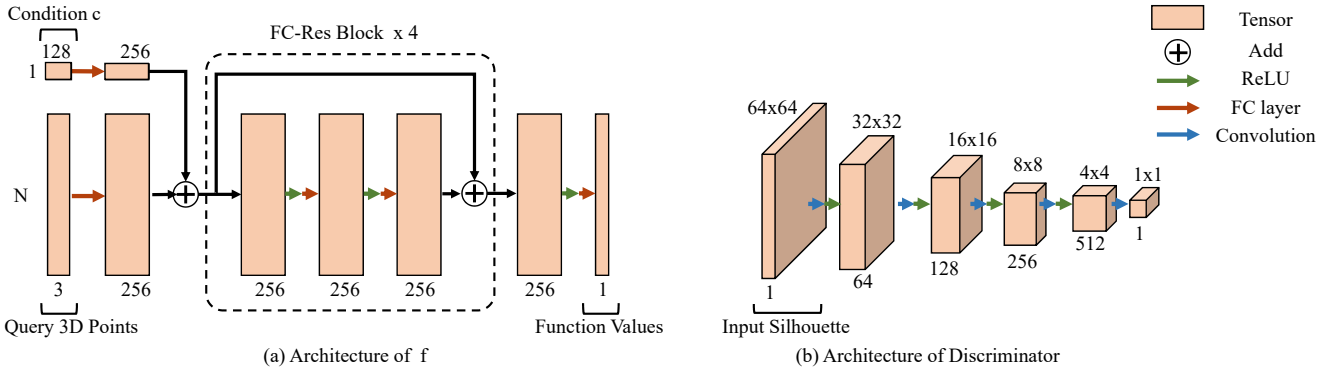

Figure 7: Networks' architectures. (a) The architecture of f in Section 4.2 and 4.3. (b) The architecture of discriminator in Section 4.3.

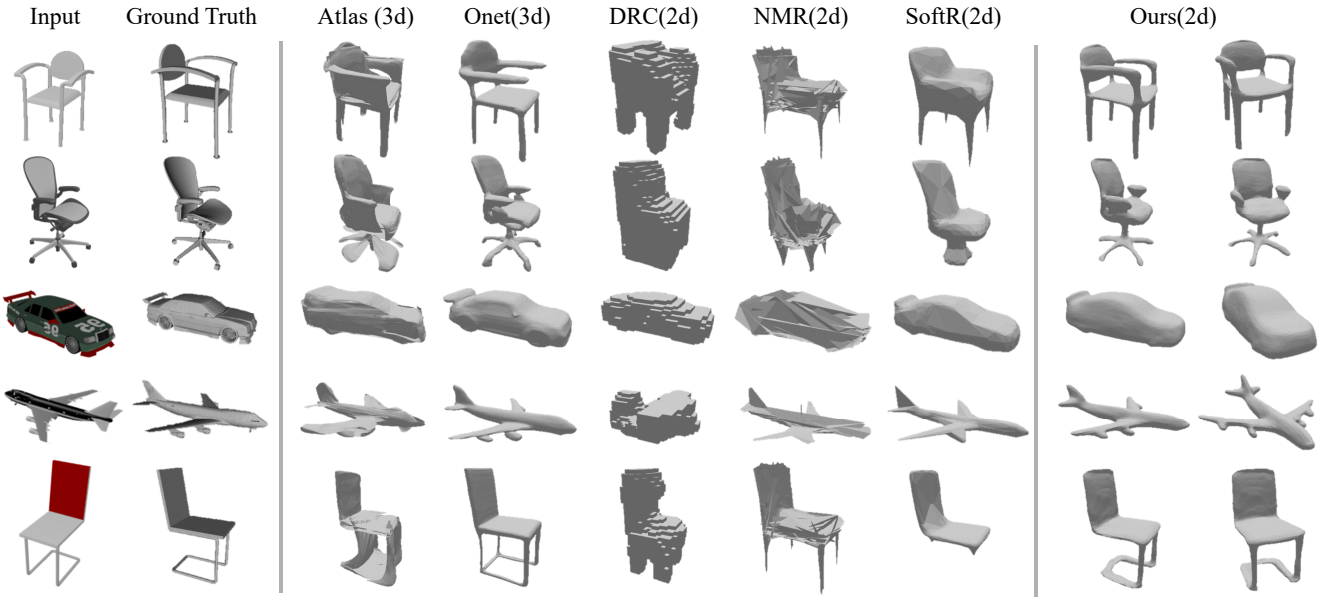

Figure 8: Qualitative results of single-image 3D reconstruction. We compare our method with three 2D-supervised methods and two 3D-supervised methods.

The possible reason for their better performance is that estimating implicit-represented 3D shapes is more difficult and complicated than estimating template-deformed ones. Besides, shapes in these categories contain many large flat surfaces, which can be easily obtained by the template's deformation. However, the template-deformation representation limits the results' variety in topology and harms the final performance for complex-topology categories.

Compared to the 3D-supervised methods, our method's performance is worse. This is not surprising as these two methods access the ground-truth 3D data, while our model is trained with 2D silhouettes. The weaker supervision inevitably leads to the loss of some 3D information. In general, our method can achieve state-of-the-art 2D-supervised results in complex-topology categories.

4.2.2. Effect of Points Sampling Strategies

We conduct some experiments to check the influence of points sampling strategies. We compare models trained with random sampling (RS), uniform sampling (US), and the proposed stratified random sampling (SRS). The results are shown in Table 3 and Figure 9. It could be seen that the results of the RS often miss some detailed parts, such as the arms in row 2. On the other hand, the results of the US can coarsely represent the shape but may lead to an over-thick effect. It is because the sampled points are distributed in a limited region, which causes some errors in unsampled space. With SRS, we can achieve better results.

4.2.3. Effect of Sharpness Parameter

The sharpness parameter in soft shading controls the clarity of the boundaries in results. We present rendering results of reconstruction network f in Figure 10. We can observe that a higher k can make the rendering closer to the standard shading effect. According to section 3.7, the gradients of black or white pixels are close to 0. So all the pixels' gradients under high k are close to 0. On the contrary, a smaller k makes the erroneous pixels shaded grey (*e.g.* the

	2D-supervised methods								3D-supervised methods			
	CD				IoU				CD		IoU	
	DRC	NMR	SoftR	Ours	DRC	NMR	SoftR	Ours	Atlas	Onet	Atlas	Onet
plane	9.02	2.88	1.96	**1.64**	34.2	46.5	50.9	**53.3**	1.18	1.52	-	59.0
bench	-	3.22	4.39	**2.03**	-	31.8	33.8	**39.1**	1.44	1.56	-	48.5
cabinet	-	3.23	**2.26**	2.48	-	66.1	**68.7**	65.2	1.82	1.45	-	75.0
car	5.56	2.52	**2.03**	2.35	44.1	63.4	**67.2**	66.0	1.35	1.21	-	74.3
chair	14.91	9.20	7.09	**5.84**	31.8	36.7	41.9	**44.4**	2.25	2.79	-	53.2
display	-	4.81	**3.73**	4.21	-	45.3	47.8	**52.2**	2.01	2.67	-	54.4
Lamp	-	12.45	9.84	**8.31**	-	27.8	32.7	**37.7**	3.91	4.15	-	40.4
speaker	-	5.61	6.58	**5.46**	-	57.4	62.5	**62.7**	3.27	3.94	-	67.6
rifle	-	2.17	**1.83**	2.28	-	42.3	**45.3**	38.9	1.40	1.73	-	47.9
sofa	-	**2.38**	2.65	2.47	-	52.6	**55.9**	54.8	1.78	2.06	-	69.3
table	-	7.62	6.90	**5.15**	-	42.5	38.0	**45.6**	2.98	6.33	-	53.5
cellphone	-	4.05	**3.07**	3.47	-	61.9	62.2	**66.8**	1.31	1.44	-	74.1
vessel	-	5.37	3.67	**3.26**	-	47.3	50.2	**53.8**	1.69	2.64	-	55.1
mean	9.83	5.04	4.31	**3.77**	36.7	47.8	50.5	**52.3**	2.03	2.57	-	59.4

Table 2: Comparison on single-view 3D reconstruction. We compare our method with three 2D-supervised methods: DRC [TZEM17], NMR [KUH18], SoftR [LLCL19] and two 3D-supervised methods: Atlas [GFK*18], Onet [MON*19]. The metrics are chamfer distance (x0.001) and 3D Intersection-over-Union (%).

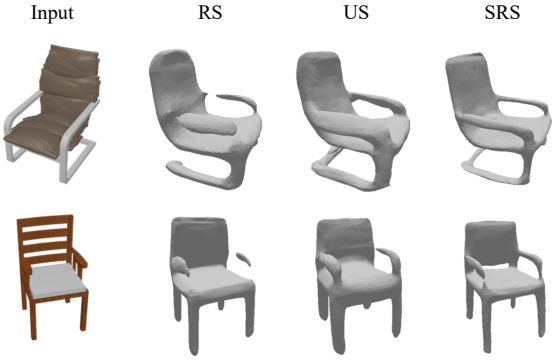

Figure 9: Effect of points sampling strategies.

	RS	US	SRS
CD	6.15	6.02	**5.84**
IoU	42.1	42.7	**44.4**

Table 3: Quantitative comparison of different points sampling strategies in the 'chair' category.

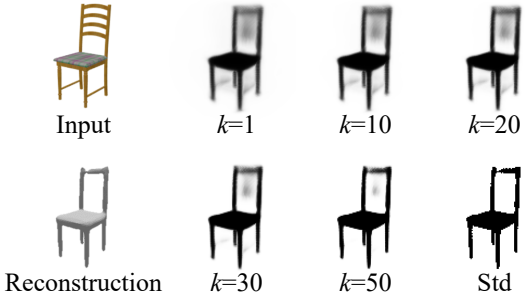

Figure 10: Rendering results under different k. The network f is not completely trained, and some regions are missing in reconstruction results. These missing regions are shaded grey under small k and white under big k.

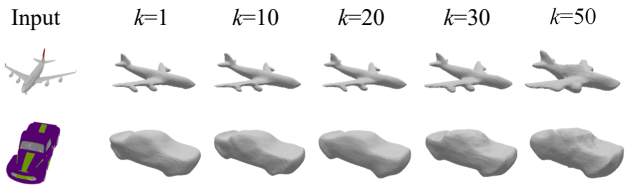

Figure 11: Reconstruction results with different k. When k exceeds about 30, the reconstruction results become defective.

missing crossbars). It means the gradients in these pixels are bigger than the correct pixels, leading the training process. We show some final reconstruction results under different sharpness parameter settings in Figure 11. The results are within the expectation. When k is not too big, the results are close. When k exceeds about 30, the results become worse.

4.3. 3D GAN Learning

The second application of DFR is to cooperate with a 3D GAN, enabling the training without 3D data. To our knowledge, this is the first attempt to train an implicit-represented 3D GAN with 2D images. Notice concurrent work [LSCL19] is not suited for this task, since the incomplete renderings can hardly fool the discriminator.

Implementation details. To generate a 3D GAN with 2D images, we combine a 2D discriminator with our implicit 3D shapes gen-

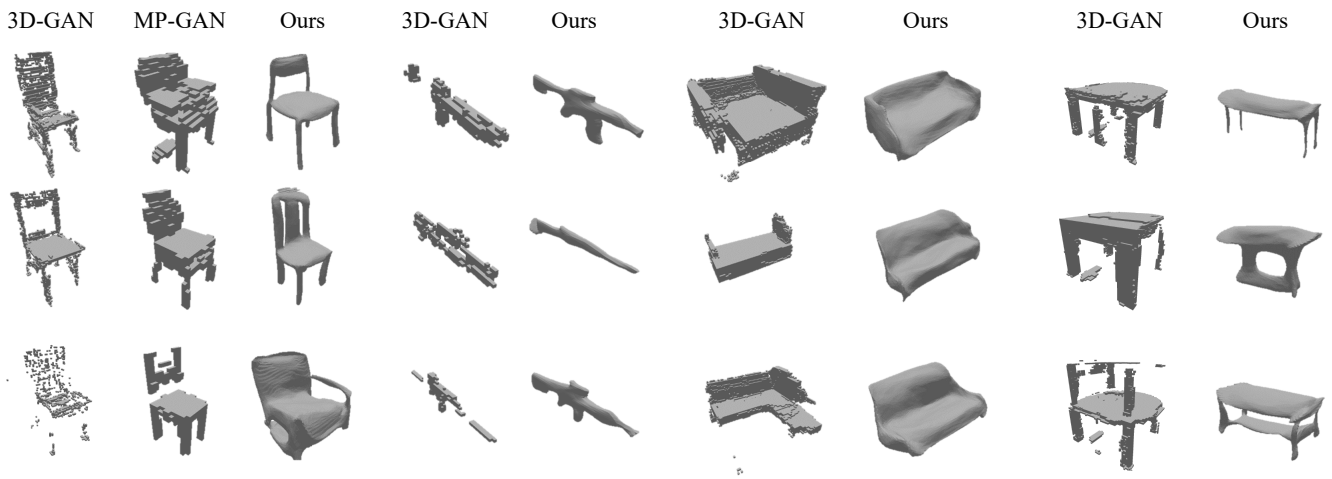

Figure 12: Qualitative comparisons of various GAN model. Our model can generate various shapes with a more plausible appearance.

erator. We take the real silhouettes as real samples to train a 2D discriminator. Then we generate implicit 3D shapes by generator and render them with DFR. The generator is expected to produce plausible 3D shapes to fool the discriminator from any perspective. During rendering, we fix the elevation to 30 and randomly sample an azimuth to decide the camera viewpoint.

Discriminator: We follow the designing of the discriminator in DCGAN [RMC15]. Its architecture is illustrated in Figure 8(b). Empirically, we find that adopting the WGAN-gp [GAA*17] improves the training process. Hence the loss for discriminator is:

$$L_D(X,Z) = \sum_{z\sim Z} D(DFR(v,G(z))) - \sum_{x\sim X} D(x) + \lambda \sum_{\hat{x}\sim \hat{X}} \left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\right)^2 \quad (11)$$

where G denotes the generator, and D denotes the discriminator. The v denotes a random viewpoint. The z is a noise vector that follows Gaussian distribution Z. The x is a real silhouette in training data X. The $\hat{X}$ is the interpolated space between real samples and generated samples, which is proposed by Gulrajani [GAA*17]. The λ controls the weight of the gradient penalty and it is set to 10.

Generator: The network f is now seen as a 3D generator. It takes a noise vector z as condition c, generating implicit 3D shape f_z from z. Its architecture is the same with 4.2. The loss function for the generator is defined as:

$$L_G(Z) = -\sum_{z\sim Z} D(DFR(v,G(z))) \quad (12)$$

During training, batch-size, learning rate, and sampling steps keep the same as Section 4.2.

Dataset. We employ the same dataset as Section 4.2. Differently, we take a single instead of a pair of images from each 3D shape for training. Besides, in this section, we do not need the viewpoint annotation for training.

	Chair	Sofa	Table	Rifle	Mean
3D-GAN	234.72	270.03	**171.83**	65.23	185.45
MP-GAN	240.31	-	-	-	240.31
Ours	**169.57**	**213.95**	192.18	**59.55**	**158.81**

Table 4: The comparison of GAN models with fid. We render the generated shapes to images and evaluate them by Inception-v3.

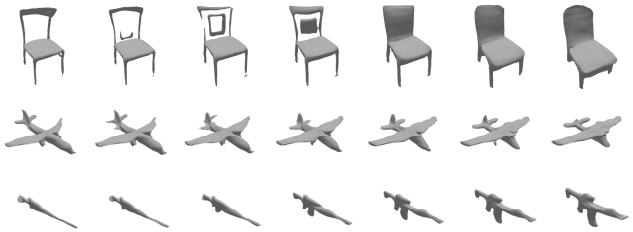

Figure 13: Interpolation of shapes in the learned latent space.

Comparisons. We take a 2D-based method MP-GAN [LDPT19] and a 3D-based method 3D-GAN [WZX*16] for comparison. Both methods are evaluated with the official pre-trained models. The provided model of MP-GAN [LDPT19] only contains the 'chair' class.

Metrics We select the fid metric [HRU*17] to evaluate the quality of generated shapes. It measures the difference between the distribution of real and synthesized samples by calculating the activations in a pre-trained perceptual model. We render all methods' results to 2D images and use the pre-trained inception-v3 to capture a higher-level visual evaluation.

Results First, we show some qualitative results in Figure 12. More qualitative results are provided in Figure 1 (b) and supplementary materials. Compared to other methods, our model generates more satisfactory results with a smooth surface and plausible details in all categories. Note that 3D-GAN is trained with existed 3D data. MP-

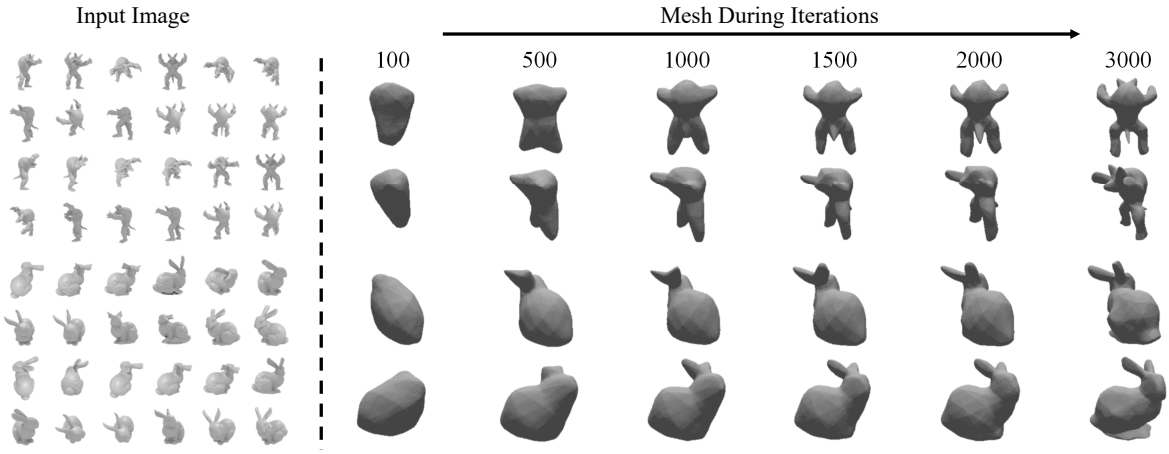

Figure 14: Two examples of multi-view fusion results. With DFR, we can fusion multi-view images to get a 3D shape. Two views of the results are shown.

GAN is trained with 2D images, but they adopt a more complex framework than ours, which contains multiple 2D discriminators and an extra viewpoint-prediction module. So, this further explains the superior ability of implicit representation and the effectiveness of our DFR. In addition, we can observe that the generated shapes are not as plausible as Section 4.2. It is because in Section 4.2, we use pairs of images as supervision for training, while here we only use independent images without correspondence information. This makes the 3D generation more difficult. The quantitative evaluation is reported in Table 4. It is consistent with qualitative results, demonstrating our results can obtain a more plausible visual effect. We also perform an interpolation operation in the noise vector z. The results are presented in Figure 13. We can observe that all the interpolated shapes are plausible.

4.4. Image-fusion for 3D modeling

Image fusion for 3D modeling is a traditional application that creates a 3D object from multiple 2D views. Different from traditional point-matching-based methods [LAAH12], our DFR enables a gradient-based image fusion.

We select two objects from The Stanford 3D Scanning Repository [LGCP05]. We first obtain 2D images from different views. Then, we construct a simple network f with three fully-connected layers. In this setting, no condition c is needed as f only represents a single 3D shape. With DFR, we render the f from all viewpoints and compute the silhouettes errors compared with input images. The fusion process is performed by minimizing all silhouettes' errors. We show the fusion process with iterations in Figure 14.

5. Limitations

There are some limitations of our work. First, we temporarily only consider the 2D silhouette supervision for 3D modeling tasks. This limits the information that can be obtained from supervision. For example, our model can not predict the concave parts of the sofa in Figure 15(a), because these parts do not appear in silhouettes from any viewpoints. Second, our point sampling strategy is a fast but approximate estimation of implicit surface. It may cause some artifacts in rendering results. An example is shown in Figure 15(b). When the sampling step is 64, the normal map and the silhouette are correct. When the sampling step is 32, some pixels' colors are wrong in the normal map. When the sampling step is 16, the silhouette misses some thin parts, and the normal map becomes even worse. Although a larger sampling number can achieve better rendering results, it may cause more consumption of memory and computation.

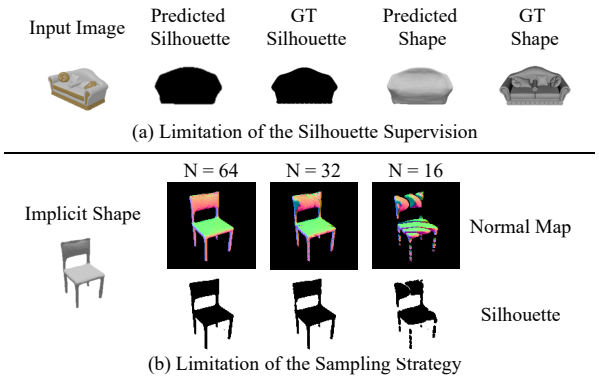

Figure 15: Two examples of our method's limitations.

6. Conclusions

In this paper, we propose the DFR: an efficient differentiable rendering for implicit 3D representation. We optimize the normal iterative ray-marching to a totally parallelized points sampling process. We design a ray integral strategy that reduces high memory consumption. A soft shading function is also raised to replace the traditional non-differentiable one. We conduct various 3D modeling experiments with 2D supervision. The results show: 1) Our framework can efficiently render the implicit function. 2) We can

learn a single-image 3D reconstruction model without 3D supervision, and it outperforms other 2D-supervised methods. 3) We can train an implicit-represented 3D GAN model with 2D images and generate plausible 3D shapes. 4) We can perform a gradient-based multi-view fusion process for 3D modeling and achieve results with satisfactory visual effects. All these experimental results prove our DFR's effectiveness and superiority. We believe our work will be useful and bring some inspirations to other researchers for their future works related to the generation of 3D shapes.

Acknowledgement

We thank the anonymous reviewers for their valuable comments. This work was supported by National High Technology Research and Development Program of China (No. 2007AA01Z334), National Natural Science Foundation of China (Nos. 61321491 and 61272219), National Key Research and Development Program of China(Nos. 2018YFC0309100, 2018YFC0309104), the China Postdoctoral Science Foundation (Grant No. 2017M621700) and Innovation Fund of State Key Laboratory for Novel Software Technology (Nos. ZZKT2018A09).

References

[ABC*16] ABADI M., BARHAM P., CHEN J., CHEN Z., DAVIS A., DEAN J., DEVIN M., GHEMAWAT S., IRVING G., ISARD M., ET AL.: Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)* (2016), pp. 265–283. 4

[CFG*15] CHANG A. X., FUNKHOUSER T., GUIBAS L., HANRAHAN P., HUANG Q., LI Z., SAVARESE S., SAVVA M., SONG S., SU H., ET AL.: Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012* (2015). 6

[CXG*16] CHOY C. B., XU D., GWAK J., CHEN K., SAVARESE S.: 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European conference on computer vision* (2016), Springer, pp. 628–644. 1, 2, 3, 6

[CZ19] CHEN Z., ZHANG H.: Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 5939–5948. 1, 2, 3

[FSG17] FAN H., SU H., GUIBAS L. J.: A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 605–613. 1, 2, 3

[GAA*17] GULRAJANI I., AHMED F., ARJOVSKY M., DUMOULIN V., COURVILLE A. C.: Improved training of wasserstein gans. In *Advances in neural information processing systems* (2017), pp. 5767–5777. 9

[GCC*17] GWAK J., CHOY C. B., CHANDRAKER M., GARG A., SAVARESE S.: Weakly supervised 3d reconstruction with adversarial constraint. In *2017 International Conference on 3D Vision (3DV)* (2017), IEEE, pp. 263–272. 2, 3

[GCM*18] GENOVA K., COLE F., MASCHINOT A., SARNA A., VLASIC D., FREEMAN W. T.: Unsupervised training for 3d morphable model regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 8377–8386. 2, 3

[GFK*18] GROUEIX T., FISHER M., KIM V., RUSSELL B., AUBRY M.: Atlasnet: A papier-mâché approach to learning 3d surface generation. In *CVPR 2018* (2018). 1, 3, 6, 8

[HRU*17] HEUSEL M., RAMSAUER H., UNTERTHINER T., NESSLER B., HOCHREITER S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems* (2017), pp. 6626–6637. 9

[HTM17] HÄNE C., TULSIANI S., MALIK J.: Hierarchical surface prediction for 3d object reconstruction. In *2017 International Conference on 3D Vision (3DV)* (2017), IEEE, pp. 412–420. 1

[HZRS16] HE K., ZHANG X., REN S., SUN J.: Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778. 6

[ID18] INSAFUTDINOV E., DOSOVITSKIY A.: Unsupervised learning of shape and pose with differentiable point clouds. In *Advances in Neural Information Processing Systems* (2018), pp. 2802–2812. 2, 3

[JJHZ20] JIANG Y., JI D., HAN Z., ZWICKER M.: Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. 3

[KB14] KINGMA D. P., BA J.: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014). 6

[KH19] KATO H., HARADA T.: Learning view priors for single-view 3d reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 9778–9787. 3, 4

[KTEM18] KANAZAWA A., TULSIANI S., EFROS A. A., MALIK J.: Learning category-specific mesh reconstruction from image collections. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 371–386. 3

[KUH18] KATO H., USHIKU Y., HARADA T.: Neural 3d mesh renderer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 3907–3916. 2, 3, 4, 5, 6, 8

[LAAH12] LAGÜELA S., ARMESTO J., ARIAS P., HERRÁEZ J.: Automation of thermographic 3d modelling through image fusion and image matching techniques. *Automation in Construction 27* (2012), 24–31. 10

[LADL18] LI T.-M., AITTALA M., DURAND F., LEHTINEN J.: Differentiable monte carlo ray tracing through edge sampling. In *SIGGRAPH Asia 2018 Technical Papers* (2018), ACM, p. 222. 2, 3

[LB14] LOPER M. M., BLACK M. J.: Opendr: An approximate differentiable renderer. In *European Conference on Computer Vision* (2014), Springer, pp. 154–169. 2

[LDPT19] LI X., DONG Y., PEERS P., TONG X.: Synthesizing 3d shapes from silhouette image collections using multi-projection generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 5535–5544. 9

[LGCP05] LEVOY M., GERTH J., CURLESS B., PULL K.: The stanford 3d scanning repository. *URL http://www-graphics. stanford. edu/data/3dscanrep 5* (2005). 10

[LLCL19] LIU S., LI T., CHEN W., LI H.: Soft rasterizer: A differentiable renderer for image-based 3d reasoning. *The IEEE International Conference on Computer Vision (ICCV)* (Oct 2019). 2, 3, 4, 5, 6, 8

[LSCL19] LIU S., SAITO S., CHEN W., LI H.: Learning to infer implicit surfaces without 3d supervision. In *Advances in Neural Information Processing Systems* (2019), pp. 8293–8304. 3, 8

[LZP*19] LIU S., ZHANG Y., PENG S., SHI B., POLLEFEYS M., CUI Z.: Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. *arXiv preprint arXiv:1911.13225* (2019). 3

[MON*19] MESCHEDER L., OECHSLE M., NIEMEYER M., NOWOZIN S., GEIGER A.: Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 4460–4470. 1, 2, 3, 6, 8

[MPJ*19] MICHALKIEWICZ M., PONTES J. K., JACK D., BAKTASHMOTLAGH M., ERIKSSON A.: Deep level sets: Implicit surface representations for 3d shape inference. *arXiv preprint arXiv:1901.06802* (2019). 1, 2, 3

[PBCC18] PALAZZI A., BERGAMINI L., CALDERARA S., CUCCHIARA R.: End-to-end 6-dof object pose estimation through differentiable rasterization. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 0–0. 2

[PFS*19] PARK J. J., FLORENCE P., STRAUB J., NEWCOMBE R., LOVEGROVE S.: Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 165–174. 1, 2, 3

[PGM*19] PASZKE A., GROSS S., MASSA F., LERER A., BRADBURY J., CHANAN G., KILLEEN T., LIN Z., GIMELSHEIN N., ANTIGA L., ET AL.: Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* (2019), pp. 8024–8035. 4, 6

[RMC15] RADFORD A., METZ L., CHINTALA S.: Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015). 9

[SZW19] SITZMANN V., ZOLLHÖFER M., WETZSTEIN G.: Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems* (2019), pp. 1119–1130. 3

[TZEM17] TULSIANI S., ZHOU T., EFROS A. A., MALIK J.: Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 2626–2634. 6, 8

[WWX*17] WU J., WANG Y., XUE T., SUN X., FREEMAN B., TENENBAUM J.: Marrnet: 3d shape reconstruction via 2.5 d sketches. In *Advances in neural information processing systems* (2017), pp. 540–550. 1, 3

[WZL*18] WANG N., ZHANG Y., LI Z., FU Y., LIU W., JIANG Y.-G.: Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 52–67. 2, 3

[WZX*16] WU J., ZHANG C., XUE T., FREEMAN B., TENENBAUM J.: Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in neural information processing systems* (2016), pp. 82–90. 2, 9

[YYY*16] YAN X., YANG J., YUMER E., GUO Y., LEE H.: Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *Advances in Neural Information Processing Systems* (2016), pp. 1696–1704. 2, 3, 4, 5