

Boosting Point Clouds Rendering via Radiance Mapping

Xiaoyang Huang¹, Yi Zhang¹, Bingbing Ni^{1*}, Teng Li², Kai Chen³, Wenjun Zhang¹

¹Shanghai Jiao Tong University, ²Anhui University, ³Shanghai AI Lab

Abstract

Recent years we have witnessed rapid development in NeRF-based image rendering due to its high quality. However, point clouds rendering is somehow less explored. Compared to NeRF-based rendering which suffers from dense spatial sampling, point clouds rendering is naturally less computation intensive, which enables its deployment in mobile computing device. In this work, we focus on boosting the image quality of point clouds rendering with a compact model design. We first analyze the adaption of the volume rendering formulation on point clouds. Based on the analysis, we simplify the NeRF representation to a spatial mapping function which only requires single evaluation per pixel. Further, motivated by ray marching, we rectify the noisy raw point clouds to the estimated intersection between rays and surfaces as queried coordinates, which could avoid *spatial frequency collapse* and neighbor point disturbance. Composed of rasterization, spatial mapping and the refinement stages, our method achieves the state-of-the-art performance on point clouds rendering, outperforming prior works by notable margins, with a smaller model size. We obtain a PSNR of 31.74 on NeRF-Synthetic, 25.88 on ScanNet and 30.81 on DTU. Code and data would be released soon.

Introduction

The rising trend of AR/VR application calls for better image quality and higher computation efficiency in rendering technology. Recent works mainly focus on NeRF-based (Mildenhall et al.) rendering due to its photo-realistic effect. Nevertheless, NeRF-based rendering suffers from heavy computation cost, since its representation assumes no explicit geometry is known, and requires burdensome spatial sampling. This drawback severely hampers its application in mobile computing devices, such as smart phones or AR headsets. On the other hand, point clouds, which have explicit geometry, are easy to be obtained as the depth sensors become prevalent and MVS algorithms (Yao et al.; Wang et al.) get powerful. It deserves more attention to develop high-performance rendering methods based on point clouds, which is so far insufficiently explored. In this work, we introduce a point clouds rendering method which achieves comparable rendering performance to NeRF.

The main difference between NeRF-based rendering and point clouds rendering is that the latter is designed upon the noisy surface of objects. On the bright side, it is a beneficial geometric prior which could greatly reduce the query times in 3D space. On the bad side, this prior is noisy and sparse, since the point clouds are generally reconstructed by MVS algorithms or collected by depth sensors. It needs additional approaches to alleviate the artifact brought by the noise and sparsity. Therefore, most of the current point clouds rendering methods require two steps. One is the spatial feature mapping, and the other is image-level refinement. The spatial feature mapping step is similar to the NeRF representation, which maps a 3D coordinate to its color, density or latent feature. The refinement step is usually implemented as a convolutional neural network. In this work, we mainly focus on the spatial feature mapping step. Previous works use point clouds voxelization (Dai et al.), learnable parameters (Rückert, Franke, and Stamminger; Kopanas et al.) or linear combination of sphere basis (Rakhimov et al.) as mapping functions. However, these methods suffer either from high computation cost, large storage requirements, or unsatisfactory rendering performance. To this end, we introduce a much simpler but surprisingly effective mapping function. Motivated by the volume rendering formulation in NeRF, we analyze its adaptation on point clouds rendering scenarios. It is concluded that in a point cloud scene, the volumetric rendering could be simplified to the modeling of the view-dependent color of the first-time intersection between the estimated surface and the ray. In other words, we augment each 3D point (i.e., most probably a surface point) with a learnable feature indicating first-hit color. Thereby the point clouds rendering task could be re-cast within the high fidelity NeRF framework, without consuming redundant computation on internal ray samples. We name it *radiance mapping*. Moreover, based on radiance mapping, we rectify the raw point cloud coordinates that are fed into the mapping function using the z-buffer in rasterization to obtain a query point which lies exactly on the camera ray. This approach allows us to obtain a more accurate geometry and avoid spatial frequency collapse. The radiance mapping function consisted of a 5-layer MLP is only 0.75M large, which is much smaller than the spatial feature mapping functions in previous works, but with notably better performance. Followed by a 2D neural renderer which compensates the sparsity and

*Corresponding Author.

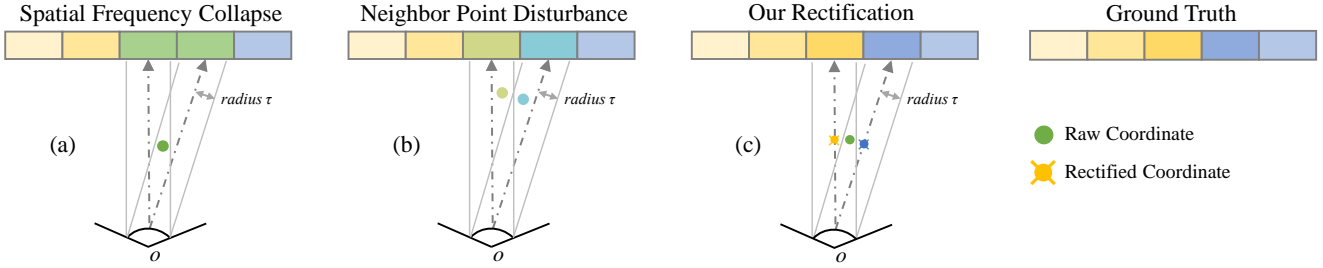


Figure 1: **(a)** *Spatial frequency collapse* occurs when using neural descriptors or raw point clouds query. The point is optimized to a green color, which is the mixing of yellow and blue. **(b)** Using raw point clouds query would additionally cause neighbor point disturbance. The points lying close have a larger impact on the feature optimization of each other. **(c)** Our coordinate rectification could alleviate the above issues. The idea is illustrated in 1D rendering.

noise in point clouds as done in previous works, our complete model is approximately 8M in total.

Our method reaches comparable rendering effect compared to NeRF, but with much less computation cost since it only needs single model inference per pixel. Compared to prior point clouds rendering methods, we obtain notable improvement in terms of image quality, with a smaller model size and simpler computation. We achieve a PSNR of 31.74 on NeRF-Synthetic (Mildenhall et al.), 25.88 on ScanNet (Dai et al.) and 30.81 on DTU (Aanæs et al.). As far as we know, It is the state-of-the-art result on this task.

Related Work

Implicit Rendering

NeRF-based Neural Radiance Fields (Mildenhall et al.) advance the neural rendering quality to a higher level. NeRF represents the scene using an MLP which predicts the color and density of a point. It projects the points along the camera ray to the pixel color with volume rendering. Following NeRF, there are various innovations which address the different challenges in NeRF representation. PixelNeRF (Yu et al.), IBRNet (Wang et al.) and DietNeRF (Jain, Tancik, and Abbeel) render novel views from only one or a few input images. NeRF-W (Martin-Brualla et al.) tackles the variable illumination and transient occluders in the wild instead of a static scene. Mip-NeRF (Barron et al.) and Mip-NeRF 360 (Barron et al.) improves the image quality by rendering anti-aliased conical frustums. NSVF (Liu et al.), PlenOctrees (Yu et al.) and TensorRF (Chen et al.) aim at accelerating the inference speed of NeRF by building a more efficient structure after scene fitting. Point-NeRF (Xu et al.) also assumes a point cloud is given like ours. But it still follows the volume rendering formulation in NeRF, which also suffers from dense spatial sampling.

Implicit Surface Rendering This line of works aim at reconstruction the implicit surfaces via neural rendering. DVR (Niemeyer et al.) learns implicit 3D representation from images by analytically deriving depth gradients from implicit differentiation. IDR (Yariv et al.) renders an implicit surface by approximating the light reflected from the surface towards the camera. UNISURF (Oechsle, Peng, and Geiger) combines implicit surface models and radiance fields together to

enable surface reconstruction without object masks. NeuS (Wang et al.) gives a theoretical proof that the classic volume rendering formulation causes error on the expectation of the object surface, and presents a solution which yields an unbiased SDF representation. Yariv et al. models the volume density as the function of the SDF representation, leading to a more accurate sampling of the camera ray.

Point Clouds Rendering

Inverse Rendering Early work (Zwicker et al.) proposes a point cloud rendering method using an Elliptical Weighted Average filter based on Gaussian Kernel. Yifan et al. enables backward propagation of surface splatting to optimize the position of point clouds to match to object geometry from images. Insafutdinov and Dosovitskiy use a differentiable point clouds projection module to unsupervisedly learn the object shape and pose from two-view images. Lin, Kong, and Lucey propose pseudo-rendering which upsamples the target image to alleviate the collision effect in discretization. Wiles et al. construct a point cloud from single-view images by using a depth regressor and spatial feature predictor, and render the point cloud with α -composition followed by a ResNet (He et al.) refinement network. The training is supervised by a photometric loss and a GAN loss (Wang et al.). Zhou et al. and Godard, Mac Aodha, and Brostow adopt a similar approach, but on a monocular depth estimation task with street-view video sequences.

View Synthesis NPBG (Aliev et al.) proposes to render novel views of a scene using point-based learnable neural descriptors and a U-Net refinement network. It adopts multi-scale rasterization to model image details in different level. Johnson, Alahi, and Fei-Fei; Dosovitskiy and Brox use a perceptual loss to optimize the network. Dai et al. propose to project the point clouds to a layered volume by voxelization. Then a 3D CNN is used to transform the volume into a set of multi-plane images and their blending weights, which form the final image by its the weighted sum. NPBG++ (Rakhimov et al.) reduces the running time upon NPBG, using a feature extractor to lift the neural descriptor feature and making it view-dependent. ADOP (Rückert, Franke, and Stamminger) renders HDR and LDR images with a coarsely-initialized point cloud and camera parameters. The point

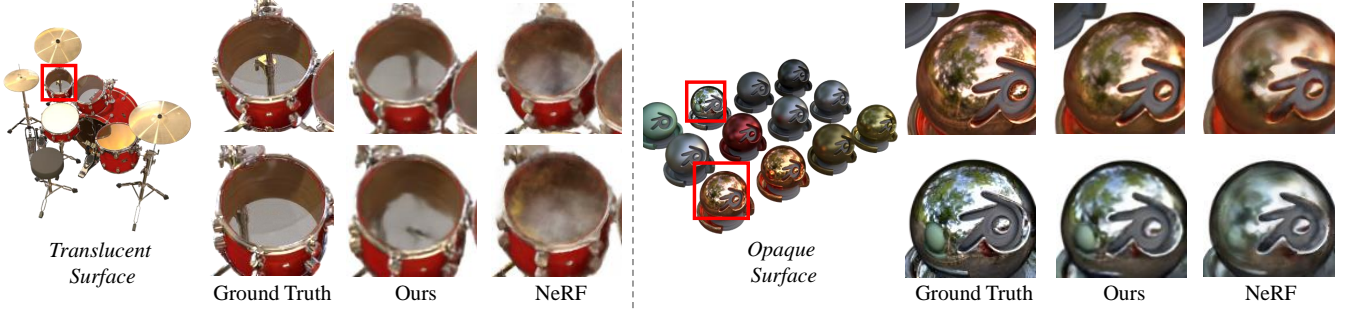


Figure 2: The translucent and opaque surfaces rendered by NeRF and our method. In the Drums scene, both methods are optimizing the color of the first-intersected surface instead of modeling the correct translucency. The second row shows the membrane from another view. In the Materials scene, our method could even render more decorate specular effect on the smooth metal balls, while NeRF generates somehow blurry artifacts. The visualization in the original NeRF paper (Mildenhall et al.) reveals the same artifact. We owe this superiority to the **explicit geometry** provided by point clouds.

clouds, camera poses and the 2D refinement network are jointly optimized. Kopanas et al. perform scene optimization for each view based on bi-directional Elliptical Weighted Average splatting. Ost et al. promote point clouds to implicit light fields to allow fast inference in view synthesis. READ (Li et al.) adopt a similar approach to NPBG++ to synthesize photo-realistic street views for autonomous driving. We analyze the most relevant works to ours in the next section.

Method

Spatial Mapping

We first analyze the spatial mapping functions in previous point clouds rendering methods. Then we introduce our radiance mapping, a simpler but more effective mapping.

Previous Mapping Functions Revisited NPBG (Aliiev et al.) attaches learnable parameters to each point as neural descriptors. The advantage of this approach is that each point feature is optimized independently, and would not be influenced by nearby point feature. This is beneficial to those scenes where surface color changes drastically. However, it also leads to a drawback that the density of point clouds impose restrictions on the representation capacity of point feature. When the point clouds are sparse, the same neural descriptor would be rasterized to multiple pixels and optimized to fit the average of multiple pixels, which harms the rendering quality. We illustrate this issue in 1D rendering in Figure 1 (a). The point is optimized to a green color, which is the mixing of yellow and blue. Since the cause of the phenomenon is analogous to the dissatisfaction of Nyquist Rate in signal processing, we call it *spatial frequency collapse*. On the other hand, when point clouds get comparatively dense due to higher quality of reconstruction or depth sensor, the size of point feature would grows proportionally, which consumes more memory for storage and training. Besides, some of the point features which are only visible in a few views might not be sufficiently optimized.

Dai et al. propose to use a 3D CNN to extract spatial feature. It first voxelizes the point clouds into a layered volume, and then adopts a 3D CNN to extract spacial feature. Due to

the high computation complexity of 3D CNN, this model is much more heavier, and not easy to deploy.

NPBG++ (Rakhimov et al.) develops a spatial mapping function motivated by sphere harmonics basis. It first uses a shared 2D CNN to extract image feature from multi-view images, and then aggregate the feature of each view into the point clouds by a linear combination of learnable basis functions over the unit sphere. This approach considers view direction as input, which would potentially generate better rendered images. However, it still suffers from proportionally increasing memory as the point clouds get more dense, similar to NPBG. Besides, it requires an additional U-Net as a image feature extractor which further increase model size.

Radiance Mapping Comparing to the above spatial mapping functions, our method is much more light-weight. Our compact representation store the view-dependent radiance of the object surface. The idea is motivated from the volumetric rendering formulation in NeRF representations (Mildenhall et al.), which take the 3D coordinate $\mathbf{x} = (x, y, z)$ and view direction $\mathbf{d} = (\theta, \phi)$ as inputs and output the color \mathbf{c} and density σ using a multi-layer perceptron (MLP) \mathbf{F}_Θ , parameterized by Θ :

$$\mathbf{c}, \sigma = \mathbf{F}_\Theta(\mathbf{x}, \mathbf{d}) \quad (1)$$

Since NeRF representations assume no explicit geometry exists, each point lying on the camera ray $\mathbf{r} = \mathbf{o} + t\mathbf{d}$ are queried and aggregated to obtain the final pixel color $C(\mathbf{r})$:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t) \sigma(\mathbf{r}(t)) \mathbf{c}(\mathbf{r}(t), \mathbf{d}) dt \quad (2)$$

$$T(t) = \exp \left(- \int_{t_n}^t \sigma(\mathbf{r}(s)) ds \right) \quad (3)$$

$T(t)$ denotes the accumulated transmittance along the ray. Suppose the camera ray first intersects with an object surface at point $\mathbf{p}_o = \mathbf{o} + t_o\mathbf{d}$. For those points that lie in front of \mathbf{p}_o , which satisfy $\{\mathbf{p} = \mathbf{o} + t\mathbf{d} | t < t_o\}$, would correspond to $\sigma_o = 0$. Substituting it into Eqn. 3 yields $\{T(t) = 0 | t < t_o\}$. Then we consider the points that lie

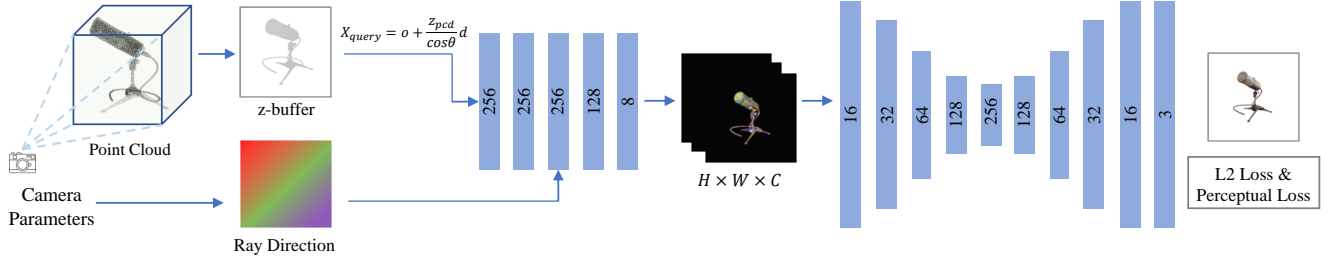


Figure 3: Our end-to-end point clouds rendering pipeline. We use the z-buffer obtained from the rasterization stage to perform coordinate rectification. The rectified coordinates are fed into the radiance mapping MLP. The output feature map is sent to the refinement stage, which is implemented as a U-Net. We train the MLP and U-Net with a weighted sum of L2-loss and perceptual loss.

behind \mathbf{p}_o , which satisfy $\{\mathbf{p} = \mathbf{o} + t\mathbf{d} | t > t_o\}$. One case is that the density of \mathbf{p}_o equals to a comparatively large value, meaning that the object surface is opaque. In this case we have $\{T(t) \approx 0 | t > t_o\}$ according to Eqn. 3. Another case is that the density of \mathbf{p}_o equals to a small value, in which the object surface is translucent. We demonstrate that the latter case could be discarded in point clouds rendering. (i) No matter the point clouds are reconstructed by MVS algorithms or collected by depth sensors, only the first intersected surface would be revealed in the point clouds. It is impractical and not necessary to evaluate the surfaces behind the first one. (ii) Even if only the first-intersected surface is evaluated, the correct pixel color could still be modeled in practice, since the point color $c(\mathbf{r}(t_o), \mathbf{d})$ would be fitted to the pixel color $C(\mathbf{r})$ during optimization. Figure 2 (left) shows the Drums scene in NeRF-Synthetic dataset, where the drum membrane is a translucent surface. We could see that the drum shelf behind the membrane is a thin column. If the translucency is modeled correctly, we should at least see the outline of the shelf. But in NeRF results we could see no outline but blurry artifact. It means that the density of the membrane predicted by NeRF is comparatively large such that the shelf behind is invisible. With no regard to the points that lie behind the intersection p_o , our method could still obtain similar results, or even better. Please refer to experiment section for more analysis. Given that $\{T(t) = 0 | t < t_o\}$ and $\{T(t) \approx 0 | t > t_o\}$, the pixel color $C(\mathbf{r})$ becomes a single evaluation on \mathbf{p}_o according to Eqn. 2. Thus we simply remove the density prediction in NeRF representation (Eqn. 1). Moreover, we output the latent feature \mathbf{L} that encodes the color to a higher dimension instead of RGB as prior works do. Then Eqn. 1 is reformulated as

$$\mathbf{L} = \mathbf{F}_\Theta(\mathbf{x}, \mathbf{d}) \quad (4)$$

Since we do not need to predict the density, we reduce the number of MLP layers from 13 in NeRF to 5, yielding a model size as small as $0.75M$. This compact architecture enables fast inference and small storage requirement. As compared in Table 4, our spatial mapping model is notably smaller than its counterparts in prior works. Since our model is light-weight with respect to model size and computation and needs only single time inference for each pixel, it is

possible to deploy it in mobile computing device like smart phones or AR/VR headsets.

Based on our radiance mapping, we further improve the performance by rectifying the raw coordinates of point clouds. One critical characteristics of point clouds is that it is generally noisy and sparse, due to unsatisfactory reconstruction or measurement quality. Specifically, the points \mathbf{p}_{pcd} in a point cloud actually lie around the object surface \mathbf{p}_{obj} with a spatial noise ϵ . We denote its coordinate with $\mathbf{x}_{pcd} = \mathbf{x}_{obj} + \epsilon$. If we feed the raw point clouds into Eqn. 4, we obtain $\mathbf{L}_{pcd} = \mathbf{F}_\Theta(\mathbf{x}_{obj} + \epsilon, \mathbf{d})$, meaning that the queried latent code is actually from a nearby coordinate around the surface. The noise is mainly handled by adopting a coordinate rectification step before we feed them into the MLP. As shown in Figure 1 (b), after rasterization, we obtain the closest point to the camera position and its z-buffer for each pixel via depth test. We conduct ray marching to find the estimated intersection between the ray and the approximate surface. Specifically, we calculate the point which share the same z-buffer with the closest point and also lies on the ray:

$$\mathbf{x}_{query} = \mathbf{o} + \frac{z_{pcd}}{\cos \theta} \times \mathbf{d} \quad (5)$$

where \mathbf{o} and \mathbf{d} are the camera position and the normalized direction, and θ is the angle between the ray and the z axis pointing to the center of the scene. This simple approach has obvious benefits over prior art. This is because that using the raw point clouds would lead to spatial frequency collapse as conventional neural descriptors do. Namely, when the point clouds are sparse, the same point would be rasterized to multiple pixels. In other words, nearby pixels are highly probable to be queried to the same latent feature from the MLP mapper, leading to blurred results. Also, the point density of the raw point clouds has a non-uniform distribution on different parts. Thus the color of the closer points would have a larger impact on each other, as shown in Figure 1 (b). In contrast, our proposed rectification offers a more uniformly-distributed points density. Thus it results in more accurate geometry recovery. Empirical results also prove the effectiveness of this approach.

Table 1: Results on three datasets. For NeRF-Synthetic, the results are averaged on Ficus, HotDog and Mic, following NPBG++.

Method	NeRF-Synthetic			PSNR↑	ScanNet		PSNR↑	DTU	
	PSNR↑	SSIM↑	LPIPS↓		SSIM↑	LPIPS↓		SSIM↑	LPIPS↓
NPBG	30.47	0.957	0.046	25.08	0.737	0.459	30.24	0.895	0.130
NPBG++	30.52	0.960	0.042	25.24	0.755	0.444	30.25	0.894	0.131
Ours	31.74	0.968	0.029	25.88	0.794	0.414	30.81	0.904	0.128
NeRF	33.07	0.989	0.064	25.63	0.765	0.436	31.03	0.909	0.127

Table 2: Per-scene PSNR in NeRF-Synthetic.

Method	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
NPBG	28.81	23.57	28.23	32.03	27.72	27.24	31.16	26.04	28.10
NPBG++	28.72	23.60	28.11	32.22	27.84	27.12	31.23	26.11	28.12
Ours	31.13	24.51	29.09	33.20	26.62	28.03	32.94	26.14	28.96
NeRF	33.00	25.01	30.13	36.18	32.54	29.62	32.91	28.65	31.01

The End-to-End Point Clouds Rendering Pipeline

Our pipeline could be divided into three stages: rasterization, radiance mapping and refinement. In rasterization stage, we first transform the point clouds from world space to camera space. For each pixel, we search the points that lie within a radius threshold τ to the ray omitted from the camera to the pixel. The obtained points with their properties are so-called *fragments*. Since this searching process is not required to be differentiable, it could be implemented by hardware-accelerated framework such as OpenGL (Shreiner, Group et al.) and ran in real time. In our experiments, we find the OpenGL-based implementation are 8-10 times faster than a pure software implementation, such as PyTorch3D (Johnson et al.). Based on the fragments obtained from the per-pixel search, we keep the closest point to the camera for the occupied pixels and abandon the rest, since we only evaluate the first intersected surface. Note that there are some unoccupied pixels that no points are found within the threshold τ . These pixels leads to the artifact caused by the sparsity of point clouds and would be processed in the refinement stage. We transform the raw coordinates of the occupied pixels to the queried coordinates using the rectification in Eqn. 5. Then we transform the queried coordinates $\mathbf{x}_{query} = N_{occ} \times 3$ with position encoding introduced in NeRF (Mildenhall et al.) and feed them into our spatial mapping function \mathbf{F}_{Θ} using Eqn. 4, and obtain the latent feature $\mathbf{L} = N_{occ} \times C$. Then we re-organize these features back to a $H \times W \times C$ feature map, while we assign zero to the feature of those unoccupied pixels. The feature map output by MLP goes into the refinement stage. Unlike NeRF representation which uses a pure MLP design, the refinement stage is indispensable for point clouds rendering. The spatial mapping stage guarantees multi-view consistency, but outputs noisy 3D feature, while the refinement network is agnostic to the 3D geometry, but learn the texture of the images relying on the capability of convolution layers. The refinement stage is implemented as a 2D U-Net (Ronneberger, Fischer, and Brox). After the feature map is processed by the U-Net, we obtain the rendered image in shape of $H \times W \times 3$. We su-

pervised the training by using the weighted sum of L2-loss and perceptual loss (Johnson, Alahi, and Fei-Fei; Dosovitskiy and Brox) as done in (Dai et al.), whose weights are 1 and 0.01 respectively.

Experiments

Benchmark Evaluation

Settings We experiment on three datasets: NeRF-Synthetic (Mildenhall et al.), ScanNet Dai et al. and DTU Aanæs et al.. We give a detailed description on them in the appendix. To speed up the training process, we only rasterize the point clouds once and save the fragments for reusing, since the training and testing views are fixed. We set the radius threshold τ as $5e-3$ for Nerf-Synthetic, $1.5e-2$ for ScanNet, and $3e-3$ for DTU. The selection of τ depends on the density of point clouds. If τ is set too large, the rasterized points would be inaccurate. If τ is set too small, then more pixels would become unoccupied, causing a so-called bleeding issue. We employ the same positional encoding to the input queried coordinates and ray directions as done in NeRF. The output feature dimensions of each MLP layer are 256, 256, 256, 128 and 8 respectively. The input ray directions are concatenated with the output of second fully-connected layer. We use the same U-Net architecture as NPBG, where gated blocks are employed, consisting of gated convolution, ReLU and instance normalization layers. The U-Net down-samples the feature map for four times. The feature dimensions of each layer are 16, 32, 64, 128, 256. We use the Adam optimizer for training, with a batch size of 1. The initial learning rates of MLP and U-Net are $5e-4$ and $1.5e-4$ respectively, which are multiplied by 0.9999 in each step. We compare our methods with NPBG, NPBG++ and NeRF, using PSNR, SSIM and LPIPS as evaluation metrics.

Results We show the overall quantitative results in Table 1. Our method outperform prior arts with notable margins on each dataset, in terms of PSNR, SSIM and LPIPS. Compared to NeRF, our method achieves even better re-

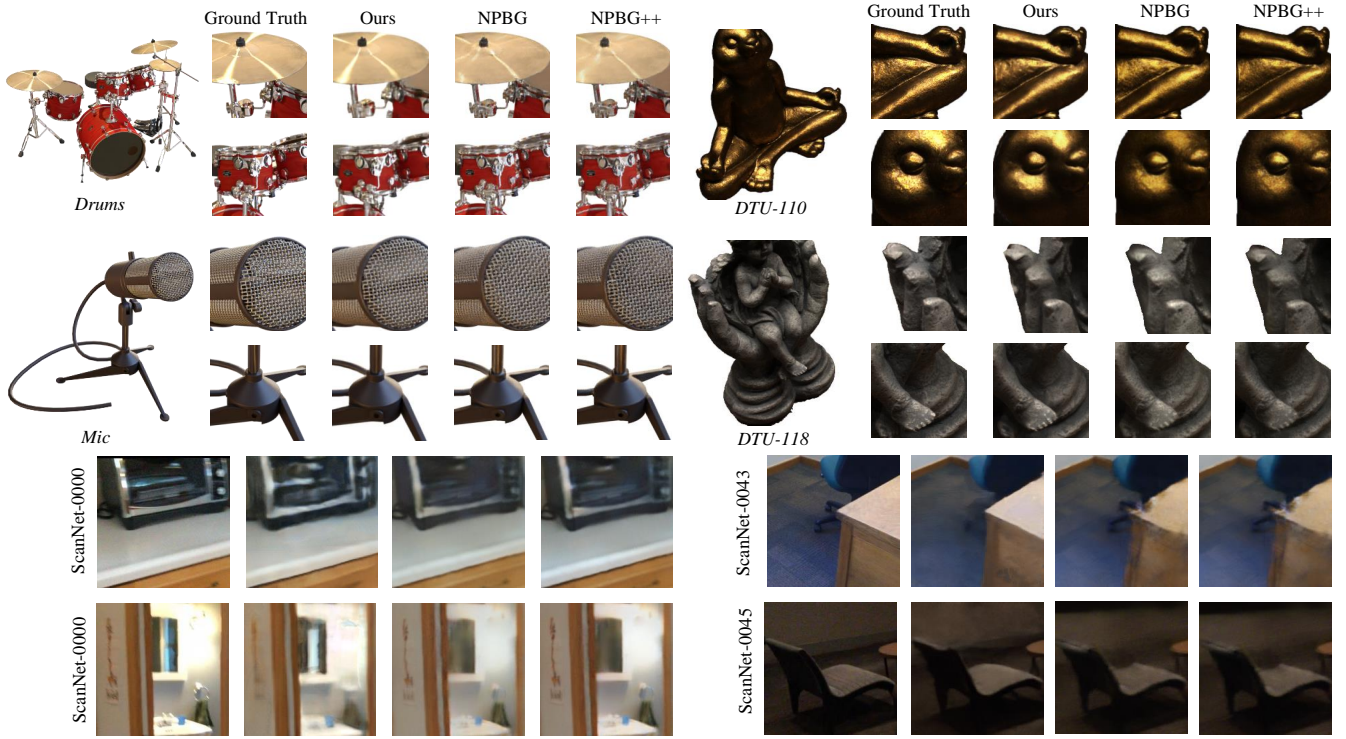


Figure 4: We compare the rendered novel views between our method and prior works. From the Drums scenes, we could see that the reflection rendered by our method is more photo-realistic than NPBG and NPBG++. Similar phenomenon is also observed in DTU-110. Besides, our method is able to produce elaborate details, such as the fine holes in the Mic scene, and the toes in DTU-118. We urge readers to zoom in the images for better observations.

sults on ScanNet, and comparable results on the other two datasets. It demonstrates the effectiveness of our methods under various scenes. Table 2 show the PSNR on each scene in NeRF-Synthetic. Please refer to the appendix for all metrics. From Table 2, we could see our methods achieve better results than NPBG and NPBG++ on 6 scenes out of 8, except for Lego and Ship. We visualize the results on these two scenes and their noisy point clouds in the appendix. In Figure 4, we illustrate the zoom-in views of four scenes. From the Drums scenes, we could see that the reflection rendered by our method is more photo-realistic than NPBG and NPBG++. Similar results are observed in DTU-110. Besides, our method is able to produce elaborate details, such as the fine holes in the Mic scene, and the toes in DTU-118. More results are presented in the appendix and supplementary videos. We argue the reason that NPBG and NPBG++ lose fine details is the spatial frequency collapse and neighbor point disturbance issues caused by the discrete neural descriptors. Figure 2 compare our method against NeRF. As analyzed before, our method and NeRF both model the translucency by optimizing the color of first-intersected surface. Our method avoids blurry artifact by leveraging the explicit geometry provided by point clouds. Besides, from the Materials scene, we observe that our method produces better specular reflection effect.

Complexity Analysis

(a) Model Size Our model includes a 5-layer MLP for radiance mapping and a U-Net for refinement. Their sizes are 0.75M and 7.3M respectively. The NeRF MLP has a size of 4.8M. NPBG (Aliiev et al.) uses the same U-Net architecture as ours, but its neural descriptors are proportional to the number of points. Demonstrated with ScanNet, whose point clouds are about in the size of 2 millions points, the model size of the neural descriptors would be 8M. NPBG++ (Rakhimov et al.) also use neural descriptors and a refinement U-Net. One difference is that it uses an additional U-Net as the image feature extractor to obtain the neural descriptor. Their sizes are 53M in total. Besides, NPBG++ uses a smaller U-Net in refinement stage, whose size is 1.2M. PlenOctree has a much larger model size as 440M. **(b) Time Complexity** In Table 4, we show the Frame Per Second (FPS) of each method. All results are measured on a TiTAN Xp with an image size of 800×800 . The point clouds-based methods (ours, NPBG and NPBG++) require rasterization for pre-processing (All implemented in OpenGL). NPBG and NPBG++ need to rasterize the point clouds five times into five different resolution to feed into the refinement U-Net, while ours only need one. Therefore their FPS are slightly lower than ours. Our method outperforms these two methods with a FPS of 20. NeRF needs burdensome pre-processing, which severely slower the speed, yielding a 0.015 FPS. Point-NeRF follows the volume rendering for-

Table 3: **(a)** We verify our method under various density of point clouds. $1\times$, $10\times$ and $50\times$ denote down-sampling the point clouds by 1, 10 and 50 times. *w/o Rect.* denotes our method without coordinate rectification. The sparser the point cloud is, the more improvement brought by our method. **(b)** We analyze the influence of the refinement network capacity. *Quad dim* denotes increasing the feature dimension by a factor of 4. Results are obtained using 400×400 resolution due to GPU memory limitations. The three metrics are PSNR \uparrow , SSIM \uparrow and LPIPS \downarrow respectively.

Method	Chair $\uparrow / \uparrow / \downarrow$	Drums $\uparrow / \uparrow / \downarrow$	Lego $\uparrow / \uparrow / \downarrow$	Mic $\uparrow / \uparrow / \downarrow$
NPBG++, $1\times$	29.33 / 0.950 / 0.045	24.68 / 0.919 / 0.067	27.27 / 0.905 / 0.080	32.72 / 0.971 / 0.026
Ours, $1\times$, w/o Rect.	32.56 / 0.969 / 0.024	24.57 / 0.923 / 0.056	26.48 / 0.898 / 0.069	32.36 / 0.979 / 0.017
Ours, $1\times$	33.63 / 0.977 / 0.021	25.05 / 0.931 / 0.051	27.16 / 0.910 / 0.066	33.44 / 0.983 / 0.014
NPBG++, $10\times$	29.20 / 0.949 / 0.047	23.90 / 0.903 / 0.085	27.41 / 0.908 / 0.091	32.00 / 0.965 / 0.033
Ours, $10\times$, w/o Rect.	31.24 / 0.958 / 0.030	23.77 / 0.910 / 0.069	25.88 / 0.887 / 0.083	31.42 / 0.975 / 0.022
Ours, $10\times$	32.46 / 0.967 / 0.028	24.26 / 0.919 / 0.065	26.46 / 0.900 / 0.077	32.91 / 0.981 / 0.016
NPBG++, $50\times$	27.82 / 0.929 / 0.058	22.72 / 0.876 / 0.104	25.72 / 0.871 / 0.119	31.21 / 0.959 / 0.039
Ours, $50\times$, w/o Rect.	29.89 / 0.945 / 0.038	22.43 / 0.888 / 0.088	24.67 / 0.867 / 0.101	29.99 / 0.968 / 0.026
Ours, $50\times$	31.34 / 0.959 / 0.034	22.99 / 0.900 / 0.082	25.51 / 0.888 / 0.088	31.64 / 0.975 / 0.021
Ours, $1\times$, Qurd dim	34.13 / 0.980 / 0.019	25.53 / 0.937 / 0.044	27.46 / 0.918 / 0.055	34.44 / 0.987 / 0.009

Table 4: Comparisons of model size and Frame Per Second (FPS). For point clouds rendering methods, the model size is separated into spatial mapping and refinement stages. All results are measured on a TITAN Xp.

Method	Model Size \downarrow	FPS \uparrow
NPBG	(8+7.3)M	17
NPBG++	(53+1.2)M	18
Ours	(0.75+7.3)M	20
NeRF	4.8M	0.015
Point-NeRF	18.5M	0.125
PlenOctrees	440M	110

mulation like NeRF with point clouds as input. Hence it also suffers from heavy sampling and high computation cost, with an FPS of 0.125.

Ablation Study

Point clouds Density We evaluate our methods under different point clouds density. We select four scenes from NeRF-Synthetic dataset (Chair, Drums, Lego and Mic) for evaluation. We conduct these experiments with a 400×400 resolution, without affecting the conclusion. Specifically, we uniformly down-sample the original point clouds by a factor of 10 and 50 respectively, to explore how much improvement the proposed method brings under different point cloud density. We compare our method against NPBG++ and our baseline without coordinate rectification. We show the PSNR, SSIM, and LPIPS in Table 3. It’s seen that our method achieves best performance in almost all cases. Moreover, the sparser the point cloud is, the greater improvement is brought by our method over NPBG++ and the baseline. The reason is that in NPBG++ and the baseline, the same neural feature would be rasterized into multiple pixels and optimized to fit the average of these pixels, which would harms the rendering quality. When the point cloud becomes sparser, this issue becomes more serious. Our method with

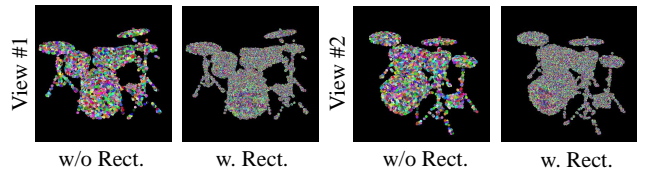


Figure 5: Rasterized feature of our method and baseline (w/o Rect.). Without coordinate rectification, the same neural feature would be rasterized into multiple pixels and optimized to fit the average of these pixels, which would harms the rendering quality. Our method could notably allivete the issue.

coordinate rectification could notably alleviate this issue. In Figure 5, we illustrate the rasterized feature of our method and the baseline, regarding three channels as RGB. NPBG and NPBG++ has similar effect as baseline (w/o Rect.) does.

Capacity of Refinement Network Since the refinement network is actually playing the role of denoising or inpainting, it would be important to know how the capacity of the network influence the image quality. We increase the channel dimension in each layer of the refinement network by a factor of four, yielding a model size of 127M. As shown in the bottom row in Table 3, increasing the capacity further improve our performance.

Conclusion

In this work, we propose a point clouds rendering method that achieve photo-realistic rendering quality. We begin with revisiting the spatial mapping functions in prior art and then propose our radiance mapping based on the adaptation of volumetric rendering formulation on point clouds. Further, we propose coordinate rectification which alleviate the spatial frequency collapse and neighbor point disturbance issues. Composed of rasterization, spatial mapping and the refinement stages, Our method achieves the state-of-the-art performance on point clouds rendering task.

References

- Aanæs, H.; Jensen, R. R.; Vogiatzis, G.; Tola, E.; and Dahl, A. B. 2016. Large-scale data for multiple-view stereopsis. *International Journal of Computer Vision*, 120(2): 153–168.
- Aliiev, K.-A.; Sevastopolsky, A.; Kolos, M.; Ulyanov, D.; and Lempitsky, V. 2020. Neural point-based graphics. In *Euro-pean Conference on Computer Vision*, 696–712. Springer.
- Barron, J. T.; Mildenhall, B.; Tancik, M.; Hedman, P.; Martin-Brualla, R.; and Srinivasan, P. P. 2021. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 5855–5864.
- Chen, A.; Xu, Z.; Geiger, A.; Yu, J.; and Su, H. 2022. TensorRF: Tensorial Radiance Fields. *arXiv preprint arXiv:2203.09517*.
- Dai, A.; Chang, A. X.; Savva, M.; Halber, M.; Funkhouser, T.; and Nießner, M. 2017. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 5828–5839.
- Dai, P.; Zhang, Y.; Li, Z.; Liu, S.; and Zeng, B. 2020. Neural point cloud rendering via multi-plane projection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7830–7839.
- Dosovitskiy, A.; and Brox, T. 2016. Generating images with perceptual similarity metrics based on deep networks. *Advances in neural information processing systems*, 29.
- Godard, C.; Mac Aodha, O.; and Brostow, G. J. 2017. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 270–279.
- He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.
- Insafutdinov, E.; and Dosovitskiy, A. 2018. Unsupervised learning of shape and pose with differentiable point clouds. *Advances in neural information processing systems*, 31.
- Jain, A.; Tancik, M.; and Abbeel, P. 2021. Putting nerf on a diet: Semantically consistent few-shot view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 5885–5894.
- Johnson, J.; Alahi, A.; and Fei-Fei, L. 2016. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, 694–711. Springer.
- Johnson, J.; Ravi, N.; Reizenstein, J.; Novotny, D.; Tulsiani, S.; Lassner, C.; and Branson, S. 2020. Accelerating 3d deep learning with pytorch3d. In *SIGGRAPH Asia 2020 Courses*, 1–1.
- Kopanas, G.; Philip, J.; Leimkühler, T.; and Drettakis, G. 2021. Point-Based Neural Rendering with Per-View Optimization. In *Computer Graphics Forum*, volume 40, 29–43. Wiley Online Library.
- Li, Z.; Li, L.; Ma, Z.; Zhang, P.; Chen, J.; and Zhu, J. 2022. READ: Large-Scale Neural Scene Rendering for Autonomous Driving. *arXiv:2205.05509*.
- Lin, C.-H.; Kong, C.; and Lucey, S. 2018. Learning efficient point cloud generation for dense 3d object reconstruction. In *proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.
- Liu, L.; Gu, J.; Zaw Lin, K.; Chua, T.-S.; and Theobalt, C. 2020. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33: 15651–15663.
- Martin-Brualla, R.; Radwan, N.; Sajjadi, M. S.; Barron, J. T.; Dosovitskiy, A.; and Duckworth, D. 2021. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7210–7219.
- Mildenhall, B.; Srinivasan, P. P.; Tancik, M.; Barron, J. T.; Ramamoorthi, R.; and Ng, R. 2020. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, 405–421. Springer.
- Niemeyer, M.; Mescheder, L.; Oechsle, M.; and Geiger, A. 2020. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3504–3515.
- Oechsle, M.; Peng, S.; and Geiger, A. 2021. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 5589–5599.
- Ost, J.; Laradji, I.; Newell, A.; Bahat, Y.; and Heide, F. 2021. Neural Point Light Fields. *CoRR*, abs/2112.01473.
- Rakhimov, R.; Ardelean, A.-T.; Lempitsky, V.; and Burnaev, E. 2022. NPBG++: Accelerating Neural Point-Based Graphics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.
- Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, 234–241. Springer.
- Rückert, D.; Franke, L.; and Stamminger, M. 2021. Adop: Approximate differentiable one-pixel point rendering. *arXiv preprint arXiv:2110.06635*.
- Shreiner, D.; Group, B. T. K. O. A. W.; et al. 2009. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. Pearson Education.
- Wang, F.; Galliani, S.; Vogel, C.; Speciale, P.; and Pollefeys, M. 2021a. Patchmatchnet: Learned multi-view patch-match stereo. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 14194–14203.
- Wang, P.; Liu, L.; Liu, Y.; Theobalt, C.; Komura, T.; and Wang, W. 2021b. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction. In *Advances in Neural Information Processing Systems*.
- Wang, Q.; Wang, Z.; Genova, K.; Srinivasan, P. P.; Zhou, H.; Barron, J. T.; Martin-Brualla, R.; Snavely, N.; and

- Funkhouser, T. 2021c. Ibrnet: Learning multi-view image-based rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4690–4699.
- Wang, T.-C.; Liu, M.-Y.; Zhu, J.-Y.; Tao, A.; Kautz, J.; and Catanzaro, B. 2018. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 8798–8807.
- Wiles, O.; Gkioxari, G.; Szeliski, R.; and Johnson, J. 2020. Synsin: End-to-end view synthesis from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7467–7477.
- Xu, Q.; Xu, Z.; Philip, J.; Bi, S.; Shu, Z.; Sunkavalli, K.; and Neumann, U. 2022. Point-NeRF: Point-based Neural Radiance Fields. *arXiv preprint arXiv:2201.08845*.
- Yao, Y.; Luo, Z.; Li, S.; Fang, T.; and Quan, L. 2018. Mvs-net: Depth inference for unstructured multi-view stereo. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 767–783.
- Yariv, L.; Gu, J.; Kasten, Y.; and Lipman, Y. 2021. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems*, 34.
- Yariv, L.; Kasten, Y.; Moran, D.; Galun, M.; Atzmon, M.; Ronen, B.; and Lipman, Y. 2020. Multiview neural surface reconstruction by disentangling geometry and appearance. *Advances in Neural Information Processing Systems*, 33: 2492–2502.
- Yifan, W.; Serena, F.; Wu, S.; Öztireli, C.; and Sorkine-Hornung, O. 2019. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics (TOG)*, 38(6): 1–14.
- Yu, A.; Li, R.; Tancik, M.; Li, H.; Ng, R.; and Kanazawa, A. 2021a. Plenotrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 5752–5761.
- Yu, A.; Ye, V.; Tancik, M.; and Kanazawa, A. 2021b. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4578–4587.
- Zhou, T.; Brown, M.; Snavely, N.; and Lowe, D. G. 2017. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1851–1858.
- Zwicker, M.; Pfister, H.; Van Baar, J.; and Gross, M. 2001. Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 371–378.

Datasets

NeRF-Synthetic NeRF-Synthetic (Mildenhall et al.) is a high quality synthetic dataset containing pathtraced images of 8 objects that exhibit complex geometry and realistic non-Lambertian materials. For each object, there are 100 images for training and 200 images for testing. We use the point clouds provided by Point-NeRF (Xu et al.). During training stage, we conduct data augmentation by random scaling and cropping as done in prior works. We set the training window size as 800×800 . At test time, we render novel view images with the original resolution 800×800 . In main paper (Table 1), we report the results on Ficus, HotDog and Mic, as done in NPBG++ (Rakhimov et al.). Besides, we additionally conduct experiments on all eight scenes and show their PSNR in main paper (Table 2), while the rest metrics are shown in the appendix (Table 5, 6).

ScanNet ScanNet (Dai et al.) is a RGBD dataset of indoor scenes. We evaluate our method on three scenes $\{scene0000_00, scene0043_00 \text{ and } scene0045_00\}$ as done in NPBG++ (Rakhimov et al.) and follow their train-test-split. Specifically, if there are less than 2000 frames in the scene, we take 100 frames with equal spacing in the image stream. If there are more than 2000 frames, we take every 20th image such that the training views would not be too sparse. *scene0000_00* belongs to the latter case and the other two belong to the former. Then, we sample 10 frames at a fixed interval for testing and the rest for training. The training window size is set to 720×720 and the test resolution is 960×1200 .

DTU DTU (Aanaes et al.) is a multi-view stereo dataset captured at a resolution of 1200×1600 . We validate our approach on three scenes $\{scan110, scan114, scan118\}$ as done in NPBG++ (Rakhimov et al.) and use the same point clouds as theirs, which are obtained by PatchMatchNet (Wang et al.). We also perform the same train-test-split in each scene. NPBG++ evaluate the whole images including the background. In our empirical observations, we find that the background is dark and far from the camera, producing a extremely noisy point cloud. Although the models are able to render high-quality details of the objects, the numerical results are still very low. We argue that such evaluation settings do not match our visual effects and is not reasonable. Therefore, we mask out the background both in training and testing using the binary segmentation masks provided by IDR (Yariv et al.). All methods are evaluated using this settings. The training window size is set to 800×800 and the test resolution 1200×1600 .

Limitation

The proposed method produces photo-realistic images based on point clouds, whose performance is comparable to NeRF. However, similar to previous point clouds rendering methods, our performance is restricted by the point clouds quality. Specifically, when the point clouds quality is extremely poor, such as the one in the Lego scene in NeRF-Synthetic, our performance would degrade severely, while NeRF is hardly affected. Besides, although our method produces bet-

ter visual effects in some cases such as specular reflection compared to NeRF, we does not outperform NeRF in terms of numerical results.

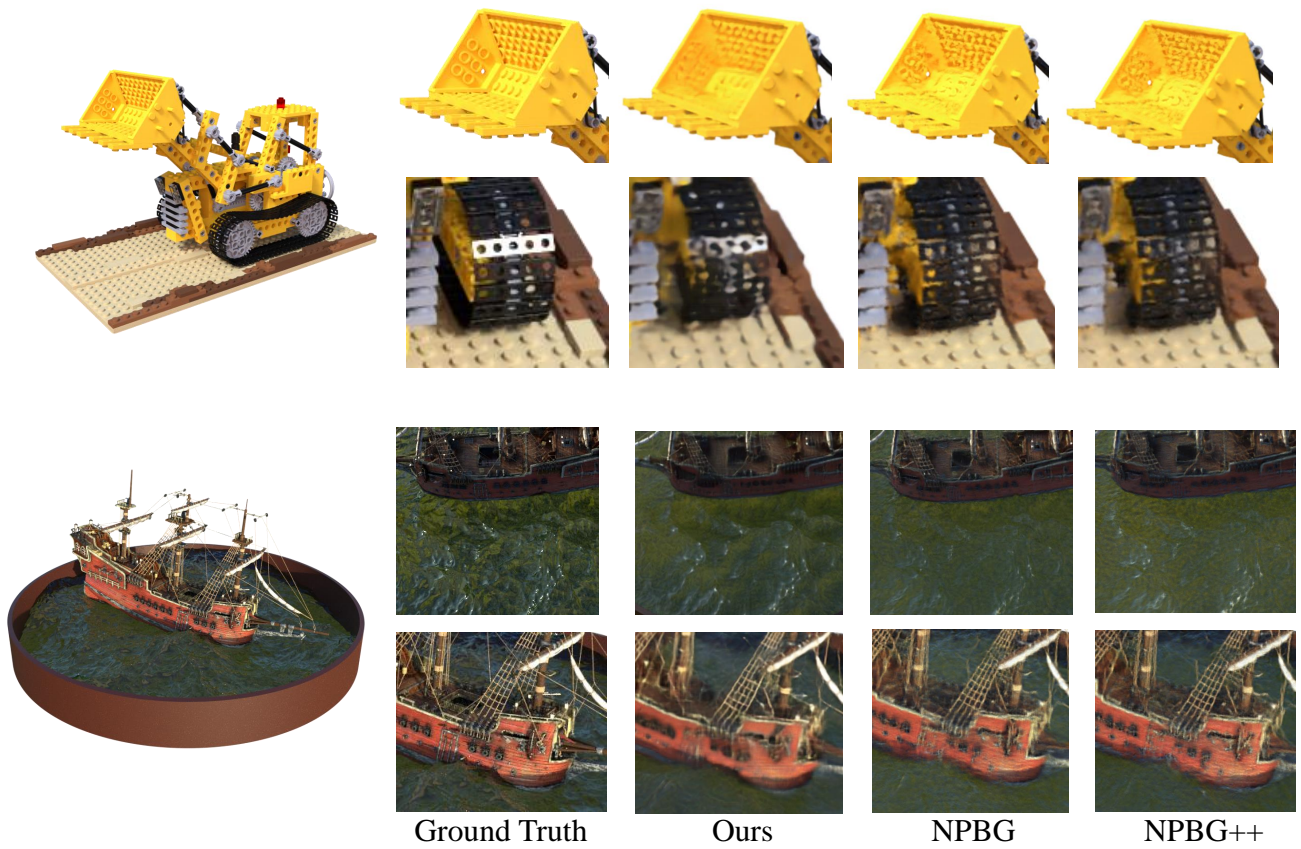


Figure 6: Qualitative results on Lego and Ship scenes. In Lego scene, a layer of points are wrongly reconstructed by MVSNet, which lie ahead of the real surface on the shovel (See first row images). With the wrong geometry, fitting one view well might result in 3D inconsistency in another view, leading to unusual artifact for all three point-based methods. The second row images of Lego shows that our method produce better specular effect compared to NPBG and NPBG++. We also show the noisy point clouds of these two scenes for demonstration.

Table 5: SSIM \uparrow on each scene in NeRF-Synthetic

Method	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
NPBG	0.954	0.902	0.942	0.960	0.919	0.922	0.970	0.812	0.923
NPBG++	0.961	0.910	0.947	0.961	0.923	0.925	0.972	0.822	0.928
Ours	0.953	0.924	0.958	0.964	0.902	0.945	0.983	0.824	0.932
NeRF	0.992	0.956	0.985	0.991	0.988	0.982	0.992	0.928	0.977

Table 6: LPIPS \downarrow on each scene in NeRF-Synthetic

Method	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
NPBG	0.047	0.093	0.046	0.055	0.089	0.076	0.037	0.171	0.077
NPBG++	0.048	0.092	0.043	0.053	0.089	0.074	0.030	0.178	0.076
Ours	0.040	0.068	0.035	0.038	0.085	0.050	0.014	0.159	0.061
NeRF	0.046	0.091	0.044	0.121	0.050	0.063	0.028	0.206	0.081

Table 7: PSNR \uparrow on each scene in ScanNet and DTU

Method	ScanNet-0000	ScanNet-0043	ScanNet-0045	DTU-110	DTU-114	DTU-118
NPBG	22.24	25.27	27.75	29.20	28.05	33.47
NPBG++	22.25	25.43	28.03	29.50	27.84	33.43
Ours	23.79	25.26	28.59	30.79	27.91	33.72
NeRF	22.10	25.64	29.16	30.32	28.38	34.38

Table 8: SSIM \uparrow on each scene in ScanNet and DTU

Method	ScanNet-0000	ScanNet-0043	ScanNet-0045	DTU-110	DTU-114	DTU-118
NPBG	0.695	0.830	0.686	0.907	0.873	0.904
NPBG++	0.712	0.821	0.731	0.910	0.865	0.909
Ours	0.748	0.844	0.789	0.924	0.863	0.926
NeRF	0.710	0.833	0.753	0.916	0.870	0.942

Table 9: LPIPS \downarrow on each scene in ScanNet and DTU

Method	ScanNet-0000	ScanNet-0043	ScanNet-0045	DTU-110	DTU-114	DTU-118
NPBG	0.474	0.421	0.482	0.124	0.143	0.123
NPBG++	0.456	0.412	0.465	0.125	0.148	0.121
Ours	0.440	0.389	0.415	0.114	0.154	0.117
NeRF	0.461	0.402	0.447	0.124	0.140	0.116