Introduction

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 devices. In this work, we focus on boosting the image quality of point clouds rendering with a compact model design. We first analyze the adoption 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 cloud 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 are publicly available in https://github.com/seanywang0408/RadianceMapping.

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 devices. In this work, we focus on boosting the image quality of point clouds rendering with a compact model design. We first analyze the adoption 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 cloud 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 are publicly available in https://github.com/seanywang0408/RadianceMapping.

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 (Rücker, 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...
Spatial Frequency Collapse

Neighbor Point Disturbance

Our Rectification

Ground Truth

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.

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 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 cloud 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. Insaftudinov 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 multiscale 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 (Maturana and Scherer; Yang et al.) is used to transform the volume into a set of multi-plane images and their blending weights, which form the final im-
age 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 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 (Aliev 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 grow 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 \( \sigma \) using a multi-layer perceptron (MLP) \( F_\Theta \), parameterized by \( \Theta \):

\[
c, \sigma = F_\Theta(x, d)
\]

(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 \( C(\mathbf{r}) \):

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

(2)

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

(3)
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.

\( T(t) \) denotes the accumulated transmittance along the ray. Suppose the camera ray first intersects with an object surface at point \( p_o = o + t_o d \). For those points that lie in front of \( p_o \), which satisfy \( \{ p = o + td | t < t_o \} \), 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 behind \( p_o \), which satisfy \( \{ p = o + td | t > t_o \} \). One case is that the density of \( p_o \) 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 \( p_o \) 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 \( c(r(t_o), d) \) would be fitted to the pixel color \( C(r) \) 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 shell behind is invisible. Without 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 \( C(r) \) becomes a single evaluation on \( p_o \) according to Eqn. 2. Thus we simply remove the density prediction in NeRF representation (Eqn. 1). Moreover, we output the latent feature \( L \) that encodes the color to a higher dimension instead of RGB as prior works do. Then Eqn. 1 is reformulated as

\[
L = F_0(x, d) \tag{4}
\]

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 \( p_{pcd} \) in a point cloud actually lie around the object surface \( p_{obj} \) with a spatial noise \( \epsilon \). We denote its coordinate with \( x_{pcd} = x_{obj} + \epsilon \). If we feed the raw point clouds into Eqn. 4, we obtain \( L_{pcd} = F_0(x_{obj} + \epsilon, d) \), 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:

\[
x_{query} = o + \frac{z_{pcd}}{\cos \theta} \times d \tag{5}
\]
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 \( \tau \) 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 \( \tau \). 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 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 supervised 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.). 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 \( \tau \) as 5e-3 for NeRF-Synthetic, 1.5e-2 for ScanNet, and 3e-3 for DTU.

<table>
<thead>
<tr>
<th>Method</th>
<th>PSNR↑</th>
<th>SSIM↑</th>
<th>LPIPS↓</th>
<th>PSNR↑</th>
<th>SSIM↑</th>
<th>LPIPS↓</th>
<th>PSNR↑</th>
<th>SSIM↑</th>
<th>LPIPS↓</th>
<th>DTU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>31.74</td>
<td>0.968</td>
<td>0.029</td>
<td>25.88</td>
<td>0.794</td>
<td>0.414</td>
<td>30.81</td>
<td>0.904</td>
<td>0.128</td>
<td></td>
</tr>
<tr>
<td>NeRF</td>
<td>33.07</td>
<td>0.989</td>
<td>0.064</td>
<td>25.63</td>
<td>0.765</td>
<td>0.436</td>
<td>31.03</td>
<td>0.909</td>
<td>0.127</td>
<td></td>
</tr>
<tr>
<td>NPBG++</td>
<td>30.52</td>
<td>0.960</td>
<td>0.042</td>
<td>25.24</td>
<td>0.755</td>
<td>0.444</td>
<td>30.25</td>
<td>0.894</td>
<td>0.131</td>
<td></td>
</tr>
<tr>
<td>NPBG</td>
<td>30.47</td>
<td>0.957</td>
<td>0.046</td>
<td>25.08</td>
<td>0.737</td>
<td>0.459</td>
<td>30.24</td>
<td>0.895</td>
<td>0.130</td>
<td></td>
</tr>
<tr>
<td>Chair</td>
<td>28.81</td>
<td>25.37</td>
<td></td>
<td>32.03</td>
<td>27.72</td>
<td>25.24</td>
<td>30.25</td>
<td>27.72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drums</td>
<td>28.72</td>
<td>23.60</td>
<td>28.11</td>
<td>32.22</td>
<td>27.84</td>
<td>23.60</td>
<td>32.16</td>
<td>23.60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ficus</td>
<td>31.13</td>
<td>24.51</td>
<td>29.09</td>
<td>33.20</td>
<td>27.84</td>
<td>23.57</td>
<td>32.03</td>
<td>23.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hotdog</td>
<td>31.13</td>
<td>24.51</td>
<td>29.09</td>
<td>33.20</td>
<td>27.84</td>
<td>23.57</td>
<td>32.03</td>
<td>23.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lego</td>
<td>31.13</td>
<td>24.51</td>
<td>29.09</td>
<td>33.20</td>
<td>27.84</td>
<td>23.57</td>
<td>32.03</td>
<td>23.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Materials</td>
<td>31.13</td>
<td>24.51</td>
<td>29.09</td>
<td>33.20</td>
<td>27.84</td>
<td>23.57</td>
<td>32.03</td>
<td>23.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mic</td>
<td>31.13</td>
<td>24.51</td>
<td>29.09</td>
<td>33.20</td>
<td>27.84</td>
<td>23.57</td>
<td>32.03</td>
<td>23.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ship</td>
<td>31.13</td>
<td>24.51</td>
<td>29.09</td>
<td>33.20</td>
<td>27.84</td>
<td>23.57</td>
<td>32.03</td>
<td>23.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>31.13</td>
<td>24.51</td>
<td>29.09</td>
<td>33.20</td>
<td>27.84</td>
<td>23.57</td>
<td>32.03</td>
<td>23.57</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Results on three datasets. For NeRF-Synthetic, the results are averaged on Ficus, HotDog and Mic, following NPBG++.
with NPBG, NPBG++, and NeRF, using PSNR, SSIM and LPIPS as evaluation metrics.

**Results** We show the quantitative results in Table 1. For NeRF-Synthetic, we report the results on Ficus, HotDog and Mic in Table 1, as done in NPBG++ (Rakhimov et al.). Besides, we additionally conduct experiments on all eight scenes and show their PSNR in Table 2. Our method outperforms prior arts with notable margins on each dataset in terms of PSNR, SSIM and LPIPS. Compared to NeRF, our method achieves comparable results on NeRF-Synthetic and DTU, and even better results on ScanNet. It demonstrates the effectiveness of our methods under various scenes. Table 2 shows per-scene PSNR in NeRF-Synthetic. 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. 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 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.

**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 (Aliev et al.) uses the same U-Net architecture as ours (7.3M), 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 61M. Hence the total size of NPBG is 68.3M. NPBG++ (Rakhimov et al.) uses a convolution neural network in the size of 17.5M as the image feature extractor to obtain the neural descriptor. Besides, NPBG++ uses a smaller U-Net in refinement stage, whose size is 1.2M. It results in a total model size of 18.7M. 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 formulation like NeRF with point clouds as input. Hence it also suffers from heavy sampling and high computation cost, with an FPS of 0.125.
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 harm the rendering quality. Our method could notably alleviate the issue.

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 clouds density. We compare our method against NPBG++ 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.

Table 3: (a) We verify our method under various density of point clouds. 1×, 10× and 50× 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↑, SSIM↑ and LPIPS↓ respectively.

<table>
<thead>
<tr>
<th>Method</th>
<th>Model Size ↓</th>
<th>FPS ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPBG</td>
<td>(61+7.3)M</td>
<td>17</td>
</tr>
<tr>
<td>NPBG++</td>
<td>(17.5+1.2)M</td>
<td>18</td>
</tr>
<tr>
<td>Ours</td>
<td>(0.75+7.3)M</td>
<td>20</td>
</tr>
<tr>
<td>NeRF</td>
<td>4.8M</td>
<td>0.015</td>
</tr>
<tr>
<td>Point-NeRF</td>
<td>18.5M</td>
<td>0.125</td>
</tr>
<tr>
<td>PlenOctrees</td>
<td>440M</td>
<td>110</td>
</tr>
</tbody>
</table>

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.
Acknowledgements
This work was supported by National Science Foundation of China (U20B2072, 61976137). This work was also partially supported by Grant YG2021ZD18 from Shanghai Jiao Tong University Medical Engineering Cross Research. This work was also partially supported by the Shanghai Committee of Science and Technology, China (Grant No. 20DZ1100800).

References


