NeuralEditor: Editing Neural Radiance Fields via Manipulating Point Clouds

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
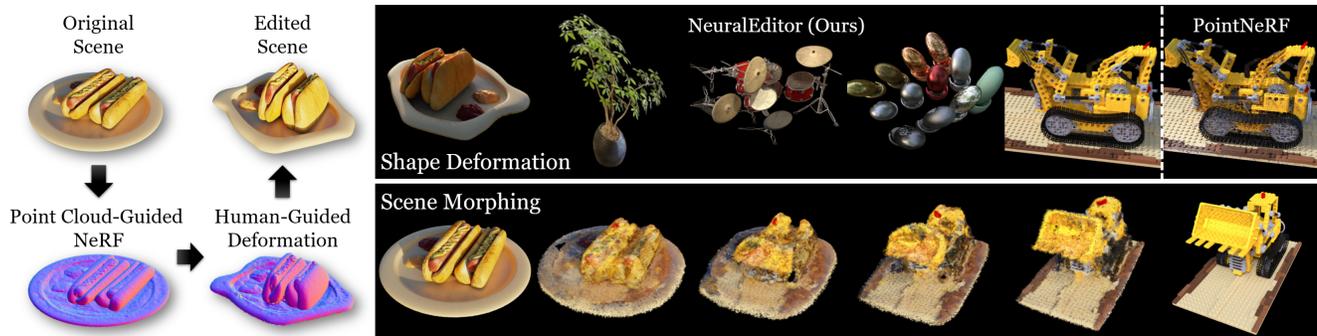
Figure 1. Our NeuralEditor offers *native* support for general and flexible shape editing of neural radiance fields via manipulating point clouds. By generating a precise point cloud of the scene with a novel point cloud-guided NeRF model, our NeuralEditor produces high-fidelity rendering results in both shape deformation and more challenging scene morphing tasks.

Abstract

This paper proposes NeuralEditor that enables neural radiance fields (NeRFs) natively editable for general shape editing tasks. Despite their impressive results on novel-view synthesis, it remains a fundamental challenge for NeRFs to edit the shape of the scene. Our key insight is to exploit the explicit point cloud representation as the underlying structure to construct NeRFs, inspired by the intuitive interpretation of NeRF rendering as a process that projects or "plots" the associated 3D point cloud to a 2D image plane. To this end, NeuralEditor introduces a novel rendering scheme based on deterministic integration within K-D tree-guided density-adaptive voxels, which produces both high-quality rendering results and precise point clouds through optimization. NeuralEditor then performs shape editing via mapping associated points between point clouds. Extensive evaluation shows that NeuralEditor achieves state-of-the-art performance in both shape deformation and scene morphing tasks. Notably, NeuralEditor supports both zero-shot inference and further fine-tuning over the edited scene. Our code, benchmark, and demo video are available at im-mortalco.github.io/NeuralEditor.

1. Introduction

Perhaps the most memorable shot of the film *Transformers*, *Optimus Prime* is seamlessly transformed between a humanoid and a Peterbilt truck – such free-form editing of 3D objects and scenes is a fundamental task in 3D computer vision and computer graphics, directly impacting applications such as visual simulation, movie, and game industries. In these applications, often we are required to manipulate a scene or objects in the scene by editing or modifying its shape, color, light condition, *etc.*, and generate visually-faithful rendering results on the edited scene efficiently. Among the various editing operations, shape editing has received continued attention but remains challenging, where the scene is deformed in a human-guided way, while all of its visual attributes (*e.g.*, shape, color, brightness, and light condition) are supposed to be natural and consistent with the ambient environment.

State-of-the-art rendering models are based on implicit neural representations, as exemplified by neural radiance field (NeRF) [27] and its variants [3, 33, 37, 39, 48]. Despite their impressive novel-view synthesis results, most of the NeRF models substantially lack the ability for users to adjust, edit, or modify the shape of scene objects. On the other hand, shape editing operations can be natively applied to explicit 3D representations such as point clouds and meshes.

Inspired by this, we propose *NeuralEditor* – a general and flexible approach to editing neural radiance fields via manipulating point clouds (Fig. 1). Our *key insight* is to benefit from the best of both worlds: the superiority in rendering performance from implicit neural representation

combined with the ease of editing from explicit point cloud representation. NeuralEditor enables us to perform a wide spectrum of shape editing operations in a consistent way.

Such introduction of point clouds into NeRF for general shape editing is rooted in our interpretation of *NeRF rendering as a process that projects or "plots" the associated 3D point cloud to a 2D image plane*. Conceptually, with a dense enough point cloud where each point has an opacity and its color is defined as a function of viewing direction, directly plotting the point cloud would achieve similar visual effects (*i.e.*, transparency and view-dependent colors) that are rendered by NeRF. This *intrinsic integration* between NeRF and point clouds underscores the advantage of our NeuralEditor over existing mesh-based NeRF editing methods such as NeRF-Editing [51], Deforming-NeRF [44], and CageNeRF [30], where the process of constructing and optimizing the mesh is separated from the NeRF modeling, making them time-consuming. More importantly, with the point cloud constructed for a scene, the shape editing can be natively defined as and easily solved by just moving each point into the new, edited position and re-plotting the point cloud. Therefore, our approach supports more general scene editing operations which are difficult to achieve via mesh-guided space deformation.

The key component in our NeuralEditor lies in a point cloud-guided NeRF model that natively supports general shape editing operations. While the recent method Point-NeRF [43] has demonstrated improved novel-view synthesis capability based on point clouds, it is not supportive to shape editing. Our idea then is to exploit the underlying point cloud in ways of not only optimizing its structure and features (*e.g.*, adaptive voxels) for rendering, but also extracting additional useful attributes (*e.g.*, normal vectors) to guide the editing process. To this end, we introduce K-D trees [4] to construct density-adaptive voxels for efficient and stable rendering, together with a novel deterministic integration strategy. Moreover, we model the color with the Phong reflection [31] to decompose the specular color and better represent the scene geometry.

With a much more precise point cloud attributed to these improvements, our NeuralEditor achieves high-fidelity rendering results on deformed scenes compared with prior work as shown in Fig. 1, even in a *zero-shot* inference manner without additional training. Through fast fine-tuning, the visual quality of the deformed scene is further enhanced, almost perfectly consistent with the surrounding light condition. In addition, under the guidance of a point cloud diffusion model [24], NeuralEditor can be naturally extended for smooth *scene morphing* across multiple scenes, which is difficult for existing NeRF editing work.

Our contributions are four-fold. (1) We introduce NeuralEditor, a flexible and versatile approach that makes neural radiance fields editable through manipulating point clouds. (2) We propose a point cloud-guided NeRF model based on K-D trees and deterministic integration, which produces precise point clouds and supports general scene editing. (3) Due to the lack of publicly available benchmarks for shape editing, we construct and release a reproducible benchmark that promotes future research on shape editing. (4) We investigate a wide range of shape editing tasks, covering both shape deformation (as studied in existing NeRF editing work) and challenging scene morphing (a novel task addressed here). NeuralEditor achieves state-of-the-art performance on all shape editing tasks in a unified framework, without extra information or supervision.

2. Related Work

Neural Scene Representation. Traditional methods model scenes with explicit [2, 11, 15, 20, 34, 38] or implicit [6, 19, 26, 28, 40] 3D geometric or shape representations. Initiated by NeRF [27], leveraging implicit neural networks to represent scenes and perform novel-view synthesis has become a fast-developing field in 3D vision [8,9]. While most of the follow-up work focuses on improving aspects such as the rendering realism [3, 10, 21, 37], efficiency [33, 41, 48], and cross-scene generalization [5, 39, 43, 49], the scene editing capability is substantially missing in the NeRF family which we address in this paper. In addition, we exploit K-D tree-guided point clouds as the underlying structure, different from other NeRF variants based on octrees [21, 33, 48] or plain voxels [41].

Point-Based NeRFs. Recently, using point clouds to build a NeRF model has shown better encoding of scene shape and improved rendering performance, as represented by PointNeRF [43]. PointNeRF proposes a point initialization network to produce the initial point cloud together with the point features, which is further optimized by a pruning and growing strategy. While both PointNeRF and our NeuralEditor employ point clouds as the underlying structure, NeuralEditor better exploits useful information within the point clouds: PointNeRF only directly uses the locations of points; by contrast, NeuralEditor *considers the point cloud more as a geometrical shape* and extracts relevant information like normal vectors, which plays an important role in rendering and shape editing. Importantly, our approach is designed to support scene editing, in contrast to PointNeRF.

Scene Editing via NeRFs. Different types of scene editing have been studied under NeRFs. EditNeRF [22], ObjectNeRF [45], and DistillNeRF [18] perform simple shape and color editing for objects specified with human-input scribble, pixel, segment, language, *etc*. NeuPhysics [32] edits a dynamic scene via physics parameters. CCNeRF [35] proposes an explicit NeRF representation with tensor rank decomposition to support scene composition. INSP-Net [42] considers filter editing like denoising. Such work cannot address 3D shape editing and only sup-

ports simple editing operations, like object selection, similarity transformation, or limited shape deformation.

3D Shape Editing. Traditional representation methods support keypoint-based shape editing [13, 14, 16, 25, 36, 47, 52] with meshes [29, 50], which cannot be directly applied to implicit representations used by NeRF. Existing NeRF editing work primarily studies a particular shape editing task, mesh deformation, and addresses it in a common paradigm [30, 44, 51]: A mesh of the scene is first constructed by either exporting it from a trained NeRF with the Marching Cubes algorithm [23], or optimizing close-to-surface cages along with training. After the user deforms the mesh, the deformed scene is rendered by deforming the space and bending the viewing rays in the original scene with the trained NeRF. Doing so requires extra efforts to convert *implicit* scene representation to *explicit* mesh, which might not be precise enough, and only supports *continuous* shape editing that can be converted to space deformation. On the contrary, our NeuralEditor directly maintains and utilizes alternative explicit scene representation – *the point cloud which is intrinsically integrated with NeRF*, making NeuralEditor require no extra efforts and support more general shape editing tasks like scene morphing. NeuralEditor supports both zero-shot inference and further fine-tuning over the edited scene, while prior work cannot.

3. NeuralEditor: Point Cloud-Guided NeRF

We propose a novel point cloud-guided NeRF model, NeuralEditor – it not only achieves realistic rendering results in the novel-view synthesis task, but also produces a point cloud that precisely describes the shape of the scene, thus facilitating general shape editing tasks. As illustrated in Fig. 2, we leverage the K-D trees [4] to construct *density-adaptive* voxels (which also naturally enable us to skip empty spaces), and introduce *deterministic* spline integration for rendering. We use the Phong reflection to model the color along with the normal vectors obtained from the underlying point cloud. With our enhanced point cloud optimization, NeuralEditor obtains much more precise underlying point clouds, compared with noisy and imprecise outputs of state-of-the-art PointNeRF [43] (as shown in Sec. 5).

3.1. K-D Tree-Guided Voxels

To render with points, we construct multi-scale density-adaptive voxels based on K-D trees [4], namely, *K-D voxels*. K-D trees are a data structure constructed on K-dimensional points, where $K = 3$ for 3D points. As a special decision tree, K-D tree's each node divides the point set into two equal-sized parts with axis-parallel criterion.

For each K-D tree's node, we compute its bounding box by taking the minimum and maximum x, y, z coordinates in its subtree and with proper padding margins. As we divide the points in a top-down manner in one of the x, y, z direc-
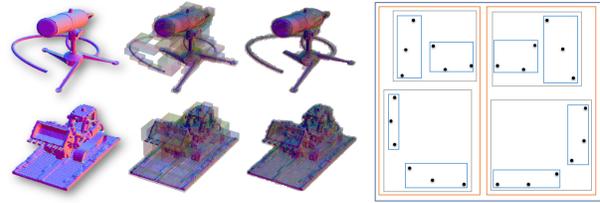
Figure 2. **Our K-D Voxels. Column 1:** original point clouds constructed from two scenes, colored with normal vector directions. **Column 2:** upper-level voxels which coarsely represent the shape. **Column 3:** lower-level voxels which tightly cover the shape. **Right:** Visualization of K-D voxels on a 2D point cloud. Each color represents boxes of nodes on each level of the K-D tree. Lower-level boxes containing fewer points cover the shape more tightly, and vice versa for higher-level boxes.

tions, in each layer of the K-D tree, different nodes' bounding boxes are *mutually exclusive*. Therefore, the bounding boxes can be regarded as voxels. As boxes in the upper layers contain more points (larger voxels), while those in the lower layers contain fewer points (smaller voxels), we natively obtain a *multi-scale* voxel construction from one K-D tree. As shown in Fig. 2, voxels from the large to small scales represent the shape of the scene from coarse to fine.

3.2. Rendering Over K-D Voxels

We now introduce a rendering scheme that exploits K-D voxels to perform all the sub-procedures associated with rendering in a unified way. This scheme enables us to render more naturally, efficiently, and even deterministically, meanwhile it also simplifies some widely-adopted design choices in conventional NeRF rendering.

Skipping Empty Spaces. In NeRF rendering, we are supposed to focus only on the *surface* of scene objects. As shown in Fig. 2, all our K-D voxels are produced to stick to the surface of objects, which the point cloud is constructed to describe. Such a property allows us to avoid explicitly "skipping" empty spaces, which often requires extra consideration in most NeRF models – only considering the space inside a voxel automatically focuses on the surface; as the depth of the voxel's node goes deeper, it becomes closer to the surface. Moreover, during the construction of the K-D tree, the points at each node are divided within its sub-nodes, and the node's voxel fully covers all its sub-nodes. This further provides us with a native *top-down recursive* procedure to locate the voxels intersected with the querying ray: We start from the root node, and recurse on the sub-nodes until we (1) reach a pre-set node depth (or equivalently, a pre-set voxel size) and then query within the associated voxel, or (2) stop recursion on non-intersected nodes.

Density-Adaptive Rendering. An important design in NeRFs is the coarse-to-fine strategy for density-adaptive rendering, so that more points are sampled in high-volume density areas. Our K-D voxels natively support such a design *without additional bells and whistles*. This is because
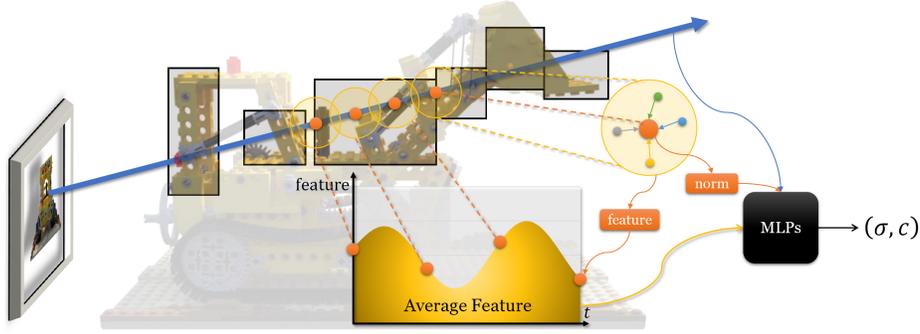
Figure 3. **Our NeuralEditor architecture.** We propose deterministic spline integration for KNN-based point features over each K-D tree-guided density-adaptive voxel, and model the color via Phong reflection with normal vectors estimated from the point cloud's shape.

voxels in the same K-D tree layer contain the same number of points. As the point density can be regarded as an approximation of volume density, all such voxels have the same density. Therefore, we directly use K-D voxels to guide the density-adaptive rendering. Specifically, we conduct the rendering process at the voxels of some bottom layers in the K-D tree. For each querying ray, we use the aforementioned recursive procedure to locate the minimal intersected voxels that are deep enough. Here "minimal" means that the ray intersects with the node's voxel, but does not intersect with any sub-node's voxel. These intersected voxels divide the querying ray into several segments (Fig. 3). The segments covered by a voxel are close to the surface and used for rendering, while those not covered are in empty spaces.

Deterministic Spline Integration. DIVeR [41] shows that deterministic integration outperforms stochastic integration in NeRF rendering. So we perform a deterministic integration to obtain the segment's feature within each voxel. Since we do not necessarily have points at the voxel's vertices, the trilinear interpolation used in DIVeR is not feasible here. Instead, we use spline integration. For the i-th intersected voxel in the ray passing order, we uniformly select points in this segment, and integrate their features to obtain the average feature f_i of the segment of the i-th voxel:

$$f_i = \frac{1}{r_i - l_i} \int_{l_i}^{r_i} \textbf{feature}(o + t \cdot d) \mathrm{d}t, \quad (1)$$

where $[l_i, r_i]$ is the intersection interval, and o and d are the source and direction of the querying ray, respectively. This average feature f_i can be interpreted as the feature of *a representative point* p_i located somewhere in the segment.

KNN-Based Feature Aggregation. For each uniformly selected point q in the segment during spline integration, we obtain its feature via weighted interpolation from the features of its K nearest neighbors (KNN) in the point cloud:

$$\textbf{feature}(q) = \sum_{p_j \in \mathrm{KNN}(q;K)} k_j e_j,$$

$$\{k_j\} = \underset{p_j \in \mathrm{KNN}(q;K)}{\mathrm{SoftMax}} \left(\log \gamma_j - \log \|q - p_j\|_2^2 \right), \quad (2)$$

where for each point p_j in the point cloud, we parameterize its confidence γ_j and point feature e_j as in PointNeRF.
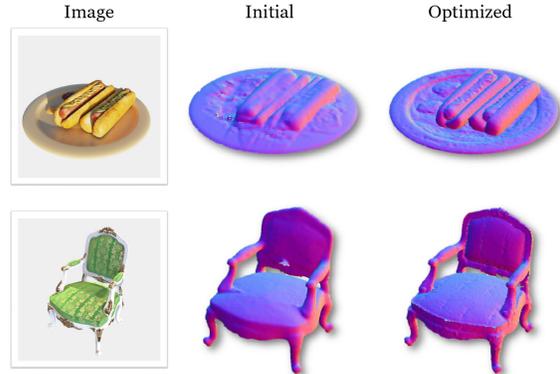
Figure 4. In the two scenes of NeRF Synthetic [27], NeuralEditor optimizes the rough initial point cloud to a precise point cloud. The points are colored with their normal vectors.

Phong Reflection-Based Color Modeling with Point Cloud Normal Vectors. To obtain the volume density σ_i and color c_i of the representative point p_i in the i-th voxel, we use the Phong reflection model [31]. As we have the underlying point cloud, we use Open3D [53] to estimate the normal vector for each point, and integrate these vectors over the interval to get an average normal vector n_i. Such information better characterizes the shape of point clouds (scenes), which plays an important role in Phong-based rendering and also implicitly facilitates optimizing the point clouds (Sec. 3.3). Consistent with RefNeRF [37], we use multiple multilayer perceptrons (MLPs) to model other attributes, including volume density, tint, roughness, and diffuse and specular color, and use the Phong formula to calculate the final c_i with these attributes. Finally, we aggregate c_i of all segments on the ray to obtain the final color:

$$c_{\mathrm{pixel}} = \sum_{i \geq 1} \tau_i \cdot (1 - \exp(-(r_i - l_i)\sigma_i) \cdot c_i,$$

$$\tau_i = \exp\left(-\sum_{i'=1}^{i-1} (r_{i'} - l_{i'})\sigma_{i'} \right). \quad (3)$$

3.3. Point Cloud Optimization

Point Cloud Initialization. To start training, we need a coarse initial point cloud. Consistent with PointNeRF, we

use a point generation network, which consists of a multi-view stereo (MVS) model [12] based on a 3D convolutional neural network (CNN), to generate the points' coordinates and confidence values, and another 2D CNN [46] to generate their initial features. This network was pre-trained on the DTU training dataset [1], and can generalize to other datasets and scenes. As shown in Fig. 4, the initial point cloud generated by such a network is coarse and noisy.

Explicit Optimization via Pruning and Growing. We perform a similar pruning and growing procedure as in PointNeRF, to prune outliers with low confidence γ_j and fill holes in the point clouds. We make several important modifications over PointNeRF, and integrate this procedure with our deterministic integration (details in the supplementary).

Implicit Optimization with Normal Vectors. In addition to the explicit optimization, the point cloud is also optimized implicitly during training through the adjustment of point confidence γ_j. When computing the average normal vector for rendering, we aggregate normal vectors of nearby points weighted with their distance and confidence, where the confidence of noisy or inaccurate points with potentially abnormal normal vectors is adjusted accordingly. Moreover, we apply the normal vector regularization losses from RefNeRF [37] to supervise the points' confidence w.r.t. their normal vectors. These strategies collectively provide *implicit but more tailored* ways to optimize the point clouds. With both explicit and implicit optimization, NeuralEditor obtains very precise point clouds (Fig. 4).

4. Shape Editing with NeuralEditor

Formulation of General Shape Editing Tasks. We define the shape editing tasks based on *indexed* point clouds. To this end, we first re-define an indexed point cloud P as a mapping from a point index j to the corresponding point p_j,

$$P : j \rightarrow p_j, \text{ where } p_j \in \mathbb{R}^3, j = 1, \cdots, |P|. \quad (4)$$

A shape editing task is defined as *another* indexed point cloud $Q(P)$,

$$Q(P) : j \rightarrow q_j, \text{ where } q_j \in \mathbb{R}^3 \cup \{\varnothing\}, j = 1, \cdots, |P|, \quad (5)$$

describing a shape editing task whether the j-th point moves from p_j to q_j or is deleted in the deformation if $q_j = \varnothing$. With this definition, $Q(P)$ can be an *arbitrary* point cloud with points properly matched to points in P by same indices, regardless of connectivity or continuity.

Our formulation represents a broad range of shape editing tasks. The mesh editing tasks in NeRF-Editing [51], Deforming-NeRF [44], and CageNeRF [30] can be more simply and clearly defined here. For example, in NeRF-Editing, a mesh is exported from a general NeRF model, deformed manually, and converted to an "offset" or a continuous space deformation. We can depict such a task without
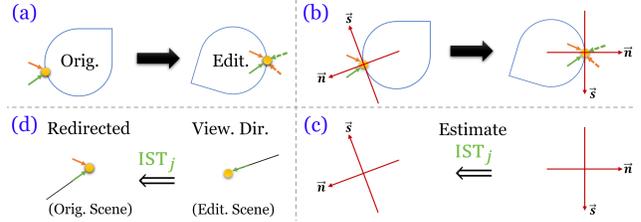
Figure 5. **Infinitesimal surface transformation (IST).** (a) As the view-dependent colors are modeled as absolute viewing directions, they (solid arrows at the right) are different from the correct colors (dashed arrows at the right) after deformation. We solve this by (b) constructing a local coordinate system near the j-th point and (c) modeling IST *from* the edited scene *to* the original scene with the coordinate systems, so as to (d) redirect the viewing direction to the original scene when rendering the edited scene.

"offsets," by recording only the *final location* for *each point* without extra information. Notably, our formulation even models those whose deformation is *not continuous* in the space, *e.g.*, cutting a scene into two parts, and thus cannot be covered and solved by NeRF-Editing, Deforming-NeRF, or CageNeRF, as shown in the supplementary.

Editing Shape by Moving Points. We design our shape editing scheme with NeuralEditor. This is achieved by interpreting NeRF rendering as "plotting" the sampled points over the viewing ray. If we render the scene by naively plotting the point cloud P, the shape editing task can be addressed by replacing each point's coordinate from p_j to q_j. For NeuralEditor, we similarly replace the underlying point cloud from P to $Q(P)$, while maintaining the confidence values and features. This method is general and can also be applied to any point-based NeRF model like PointNeRF.

Correcting View-Dependence with Infinitesimal Surface Transformation (IST). The editing method above can already obtain reasonable results. However, as illustrated in Fig. 5, the modeled view-dependent colors record the *absolute* viewing direction, making them incorrect after deformations that change their orientation.

To solve this issue, we model the infinitesimal surface transformation (IST) for each point to redirect the viewing ray in the correct direction. We construct a local coordinate system for each point to represent the orientation of the infinitesimal surface, using its normal vector and two point indices that are neighbors of the j-th point in both P and $Q(P)$. By comparing these two coordinate systems, we can obtain an affine transformation IST_j for the j-th point to redirect the querying view direction (Fig. 5). This procedure is different from modeling space deformation [30, 44, 51], as we only need to model a simple affine transformation at each point, while those methods model a complicated, continuous, and non-linear deformation in the whole space.

Our proposed method requires a precise point cloud with normal vector-based color modeling. It is thus incompatible with PointNeRF, as PointNeRF is unable to obtain a desired point cloud to estimate the surface normal vectors.

Fine-Tuning on Deformed Scene. Using the shape editing scheme introduced above, we can apply shape deformation on the scene modeled by our NeuralEditor *without any modification* to the model architecture or rendering scheme, which means that the resulting model is still a valid, fully functional NeuralEditor. Therefore, we can further fine-tune NeuralEditor on the deformed scene if the ground truth is available. We can even fine-tune the infinitesimal surface transformation with other parameters, to rapidly adjust toward better ambient consistency. This makes NeuralEditor desirable *in practice*, since in most applications, the final goal is not a zero-shot inference, but to *fit* the deformed scene well with reduced cost. By supporting fine-tuning, our NeuralEditor aligns well with and achieves this goal.

As another point-based NeRF model, PointNeRF supports fine-tuning but cannot leverage infinitesimal surface transformation fine-tuning to further optimize the performance. On the other hand, mesh-based NeRF editing models [30, 44, 51] do not support fine-tuning well: With deforming the space instead of the scene, these models' rendering scheme has highly changed. In their rendering process, a ray may go through a long, irregular way to reach the scene's surface. As the modeled space deformation might not be precise, it could be hard to tune the irregular space well, and even hurt other parts of the trained NeRF model. Such issues occur especially for some spaces with non-uniform density, since most of their model components (*e.g.*, positional encodings, voxels) are not designed to deal with non-uniform spaces. Among all these methods, only our NeuralEditor has complete support for fine-tuning.

5. Experiment

Point Cloud Generation. The underlying point cloud is fundamental to all editing tasks. Fig. 6 first provides a qualitative comparison of point clouds generated by our NeuralEditor and PointNeRF [43] on NeRF Synthetic [27]. Ours are much more precise with sharper details, *e.g.*, the mayonnaise on the Hotdog's sausage, the uneven texture on the Chair's cushion, the edge of the Mic's stand, and the Lego brick's studs. By contrast, PointNeRF's point clouds are blurred and noisy, lose most of the details, and even contain obvious shape deflects on the Hotdog's plate and Chair's backrest. This shows that while the point cloud generation task is challenging, NeuralEditor generates a superprecise point cloud which is crucial for shape editing tasks.

Experimental Settings. We mainly conduct experiments based on scenes from the NeRF Synthetic (NS) dataset. NS is a widely-used NeRF benchmark constructed from Blender [7] scenes. Due to the lack of publicly available benchmarks for shape editing, we use Blender to construct a reproducible benchmark, including the ground truth of edited scenes for evaluation and fine-tuning. Our shape editing tasks cover *all eight scenes* in NS, while prior
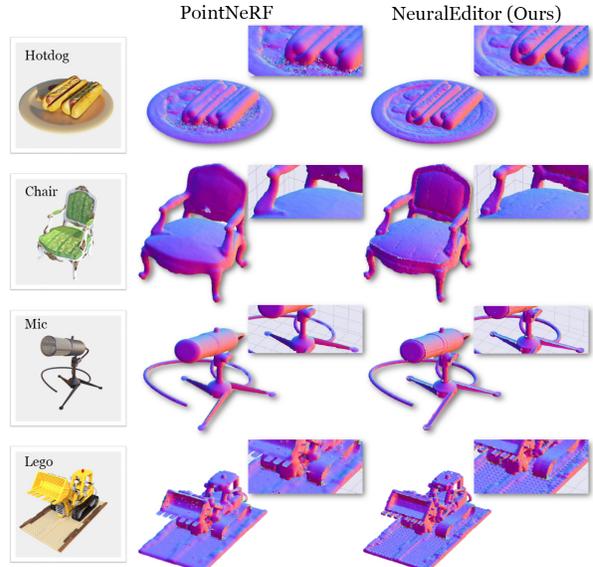
Figure 6. NeuralEditor generates much more precise point clouds than PointNeRF [43] in the four scenes of NeRF Synthetic [27]. The points are colored with their normal vectors.

Figure 7. With too coarse cages, DeformingNeRF [44] is unable to perform the deformation faithfully, leading to poor results.

work [30, 44, 51] only picks a few scenes. The provided images for NS scenes are with opacity, and there is no requirement for the background color. We evaluate and visualize the results on a black background, for better contrast and clearer detail visualization. In the supplementary, we show the results on a white background with same conclusions.

Shape Editing Tasks. We evaluate our model on two types of shape editing tasks, as shown in Fig. 1:

(I) Shape (Mesh) Deformation Task. We consider the shape deformation task as in [30, 44, 51]: deform the shape of a scene in a human-guided way. To construct our deformation tasks from NS and obtain the ground truth, we apply the shape deformation simultaneously to the scene and our point cloud within the provided Blender file. We perform both zero-shot inference and fine-tuning, and compare our rendering results with the ground truth. As domenstrated in Figs. 7 and 8, our deformation tasks are much more precise and *aggressive*, compared with those in previous work [30, 44, 51]. In the supplementary, we also design a non-continuous deformation task and deformation tasks on the real-world dataset Tanks and Temples [17] (with zero-shot inference only, as there is no ground truth available).

(II) Scene Morphing Task. We address a more challenging shape editing task that has *not* been investigated in prior NeRF editing work: the scene morphing task. Given two scenes A and B, we should construct a path to gradually

Model	Zero-Shot Inference, PSNR ↑								Fine-Tune for 1 Epoch, PSNR ↑							
	Chair	Hotdog	Lego	Drums	Ficus	Materials	Mic	Ship	Chair	Hotdog	Lego	Drums	Ficus	Materials	Mic	Ship
DeformingNeRF [44]	18.84	-	13.10	-	-	-	-	-	-	-	-	-	-	-	-	-
PointNeRF [43]	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**

Table 1. NeuralEditor *significantly and consistently* outperforms PointNeRF and Naive Plotting on all deformed scenes of NeRF Synthetic in peak signal-to-noise ratio (PSNR), 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. Comparison results under other metrics are in the supplementary.
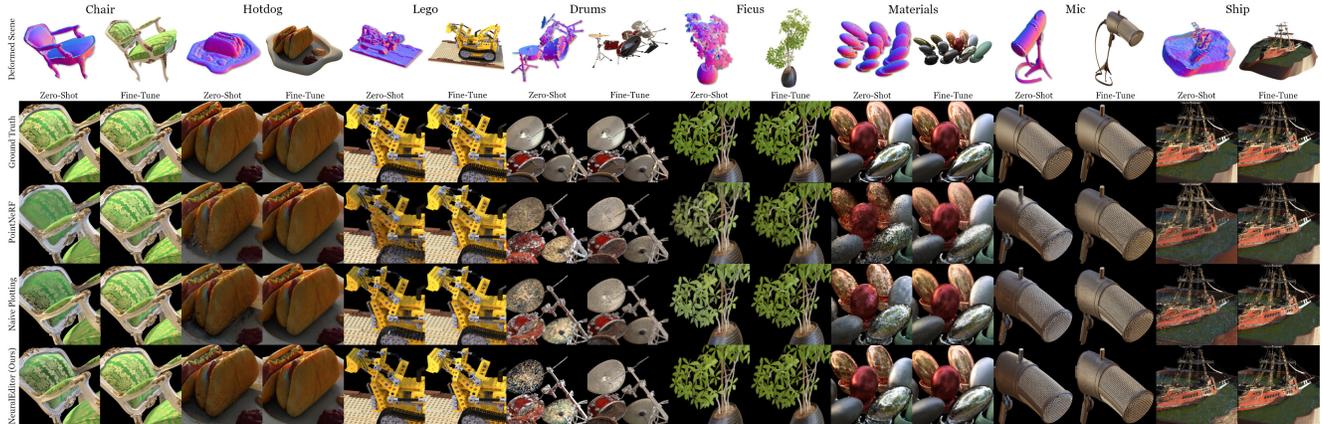
Figure 8. NeuralEditor produces superior rendering results to PointNeRF, with significantly fewer artifacts in zero-shot inference. Fine-tuning further improves the *consistency of rendering with the ambient environment*. We use a black background for better visualization.

change one scene to the other, and the intermediate scenes should have reasonable appearances. We are required to render all intermediate scenes. For this task, we use the point cloud diffusion model [24] to generate intermediate point clouds with *latent space interpolation* between A and B as in [24], and we introduce a K-D tree-based [4] matching algorithm to match the adjacent points to fix the indices of the intermediate scenes. To render an intermediate scene, we apply the shape transformation to the NeuralEditor models trained for scenes A and B, and then interpolate the rendering features to obtain the features of the intermediate scene for rendering.

NeuralEditor Variants. (1) Full NeuralEditor: Our complete model with all components. (2) NeuralEditor *without* infinitesimal surface transformation (IST): We remove the maintenance and optimization of infinitesimal surface transformation in scene editing. This key variant of NeuralEditor enables us to evaluate the importance of IST as well as other components of our model. The *full ablation study* of NeuralEditor is in the supplementary.

Baselines. We compare NeuralEditor against different types of baselines as follows. (1) Naive Plotting: We use the point cloud generated by NeuralEditor, computing each point's opacity and view-dependent colors with their point features. We render the scene by directly plotting/projecting the point cloud to the camera plane. (2) PointNeRF [43]: For the shape deformation task, we apply the same deformation to the point clouds generated by PointNeRF. For the scene morphing task, we apply the matching algorithm to

the point clouds generated by PointNeRF and the same intermediate point clouds generated by the point cloud diffusion model [24] for fairness. (3) DeformingNeRF [44]: DeformingNeRF is not compatible with the scene morphing task. For the shape deformation task, we perform the same deformation on vertices of given cages. Note that DeformingNeRF only released trained models for Lego and Chair, so we can only evaluate it on these two scenes. While other models [30, 51] support NeRF-based shape deformation via cages or exported meshes, we were unable to use them as baselines – they did not provide executable code nor their deformed scenes for us to evaluate on their tasks.

Shape (Mesh) Deformation Results. The qualitative comparison is shown in Fig. 8. Both PointNeRF and Naive Plotting have many artifacts, like blur, wrong color, black or white shadows, noise, *etc.*, whereas our powerful NeuralEditor produces clean and realistic rendering results. After fine-tuning, NeuralEditor shows a significant improvement with better rendering results than PointNeRF, indicating that NeuralEditor is able to achieve higher consistency with the surrounding ambient environment. Notably, in the Materials scene (the 6th scene from left), only our NeuralEditor generates reasonable reflection, while both baselines show blurry and visually messy results. Also in the Drums scene, the bottom face of the gong is not visible in any of the training views in the original scene, so all models render poor results in zero-shot inference. However, after only fast fine-tuning, NeuralEditor is able to precisely model the previously unknown surface and generate a sub-
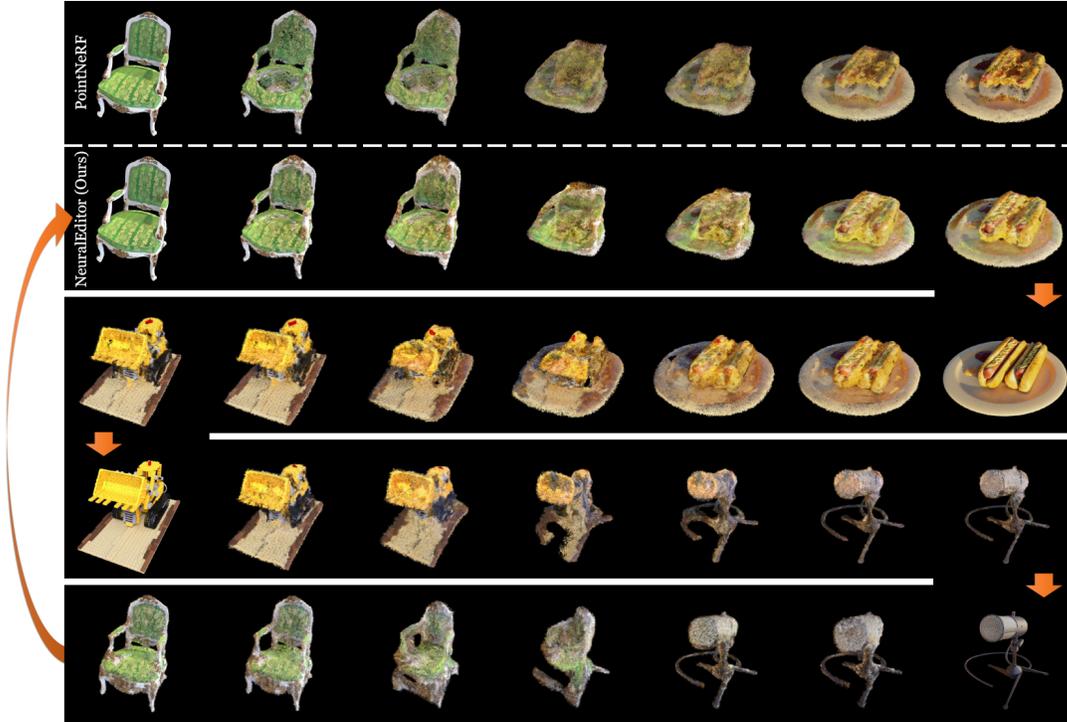
Figure 9. Our NeuralEditor produces *smooth morphing* results between Chair, Hotdog, Lego, and Mic in the NeRF Synthetic dataset, while PointNeRF produces results with blurry textures, black shadows, and gloomy, non-smooth colors. The rendering results in the looped morphing process are arranged in the shape of the numerical digit "3," indicated by the dividing lines and arrows.

stantially better result than PointNeRF, highlighting NeuralEditor's strength in fast-fitting. All these results demonstrate that NeuralEditor can handle various visual effects and make them consistent in the deformed scene. We provide the figure with higher resolution in the supplementary.

The quantitative comparison is summarized in Table 1. We observe that: (1) Our NeuralEditor consistently outperforms all the baselines and variants for both zero-shot inference and fine-tuning settings. (2) With the precise point cloud generated by NeuralEditor, the Naive Plotting baseline even consistently outperforms PointNeRF. (3) Our 'w/o IST' variant has a comparable performance to Naive Plotting with the same point cloud and features in the zero-shot inference setting, but after fine-tuning its performance is significantly higher than Naive Plotting, validating the capability of NeuralEditor in NeRF modeling.

Notably, DeformingNeRF [44] performs poorly in our benchmark with significantly lower metric values. As shown in Fig. 7, the cages provided by DeformingNeRF are too coarse, and cannot even cover the whole scene. Therefore, DeformingNeRF cannot faithfully perform the precise deformation in our benchmark, leading to poor rendering results. On the contrary, both PointNeRF and our NeuralEditor at least faithfully perform the deformation, showing that point cloud is necessary for precise shape editing.

Scene Morphing Results. The morphing results between 4 NeRF Synthetic scenes are shown in Fig. 9. The morphing process starts from Chair, morphs to Hotdog, Lego, Mic, and at last turns back to Chair. NeuralEditor produces smooth rendering results on the point cloud diffusion-guided [24] intermediate scenes, mixing the textures of the two scenes in a reasonable way. In comparison, the rendering results produced by PointNeRF are unsatisfactory, with blurry textures, black shadows, and gloomy, non-smooth colors. These results show that our NeuralEditor can render challenging intermediate morphing scenes and achieve decent results *with only the input of moved points*.

6. Conclusion

This paper proposes NeuralEditor, a point cloud-guided NeRF model that supports general shape editing tasks by manipulating the underlying point clouds. Empirical evaluation shows NeuralEditor to produce rendering results of much higher quality than baselines in a zero-shot inference manner, further significantly improving after fast fine-tuning. NeuralEditor even supports smooth scene morphing between multiple scenes, which is difficult for prior work. We hope that our work can inspire more research on point cloud-guided NeRFs and 3D shape and scene editing tasks.

References

[1] Henrik Aanæs, Rasmus Ramsbøl Jensen, George Vogiatzis, Engin Tola, and Anders Bjorholm Dahl. Large-scale data for multiple-view stereopsis. *International Journal of Computer Vision*, 120(2):153–168, 2016. 5

[2] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas J. Guibas. Learning representations and generative models for 3D point clouds. In *ICML*, 2018. 2

[3] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, 2021. 1, 2

[4] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. 2, 3, 7

[5] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. MVSNeRF: Fast generalizable radiance field reconstruction from multi-view stereo. In *ICCV*, 2021. 2

[6] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *CVPR*, 2019. 2

[7] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 6

[8] Frank Dellaert and Lin Yen-Chen. Neural volume rendering: NeRF and beyond. *arXiv:2101.05204*, 2021. 2

[9] Kyle Gao, Yina Gao, Hongjie He, Denning Lu, Linlin Xu, and Jonathan Li. NeRF: Neural radiance field in 3D vision, a comprehensive review. *arXiv:2101.05204*, 2021. 2

[10] Yuan-Chen Guo, Di Kang, Linchao Bao, Yu He, and Song-Hai Zhang. NeRFReN: Neural radiance fields with reflections. In *CVPR*, 2022. 2

[11] Peter Hedman, Tobias Ritschel, George Drettakis, and Gabriel Brostow. Scalable inside-out image-based rendering. *ACM Trans. Graph.*, 35(6):231:1–231:11, 2016. 2

[12] Po-Han Huang, Kevin Matzen, Johannes Kopf, Narendra Ahuja, and Jia-Bin Huang. DeepMVS: Learning multi-view stereopsis. In *CVPR*, 2018. 5

[13] Alec Jacobson, Ilya Baran, Ladislav Kavan, Jovan Popović, and Olga Sorkine. Fast automatic skinning transformations. *ACM Trans. Graph.*, 31(4), 2012. 3

[14] Tomas Jakab, Richard Tucker, Ameesh Makadia, Jiajun Wu, Noah Snavely, and Angjoo Kanazawa. Keypointdeformer: Unsupervised 3D keypoint discovery for shape control. In *CVPR*, 2021. 3

[15] Mengqi Ji, Juergen Gall, Haitian Zheng, Yebin Liu, and Lu Fang. Surfacenet: An end-to-end 3D neural network for multiview stereopsis. In *ICCV*, 2017. 2

[16] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.*, 24(3):561–566, 2005. 3

[17] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and Temples: Benchmarking large-scale scene reconstruction. *ACM Trans. Graph.*, 36(4), 2017. 6

[18] Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing NeRF for editing via feature field distillation. In *NeurIPS*, 2022. 2

[19] Marc Levoy and Pat Hanrahan. Light field rendering. In *SIGGRAPH*, 1996. 2

[20] Fayao Liu, Chunhua Shen, Guosheng Lin, and Ian Reid. Learning depth from single monocular images using deep convolutional neural fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):2024–2039, 2015. 2

[21] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. In *NeurIPS*, 2020. 2

[22] Steven Liu, Xiuming Zhang, Zhoutong Zhang, Richard Zhang, Junyan Zhu, and Bryan C. Russell. Editing conditional radiance fields. In *ICCV*, 2021. 2

[23] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH*, 1987. 3

[24] Shitong Luo and Wei Hu. Diffusion probabilistic models for 3D point cloud generation. In *CVPR*, 2021. 2, 7, 8

[25] Bruce Merry, Patrick Marais, and James Gain. Animation space: A truly linear framework for character animation. *ACM Trans. Graph.*, 25(4):1400–1423, 2006. 3

[26] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *CVPR*, 2019. 2

[27] 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, 4, 6, 8

[28] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3D representations without 3D supervision. In *CVPR*, 2020. 2

[29] Jesús Nieto and Toni Susin. Cage based deformations: A survey. *Lecture Notes in Computational Vision and Biomechanics*, 7:75–99, 2013. 3

[30] 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. 2, 3, 5, 6, 7

[31] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, 1975. 2, 4

[32] Yi-Ling Qiao, Alexander Gao, and Ming C. Lin. NeuPhysics: Editable neural geometry and physics from monocular videos. In *NeurIPS*, 2022. 2

[33] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 1, 2

[34] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3D. *ACM Trans. Graph.*, 25(3):835–846, 2006. 2

[35] Jiaxiang Tang, Xiaokang Chen, Jingbo Wang, and Gang Zeng. Compressible-composable NeRF via rank-residual decomposition. In *NeurIPS*, 2022. 2

[36] Jean-Marc Thiery, Julien Tierny, and Tamy Boubekeur. Jacobians and hessians of mean value coordinates for closed triangular meshes. *The Visual Computer*, 30(9):981–995, 2014. 3

[37] 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. 1, 2, 4, 5

[38] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2Mesh: Generating 3D mesh models from single RGB images. In *ECCV*, 2018. 2

[39] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. IBRNet: Learning multi-view image-based rendering. In *CVPR*, 2021. 1, 2

[40] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. NeX: Real-time view synthesis with neural basis expansion. In *CVPR*, 2021. 2

[41] Liwen Wu, Jae Yong Lee, Anand Bhattad, Yu-Xiong Wang, and David Forsyth. DIVeR: Real-time and accurate neural radiance fields with deterministic integration for volume rendering. In *CVPR*, 2022. 2, 4

[42] Dejia Xu, Peihao Wang, Yifan Jiang, Zhiwen Fan, and Zhangyang Wang. Signal processing for implicit neural representations. In *NeurIPS*, 2022. 2

[43] 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. 2, 3, 6, 7

[44] Tianhan Xu and Tatsuya Harada. Deforming radiance fields with cages. In *ECCV*, 2022. 2, 3, 5, 6, 7, 8

[45] 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 *ICCV*, 2021. 2

[46] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. MVSNet: Depth inference for unstructured multi-view stereo. In *ECCV*, 2018. 5

[47] Wang Yifan, Noam Aigerman, Vladimir G Kim, Siddhartha Chaudhuri, and Olga Sorkine-Hornung. Neural cages for detail-preserving 3D deformations. In *CVPR*, 2020. 3

[48] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021. 1, 2

[49] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021. 2

[50] Yu-Jie Yuan, Yu-Kun Lai, Tong Wu, Lin Gao, and Ligang Liu. A revisit of shape editing techniques: From the geometric to the neural viewpoint. *Journal of Computer Science and Technology*, 36(3):520–554, 2021. 3

[51] 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. 2, 3, 5, 6, 7

[52] Yuzhe Zhang, Jianmin Zheng, and Yiyu Cai. Proxy-driven free-form deformation by topology-adjustable control lattice. *Computers & Graphics*, 89:167–177, 2020. 3

[53] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. 4