Frequency-Modulated Point Cloud Rendering with Easy Editing

Yi Zhang*, Xiaoyang Huang*, Bingbing Ni†, Teng Li², Wenjun Zhang¹
¹Shanghai Jiao Tong University, Shanghai 200240, China ²Anhui University
{yizhangphd, huangxiaoyang, niblingbing}@sjtu.edu.cn

Abstract

We develop an effective point cloud rendering pipeline for novel view synthesis, which enables high fidelity local detail reconstruction, real-time rendering and user-friendly editing. In the heart of our pipeline is an adaptive frequency modulation module called Adaptive Frequency Net (AFNet), which utilizes a hypernetwork to learn the local texture frequency encoding that is consecutively injected into adaptive frequency activation layers to modulate the implicit radiance signal. This mechanism improves the frequency expressive ability of the network with richer frequency basis support, only at a small computational budget. To further boost performance, a preprocessing module is also proposed for point cloud geometry optimization via point opacity estimation. In contrast to implicit rendering, our pipeline supports high-fidelity interactive editing based on point cloud manipulation. Extensive experimental results on NeRF-Synthetic, ScanNet, DTU and Tanks and Temples datasets demonstrate the superior performances achieved by our method in terms of PSNR, SSIM and LPIPS, in comparison to the state-of-the-art. Code is released at https://github.com/yizhangphd/FreqPCR.

1. Introduction

Photo-realistic rendering and editing of 3D representations is a key problem in 3D computer vision and graphics with numerous applications, such as computer games, VR/AR, and video creation. In particular, recently introduced neural radiance field (NeRF) [21] has inspired some follow-up works aiming to editable rendering [15, 17, 20, 47, 49, 55]. Due to the deeply coupled black-box network, NeRF-based object-level editing usually requires a pre-trained segmentation model to separate the objects to be edited [17, 55]. Although some recent voxel-based variants of NeRF [47, 58] achieve multi-scene composition, they still lack the ability to extract target objects from voxels.

In contrast to implicit rendering, point cloud rendering [1, 6, 13, 18, 33, 36, 57] is a promising editable rendering model. On the one hand, explicit 3D representations are better for interactive editing. On the other hand, the geometric priors provided by point clouds can help us avoid massive sampling in volume rendering methods, which can meet the requirements of some real-time applications. As a class of representative point cloud rendering methods, NPBG and NPBG++ [1, 33] achieve real-time rendering by using point-wise features for encoding appearance information and an U-Net [35] for decoding, respectively. However, the parameter quantity increases with the size of point cloud, which may limit their application due to the excessive computational and memory complexity. Huang et al. [13] combine point clouds with implicit rendering, where explicit point clouds are used to estimate the geometry, and implicit radiance mapping is used to predict view-dependent appearance of surfaces. However, quantitative evaluation of its rendering results is significantly lower than that of implicit rendering methods such as NeRF [21], mainly due to the following reasons: 1) the color of each viewing ray only depends on

Figure 1. We propose a novel frequency-modulated point cloud rendering pipeline, which enables high fidelity local detail reconstruction, real-time rendering and user-friendly editing. Specifically, we can render sharper textures with 1700× faster than NeRF [21] and support both object-level editing and scene composition.
a single surface point, thus without multiple sample color aggregation for error attenuation, surface based rendering techniques require radiance mapping to have a more precise and expressive frequency encoding ability; and 2) defects of the point cloud geometry reconstructed by MVSNet [56] cause wrong surface estimation. To this end, we introduce Adaptive Frequency Net (AFNet) to improve frequency expression ability of the radiance mapping and a preprocessing module for point cloud geometry optimization.

Radiance mapping, also known as radiance field, is a type of Implicit Neural Representation (INR). There have been some studies [2, 7, 32, 41, 45, 46, 60] on the expressive power and inductive bias of INRs. The standard Multi-layer Perceptrons (MLPs) with ReLU activation function are well known for the strong spectral bias towards reconstructing low frequency signals [32]. Some recent works [7, 41, 46] introduce strategies to enhance the high-frequency representation of MLPs from a global perspective. However, from a local perspective, the frequencies of a 3D scene are region-dependent and most real objects are composed by both weak textured regions and strong textured ones. Motivated by this, we design a novel adaptive frequency modulation mechanism based on HyperNetwork architecture [11], which learns the local frequency and injects it into adaptive frequency activation layers to modulate the implicit radiance signal. The proposed mechanism can predict suitable frequency without frequency supervision and modulate the radiance signal with adaptive frequency basis support to express more complex textures at negligible computational overhead.

Previous surface point-based works [1, 13, 33] could not optimize the point cloud geometry because they keep only the closest point as a surface estimation for each ray. But if we sample multiple points per ray during rendering, it will greatly reduce the rendering speed. Therefore, we use the volume rendering method as a preprocessing module to optimize the point cloud geometry. Specifically, we keep more points in the pixel buffer and learn the opacity of each point based on volume rendering. We find in our experiments that for some poorly reconstructed scenes, point cloud geometry optimization can improve the rendering PSNR by 2-4dB and avoid rendering artifacts.

For rigid object editing, we follow the deformation field construction [28–31, 48] to render the edited point cloud. Point cloud can be seen as a bridge between user editing and deformation field to achieve interactive editing and rendering. Users only need to edit the point cloud, and the deformation field between the original 3D space and the deformed space is easy to obtain by querying the corresponding transformations performed on point cloud. Moreover, to avoid cross-scene training in multi-scene composition application, we develop a masking strategy based on depth buffer to combine multiple scenes in pixel level.

We evaluate our method on NeRF-Synthetic [21], ScanNet [5], DTU [14] and Tanks and Temples [16] datasets and comprehensively compare the proposed method with other works in terms of performance (including PSNR, SSIM and LPIPS), model complexity, rendering speed, and editing ability. Our performance outperforms NeRF [21] and all surface point-based rendering methods [1, 13, 33], and is comparable to Compressible-composable NeRF (CCNeRF), i.e., the latest NeRF-based editable variant. We achieve a real-time rendering speed of 39.27 FPS on NeRF-Synthetic, which is 1700× faster than NeRF and 37× faster than CCNeRF. We also reproduce the scene editing of Object-NeRF [55] and CCNeRF [47] on ToyDesk [55] and NeRF-Synthetic [21] dataset, respectively. As shown in Fig. 1, we achieve real-time rendering with sharper details and user-friendly editing. The above results demonstrate that our method is comprehensive in terms of both rendering and editing and has great application potentials.

2. Related Work

Implicit Neural Representations. The goal of INRs is to encode a continuous target signal using a neural network, by representing the mapping between input coordinates and signal values. This technique has been used to represent various objects such as images [23, 43], shapes [4, 9, 27], scenes [21, 42], and textures [12, 25]. Tanick et al. [46] propose Fourier Feature Network (FFN), showing that feeding input coordinates through a Fourier feature mapping enables MLP to learn high-frequency functions in low dimensional domains. Sitzmann et al. [41] propose to use MLP with sinusoidal activations. Fathony et al. propose MFN [7], which employs a Hadamard product between linear layers and nonlinear activation functions. In [60], Yüce et al. prove that FFN [46], SIREN [41] and MFN [7] have the same expressive power as a structured signal dictionary.

3D Representations for View Synthesis. Implicit representations often use neural networks to encode 3D scenes [21, 24, 26, 27, 39, 50, 51, 61]. NeRF [21] uses a 5D function to represent the scene and applies volumetric rendering for view synthesis. Many recent works combine NeRF with some explicit representation to improve rendering quality or speed. NSVF [19] uses a sparse voxel octree to achieve fast rendering. Yu et al. propose PlenOctrees [58], a data structure derived from NeRF [21] which enables highly efficient rendering. Plenoxels [8] represents a scene as a sparse 3D grid with spherical harmonics. Xu et al. [52] propose Point-NeRF, which is a localized neural representation, combining volumetric radiance fields with point clouds. Instant-NGP [22] uses multi-resolution hashing for efficient encoding and also leads to high compactness. TensoRF [3] factorizes the scene tensor into lower-rank components. Huang et al. [13] com-
bine point cloud with implicit radiance mapping to achieve the state-of-the-art performance of point cloud rendering. However, their performance is still worse than NeRF [21]. In this work, we greatly improve the performance of point cloud rendering by introducing AFNet and a preprocessing module for geometry optimization.

**Editable Rendering.** Editing ability is important for 3D representations. Some recent works [10, 44, 53–55, 59] improve NeRF to editable version. Yang et al. [55] propose Object-NeRF for editable rendering, while it requires a 2D segmentation model to extract the target object and renders at a slow speed. Yuan et al. propose NeRF-Editing [59], which allows users to perform controllable shape deformation on the implicit representation. Yang et al. [54] present a mesh-based representation by encoding the implicit field with disentangled geometry and texture codes on mesh vertices, which facilitates a set of editing functionalities. Tang et al. propose CCNeRF [47], which could transform and compose several scenes into one scene with a small model.

### 3. Method

In this section, we first revisit the state-of-the-art point cloud rendering technique and discuss its dilemma of insufficient frequency representation, in Sec. 3.1. In Sec. 3.2, we present our proposed frequency-modulated radiance mapping technique to address above limitation, followed by the introduction to our end-to-end point cloud rendering pipeline in Sec. 3.3. We also introduce a novel point cloud geometry optimization pipeline as an optional preprocessing module in Sec. 3.4. Applications of the above pipeline on interactive object level editing and multi-scene composition are presented in Sec. 3.5.

#### 3.1. Revisiting Point Cloud Rendering

Some traditional point cloud rendering methods focus on fast rendering of large-scale point clouds [37, 38], while they are not as photorealistic as neural rendering (e.g., not smooth and lack of view-dependent appearance). Different from traditional works, our approach builds on a state-of-the-art point-based neural rendering method [13], which utilizes point cloud to determine the position of the surface by the following coordinate rectification

\[
x = o + \frac{z}{\cos \theta} \times d,
\]

where \(o\), \(d\) and \(\theta\) denote camera position, normalized ray direction and angle between the ray and the \(z\)-axis pointing to the center of the scene, respectively. In practice, \(z\) value is obtained by querying the depth buffer during rasterization. As point clouds are discrete, each point is expanded to a disk of radius \(r\) during rasterization [63]. The estimated surface coordinates \(x\) together with ray direction \(d\) are fed to an MLP \(F_\Theta\) (\(\Theta\) is the network parameter set) to obtain the features \(L\) which encode the color of the surface point

\[
L = F_\Theta(x, d).
\]

Compared to volume rendering-based models [21, 52], this method avoids heavy sampling by estimating surface coordinates from point clouds when calculating ray color, and thus achieving real-time rendering. As point clouds are usually noisy and the initial depth obtained by rasterization might also be inaccurate, previous methods [1, 13, 33] often use U-Net structure [35] for further refinement.

**Limitations.** The limitations of the above surface point-based rendering scheme are obvious. Namely, the point cloud quality is usually poor due to sparsity and large number of outliers, which leads to lower-fidelity rendering especially in high frequency shapes, i.e., highly-3D-textured regions. Simply increasing the network can not solve the problem. On the contrary, prevailing volumetric rendering techniques such as NeRF [21] do not suffer from this issue (i.e., usually with higher rendering quality), since through each sampled ray there exist multiple sampling points near the surface fused together to contribute the target color, which is more stable and sufficient for dealing with high frequency shapes. Therefore, it is highly demanded to enhance the frequency expression ability of point cloud based implicit surface radiance mapping method at a relatively low computational cost overhead.

#### 3.2. Frequency-Modulated Radiance Mapping

**Motivation.** To achieve radiance mapping, also known as radiance field, previous works input a spatial coordinate and infer the color or other features associated with this spatial position via standard MLP with ReLU activation layer. However, this classical neural network is well-known for its strong spectral bias towards generating lower frequency patterns [32], which has obvious difficulty in handling high frequency shape details, e.g., 3D textures of objects and occupancy of edges. Recent works [7, 41, 46] analyze the expres-
Adaptive Frequency Estimation and Modulation. The major working mechanism of the proposed AFNet is to first estimate the local frequency information for each point, and then this frequency encoding is injected to the proposed Adaptive Frequency Activation Layers (AF Layers) which adaptively select a proper set of frequency component, providing a more precise support for generating high-fidelity reconstruction, as shown in Fig. 3. Therefore, the core building block of the proposed AFNet includes 1) AF Layers and 2) a hypernetwork for local frequency estimation, which are introduced in detail as follows.

Adaptive Frequency Activation Layer. Assuming local frequency information is successfully estimated and encoded as scalar frequencies and phases, our AF layers first encode the input radiance signal with a nonlinear filter (we use a sinusoidal filter) parameterized by the obtained point-wise adaptive frequency and phase. Then, a Hadamard product between the filtered feature and the input feature is taken as the activated output, which could be regarded as a frequency modulation on the input signal. Mathematically, suppose that the input feature of \( i \)-th layer is \( X^{(i)} \in \mathbb{R}^{M \times C} \), in which \( M \) and \( C \) are point number and feature dimension respectively, and the adaptive frequencies and phases are \( \omega^{(i)} \), \( \phi^{(i)} \in \mathbb{R}^{M \times 1} \), whose dimensions will be broadcast to \( \mathbb{R}^{M \times C} \) during computation. Then the adaptive frequency activation layers can be expressed as:

\[
Y^{(i)} = X^{(i)} \circ \sin(\omega^{(i)} \circ X^{(i)} + \phi^{(i)}),
\]

The design of Eqn. 3 has a theoretical justification. We measure the local frequency by the derivative of the color with respect to the spatial coordinate, and adaptively adjust the derivative by multiplying the feature \( X^{(i)} \) by the predicted frequency \( \omega^{(i)} \). We then use the sine function to constrain the numerical range. The final form, \( X^{(i)} \circ \sin(\cdot) \), is borrowed from MFN [7].

Hypernetwork for Frequency Estimation. The adaptive frequencies and phases are expected to contain important information of the 3D texture variation on the surface position; however, to obtain the accurate frequency representation of each point is non-trivial. Inspired by HyperNetwork [11], an approach of using a small network to generate the weights for a larger network, we use a two-layer MLP as hypernetwork to infer the frequency and phase parameters from each spatial point. In Fig. 2, we show the predicted frequency of the 4-th adaptive frequency activation layer. As seen in the RGB images, the back of the chair and the base of the ball have weak texture, while the frequency predicted by hypernetwork is closed to 0 (shown as black color). For strongly textured regions, such as patterned cushions in chair and the metal material, the hypernetwork predicts large frequency (shown as grey and white). These observations empirically indicates that the proposed hypernetwork is capable of learning correct frequency information even there is NO supervision on frequency label.

3.3. End-to-end Rendering Pipeline

Our pipeline consists of three stages: surface estimation, radiance mapping and refinement, as shown in Fig. 3. In the surface estimation stage, we first rasterize the point cloud \( P = \{p_1, p_2, \ldots, p_N\} \in \mathbb{R}^{N \times 3} \) to obtain the depth buffer \( z \in \mathbb{R}^{H \times W} \) at a given view. Specifically, we transform the point cloud from the world coordinate space to the normalized device coordinate space, in which \( p_k \in [-1, 1] \times [-1, 1] \times [z_{near}, z_{far}], k = 1, 2, ..., N \). Similarly, we transform the pixel coordinates \( I \in \mathbb{R}^{H \times W \times 2} \) to normalized space such that \( I_{i,j} \in [-1, 1]^2 \). Then, each point \( p_k \) is expanded to a disk of radius \( r_k \) in \( xOy \) plane.
We design a volume rendering-based pipeline for image denoising. We introduce a preprocessing module in our framework, to further enhance rendering quality. Then, we use Eqn. 1 to estimate the coordinate of the surface where each ray intersects.

In radiance mapping stage, the estimated surface coordinates are fed into the proposed AFNet to predict the raw RGB values and feature vectors. In the refinement stage, the feature vectors will be fed into an U-Net to predict the offsets. The final RGBs are obtained by adding the raw RGB values and offsets. AFNet and U-Net are trained with ground truth images as supervision by using L2 loss and perceptual loss.

3.4. Preprocessing: Geometry Optimization

In the meantime, point clouds obtained by MVS method [56] or 3D scans inevitably contain noise or missing parts of the surface, which harms the rendering quality or even produces artifacts. In view of this, we design a point cloud geometry optimization strategy consisting of a point cloud denoising step and a completion step, considered as an optional preprocessing module in our framework, to further enhance rendering quality.

Denoising. We design a volume rendering-based pipeline specifically to optimize the point cloud geometry. In rasterization, we maintain the depth buffer of \( k \) points instead of the nearest point. The depth values are used to compute the rectified coordinates of the points using Eqn. 1. The rectified coordinates, combined with ray directions, are fed into a MLP to estimate colors. Besides, an optimized opacity parameter \( \alpha_i \) is assigned to each point \( p_i \). Pixel RGB values are obtained using the volume rendering equation. Inspired by Point-NeRF [52], we train the MLP and the opacity parameters to fit the ground-truth images with the following sparse regularization:

\[
\mathcal{L}_{\text{sparse}} = \frac{1}{N} \sum_{i=1}^{N} \left[ \log(\alpha_i) + \log(1 - \alpha_i) \right].
\]

When the training converges, we remove those low-opacity points with a threshold \( \mu \).

Completion. For each view in training set, if a pixel in the ground truth image is occupied, but the buffer obtained by rasterization is empty, we add a set of points with opacity parameters along the pixel ray in the point clouds. To determine the depth range of the added points, we refer to the depth range of adjacent pixel buffer. Although the completion step introduces additional noise, it would be removed in the denoising stage of the next iteration.

Above denoising and completion steps run alternatively until convergence. Please see supplementary material for more details.

3.5. Scene Editing

In this section, we introduce an editing method based on the proposed rendering pipeline. At object level, we leverage point clouds to bridge the gap between user editing and deformation field. At scene level, the proposed method can combine different scenes without cross-scene training.

Object-level Editing. As an explicit 3D representation, point clouds are ideal for interactive editing. After editing target points, we can estimate the new surface coordinates of the edited point cloud by rasterization. Since radiance mapping encodes appearance of the surface in original 3D space, we need to transform the estimated surface points to original encoding space, which can be regarded as a deformation of the original space, i.e., deformation field [28, 31]. Specifically, we keep the indices of the points in pixel buffer during rasterization and query the transformations performed on those points. Then we perform the corresponding inverse transformation on the estimated surface coordinates to obtain the locations in original 3D space, which replaces the calculation of deformation field, as shown in Fig. 4. Finally, we feed the transformed surface points to radiance mapping module followed by refinement module to obtain the rendering result.

Scene Composition. A direct way to render a combined point cloud is to use respective AFNets for radiance mapping, and a shared U-Net for refinement. However, due to the inductive bias of U-Net, the texture of different scenes will affect each other, which would reduce the rendering quality. Moreover, cross-scene training of U-Net impairs the flexibility of editing, i.e., adding a new scene to an existing composition requires retraining together. To solve the above problems, we compose the scenes in pixel level. As
Table 1. Quantitative evaluation on NeRF-Synthetic [21], Tanks and Temples [16], ScanNet [5] and DTU [14] datasets. Since our work focuses on surface point-based method, which supports real-time and editable rendering, we mainly compare with NPBG [1], NPBG++ [33] and Huang et al. [13]. We also show the performance of NeRF [21] and CCNeRF [47] for reference.

<table>
<thead>
<tr>
<th>Method</th>
<th>NeRF-Synthetic</th>
<th>Tanks&amp;Temples</th>
<th>ScanNet</th>
<th>DTU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSNR↑</td>
<td>SSIM↑</td>
<td>LPIPS↓</td>
<td>PSNR↑</td>
</tr>
<tr>
<td>NeRF [21]</td>
<td>31.01</td>
<td>0.947</td>
<td>0.081</td>
<td>25.78</td>
</tr>
<tr>
<td>CCNeRF-CP [47]</td>
<td>30.55</td>
<td>0.935</td>
<td>0.076</td>
<td>27.01</td>
</tr>
<tr>
<td>CCNeRF-HY-S [47]</td>
<td>31.22</td>
<td>0.947</td>
<td>0.074</td>
<td>27.53</td>
</tr>
<tr>
<td>NPBG [1]</td>
<td>28.10</td>
<td>0.923</td>
<td>0.077</td>
<td>25.97</td>
</tr>
<tr>
<td>NPBG++ [33]</td>
<td>28.12</td>
<td>0.928</td>
<td>0.076</td>
<td>26.04</td>
</tr>
<tr>
<td>Huang et al. [13]</td>
<td>28.96</td>
<td>0.932</td>
<td>0.061</td>
<td>26.35</td>
</tr>
<tr>
<td>Ours</td>
<td>31.24</td>
<td>0.950</td>
<td>0.049</td>
<td>27.79</td>
</tr>
</tbody>
</table>

Figure 5. Qualitative evaluations on NeRF-Synthetic [21] dataset. Our method renders sharper detailed textures without artifacts.

shown in Fig. 4, we first rasterize the combined point clouds to determine which scene each pixel belongs to and generate a mask for each scene. Then we render each scene separately and multiply the masks with the rendered images. We combine the masked images to obtain the final result.

4. Experiments

4.1. Experimental Settings

Datasets and Compared Methods. Since our work focuses on surface point-based rendering methods, which enable real-time and editable rendering, we mainly compare with the following methods: 1) NPBG [1]: A famous surface point-based rendering method which enables real-time rendering and editing; 2) NPBG++ [33]: The improved version of NPBG; and 3) Huang et al. [13]: The state-of-the-art surface point-based rendering pipeline. We first compare our performance with the above methods on NeRF-Synthetic [21], Tanks and Temples [16], ScanNet [5] and DTU [14] datasets. To further demonstrate the superiority of our method, we conduct a comprehensive comparison with volume rendering-based methods, including NeRF [21] and its variants, on NeRF-Synthetic [21] and Tanks and Temples [16] datasets. Please see supplementary material for more details.

Implementation Details. There are two ways of rasterization: PyTorch3D [34] for headless server and OpenGL [40] for real-time rendering. Since our training is performed on a headless server, our main results are based on PyTorch3D rasterization. The widths of AFNet and hypernetwork are both 256, and camera ray directions are concated to the third layer output feature of AFNet. Moreover, we modify the positional encoding of NeRF [21] to make it more suitable for modulation, which is described in detail in supplementary material. The architecture of our U-Net is consistent with NPBG [1], i.e., gated blocks are used, consisting of gated convolution and instance normalization. The U-Net down-samples the feature map for four times and the dimensions of each layer are 16, 32, 64, 128, 256. We train the AFNet and U-Net with L2 loss and perceptual loss [62] on a single NVIDIA GeForce RTX 3090 GPU. For preprocessing of point cloud optimization, we use only L2 loss and a 5-layers MLP with 256 width, where ray directions are concated to the third layer output feature. We use Adam optimizer for training, with a batch size of 2. The ini-
<table>
<thead>
<tr>
<th>Method</th>
<th>Size(MB)</th>
<th>FPS</th>
<th>Object</th>
<th>Scene</th>
<th>NeRF-Synthetic</th>
<th>Tanks&amp;Temples</th>
</tr>
</thead>
<tbody>
<tr>
<td>NeRF [21]</td>
<td>5.0</td>
<td>0.023</td>
<td>✗</td>
<td>✗</td>
<td>31.01</td>
<td>25.78</td>
</tr>
<tr>
<td>NSVF [19]</td>
<td>16.0</td>
<td>0.815</td>
<td>✗</td>
<td>✓</td>
<td>31.75</td>
<td>28.40</td>
</tr>
<tr>
<td>Object-NeRF [55]</td>
<td>121.2</td>
<td>0.1</td>
<td>✓</td>
<td>✗</td>
<td>31.19</td>
<td>25.96</td>
</tr>
<tr>
<td>PlenOctrees [58]</td>
<td>1976.3</td>
<td>168</td>
<td>✗</td>
<td>✓</td>
<td>31.71</td>
<td>27.99</td>
</tr>
<tr>
<td>Point-NeRF [52]</td>
<td>20.0</td>
<td>0.125</td>
<td>✗</td>
<td>✓</td>
<td>33.00</td>
<td>29.61</td>
</tr>
<tr>
<td>Plenoxels [8]</td>
<td>778.1</td>
<td>15</td>
<td>✗</td>
<td>✓</td>
<td>31.71</td>
<td>27.43</td>
</tr>
<tr>
<td>TensoRF [3]</td>
<td>71.8</td>
<td>1.15</td>
<td>✗</td>
<td>✓</td>
<td>33.14</td>
<td>28.56</td>
</tr>
<tr>
<td>Instant-NGP [22]</td>
<td>63.3</td>
<td>60</td>
<td>✗</td>
<td>✓</td>
<td>33.18</td>
<td>28.78</td>
</tr>
<tr>
<td>CCNeRF-CP [47]</td>
<td>4.4</td>
<td>1.05</td>
<td>✗</td>
<td>✓</td>
<td>30.55</td>
<td>27.01</td>
</tr>
<tr>
<td>CCNeRF-HY-S [47]</td>
<td>68.9</td>
<td>1.05</td>
<td>✗</td>
<td>✓</td>
<td>31.22</td>
<td>27.53</td>
</tr>
<tr>
<td>NPBG [1]</td>
<td>44.2</td>
<td>33.4</td>
<td>✓</td>
<td>✓</td>
<td>28.10</td>
<td>25.97</td>
</tr>
<tr>
<td>NPBG++ [33]</td>
<td>28.6</td>
<td>35.4</td>
<td>✓</td>
<td>✓</td>
<td>28.12</td>
<td>26.04</td>
</tr>
<tr>
<td>Huang et al. [13]</td>
<td>18.5</td>
<td>39.1</td>
<td>✓</td>
<td>✓</td>
<td>28.96</td>
<td>26.35</td>
</tr>
<tr>
<td>Ours</td>
<td>11.8</td>
<td>39.3</td>
<td>✓</td>
<td>✓</td>
<td>31.24</td>
<td>27.79</td>
</tr>
</tbody>
</table>

Table 2. Comprehensive evaluation. We report the model size (including point cloud size for point-based methods), FPS on NeRF-Synthetic [21] dataset with 800×800 resolution, editing ability of object-level transform and scene composition, and performance on NeRF-Synthetic [21] and Tanks and temples [16] datasets. The results demonstrate the excellent comprehensiveness of our method.

4.2. Results of View Synthesis

We present quantitative comparison results on three performance metrics (PSNR, SSIM and LPIPS) in Tab. 1. As can be seen, we outperform the previous point-based methods with a significant margin on all datasets. Our method outperforms NeRF [21] and is comparable to the high-profile version of CCNeRF [47]. Figs. 5 and 6 present a qualitative comparison on NeRF-Synthetic [21] and Tanks and Temples [16] dataset. As can be seen, our method produces sharper details without introducing artifacts, especially for highly textured regions.

Comprehensive Analysis. To further illustrate the superiority of our method, we expand the quantitative experiments with more comparison works in Tab. 2, and comprehensively compare them from four aspects: model size, rendering speed, editing ability, and performance. It can be seen that almost all the follow-up works of NeRF are difficult to take into account the four aspects at the same time. PlenOctrees [58] achieves the state-of-the-art FPS, but its application scenarios are limited due to its huge model size. Instant-NGP [22] has high performance and real-time rendering speed, but lacks editing abilities. Object-NeRF [55] and CCNeRF [47] can achieve object-level editing and scene composition respectively, but cannot achieve real-time rendering. Our method achieves high performance and flexible editing with a small model size while rendering in real-time.

Ablation Study. The frequencies predicted by our hypernetwork are highly correlated with the true frequencies reflected by local variance, as shown in Fig. 7. Note that there is a reasonable gap between our predicted 3D texture frequencies and 2D image frequencies. For example, in Ficus scene, the occlusion between leaves and background produces a high frequency region in 2D image, but the 3D texture of leaves is weak (just green). Moreover, we claim that the frequencies predicted by hypernetwork are adaptive, which can modulate the radiance signal better than actual local variance. To illustrate this, we inject the local variance of 2D images into AF layers without hypernetwork learning. The evaluation results in Tab. 3 show that the actual local variance is difficult to modulate the radiance signal. Also, point cloud geometry optimization is critical for some poorly reconstructed scenes. In Tab. 3, we show the quantitative results on PSNR and point scale. The preprocessing module improves performance while significantly reducing the size of the point cloud. In Fig. 8, we show the point clouds, depth maps and rendered images of Lego and Hotdog scenes. As can be seen, the noises near the crawler.
<table>
<thead>
<tr>
<th>PSNR</th>
<th>loc. var.</th>
<th>30.25</th>
<th>23.48</th>
<th>29.90</th>
<th>30.33</th>
<th>29.49</th>
<th>26.55</th>
<th>32.22</th>
<th>24.47</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o. pre.</td>
<td>32.46</td>
<td>25.15</td>
<td>30.95</td>
<td>33.10</td>
<td>27.51</td>
<td>28.96</td>
<td>33.35</td>
<td>26.78</td>
<td></td>
</tr>
<tr>
<td>ours</td>
<td>33.06</td>
<td>25.95</td>
<td>32.19</td>
<td>35.82</td>
<td>31.56</td>
<td>29.69</td>
<td>33.64</td>
<td>27.97</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Ablation analysis of adaptive frequency and point cloud geometry optimization. We inject the local variance extracted from the ground truth images as frequency information without using hypernetwork to predict frequency, denoted as loc. var. We also show the PSNR of training without preprocessing of point cloud geometry optimization (w/o. pre.) and the number of points before and after preprocessing.

![Point Cloud Depth Rendering](image)

Figure 8. We show the point clouds, depth maps and rendered images with and without preprocessing of point cloud geometry optimization (w/o. pre. and w/o. pre.). The noise near the crawler and the shovel in Lego scene causes artifacts and the missing part of Hotdog causes some blanks in the rendering result, while our preprocessing module optimizes the point cloud geometry and remove those artifacts.

![Duplicate & Move Rotation Scaling](image)

Figure 9. Object-level editing on ToyDesk [55] dataset. We can scale, duplicate, translate and rotate the objects selected by users.

![Original After Editing](image)

Figure 10. Scene composition on NeRF-Synthetic [21] and Tanks and Temples [16] datasets. These scenes only need to be trained separately. And we can edit objects while combining the scenes.

In Fig. 10, our editing algorithm handles complex occlusion relationships. Moreover, object-level editing and scene composition are performed at the same time in the last two examples in Fig. 10, i.e., the balls in Materials scene are moved and composed with other scenes. More editing results can be found in supplementary material.

5. Conclusion

We develop a novel point cloud rendering pipeline which enables high fidelity reconstruction, real-time rendering and user-friendly editing. For the novel view synthesis task, experiments on major benchmarks demonstrate the proposed method outperforms existing point cloud rendering methods and achieves the state-of-the-art.

Limitations and Future Work. Our current editing method cannot relight the edited objects (i.e., the transformed objects retain their original appearance), handle non-rigid object and appearance editing. Moreover, the rendering speed of the edited scenes will be reduced. We will improve these shortcomings in the future.

Acknowledgment. This work was supported by National Science Foundation of China (U20B2072, 61976137). This work was also partly supported by SJTU Medical Engineering Cross Research Grant YG2021ZD18.
References


[52] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf:


