Frequency-Modulated Point Cloud Rendering with Easy Editing Supplementary Material

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

1. Implementation Details

1.1. End-to-end Rendering Pipeline

Rasterization Radius. During rasterization, each point is expanded to a disk to compensate for point cloud sparsity. We propose the following two heuristic rules to select disk radius: 1) the disk should cover the gaps between the points, otherwise the points from the occluded surfaces and the background can be seen through the front surface (so-called bleeding problem); 2) the radius should be as small as possible, otherwise it will lead to inaccurate depth estimation and the edges of objects will expand. In practice, the selection of the radius depends on the density of point cloud, and we can adjust the radius by observing the depth map obtained by rasterization. We also conduct an ablation study on radius, and it can be seen from Tab. 1 that the effect of the radius is not significant.

radius	3e-3	4e-3	5e-3	6e-3	7e-3
Hotdog	34.62	35.25	35.82	35.37	34.95
Mic	33.64	33.49	33.40	33.14	32.73

Table 1. PSNR of Hotdog and Mic scenes under different rasterization radius. The effect of radius is not significant.

Radiance Mapping and Refinement. We follow the position encoding form of NeRF [7], but we narrow the encoding interval to provide more fine-grained basis function support for frequency modulation, shown as follows:

$$\begin{aligned}\gamma(\mathbf{p}) = \big(&\sin\left(2^{0}\pi\mathbf{p}\right), \cos\left(2^{0}\pi\mathbf{p}\right), \\ &\sin\left(2^{0.5}\pi\mathbf{p}\right), \cos\left(2^{0.5}\pi\mathbf{p}\right), \\ &\cdots, \\ &\sin\left(2^{L-1}\pi\mathbf{p}\right), \cos\left(2^{L-1}\pi\mathbf{p}\right)\big).\end{aligned} \tag{1}$$

We set L as 10 and 2 for spatial coordinates and view directions, respectively. Our AFNet has five layers, the input feature dimension is 120, and hidden dimensions are 256, 256, 256 and 128. The output dimension is 11 (including 3-channel raw image and 8-channel feature map). Gated convolution block used in U-Net is shown in Fig. 1. For upsampling and downsampling in U-Net, we use nearest neighbor interpolation and average pooling respectively.

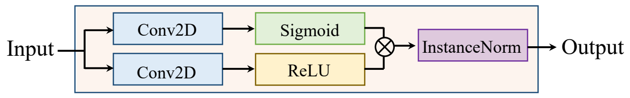

Figure 1. Gated convolution block.

Training. During training stage, we conduct data augmentation by random scaling and cropping, and we found that random scaling is critical for the training of U-Net. The loss function is shown as follows:

$$\mathcal{L} = \mathcal{L}_{\text{RGB}} + \lambda\mathcal{L}_{\text{perceptual}}, \tag{2}$$

and we set λ as 5e-3 in our experiments. Thanks to the geometric prior, our model requires only ten minutes of training to achieve realistic rendering, but takes about 10 hours to almost converge, as shown in Fig. 2. In order to achieve complete convergence, more than ten hours of fine-tuning is required.

1.2. Point Cloud Geometry Optimization

Denoising We set $k = 16$, *i.e.*, keep 16 points in each pixel buffer. Each point $\mathbf{p}_i$ in point cloud is assigned an opacity parameter α_i, and the color $\mathbf{c}_i$ is predicted by MLP. The ray color is obtained as the following discrete volume rendering equation:

$$\hat{C}(\mathbf{r}) = \sum_{i=0}^{k-1} T_i\alpha_i\mathbf{c}_i, \quad \text{where } T_i = \prod_{j=0}^{i-1}(1-\alpha_j). \tag{3}$$

We optimize the L2 distance of predicted image and ground truth with sparse regularization as follows:

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_{\text{RGB}} + \lambda_{\text{sparse}}\mathcal{L}_{\text{sparse}}, \\ \mathcal{L}_{\text{RGB}} &= \left|\left|\hat{C} - C_{gt}\right|\right|_2^2, \\ \mathcal{L}_{\text{sparse}} &= \frac{1}{N}\sum_{i=1}^{N}\left[\log\left(\alpha_i\right) + \log\left(1-\alpha_i\right)\right].\end{aligned} \tag{4}$$

*Equal contribution.
†Corresponding author: Bingbing Ni.

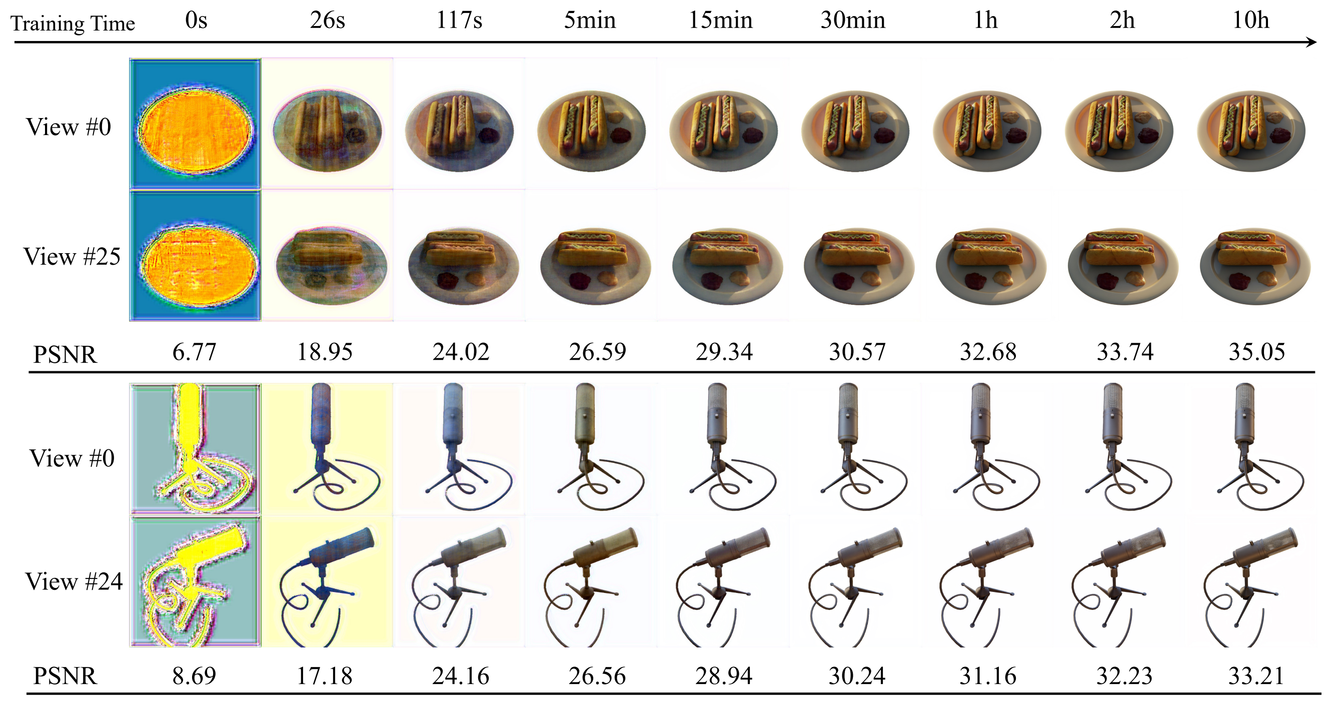

Figure 2. Rendering results during training. One Epoch consumes about 12.8s on NeRF-Synthetic. Our model requires only ten minutes of training to achieve realistic rendering, but takes about 10 hours to almost converge. In order to achieve complete convergence, more than ten hours of fine-tuning is required.

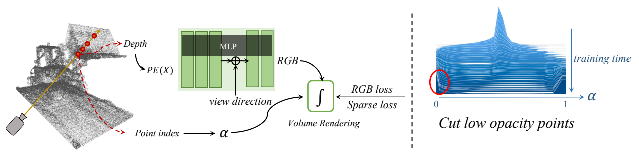

Figure 3. Denoising step of preprocessing.

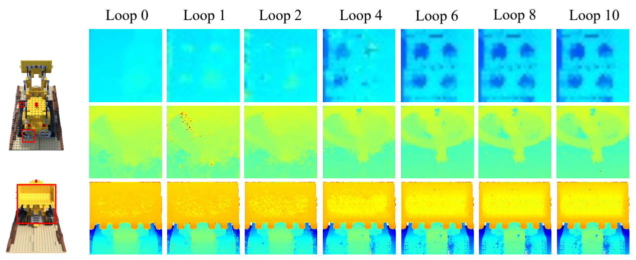

Figure 4. Depth map during geometry optimization. We remove outliers near the track and shovel.

We set $\lambda_{\text{sparse}} = 5\text{e-}4$ in our experiments, and the learning rate of MLP and opacity parameters α are set as 5e-4 and 0.01, respectively. And for scenes without background, we also add transparency loss, *i.e.* L2 distance of predicted transparency and ground truth transparency. When the training converges, we remove those low-opacity points, as shwon in Fig. 3.

In experiments, we perform 4-20 loops (depends on the raw point cloud quality) of point cloud denoising and completion step alternately. For every denosing step, we train for 300 Epochs. Fig. 4 shows the depth map computed by volume rendering equation (replace colors as depth values of sample points) during optimization. Fig. 5 shows the point cloud after completion. Although the completion step introduces additional noise, it would be removed in the denoising stage of the next iteration.

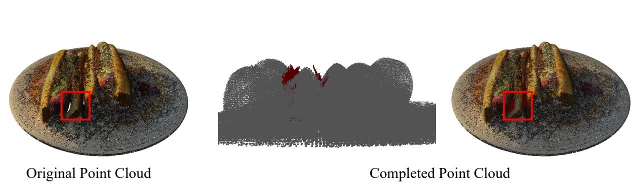

Figure 5. Original and completed point cloud. When an empty buffer is detected, we add a set of points with opacity parameters along the pixel ray in the point clouds, shown as red points. Although the completion step introduces additional noise, it would be removed in the denoising stage of the next iteration.

1.3. Editing Details

In practice, the part of point cloud that needs to be edited (*i.e.*, the object) is stored in the form of a mask. We use PCL library [9] to implement a simple interactive selection function, as shown in Fig. 7. In fact, users can obtain this mask by selecting points using any interactive software.

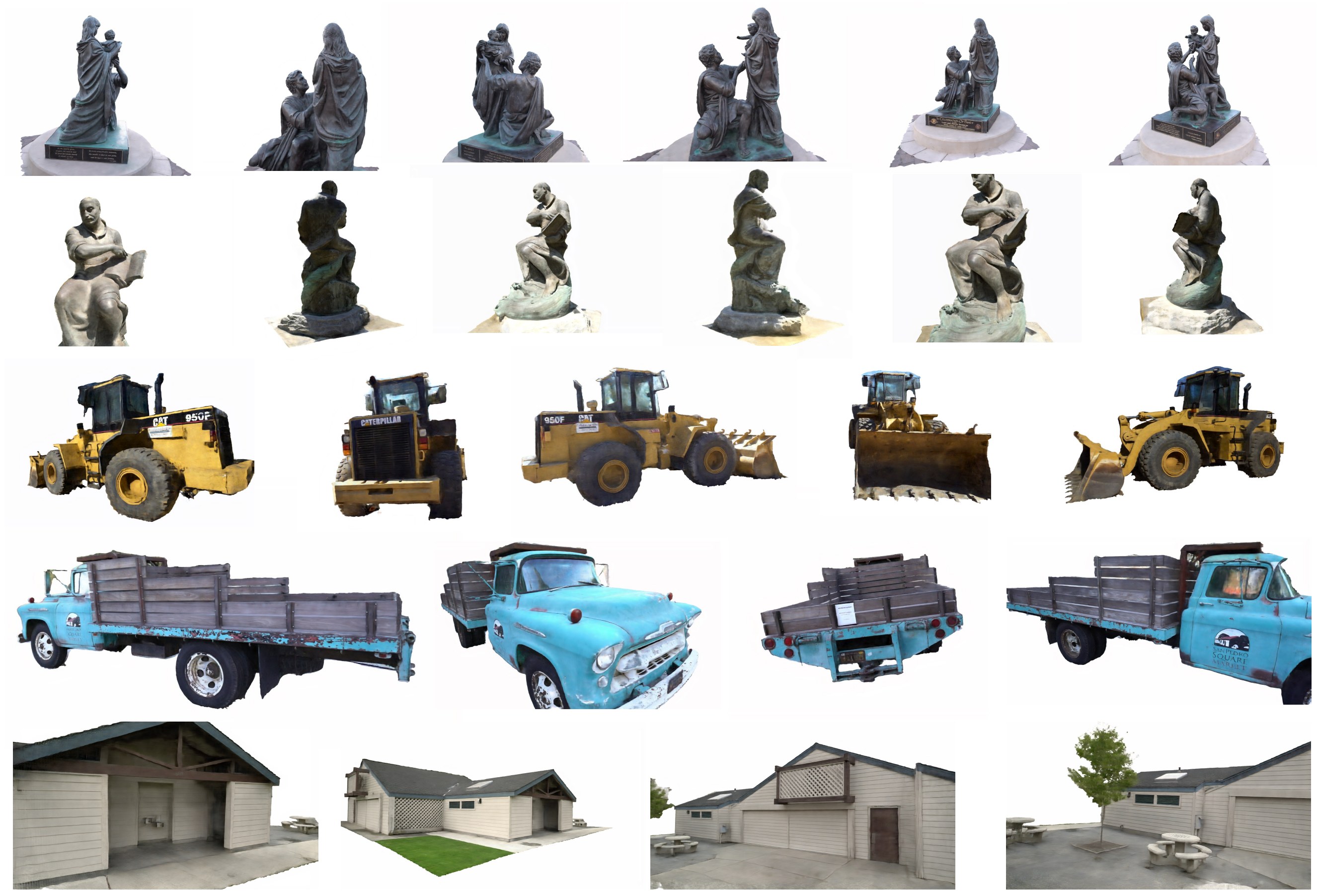

Figure 6. Some rendering results on test views of Tanks and Temples dataset.

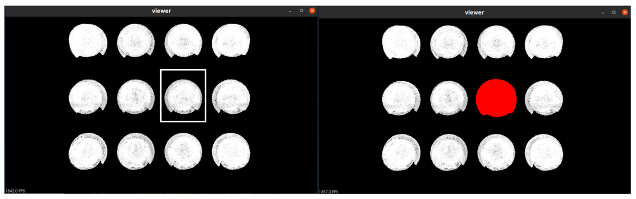

Figure 7. Point cloud selection interface.

For scene composition, since the points on the edge of the object will expand, the mask of the scene will also expand outward. We design a simple strategy to avoid artifacts at edges by shrinking the masks determined by the index buffers, please refer to the code for specific implementation.

2. Experimental Details

2.1. Datasets

- NeRF-Synthetic [7] is a high quality synthetic dataset containing pathtraced images of 8 objects. For each object, there are 100 frames for training and 200 frames for testing. The initial point clouds we used are generated by MVSNet [12]. We set the training size as 800×800 and the scaling factor is $[0.5, 1.5]$ of the side length. At test time, we render with the original resolution 800×800.

- We use a subset of Tanks and Temples [5] dataset, which is from NSVF [6], containing five scenes of real objects. We also use the foreground masks provided by NSVF. Each scene contains 152-384 images of size 1920×1080. Due to the size limitation of U-Net, we resize the test resolution to 1920×1056, and training size is 640×640. The initial point clouds we used are provided by MVSNet [12].

- ScanNet [2] is a RGBD dataset of indoor scenes. We evaluate on $scene0000\_00$, $scene0043\_00$, and $scene0045\_00$, as done in NPBG++ [8] and follow their training-test split. Specifically, if there are more than 2000 frames in the scene, we select every 20-th image such that training views would not be too sparse. Otherwise, we take 100 frames with an equal interval

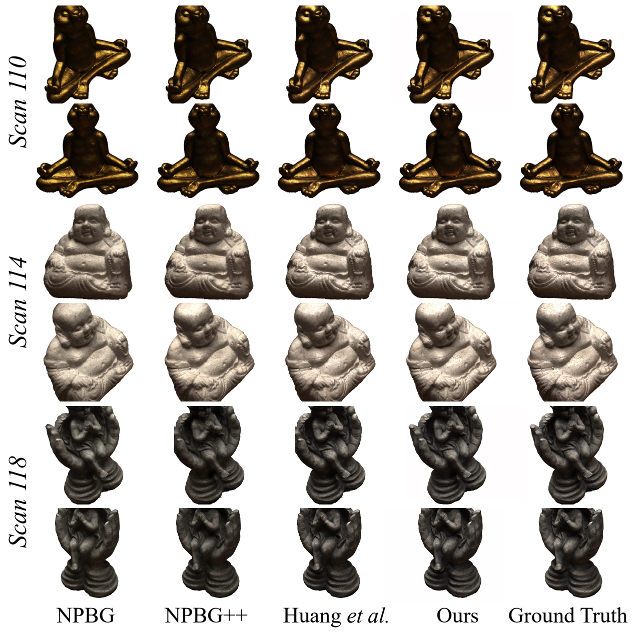

Figure 8. Qualitative comparison on DTU dataset.

in the image stream. Then, we select 10 frames at a fixed interval for testing and the rest for training. We set the training size as 720×720 and the test resolution is 960×1200. The initial point clouds are obtained by the provided depth maps.

- DTU [4] is a multi-view stereo dataset with a resolution of 1200×1600. We evaluate on $scan110$, $scan114$ and $scan118$, and use the same point clouds and training-test split as NPBG++ [8]. We mask out the background in training using the binary segmentation masks provided by IDR [13]. The training patch size is set as 800×800 and the test resolution 1200×1600.
- For ToyDesk [11] dataset, we only perform training and editing without evaluation.

2.2. Compared Methods

- NPBG [1]: A famous point-based rendering method, which uses point-wise features to encode the appearance of each surface point and an U-Net for decoding.
- NPBG++ [8]: The improved version of NPBG, which predicts the descriptors with a feature extractor and makes the neural descriptors view-dependent.
- Huang *et al.* [3]: The state-of-the-art point-based neural rendering, which combines explicit point clouds and implicit radiance mapping. However, its performance is still lower than that of NeRF, due to the weak frequency expressiveness and lack of geometric optimization.

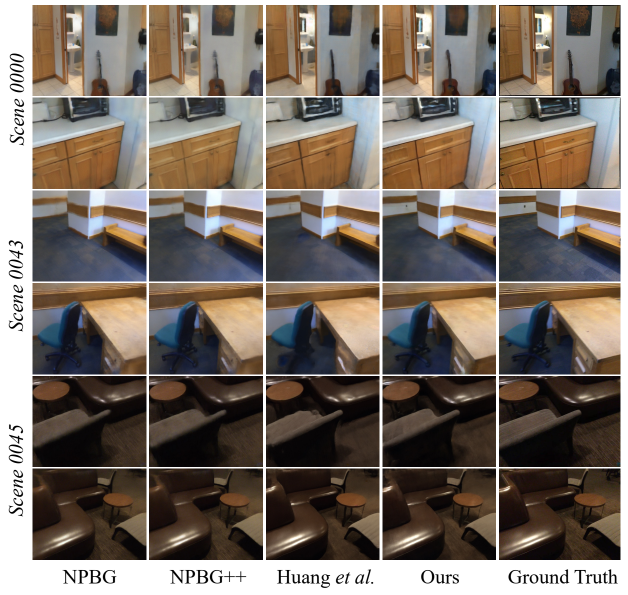

Figure 9. Qualitative comparison on ScanNet dataset.

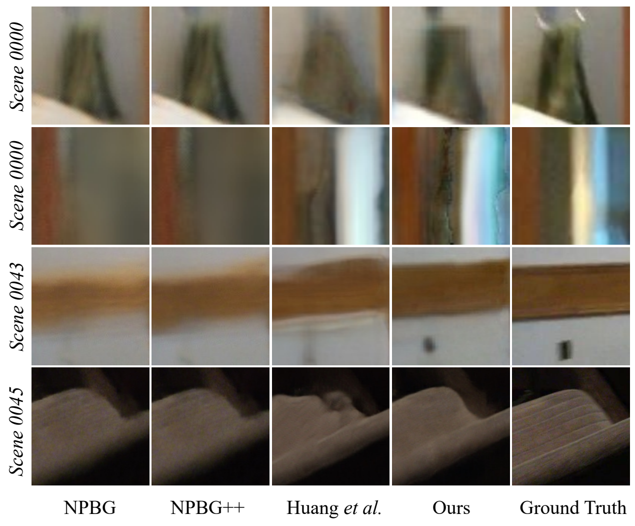

Figure 10. Qualitative comparison in detail on ScanNet dataset. Our rendering results are closer to ground truth.

- CCNeRF [10]: The latest editable variant of NeRF [7], which represents a 3D scene as a compressible 3D Tensor. Due to the massive sampling and calculation of tensor decomposition, the rendering speed is only 1.05 FPS.

2.3. Detailed Results

We present some rendering results on Tanks and Temples dataset in Fig. 6, and qualitative comparisons on DTU

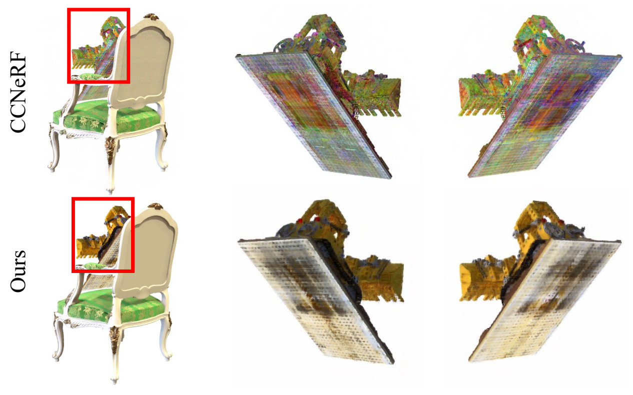

Figure 11. Extreme views of Lego scene that may be encountered in editing. We render reasonable results while CCNeRF renders distorted colors.

and ScanNet datasets in Figs. 8 and 9, respectively. A comparison of details on ScanNet dataset is shwon in Fig. 10. Although the rendering results look similar overall as seen from Figs. 8 and 9, we are better at some details, as shwon in Fig. 10. Due to the blurring of some training images in ScanNet dataset, some rendering results of novel views are also blurred. For the compared methods, we all use the same experimental configuration, which is fair. We present quantitative evaluation for each scene of each dataset in Tabs. 2, 3 and 4. More editing results are shown in Figs. 11, 12 and 13.

References

[1] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. In *European Conference on Computer Vision*, pages 696–712. Springer, 2020. 4, 8, 9, 10

[2] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5828–5839, 2017. 3

[3] Xiaoyang Huang, Yi Zhang, Bingbing Ni, Teng Li, Kai Chen, and Wenjun Zhang. Boosting point clouds rendering via radiance mapping. *arXiv preprint arXiv:2210.15107*, 2022. 4, 9, 10

[4] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 406–413, 2014. 4

[5] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4):1–13, 2017. 3

[6] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020. 3

[7] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 1, 3, 4, 8, 9, 10

[8] Ruslan Rakhimov, Andrei-Timotei Ardelean, Victor Lempitsky, and Evgeny Burnaev. Npbg++: Accelerating neural point-based graphics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15969–15979, 2022. 3, 4, 8, 9, 10

[9] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. IEEE. 2

[10] Jiaxiang Tang, Xiaokang Chen, Jingbo Wang, and Gang Zeng. Compressible-composable nerf via rank-residual decomposition. *arXiv preprint arXiv:2205.14870*, 2022. 4, 8, 9, 10

[11] Bangbang Yang, Yinda Zhang, Yinghao Xu, Yijin Li, Han Zhou, Hujun Bao, Guofeng Zhang, and Zhaopeng Cui. Learning object-compositional neural radiance field for editable scene rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13779–13788, 2021. 4, 7

[12] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. Mvsnet: Depth inference for unstructured multi-view stereo. In *Proceedings of the European conference on computer vision (ECCV)*, pages 767–783, 2018. 3

[13] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *Advances in Neural Information Processing Systems*, 33, 2020. 4

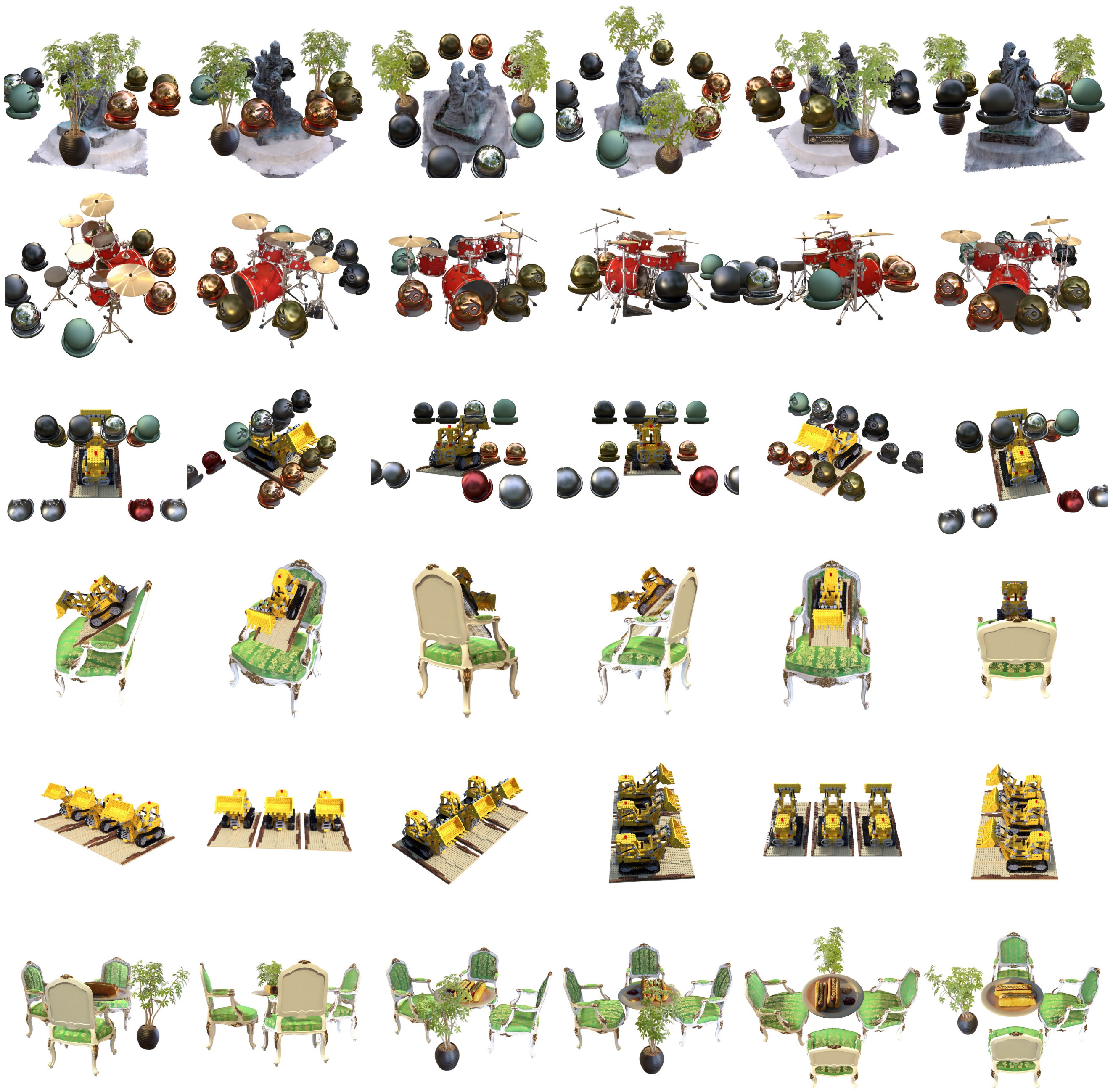

Figure 12. Editing results, including object-level editing and scene composition.

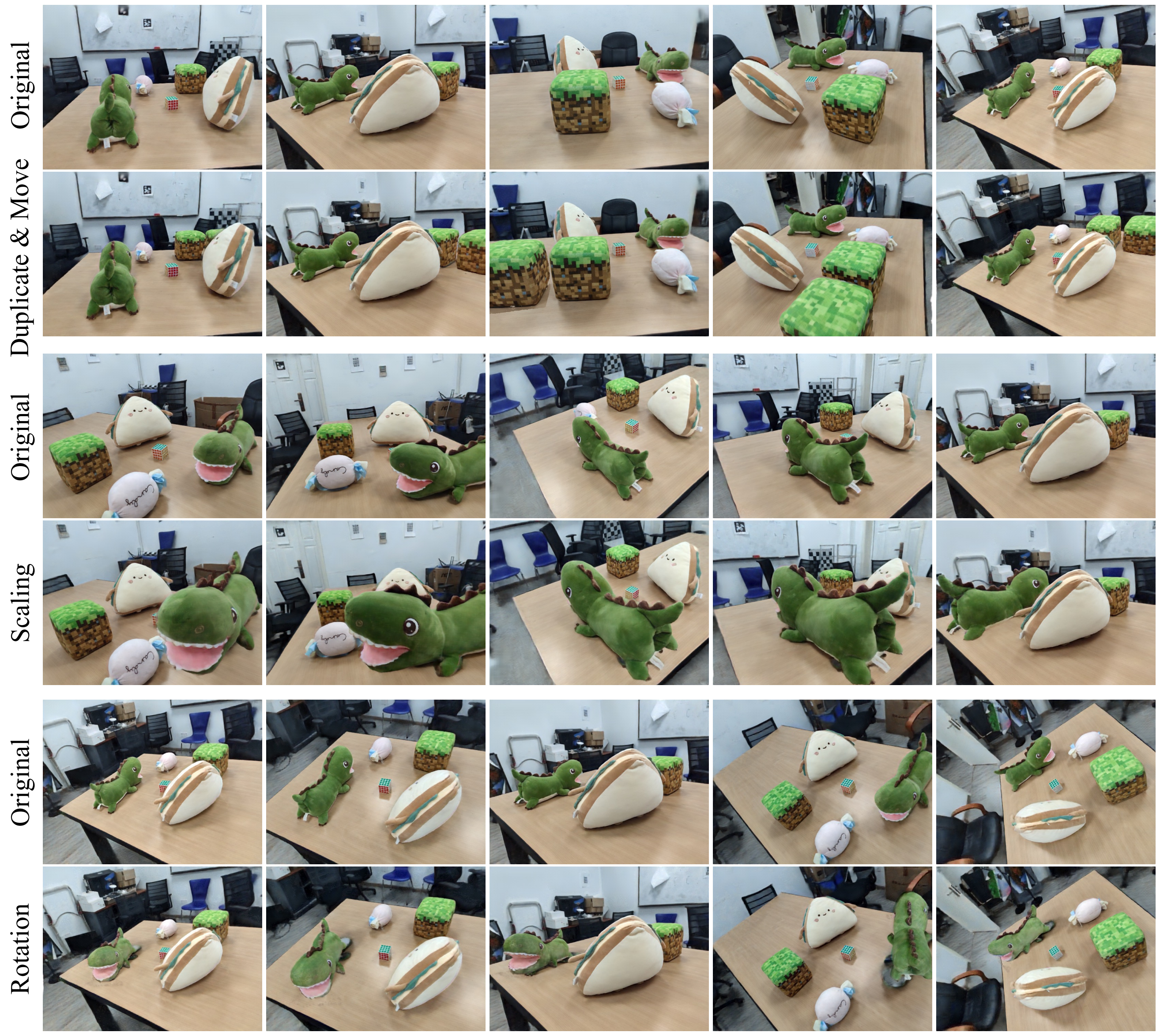

Figure 13. Object translation, rotation and scaling on ToyDesk [11] dataset. As can be seen from the last line of results, the rotation of the green object will expose the untrained local space, resulting in artifacts, which are also reflected in the results of Object-NeRF [11].

Method	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
				NeRF-Synthetic					
				PSNR↑					
NeRF [7]	33.00	25.01	30.13	**36.18**	32.54	29.62	32.91	**28.65**	31.01
CCNeRF-CP [10]	33.63	24.23	29.40	35.27	32.94	28.34	32.81	27.77	30.55
CCNeRF-HY-S [10]	**34.37**	24.76	30.04	36.04	**33.66**	28.96	33.53	28.38	31.22
NPBG [1]	28.81	23.57	28.23	32.03	27.72	27.24	31.16	26.04	28.10
NPBG++ [8]	28.72	23.60	28.11	32.22	27.84	27.12	31.23	26.11	28.12
Huang *et al.*	31.13	24.51	29.09	33.20	26.62	28.03	32.94	26.14	28.96
Ours	33.06	**25.95**	**32.19**	35.82	31.56	**29.69**	**33.64**	27.97	**31.24**
				SSIM↑					
NeRF [7]	0.967	0.925	0.964	0.974	0.961	0.949	0.980	**0.856**	0.947
CCNeRF-CP [10]	0.964	0.906	0.950	0.966	0.957	0.923	0.971	0.842	0.935
CCNeRF-HY-S [10]	**0.976**	0.918	0.962	**0.978**	**0.969**	0.935	0.983	0.853	0.947
NPBG [1]	0.954	0.902	0.942	0.960	0.919	0.922	0.970	0.812	0.923
NPBG++ [8]	0.961	0.910	0.947	0.961	0.923	0.925	0.972	0.822	0.928
Huang *et al.*	0.953	0.924	0.958	0.964	0.902	0.945	0.983	0.824	0.932
Ours	0.974	**0.938**	**0.971**	0.974	0.956	**0.955**	**0.986**	0.845	**0.950**
				LPIPS↓					
NeRF [7]	0.046	0.091	0.044	0.121	0.050	0.063	0.028	0.206	0.081
CCNeRF-CP [10]	0.037	0.111	0.055	0.057	0.037	0.082	0.031	0.196	0.076
CCNeRF-HY-S [10]	0.036	0.109	0.054	0.056	**0.036**	0.080	0.030	0.192	0.074
NPBG [1]	0.047	0.093	0.046	0.055	0.089	0.076	0.037	0.171	0.077
NPBG++ [8]	0.048	0.092	0.043	0.053	0.089	0.074	0.030	0.178	0.076
Huang *et al.*	0.040	0.068	0.035	0.038	0.085	0.050	**0.014**	0.159	0.061
Ours	**0.025**	**0.065**	**0.026**	**0.028**	0.045	**0.046**	0.015	**0.142**	**0.049**

Table 2. PSNR↑, SSIM↑ and LPIPS↓ on each scene of NeRF-Synthetic dataset.

			Tanks and Temples			
Method	Barn	Caterpillar	Family	Ignatius	Truck	Mean
			PSNR↑			
NeRF [7]	24.05	23.75	30.29	25.43	25.36	25.78
CCNeRF-CP [10]	25.84	24.02	32.13	27.24	25.84	27.01
CCNeRF-HY-S [10]	26.34	24.48	**32.75**	27.76	26.34	27.53
NPBG [1]	24.86	22.05	30.84	26.50	25.59	25.97
NPBG++ [8]	24.90	22.22	30.67	26.98	25.45	26.04
Huang *et al.* [3]	25.34	23.09	30.65	27.01	25.68	26.35
Ours	**27.01**	**24.67**	32.36	**28.83**	**26.56**	**27.79**
			SSIM↑			
NeRF [7]	0.750	0.860	0.932	0.920	0.860	0.864
CCNeRF-CP [10]	0.807	0.864	0.934	0.916	0.872	0.879
CCNeRF-HY-S [10]	0.827	**0.886**	**0.957**	0.939	0.894	0.901
NPBG [1]	0.841	0.848	0.940	0.928	0.887	0.889
NPBG++ [8]	0.842	0.863	0.943	0.933	0.878	0.892
Huang *et al.* [3]	0.841	0.855	0.946	0.936	0.886	0.893
Ours	**0.847**	0.876	0.953	**0.941**	**0.895**	**0.902**
			LPIPS↓			
NeRF [7]	0.395	0.196	0.098	0.111	0.192	0.198
CCNeRF-CP [10]	0.310	0.223	0.078	0.099	0.192	0.180
CCNeRF-HY-S [10]	0.304	0.219	0.076	0.097	0.188	0.177
NPBG [1]	0.219	0.182	0.072	0.076	0.138	0.137
NPBG++ [8]	0.197	0.181	0.075	0.068	0.131	0.130
Huang *et al.* [3]	0.195	0.179	0.069	0.077	0.131	0.130
Ours	**0.179**	**0.175**	**0.066**	**0.073**	**0.131**	**0.125**

Table 3. PSNR↑, SSIM↑ and LPIPS↓ on each scene of Tanks and Temples dataset.

	ScanNet				DTU			
Method	Scene0000	Scene0043	Scene0045	Mean	Scan110	Scan114	Scan118	Mean
				PSNR↑				
NeRF [7]	22.08	25.98	29.15	25.74	25.55	27.42	**27.78**	26.92
CCNeRF-CP [10]	21.14	25.89	26.91	24.65	25.88	27.89	26.61	26.79
CCNeRF-HY-S [10]	21.38	**26.22**	27.91	25.17	26.23	28.12	27.34	27.23
NPBG [1]	22.24	25.27	27.75	25.09	24.65	26.74	26.62	26.00
NPBG++ [8]	22.05	25.51	28.26	25.27	24.84	26.72	26.67	26.08
Huang *et al.* [3]	23.79	25.26	28.59	25.88	25.05	26.87	26.75	26.22
Ours	**24.35**	26.15	**29.48**	**26.66**	**26.45**	**28.31**	27.06	**27.27**
				SSIM↑				
NeRF [7]	0.729	0.869	0.743	0.780	0.909	0.894	0.924	0.909
CCNeRF-CP [10]	0.695	0.849	0.779	0.774	0.904	0.894	0.922	0.907
CCNeRF-HY-S [10]	0.701	0.854	0.788	0.781	0.909	**0.896**	**0.925**	**0.910**
NPBG [1]	0.695	0.830	0.686	0.737	0.907	0.873	0.904	0.895
NPBG++ [8]	0.742	0.859	0.716	0.772	0.910	0.865	0.909	0.895
Huang *et al.* [3]	0.748	0.844	0.789	0.794	**0.917**	0.863	0.919	0.900
Ours	**0.754**	**0.863**	**0.792**	**0.803**	0.915	0.892	0.916	0.908
				LPIPS↓				
NeRF [7]	0.588	0.466	0.558	0.537	0.194	0.217	0.182	0.198
CCNeRF-CP [10]	0.599	0.457	0.569	0.542	0.181	0.191	0.161	0.178
CCNeRF-HY-S [10]	0.594	0.456	0.566	0.539	0.171	0.188	0.154	0.171
NPBG [1]	0.474	0.421	0.482	0.459	0.124	0.143	0.123	0.130
NPBG++ [8]	0.457	0.410	0.477	0.448	0.125	0.148	0.121	0.131
Huang *et al.* [3]	0.440	0.389	0.415	0.415	0.124	0.154	**0.117**	0.132
Ours	**0.418**	**0.369**	**0.412**	**0.400**	**0.122**	**0.143**	0.122	**0.129**

Table 4. PSNR↑, SSIM↑ and LPIPS↓ on each scene of ScanNet and DTU datasets.