NeuralEditor: Editing Neural Radiance Fields via Manipulating Point Clouds Supplementary Material

Jun-Kun Chen[1†] Jipeng Lyu[2†] Yu-Xiong Wang[1]
[1]University of Illinois at Urbana-Champaign [2]Peking University [†]Equal Contribution
{junkun3, yxw}@illinois.edu lvjipeng@pku.edu.cn

This document contains additional descriptions (*e.g.*, implementation details, experimental setting details, *etc.*) and extra experiments (*e.g.*, ablation study, deformation on the Tanks and Temples dataset, *etc.*). The content of this document is summarized as below:

A Black/White Background 1

B Additional Ablation Study 2

C Non-Continuous Deformation Task 2

D Experiment on Tanks and Temples Dataset 2

E Intermediate Point Clouds for Scene Morphing 4

F. Other Metrics in Table 1 4

G High-Resolution Visualization Figures 4

H Implementation Details 4
H.1. K-D Voxels & Integration 4
H.2 Phong Reflection Color Modeling 4
H.3. Point Cloud Optimization 4
H.4. Training Settings 5
H.5. Matching Algorithm for Scene Morphing . . 5

I. Limitations 5
I.1 . Point Cloud-Guided NeRF 5
I.2 . Environment Modeling in Shape Deformation Task 6
I.3 . Evaluation for Scene Morphing Task 6

A. Black/White Background

In the main paper, we evaluated the models and presented the results on a black background, for better contrast and clearer detail visualization. Here, we provide experimental results for three representative deformed scenes of NeRF Synthetic [4] on a white background. As shown in Table A, we have the same conclusions as the experiment on a black background: NeuralEditor *significantly and consistently* outperforms PointNeRF [7] and Naive Plotting, in both zero-shot inference and fine-tuning settings.

Model	Zero-Shot Inference			Fine-Tune for 1 Epoch		
	Hotdog	Lego	Drums	Hotdog	Lego	Drums
	PSNR ↑					
DeformingNeRF [8]	-	12.28	-	-	-	-
PointNeRF [7]	25.48	23.84	20.52	35.71	30.10	26.71
Naive Plotting	26.87	24.70	20.92	36.00	30.94	27.45
NeuralEditor w/o IST	26.88	24.71	20.92	36.27	31.45	27.64
NeuralEditor (Ours)	**27.27**	**26.13**	**21.41**	**36.66**	**31.70**	**27.68**
	SSIM ↑					
DeformingNeRF	-	0.690	-	-	-	-
PointNeRF	0.948	0.932	0.902	0.987	0.976	0.955
Naive Plotting	0.951	0.932	0.913	0.987	0.979	0.963
NeuralEditor w/o IST	0.951	0.932	0.913	0.987	0.981	0.963
NeuralEditor (Ours)	**0.957**	**0.964**	**0.920**	**0.988**	**0.983**	**0.964**
	LPIPS AlexNet ↓					
DeformingNeRF	-	0.271	-	-	-	-
PointNeRF	0.072	0.064	0.107	0.033	0.025	0.067
Naive Plotting	0.066	0.055	0.086	0.022	0.019	0.044
NeuralEditor w/o IST	0.064	0.054	0.085	**0.021**	0.017	**0.043**
NeuralEditor (Ours)	**0.059**	**0.031**	**0.079**	**0.021**	**0.016**	**0.043**
	LPIPS VGG ↓					
DeformingNeRF	-	0.291	-	-	-	-
PointNeRF	0.079	0.088	0.108	0.053	0.054	0.080
Naive Plotting	0.080	0.089	0.093	0.047	0.049	0.062
NeuralEditor w/o IST	0.079	0.087	0.092	0.045	0.043	0.061
NeuralEditor (Ours)	**0.073**	**0.056**	**0.087**	**0.044**	**0.041**	**0.060**

Table A. Consistent with the results on a *black* background in the main paper, we have the same conclusions when using on a *white* background here: NeuralEditor *significantly and consistently* outperforms PointNeRF [7] and Naive Plotting on the three representative deformed scenes of NeRF Synthetic [4], in both zero-shot inference and fine-tuning settings. With the precise point cloud generated by NeuralEditor, even Naive Plotting consistently outperforms PointNeRF. The metrics investigated here are peak signal-to-noise ratio (PSNR), structural similarity index measure (SSIM), and learned perceptual image patch similarity (LPIPS).

Interestingly, we observe that the results on a white background have consistently worse metric values than those on a black background. We find that this phenomenon is *not specific* to our task of shape deformation. In fact, even in rendering the original scene, we observe that a white background results in lower metric values, as shown in Table B

Background Color	PSNR ↑
White [7]	32.40
Black	32.99

Table B. For the conventional rendering [4] of the original scene of NeRF Synthetic (*e.g.*, Lego here), the metric values are also slightly worse when using a white background as in prior work [4, 7] than a black background.

for the experiment of PointNeRF 20K (PointNeRF trained for 20K epochs, following an evaluation setting in [7]) on Lego of NeRF Synthetic. The impact of the background color on NeRF rendering is an interesting aspect for future investigation.

B. Additional Ablation Study

Our additional ablation study results are in Table C. We observe that:

- All components in our NeuralEditor, including infinitesimal surface transformation (IST), deterministic integration, and Phong reflection, benefit the rendering results. Notably, our NeuralEditor still outperforms PointNeRF *without* any of these components.
- For the variant without Phong reflection modeling, the zero-shot performance is close to that of the full NeuralEditor, but the performance gap becomes much larger after fine-tuning. This demonstrates that the use of visual attributes modeled by Phong reflection (*e.g.*, tint) helps fast fitting to the ambient environment.
- Our NeuralEditor's performance drops with the initial point cloud or the final point cloud produced by PointNeRF, showing the importance of a precise point cloud optimized by our NeuralEditor for the rendering task. In these experiments, we apply de-noising on the point clouds to support IST, which is not compatible with the noisy original point clouds generated by PointNeRF.
- With half of the points, the performance of NeuralEditor decreases, but it still outperforms the baseline PointNeRF. This validates that it is not the model capacity but our model design together with the precise point clouds that leads to our superior performance.
- Our NeuralEditor even supports point cloud optimization on the deformed scene, which further improves the rendering results and achieves better PSNR than fine-tuning without point cloud optimization for the same epochs. The detailed settings of fine-tuning are described in Section H.4.

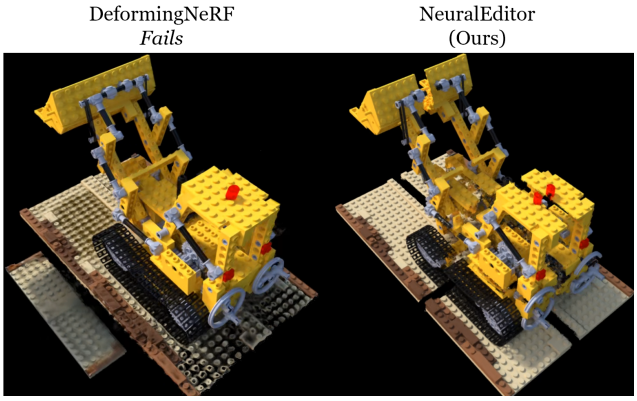


Figure A. Unlike Deforming-NeRF [8], our NeuralEditor has native support for non-continuous deformation tasks.

C. Non-Continuous Deformation Task

Figure A shows a non-continuous deformation task constructed on the Lego scene of NeRF Synthetic, by cutting the scene from the $x\mathrm{O}y$, $y\mathrm{O}z$, and $z\mathrm{O}x$ planes. NeuralEditor natively supports such deformation, while DeformingNeRF fails, further validating the superiority of NeuralEditor over cage-based methods for tackling deformation tasks.

D. Experiment on Tanks and Temples Dataset

We also present the evaluation results of our NeuralEditor and baselines Naive Plotting and PointNeRF [7] on the Tanks and Temples dataset [2], as shown in Figure B. As this dataset provides white-background ground truth images for original scenes, we use a white background for evaluation and visualization. Our NeuralEditor still produces better rendering results with fewer artifacts than baselines.

Note that Tanks and Temples is not a standard NeRF dataset but a multiview stereo (MVS) dataset. It contains some background regions that are not fully cut out, *e.g.*, the blue bar on the top of Caterpillar's cab (2nd column in Figure B) is a part of the sky background, making the NeRF rendering results blurry and noisy as some points are mistakenly grown in those regions. We propose a background sphere technique (explained in Section I.1) to improve the rendering results in this situation. While this approach is helpful, it cannot completely resolve the issue. Additionally, inconsistent exposure settings used across different views of the same scene cause the rendering results to appear blurry and flashing. Therefore, all three methods cannot achieve rendering results as clean and realistic as those on NeRF Synthetic, but ours still significantly outperforms the baselines.

Type	Variant	PSNR ↑ on Hotdog	
		Zero-Shot	Fine-Tune
NeRF Model	**Full NeuralEditor**	**27.49**	**37.22**
	− our improved point cloud-guided NeRF + PointNeRF [7]	25.95	36.08
Component	**Full NeuralEditor**	**27.49**	**37.22**
	− IST ('Ours w/o IST')	27.02	36.69
	− integration + traditional point sampling	27.48	36.69
	− deterministic integration + stochastic integration	27.46	36.87
	− NeRF modeling + plotting ('Naive Plotting')	27.01	36.38
	− normal vectors	27.26	37.00
	− Phong reflection color modeling + traditional color modeling	27.21	36.51
Point Cloud	**Full NeuralEditor**	**27.49**	**37.22**
	− point cloud optimization (w/ initial point cloud)	25.56	35.93
	− our optimized point cloud + point cloud optimized by PointNeRF	26.61	35.68
	− 50% points	26.84	36.59
Fine-Tune	Fine-tune 0 epoch ('zero-shot')	27.49	
	Fine-tune 1 epoch	37.22	
	Fine-tune 4 epochs	38.12	
	Fine-tune 10 epochs	38.56	
	Fine-tune 10 epochs w/ point cloud optimization	38.87	

Table C. Ablation study experiments show that (1) All components in NeuralEditor benefit the rendering results on deformed scenes; (2) Our NeuralEditor generates precise point clouds, which are crucial for shape editing tasks; (3) Our NeuralEditor even supports point cloud optimization during fine-tuning to further improve the rendering performance. '−' denotes that a certain component is removed, while '+' denotes that a certain component is added.

Figure B. Our NeuralEditor also generates better rendering results on deformed scenes of the Tanks and Temples [2] dataset with much fewer artifacts.

E. Intermediate Point Clouds for Scene Morphing

We show the intermediate point clouds for the scene morphing task (Figure 9) in Figure D.

F. Other Metrics in Table 1

We provide the results of the shape deformation task (Table 1 in the main paper) under the metrics of peak signal-to-noise ratio (PSNR), structural similarity index measure (SSIM), and learned perceptual image patch similarity (LPIPS) in Table D.

G. High-Resolution Visualization Figures

Here we provide the higher-resolution versions of the same experimental visualization figures in the main paper. The correspondence is listed below:

- Figure 6 (optimized point clouds): Figure E.
- Figure 7 (baseline DeformingNeRF): Figure F.
- Figure 8 (shape deformation): Figure G.
- Figure 9 (scene morphing): Figure H, with full morphing results for the baseline PointNeRF.

H. Implementation Details

H.1. K-D Voxels & Integration

We use a 21-layer K-D tree to build K-D voxels, which can contain up to 2^{21} ($\sim$ 2 million) points. Compared with PointNeRF that typically deals with 1 million points, our NeuralEditor can hold twice the number of points for higher capacity, and also achieves better results with the same magnitude of points as PointNeRF, as shown in Section B.

We use the voxels in the bottom 4 non-leaf layers for rendering, and use the additional 5th layer from the bottom only for point cloud growing. When doing integration for rendering, we uniformly select s points on the intersect interval, including the two side points, and apply spline integration using the features of these points, where we choose $s = 8, 11, 16, 22, 22$ for the 5 used layers from the bottom. For the feature of each selected point, we use its K nearest neighbors (KNN) for interpolation, where $K = 32$.

Such integration is not only for aggregating the average point features over the interval, but also for calculating the average normal vectors and average IST – we use KNN to interpolate these values in a similar manner as interpolating the features. For each segment, we use the average point feature, average normal vector, and average IST in the color modeling of the representative point.

H.2. Phong Reflection Color Modeling

Consistent with RefNeRF [6], we use several multilayer perceptrons (MLPs) that directly take the integrated average feature as input to obtain the tint, roughness, modeled normal vector, and diffuse and specular color. The modeled normal vectors are trained with and controlled by the regularization losses introduced in RefNeRF. These normal vectors may be different from the normal vectors estimated from the point cloud, and they are optimized towards the estimated normal vectors via an additional regularization loss weighted by point confidence. The effect of this somewhat redundant modeling of normal vectors is two-fold: (1) The outlier points with abnormal normal vectors can be difficult to fit into the modeled ones, so their confidence values will be driven to zero when minimizing the regularization loss; (2) The estimated normal vectors can supervise and provide extra shape information for the point features through the MLP that models the normal vectors.

H.3. Point Cloud Optimization

Pruning & Growing. Following PointNeRF, we define the defects of a point cloud as two types: *outliers* and *holes*. So any shape defection condition can be decomposed into a sequence of these two types of simple defects. We design our optimization strategy based on the "pruning and growing" (P&G) method introduced in PointNeRF, which prunes the outliers by driving their confidence values to zero during training, and grows the point cloud to probe holes by selecting points from the sampled points on the training rays.

Our strategy is different from PointNeRF in several important ways. In our growing process, we grow all rays in the training dataset at a special evaluation epoch ("growing epoch"). For each ray, we pick the point far from any point in the point cloud with the highest volume density as a candidate. The candidate point whose ray has a higher pixel rendering loss is assigned a higher priority for being added to the point cloud. We then introduce a K-D tree-guided algorithm to down-sample these growing candidates: We start from the root node, and recurse on the sub-nodes until we reach a pre-set voxel size; we then add the candidate point with the highest priority in this voxel to the point cloud, while disregarding other candidate points in the same voxel. Doing so ensures the added candidate points uniformly distributed throughout the space, by preserving only the points with the highest priorities within their respective regions. We finalize the growth of candidate points by setting their features to interpolated features and their confidence values to 0.5. Note that PointNeRF uses P&G in conjunction with a stochastic training process, whereas we apply it to our deterministic NeuralEditor during a standalone growing epoch, which enhances training stability. As a result, we obtain precise point clouds while PointNeRF cannot.

Point Cloud Denoising. It is common that a point cloud

contains some noisy or isolated points. To remove these isolated points, we use Open3D [10] to identify statistical outliers w.r.t. the K-th nearest neighbor distance of each point, where $K = 64$ in our setting. Also, as mentioned above, noisy points may have irregular estimated normal vectors, and their confidence will be driven to zero with the regularization losses introduced in Section H.2.

Point Cloud Optimization Process. We regard one "point cloud optimization process" as an additional process before one training epoch, which includes (1) applying one growing epoch to obtain grown points, (2) pruning the isolated points and the points with confidence values lower than 0.1, and (3) constructing K-D voxels with the adjusted point cloud for further training. During training (Section H.4), we apply this process with a few training epochs to optimize the point cloud.

H.4. Training Settings

Training on Original Scene. We pre-train NeuralEditor on the original scene using per-scene optimization, and obtain a model that includes a precise point cloud and its points' features for shape editing tasks. During per-scene pre-training, we start with the initial point cloud generated by the point generation network. We first train our model for 3 epochs as warm-up. From the 4th to the 12th epoch, we include an additional point cloud optimization process (Section H.3) before each training epoch. By the end of the 12th epoch, the point cloud is determined and precise enough. We keep tuning the model parameters on this underlying point cloud for up to 100 epochs, controlled with early stopping. Notably, the precise point cloud can be determined and obtained within the first 12 training epochs, even though the whole training process may take a long time.

Fine-Tuning on Deformed Scene. We apply the same training process during fine-tuning on deformed scenes. For the setting "Fine-tune 10 epochs w/ point cloud optimization" in Table C, we first train one epoch as warm-up, then train with an additional point cloud optimization process for 5 epochs, and continue to fine-tune on the optimized point cloud for the rest 4 epochs.

H.5. Matching Algorithm for Scene Morphing

To perform scene morphing, we generate the intermediate point clouds using a point cloud diffusion model [3]. However, these point clouds are *unindexed*. To render the scene with NeuralEditor, we need to assign an index to each point. This index assignment can be solved by a matching algorithm. Specifically, given point clouds $P_0, P_1, \cdots, P_n$, we can match the points in each adjacent pair of point clouds P_i and P_{i+1}, and then permute the points in P_{i+1} according to the matching to align the indices.

We design a simple matching algorithm based on K-D trees. We simultaneously build two K-D trees, one for each point cloud. At each node, we select the same division axis for the two point clouds, according to their union point set. At each leaf node, we match a pair of single points from each point cloud. Our algorithm aims to match points in the two point clouds that have similar relative locations. In an ideal case where there are numerous intermediate point clouds and every adjacent pair of point clouds is sufficiently close, this algorithm will lead to highly accurate matching results.

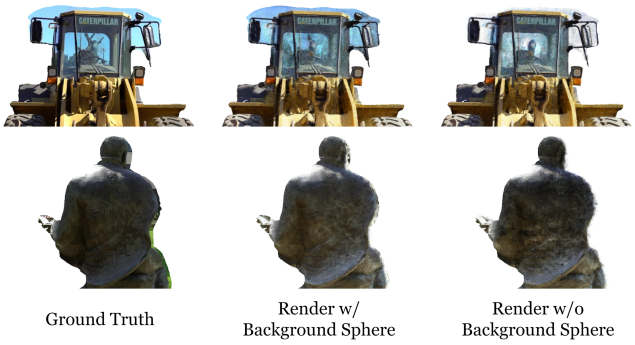


Figure C. Modeling a background sphere helps NeuralEditor to differentiate between the background and the foreground scene, thus preventing it from growing wrong points to model the background. Doing so potentially improves the robustness of the point cloud-guided NeRF model.

I. Limitations

I.1. Point Cloud-Guided NeRF

Despite offering several advantages over traditional NeRFs, the current point cloud-guided NeRF models, including ours and PointNeRF, still have some limitations. First, the scene is modeled with an explicit representation (the point cloud), which is not robust when modeling surfaces with complicated visual effects, *e.g.*, a semi-transparent blurry mirror. When the model fails to interpret such visual effects, our point cloud optimization might not generate correct points close to the surface, limiting the model's ability to improve results. Additionally, these models cannot well support strategies in NeRFReN [1] for simultaneously modeling the real-world scene and the mirrored scene to better handle mirror reflections. This is because a point cloud-guided NeRF model relies on MVS-based initialization and point cloud optimization that cannot accommodate such a form of co-optimization, which may significantly change the shape of the mirrored scene during training.

Another limitation of the point cloud-guided NeRF models is their non-robustness against inaccurate background

masks, as discussed in Section D. In the existing datasets, a mask is often given to distinguish the foreground from the background, so we only need to train NeRF on the foreground. However, if the mask is not precise, some regions of the background would be mistakenly included in the mask, as in the case of the Tanks and Temples dataset (*e.g.*, the sky in the Caterpillar scene in Figure C). Since a point cloud-guided NeRF model requires points to represent the entire scene including the background, it will try to grow the points in those background regions and use them to represent the background. On the other hand, the background regions can be far from the foreground, so the model will grow the points near the scene object rather than their true location. In normal situations, each point only represents a specific region of the scene. By contrast, for each of these mistakenly grown points, its view-dependent color is trained with different regions of the background which are inconsistent, thus resulting in abnormal rendering results in novel views.

To address this issue, we enhanced NeuralEditor by modeling a *background sphere* that covers the entire scene. Therefore, the point cloud-guided NeRF model can directly represent the scene's background using the sphere, instead of growing new points. Such a strategy was introduced for the Tanks and Temples dataset in Figure B. Figure C further analyzes the impact of this background sphere, validating its effectiveness in approximately modeling background regions without growing wrong points. Further investigation is worthwhile in other strategies to tackle this issue.

I.2. Environment Modeling in Shape Deformation Task

Neither our work nor existing methods [5, 8, 9] take into account the surrounding ambient environment when addressing the shape deformation task. These methods thus cannot assign different colors to the scene according to the changes in lighting conditions, as shown in the top of the shovel and the bent chimney in the Lego scene and the shadow on the cushion in the Chair scene (Figure G). Fortunately, our NeuralEditor, incorporating the Phong reflection's visual attributes (*e.g.*, tint), enables fast fitting to the ambient environment through fine-tuning on the deformed scene (Section B).

I.3. Evaluation for Scene Morphing Task

Quantitatively evaluating the visual realism of intermediate scenes during morphing is challenging, due to the lack of ground truth and associated metrics, as these scenes "do not exist" in the real world. Therefore, we rely mainly on visualizations for evaluation.

References

[1] Yuan-Chen Guo, Di Kang, Linchao Bao, Yu He, and Song-Hai Zhang. NeRFReN: Neural radiance fields with reflections. In *CVPR*, 2022. 5

[2] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and Temples: Benchmarking large-scale scene reconstruction. *ACM Trans. Graph.*, 36(4), 2017. 2, 3

[3] Shitong Luo and Wei Hu. Diffusion probabilistic models for 3D point cloud generation. In *CVPR*, 2021. 5, 7

[4] 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. In *ECCV*, 2020. 1, 2, 8

[5] Yicong Peng, Yichao Yan, Shengqi Liu, Yuhao Cheng, Shanyan Guan, Bowen Pan, Guangtao Zhai, and Xiaokang Yang. CageNeRF: Cage-based neural radiance field for generalized 3D deformation and animation. In *NeurIPS*, 2022. 6

[6] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. In *CVPR*, 2022. 4

[7] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-NeRF: Point-based neural radiance fields. In *CVPR*, 2021. 1, 2, 3, 7, 8

[8] Tianhan Xu and Tatsuya Harada. Deforming radiance fields with cages. In *ECCV*, 2022. 1, 2, 6, 7, 8

[9] Yu-Jie Yuan, Yang tian Sun, Yu-Kun Lai, Yuewen Ma, Rongfei Jia, and Lin Gao. NeRF-Editing: Geometry editing of neural radiance fields. In *CVPR*, 2022. 6

[10] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. 5

Model	Zero-Shot Inference								Fine-Tune for 1 Epoch							
	Chair	Hotdog	Lego	Drums	Ficus	Materials	Mic	Ship	Chair	Hotdog	Lego	Drums	Ficus	Materials	Mic	Ship
	PSNR ↑															
DeformingNeRF [8]	18.84	-	13.10	-	-	-	-	-	-	-	-	-	-	-	-	-
PointNeRF [7]	22.21	25.95	24.56	21.00	24.24	21.21	26.77	21.19	30.11	36.08	31.45	27.16	31.48	27.55	34.34	28.90
Naive Plotting	24.91	27.01	25.64	21.29	26.22	21.65	27.63	22.29	32.01	36.38	31.72	28.09	33.21	30.31	35.15	30.01
NeuralEditor w/o IST	24.92	27.02	25.65	21.29	26.24	21.64	27.64	22.28	32.24	36.69	32.79	28.30	33.34	30.40	35.28	30.08
NeuralEditor (Ours)	**25.85**	**27.49**	**27.46**	**21.84**	**27.19**	**23.18**	**27.75**	**24.16**	**32.53**	**37.22**	**32.95**	**28.35**	**33.53**	**30.82**	**35.46**	**30.44**
	SSIM ↑															
DeformingNeRF	0.865	-	0.645	-	-	-	-	-	-	-	-	-	-	-	-	-
PointNeRF	0.910	0.947	0.933	0.890	0.915	0.856	0.945	0.759	0.972	0.988	0.977	0.953	0.973	0.925	0.981	0.875
Naive Plotting	0.950	0.954	0.934	0.908	0.953	0.887	0.964	0.844	0.984	0.987	0.977	0.962	0.985	0.968	0.988	0.935
NeuralEditor w/o IST	0.950	0.954	0.934	0.908	0.953	0.887	0.964	0.845	0.984	0.988	0.982	**0.963**	0.985	0.968	0.988	0.936
NeuralEditor (Ours)	**0.963**	**0.960**	**0.966**	**0.916**	**0.960**	**0.909**	**0.966**	**0.875**	**0.986**	**0.989**	**0.983**	**0.963**	**0.986**	**0.971**	**0.989**	**0.939**
	LPIPS AlexNet ↓															
DeformingNeRF	0.071	-	0.297	-	-	-	-	-	-	-	-	-	-	-	-	-
PointNeRF	0.049	0.080	0.067	0.122	0.062	0.099	0.057	0.139	0.018	0.033	0.023	0.077	0.028	0.067	0.035	**0.073**
Naive Plotting	0.038	0.063	0.053	0.088	0.047	0.098	0.039	0.159	0.014	0.023	0.019	0.047	0.021	0.042	0.020	0.081
NeuralEditor w/o IST	0.037	0.062	0.052	0.087	0.046	0.097	0.039	0.158	0.013	0.021	0.015	0.046	0.020	0.041	**0.019**	0.080
NeuralEditor (Ours)	**0.030**	**0.057**	**0.029**	**0.080**	**0.042**	**0.076**	**0.037**	**0.126**	**0.012**	**0.020**	**0.015**	**0.045**	**0.019**	**0.035**	**0.019**	0.075
	LPIPS VGG ↓															
DeformingNeRF	0.067	-	0.291	-	-	-	-	-	-	-	-	-	-	-	-	-
PointNeRF	0.047	0.082	0.060	0.115	0.070	**0.091**	0.044	**0.153**	**0.019**	0.055	0.055	0.086	**0.039**	0.061	0.029	**0.090**
Naive Plotting	0.051	0.080	0.091	0.094	0.070	0.109	0.040	0.183	0.029	0.050	0.049	0.068	0.044	0.065	0.030	0.129
NeuralEditor w/o IST	0.051	0.079	0.088	0.093	0.069	0.107	0.040	0.182	0.027	0.048	0.043	0.066	0.042	0.064	0.029	0.127
NeuralEditor (Ours)	**0.041**	**0.074**	**0.057**	**0.088**	**0.067**	0.095	**0.038**	0.163	0.026	**0.045**	**0.042**	**0.065**	0.042	**0.058**	**0.028**	0.121

Table D. **Full comparison results of Table 1** in the main paper under all metrics. NeuralEditor *significantly and consistently* outperforms PointNeRF and Naive Plotting on all deformed scenes of NeRF Synthetic *under all metrics*, in both zero-shot inference and fine-tuning settings. Our infinitesimal surface transformation (IST) effectively improves the results by correcting the view-dependent colors. With the precise point cloud generated by NeuralEditor, even Naive Plotting consistently outperforms PointNeRF.

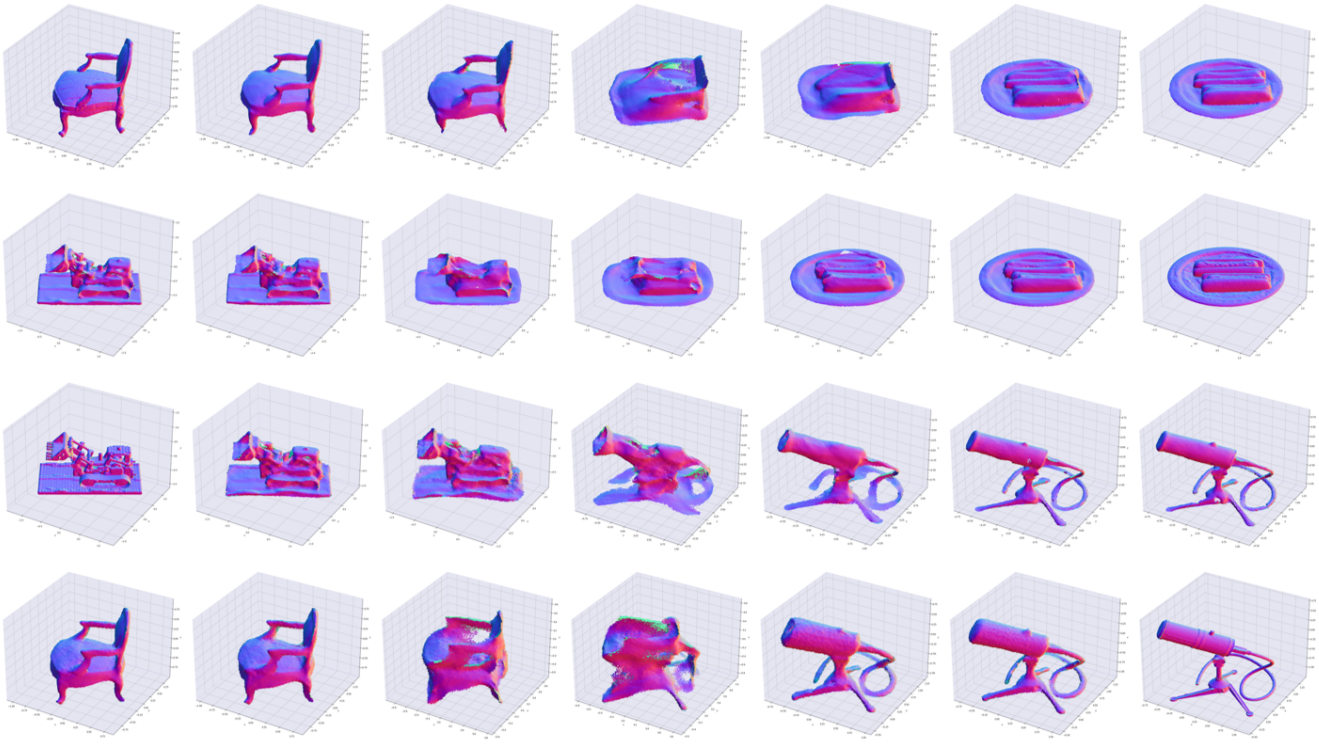


Figure D. **Intermediate point clouds for the scene morphing task corresponding to Figure 9** in the main paper, which are generated by the point cloud diffusion model [3].

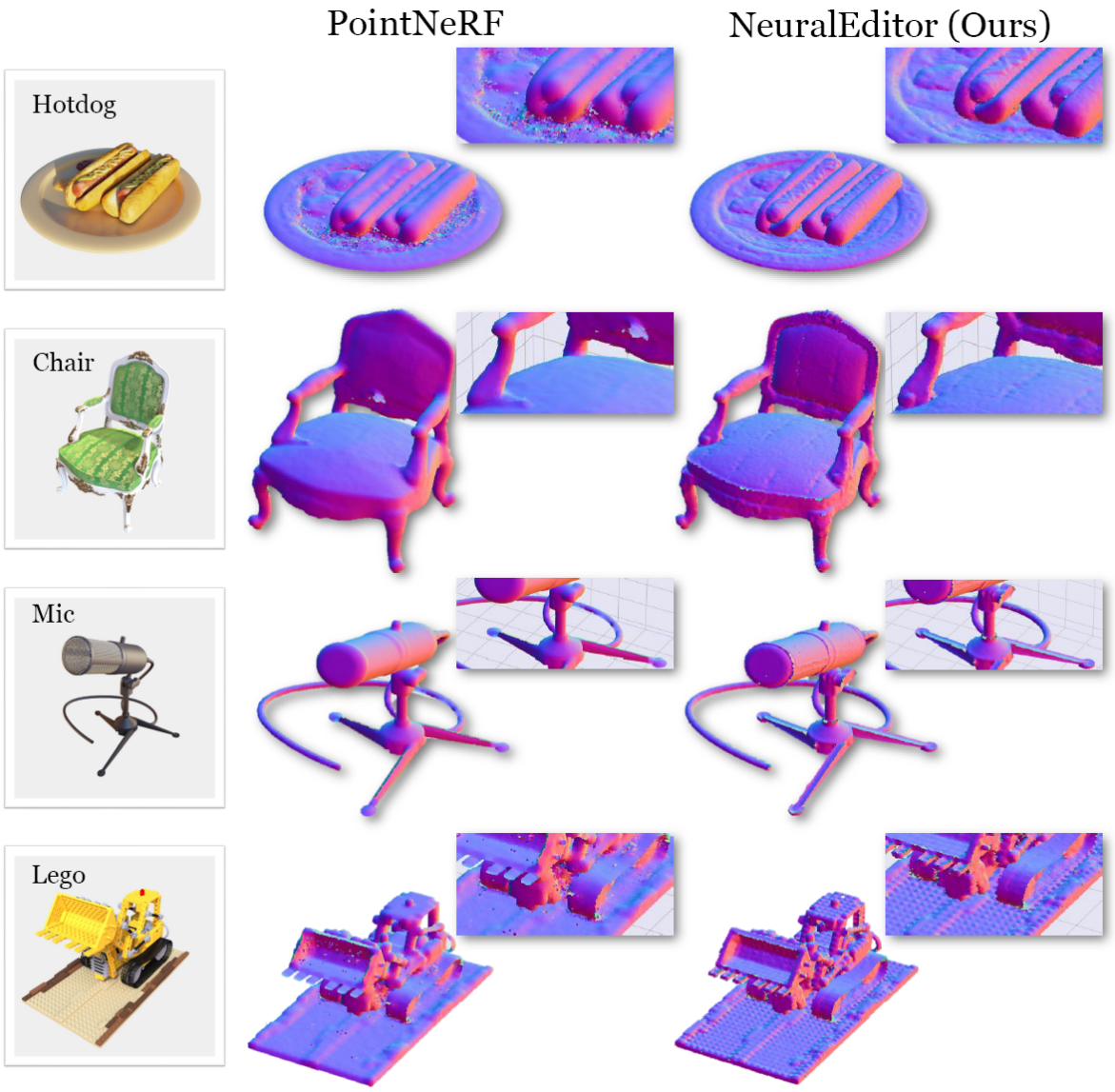


Figure E. **High-resolution version of Figure 6** in the main paper for more detailed visualization. NeuralEditor generates much more precise point clouds than PointNeRF [7] in the four scenes of NeRF Synthetic [4]. The points are colored with their normal vectors.

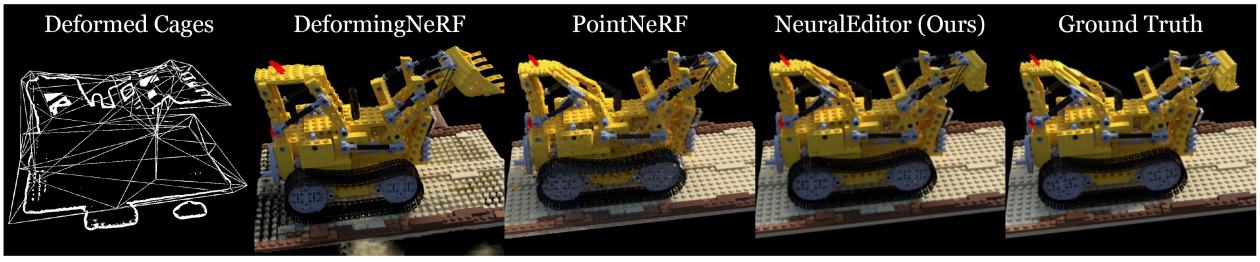


Figure F. **High-resolution version of Figure 7** in the main paper for more detailed visualization. With too coarse cages, DeformingNeRF [8] is unable to perform the deformation faithfully, leading to poor results.

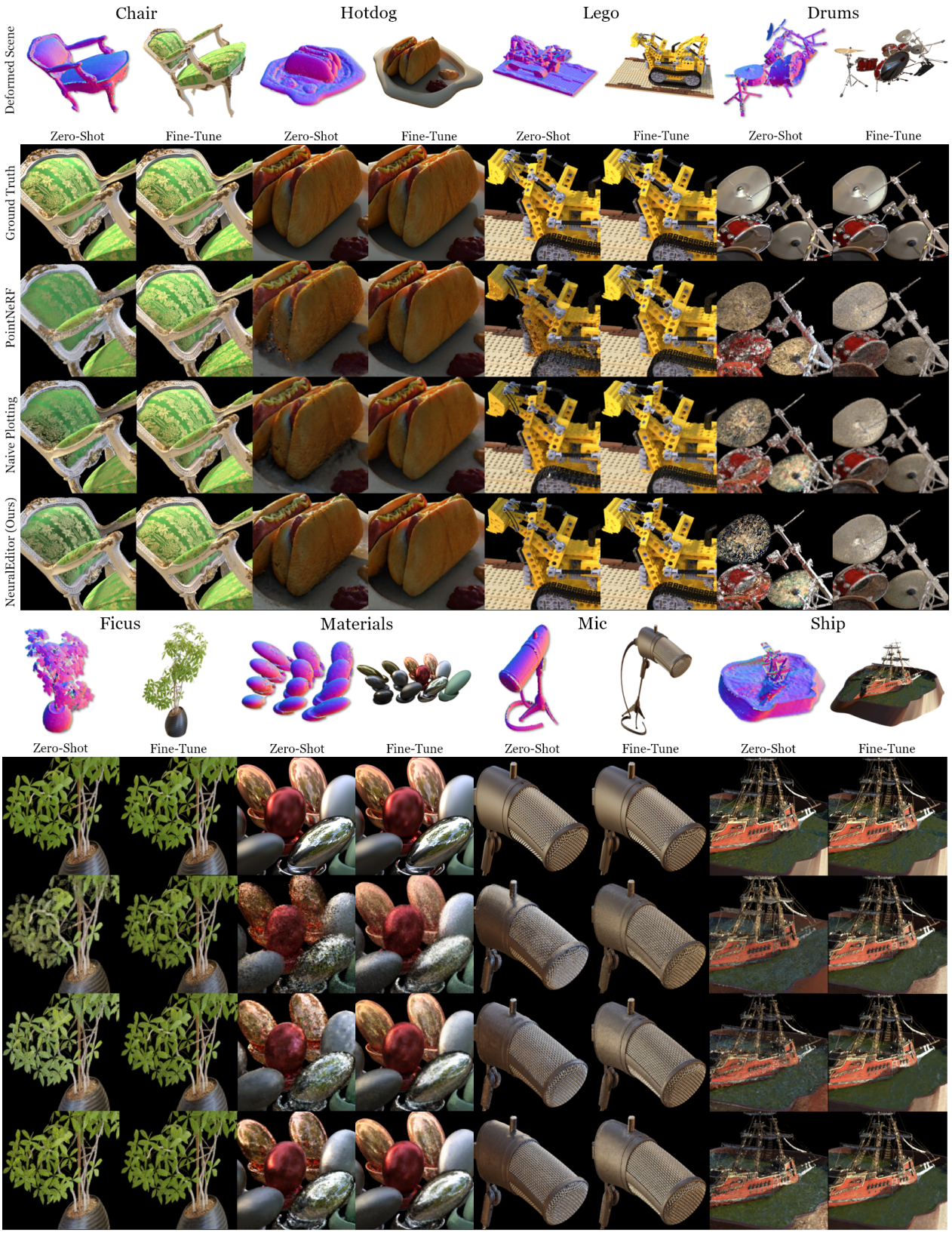
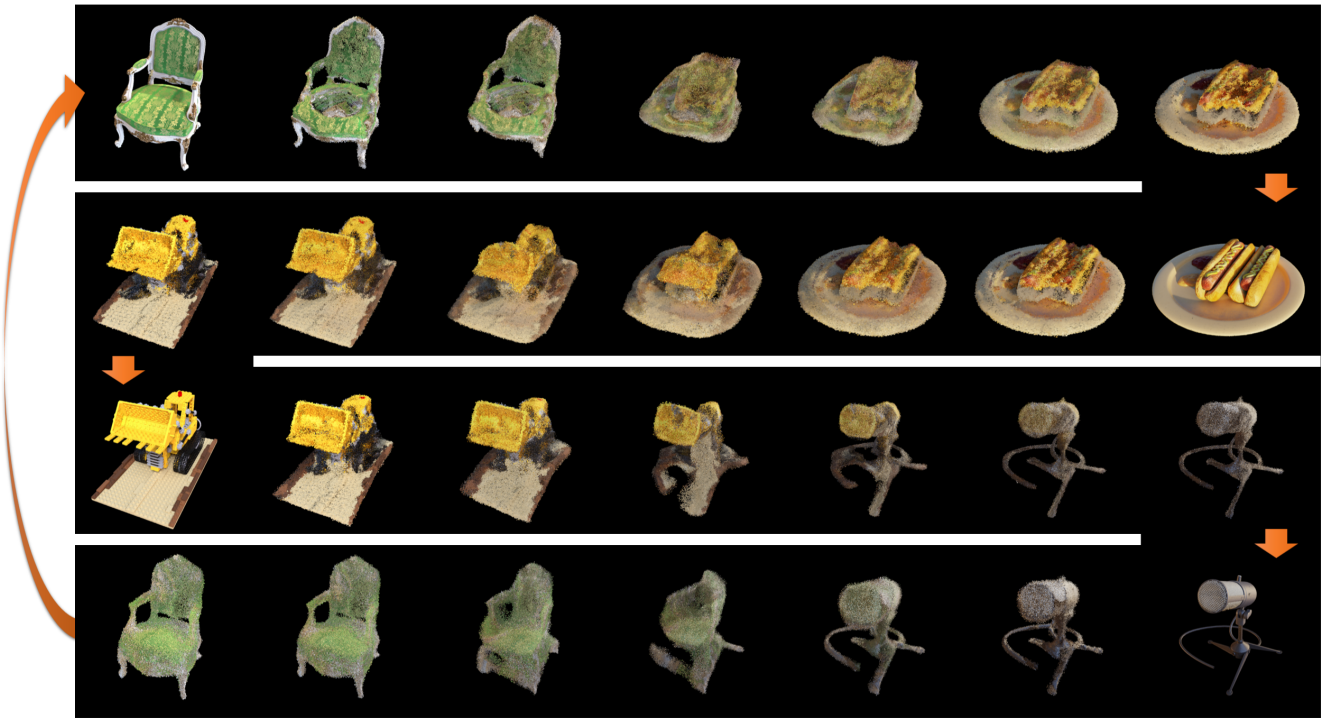
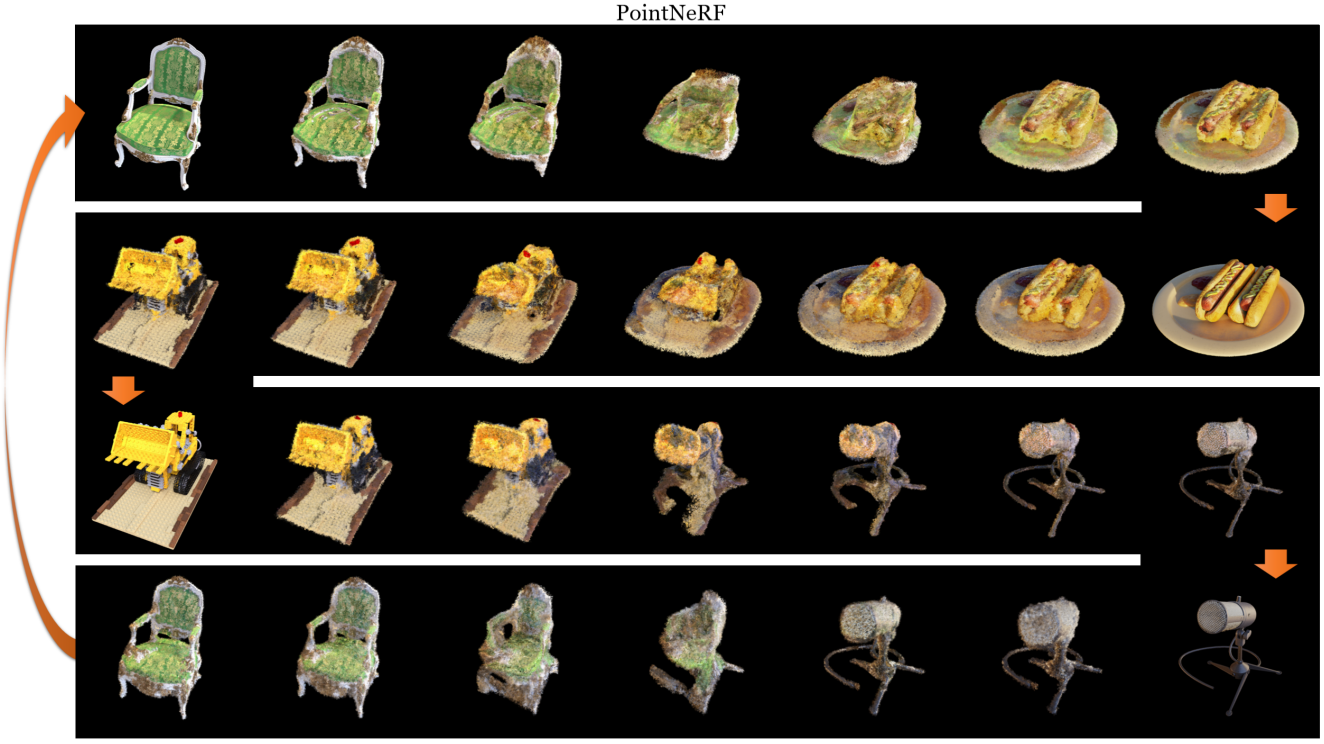


Figure G. **High-resolution version of Figure 8** in the main paper for more detailed visualization. NeuralEditor produces superior rendering results to PointNeRF, with significantly fewer artifacts in zero-shot inference. Fine-tuning further improves the *consistency of rendering with the ambient environment*. We use a black background for better visualization.



PointNeRF



NeuralEditor (Ours)

Figure H. **High-resolution, extended version of Figure 9** in the main paper for more detailed visualization. Our NeuralEditor produces *smooth morphing* results between Chair, Hotdog, Lego, and Mic in the NeRF Synthetic dataset, while PointNeRF produces results with blurry textures, black shadows, and gloomy, non-smooth colors. The rendering results in the looped morphing process are arranged in the shape of the numerical digit "3," indicated by the dividing lines and arrows. For the baseline PointNeRF, in the main paper we showed the morphing results between Chair and Hotdog due to limited space; here we include the full morphing results across all 4 scenes.