NeRFshop: Interactive Editing of Neural Radiance Fields

CLÉMENT JAMBON, Inria & Université Côte d'Azur and École Polytechnique, France
BERNHARD KERBL, Inria & Université Côte d'Azur, France
GEORGIOS KOPANAS, Inria & Université Côte d'Azur, France
STAVROS DIOLATZIS, Inria & Université Côte d'Azur, France
THOMAS LEIMKÜHLER, Max-Planck-Institut für Informatik, Germany
GEORGE DRETTAKIS, Inria & Université Côte d'Azur, France

Fig. 1. NeRFshop enables intuitive selection via scribbles and interactive editing of arbitrary NeRF scenes. We show duplicative, affine, and non-affine edits (left-to-right) with different viewpoints in the KITCHEN scene.

Neural Radiance Fields (NeRFs) have revolutionized novel view synthesis for captured scenes, with recent methods allowing interactive free-viewpoint navigation and fast training for scene reconstruction. However, the implicit representations used by these methods—often including neural networks and complex encodings—make them difficult to edit. Some initial methods have been proposed, but they suffer from limited editing capabilities and/or from a lack of interactivity, and are thus unsuitable for interactive editing of captured scenes. We tackle both limitations and introduce NeRFshop, a novel end-to-end method that allows users to interactively select and deform objects through cage-based transformations. NeRFshop provides fine scribble-based user control for the selection of regions or objects to edit, semi-automatic cage creation, and interactive volumetric manipulation of scene content thanks to our GPU-friendly two-level interpolation scheme. Further, we introduce a preliminary approach that reduces potential resulting artifacts of these transformations with a volumetric membrane interpolation technique inspired by Poisson image editing and provide a process that "distills" the edits into a standalone NeRF representation.

ACM Reference Format:
Clément Jambon, Bernhard Kerbl, Georgios Kopanas, Stavros Diolatzis, Thomas Leimkühler, and George Drettakis. 2023. NeRFshop: Interactive Editing of Neural Radiance Fields. *Proc. ACM Comput. Graph. Interact. Tech.* 6, 1 (May 2023), 21 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' addresses: Clément Jambon, Inria & Université Côte d'Azur and École Polytechnique, France, clement.jambon@polytechnique.edu; Bernhard Kerbl, Inria & Université Côte d'Azur, France, bernhard.kerbl@inria.fr; Georgios Kopanas, Inria & Université Côte d'Azur, France, georgios.kopanas@inria.fr; Stavros Diolatzis, Inria & Université Côte d'Azur, France, diolatzis@gmail.com; Thomas Leimkühler, Max-Planck-Institut für Informatik, Germany, thomas.leimkuehler@mpi-inf.mpg.de; George Drettakis, Inria & Université Côte d'Azur, France, george.drettakis@inria.fr.

1 INTRODUCTION

Neural Radiance Fields (NeRFs) [Mildenhall et al. 2020; Tewari et al. 2020] have completely revolutionized novel-view synthesis of scenes captured with multi-view photos. The most recent methods [Müller et al. 2022; Sara Fridovich-Keil and Alex Yu et al. 2022] achieve fast training of the neural representation and allow interactive rendering of the captured scene with impressive levels of realism. Internally, NeRF uses a *volumetric* representation of density and view-dependent color to encode shape and appearance, parameterized using a multi-layer perceptron (MLP) neural network; as such it is hard to directly manipulate the representation, e.g., to edit the captured scene. Some initial editing methods have been recently proposed (e.g., [Xu and Harada 2022; Yuan et al. 2022], Sec. 2.3) but they come with several limitations that hinder interactivity. Our goal is to develop a NeRF editing solution that is interactive, can work on full scenes rather than isolated "masked" objects, and allows free-form direct manipulation of the volumetric representation. To our knowledge, no previous method satisfies all of these requirements.

To achieve our goals, we draw inspiration from standard image editing tools, as well as specialized approaches like PointShop [Zwicker et al. 2002] which provide intuitive, direct-manipulation interfaces for manipulation of 2D and 3D content respectively. Specifically, we introduce a scribble-based interface that allows the user to easily select a region of space, akin e.g., to image region selection. Our method creates a *cage* mesh that encloses this space [Peng et al. [n. d.]; Xu and Harada 2022] and allows the user to manipulate the mesh vertices, enabling free-form deformation and editing, similar to other content editing tools (see Figure 1). By creating a tetrahedral representation [Garbin et al. 2022], we introduce a GPU-friendly two-level interpolation scheme, allowing interactive updates for editing. Editing complete scenes introduces specific challenges since inconsistent color and density artifacts can appear after (re)moving an object. We introduce a preliminary membrane-based interpolation approach inspired by Poisson Image Editing [Pérez et al. 2003] that reduces these artifacts. Finally, similar to "layer flattening/export" in traditional image-editing tools, we provide a "distillation" step, that collapses all edits performed and saves a hashgrid NeRF representation that can be loaded in established pipelines.

NeRFshop is the first complete method that allows *interactive* free-form editing of full NeRF scenes, opening up vast, unexplored possibilities for creative usage of captured 3D scenes. In summary, our main contributions are:

- A scribble-based interface for *interactive* object selection in NeRFs and semi-automatic cage building.
- A direct free-form volumetric manipulation approach that is interactive thanks to a GPU-friendly interpolation scheme.
- A preliminary membrane-based correction method that reduces color and density artifacts resulting from user edits, and a way to "distill" the edits into a standard NeRF representation.

Our method allows free-form edits (deformation, translation/scale/rotation, and duplication of objects), including selective removal "floaters" that are a typical artifact of NeRF reconstruction (please see supplemental video for example editing sessions). We have implemented our method in an interactive system built on top of Instant Neural Graphics Primitives [Müller et al. 2022]; we will release our source code and datasets on publication. We hope our intuitive interactive system will allow artists and researchers to experiment with creative editing and manipulation of captured scenes.

After discussing related work in the following section, in Sec. 3 we present our scribble-based selection interface that projects user annotations into the density volume, allowing us to create a cage for direct user manipulation. In Sec. 4 we describe how we tetrahedralize the cage and our design of a two-level interpolation scheme that enables interactive volumetric editing. Sec. 5

describes our preliminary membrane-based approach to reduce color and density artifacts that result from our various editing operations.

2 BACKGROUND AND RELATED WORK

Our free-form editing method builds on NeRFs [Mildenhall et al. 2020], and in particular on an interactive version based on multi-resolution hash encodings [Müller et al. 2022]. Further, we use cages [Nieto and Susín 2013] with an interpolation scheme based on Mean Value Coordinates [Ju et al. 2005] for free-form editing. We first present the relevant background for these two particular areas and then briefly review other related work. Since there has been an explosion in the volume of NeRF literature in the last few years, we refer to the excellent surveys that have recently been published for a comprehensive review [Tewari et al. 2020; Xie et al. 2022], and only discuss work that is directly related to our method.

2.1 Neural Radiance Fields and Acceleration Methods

NeRF [Mildenhall et al. 2020] represents a scene as a function f_θ with trainable parameters θ, mapping a 3D position $\mathbf{x}$ and a 3D direction $\mathbf{d}$ to a corresponding RGB color $\mathbf{c}$ and scalar density σ. Images are rendered using volumetric ray-marching [Max 1995] along a ray $\mathbf{r}$:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i \quad \text{with} \quad T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right) \tag{1}$$

where samples are taken along the ray $\mathbf{r}$ with intervals δ_i.

Initially, NeRF relied on a single large MLP to encode each scene, resulting in slow training and rendering due to numerous network evaluations per ray [Mildenhall et al. 2020]. Hybrid representations leverage spatial locality, dramatically improving training speeds, typically relying on spatial data structures to store local features: sparse voxel octrees [Sara Fridovich-Keil and Alex Yu et al. 2022; Takikawa et al. 2021], tri-planar grids [Chan et al. 2022], codebooks [Takikawa et al. 2022], hash tables [Müller et al. 2022], low-rank tensor grids [Chen et al. 2022], or point clouds [Xu et al. 2022]. These features are then aggregated locally and decoded by a relatively shallow MLP (if required).

We build our method on Instant-NGP [Müller et al. 2022], which is fully GPU-accelerated, allowing for truly interactive navigation. This leads to several constraints that we need to respect in our interactive editing solution. During ray-marching, Instant-NGP alternates between traversal, inference, and ray compaction; when rays no longer contribute meaningful samples, they are dropped and the remaining rays are reordered. The (fixed) inference workload is distributed according to the number of active rays to maintain uniform GPU occupancy. A key component of Instant-NGP is a trainable sparse occupancy grid that is queried directly from the CUDA kernels performing SIMT (Single Instruction, Multiple Threads) operations. Thus, memory coherency and minimal throughput must be enforced across ray compactions. To do so, it uses fast CUDA (micro-)kernels, maximizing memory coherency. Our solution can easily be integrated with this procedure without disrupting the original design, by introducing intermediate kernel routines that remap samples according to user-defined edits before inference.

2.2 Cage Building and Geometry Processing

Cage-based deformations are a popular and versatile tool for shape editing [Nieto and Susín 2013]. The basic principle is to build a bounding mesh—the cage—around the object, enabling the user to modify its vertices and propagate the geometric edits into the cage volume. Let $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\} \in \mathbb{R}^3$ be the original positions of the cage vertices. To displace a point $\mathbf{x}$, cage-based deformations rely

on pre-computing generalized barycentric coordinates [Floater 2015] $\{\lambda_1, \ldots, \lambda_n\} \in \mathbb{R}$ w.r.t. the canonical positions of the vertices of the cage i.e.,

$$\mathbf{x} = \sum_{i=1}^{n} \lambda_i \mathbf{v}_i \quad \text{where} \quad \sum_{i=1}^{n} \lambda_i = 1 \text{ and } \lambda_i \geq 0 \tag{2}$$

and deriving its new position $\hat{\mathbf{x}}$ as a λ-weighted sum of the displaced cage vertices $\hat{\mathbf{v}}_1, \ldots, \hat{\mathbf{v}}_n$:

$$\hat{\mathbf{x}} = \sum_{i=1}^{n} \lambda_i \hat{\mathbf{v}}_i. \tag{3}$$

This is commonly done using generalized barycentric coordinates, which come in different flavors; we choose to use Mean Value Coordinates (MVC) [Hormann and Floater 2006] for our edits due to their efficiency and simplicity.

Cage-based deformations are well suited for NeRF editing since they are a natural fit for the manipulation of volumetric data, but creating the cages by hand is a tedious and time-consuming process. Automatic cage-building algorithms address this issue [Xian et al. 2009] but often result in complex meshes, leading us to choose a more adaptive solution inspired by Bounding Proxies [Calderon and Boubekeur 2017], which also employ an adaptive cage mesh simplification scheme.

2.3 Editing NeRFs

There have been several attempts to edit the shape and appearance of NeRFs; these are typically specific, hardly interactive and/or apply only to synthetic scenes, making them unsuitable for our goals. Nonetheless, we have been inspired by many of the underlying ideas, which we discuss next.

Editing via learned decompositions. One stream of research has investigated NeRF variants that allow rigid transformations of individual objects using learned decompositions of a scene [Bing et al. 2022; Mirzaei et al. 2022; Stelzner et al. 2022; Yang et al. 2021a]. These methods typically require additional inputs such as segmentation masks [Bing et al. 2022; Yang et al. 2021b], depth [Stelzner et al. 2022], and/or require additional auxiliary networks [Bing et al. 2022; Mirzaei et al. 2022]. In contrast to our approach, these methods do not allow flexible non-rigid editing and fall short of interactivity.

Editable spatial encodings. Scene editing can be realized by employing suitable local encodings of the positional input to the NeRF. Commonly, regular grids [Lazova et al. 2022] or adaptive tree-like structures [Liu et al. 2020] are used, which allow coarse-level editing. We overcome this costly and limiting discretization and perform flexible and fine-grained continuous edits.

Reconstruction of deforming scenes. In the context of deformable scene reconstructions, non-rigid NeRFs [Kania et al. 2022; Luo et al. 2022; Park et al. 2021a,b; Pumarola et al. 2020; Tretschk et al. 2021; Weng et al. 2022] have been developed. Typically, a deformation network is used to regress displacements from a canonical configuration. Depending on the parameterization used, this provides some control over the deformation, but only within the distribution given by the input data. In contrast, our approach allows full free-form edits of static scenes.

Conditional generative approaches [Jang and Agapito 2021; Kosiorek et al. 2021; Liu et al. 2021; Wang et al. 2022] leverage large datasets to learn category-level or scene-level NeRF distributions, which are then "editable" via their latent codes. The context is however very different from our case of interactive fine-grain editing of a single captured scene.

Proxy-based editing approaches are most closely related to our approach. Here, explicit geometric proxies are extracted from the volumetric NeRF representation, allowing editing with conventional

methods. NeRF-Editing [Yuan et al. 2022] extracts an explicit simplified surface mesh from a trained NeRF, which is further turned into a tetrahedral mesh. Similar to our method, mesh edits can be mapped to the tetrahedra, allowing ray-bending; however, their surface-based reconstruction can be problematic for intricate geometries. Deforming-NeRF [Xu and Harada 2022] uses automatic cage-building, while displacements are propagated using Mean Value Coordinates (MVC) [Ju et al. 2005]. While we also use cages and MVC, Deforming-NeRF does not use a tetrahedral mesh but stores MVC coordinates on a regular lattice to avoid computing such coordinates at each sampling point; this may cause discretization artifacts. In contrast, our two-level interpolation scheme and adaptive tetrahedral lookup table allow for interactive direct volumetric deformations. VolTeMorph [Garbin et al. 2022] also uses a tetrahedral data structure for ray-bending, demonstrated for pre-processed physically-based animations and face control. To perform point-tetrahedron lookups, VolTeMorph uses a GPU-based ray-acceleration approach based on BVH traversal in contrast to our method, which is grid-based. Further, their selection and cage-building process is complex, requiring external tools and refinement, while our integrated approach handles the entire pipeline for arbitrary full scenes.

We next discuss two recently accepted methods, concurrent with ours. CageNeRF [Peng et al. [n. d.]] builds a cage by first extracting a fine-scale mesh and then optimizing for a coarser cage. Contrary to our two-level interpolation scheme, they explicitly use the deformed fine-scale mesh to perform a backward re-mapping. The method has only been demonstrated on isolated synthetic objects. CLA-NeRF [Tseng et al. 2022] trains an extra segmentation feature in addition to color and density on a collection of articulated objects and uses it to derive simple joints, allowing deformations. While they allow for free-form interactive editing, only category-level edits can be performed.

2.4 Selecting and Deforming Objects and Seamless Edits

We next briefly discuss related literature on object selection and deformation. For a more extensive introduction, please refer to the recent survey [Yuan et al. 2021]. Whenever edits are performed on a NeRF, inconsistencies at surfaces arise. We diminish these artifacts by introducing a solution that is inspired by gradient-domain editing which we briefly review below.

Selection. Object selection in photos has been explored for decades (see Szeliski [Szeliski 2022], chap. 6). In the context of NeRF, recent approaches inject semantic priors into the model. This is achieved either by utilizing pre-trained segmentation models [Fu et al. 2022; Kundu et al. 2022; Zhi et al. 2021], which are limited to predefined object categories, or by relying on general feature fields [Caron et al. 2021; Radford et al. 2021], as done in Distill-NeRF [Kobayashi et al. 2022].

Gradient-domain Editing is a standard technique used for seamless cloning, panorama stitching, etc. in the context of image and video editing [Bhat et al. 2010; Levin et al. 2004; McCann and Pollard 2008; Pérez et al. 2003]. Finite differences are used to approximate image gradients that can be edited and/or constrained, and a new image is produced by solving a Poisson equation. To avoid the expensive solver step, several methods have been proposed; in this work, we build on Instant Image Cloning [Farbman et al. 2009] that interprets the Poisson equation as a membrane interpolation scheme based on Mean Value Coordinates.

3 SELECTION AND SEMI-AUTOMATIC CAGE BUILDING

We introduce a scribble-based interface allowing the user to select objects—or a volumetric region represented as voxels—in a scene that will be edited (Fig. 2a,b). The user is then free to extend the selection (Fig. 2c), before it is turned into a cage (Fig. 2d) that can be manipulated. The cage is used to create a tetrahedral structure (Fig. 2e), which steers the volumetric deformation process.

Fig. 2. The user scribbles on the target object or volumetric region in space (a). NeRFshop then reprojects the scribbles into the 3D volume (b). We perform region growing (c) and build a cage (d). Finally, we discretize the volume using a tetrahedral mesh (e).

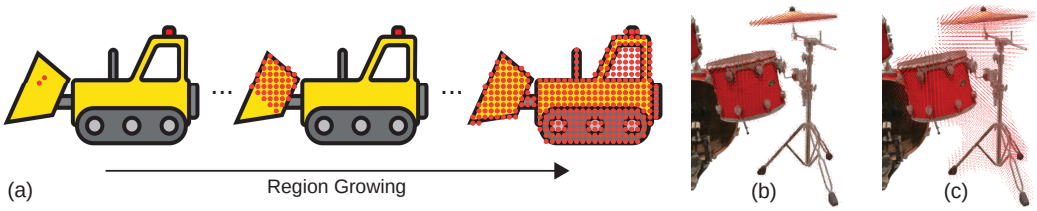


Fig. 3. (a) Starting from seed points (red dots in first image), our density-aware region growing scheme proceeds iteratively by considering direct neighbors of each valid cell. (b) The final selection contains all densities the user wants to modify. However, the resulting volume may be too fine for robust tetrahedralization and editing. (c) We perform morphological operations to simplify the shape of the selected volume.

We use multiple 3D grids to store discretized values, following the same structure as the *occupancy grids* of *Instant-NGP* [Müller et al. 2022], which uses the grid as a powerful acceleration structure to reduce the number of aggregated samples along each ray via empty space skipping. We next describe how we can exploit and extend these grid structures for the purpose of intuitive object selection.

3.1 Selection

To select a region in the NeRF scene to be manipulated, the user scribbles over that region in the current rendered view (see also supplementary video). The scribbles are then reprojected from screen-space to the underlying volume by ray-marching, using a process akin to Eq. 1. Ray-marching continues until the accumulated transmittance drops below a user-defined threshold, which we set to 10^{-3} in all our experiments except for floater removal where we set it to 0.8. The reprojected points (i.e., locations where traversal terminates) are mapped to the nearest voxel in a binary occupancy grid covering the scene. We refer to these selected cells as *seeds*.

Next, we can extend the initial, sparse selection using a region-growing procedure similar to flood filling: When invoked, we fill a queue with the cell indices of the seeds and then proceed iteratively by adding all direct neighbors of the current selection to the queue and flagging each cell based on a density threshold (Fig. 3a). The user is free to decide when to stop the expansion by performing multiple, adjustable growing steps consecutively. To allow for fine-grained control, NeRFshop provides tools to discard cells in case the user is not satisfied with the current selection.

3.2 Cage Building

With the selected grid cells ready, we now construct a cage mesh that acts as the geometric editing interface for the user. We require two properties: The cage should have a single connected component, and it should bound the selected region while staying close to the selected voxels.

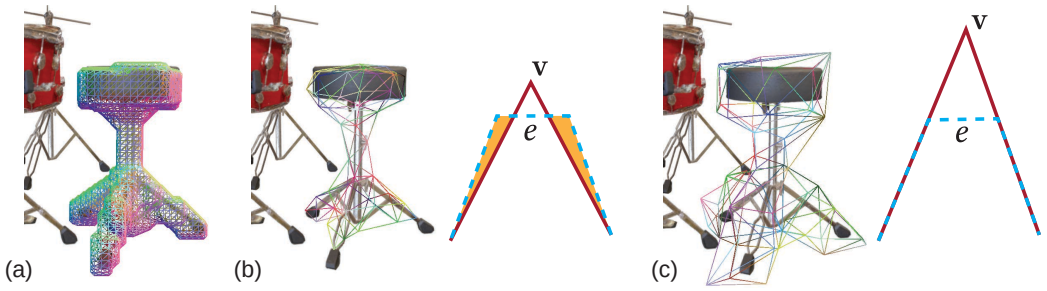


Fig. 4. (a) Original cage mesh resulting from applying Marching Cubes on the selection grid. (b) Performing decimation through edge collapses results in a mesh that does not bound the region to be edited. Let e be an edge (horizontal dashed line) that is to be collapsed into vertex $\mathbf{v}$. Without constraints, the mesh might shrink (orange region). (c) We enforce conservative bounding by including a linear constraint.

To satisfy the bounding property, we apply morphological operators [Calderon and Boubekeur 2017], specifically, a closing operation with a cube as the structuring element for the dilation step and a sphere as structuring element for the subsequent erosion. This helps achieve a grid free of gaps, simplifying the downstream steps of the pipeline by filtering the geometry and topology of the selection (Fig. 3c). The size of the structuring elements is crucial for a proper simplification of the topology of the selection. We provide the user with the possibility to choose the dilation and erosion structuring elements, as well as their corresponding radii. The result of this process is a 3D binary grid indicating the simplified selection, from which we extract a surface mesh using marching cubes [Lorensen and Cline 1987].

The mesh we obtain usually contains too many vertices and edges to enable intuitive editing and ensure interactivity (Fig. 4a). We thus seek to simplify it. However, naïvely applying standard edge collapsing techniques [Garland and Heckbert 1997] would violate the property that the cage encloses the selected region (Fig. 4b). As a remedy, we follow standard practice [Deng et al. 2011; Platis and Theoharis 2003; Sander et al. 2000] and add an extra linear constraint to guarantee strict inclusion when performing each edge collapse (Fig. 4c). To improve the stability and consistency of our decimation scheme, inspired by the work of Platis and Theoharis [Platis and Theoharis 2003], we add three extra per-edge conditions: First, the canonical link condition [Dey et al. 1998] is tested to ensure the mesh remains a manifold when collapsing an edge. Second, we avoid highly connected vertices, i.e., ones with a high valence, since these degenerate cases tend to result in unintuitive editing behavior, and they interfere with edge collapse. Thus, we simply enforce an upper valence bound. We found a valence bound of 12 to work well in practice but allow the user to adjust this value inside NeRFshop. Third, we prevent triangles from becoming too thin, again with the intention to raise the usability of the cage. To this end, we measure the triangles' compactness (for details, please see Appendix A) and make sure that no mesh simplification step increases the compactness of the involved triangles.

Finally, we apply a post-processing step using *MeshFix* [Attene et al. 2013] to avoid holes, self-intersections, and degenerate elements. In practice, we noticed that this can slightly alter the cage volume in some cases, resulting in a mesh that does not perfectly bound the selection. Nonetheless, we found this step essential to perform consistent interpolations and tetrahedralization. In the rare event where post-processing with MeshFix causes issues, the user can easily adjust the cage by hand.

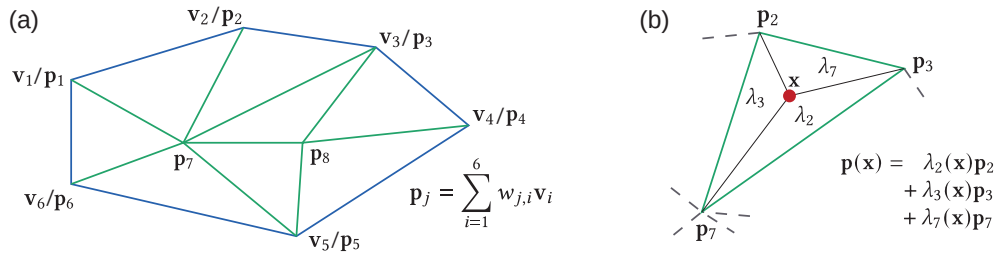


Fig. 5. We use two nested levels of interpolation: (a) Tetrahedral mesh vertices store MVC-interpolated values. (b) Point queries $\mathbf{x}$ are barycentrically interpolated from these pre-computed tetrahedral values.

4 INTERACTIVE VOLUMETRIC EDITING

The cage creation as described in the previous section allows the user to interactively manipulate the selected region. This includes both rigid transformations (i.e., all cage vertices are displaced using the same transformation), and non-rigid deformations (i.e., subsets of vertices are displaced independently). Our goal is to propagate the cage vertex displacements into corresponding manipulations of the enclosed density volume. To this end, we tetrahedralize the cage. Since we have created the cage carefully (Sec. 3), its triangle mesh is robust enough so that we can use *TetGen* [Si 2015] without modifications to extract a tetrahedral mesh from the cage. In the following, we refer to the initial configuration of the scene as the *canonical space*, and refer to the configuration of the scene after cage-based manipulation by the user as the *deformed space*.

We employ a two-level interpolation scheme, as illustrated in Fig. 5. The first level is the MVC-based interpolation done in a pre-processing step. The second level is an online barycentric interpolation inside the tetrahedron. The latter avoids the $\Omega(N)$ MVC computation during rendering at each sample location in a cage with N vertices. It further maps well to the GPU-accelerated rendering pipeline of Instant-NGP [Müller et al. 2022], which proceeds in ray batches and relies on finely optimized CUDA kernels. Our two-level interpolation scheme enables interactive cage-based editing and renders user-edited scenes at real-time frame rates.

Let $\mathbf{v}_i \in \mathbb{R}^3$ with $i \in \{1, \ldots, n\}$ be the initial positions of the cage vertices, and $\mathbf{p}_j \in \mathbb{R}^3$ $j \in \{1, \ldots, m\}$ the initial positions of the vertices of its tetrahedral mesh, the first n of which are identical to the positions of the cage vertices. In a pre-processing step, we now compute the matrix of MVC coefficients $w_{j,i} \in \mathbb{R}$ for all $\mathbf{p}_j$ with respect to all $\mathbf{v}_i$, i.e., we compute $w_{j,i}$ such that

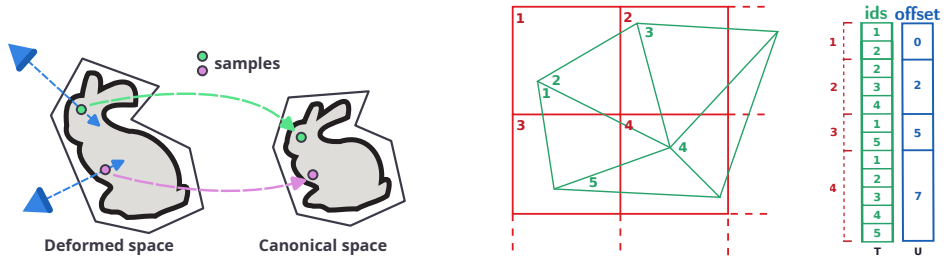
$$\mathbf{p}_j = \sum_{i=1}^{n} w_{j,i}\mathbf{v}_i \quad \text{with} \quad \sum_{i=1}^{n} w_{j,i} = 1 \quad \forall j. \tag{4}$$

When the user updates the cage vertices to new locations $\hat{\mathbf{v}}_i \in \mathbb{R}^3$, we can efficiently update the tetrahedral vertices, including interior ones, using the pre-computed MVC coordinates.

$$\hat{\mathbf{p}}_j = \sum_{i=1}^{n} w_{j,i}\hat{\mathbf{v}}_i. \tag{5}$$

In addition to the new edited position $\hat{\mathbf{p}}_j$, each vertex in the tetrahedral mesh always stores its initial position $\mathbf{p}_j$ to be used for look-ups during rendering. When deforming each tetrahedron, its local rotation matrix is estimated with SVD, which we use to account for view-dependent effects (see below).

Let us consider a single edit with a corresponding cage in canonical and deformed space. Whenever a sample $\mathbf{x}$ lies inside its deformed volume, we first determine the deformed-space tetrahedron



(a) Rendering edited objects in NeRFshop (b) Lookup table for tetrahedron intersection

Fig. 6. (a) Rendering edited scenes: during ray marching, we map all samples that intersect with deformed tetrahedra into canonical space, where we infer the values for integration. (b) We pre-compute a tetrahedron lookup table using a voxel grid, where each cell stores all intersecting tetrahedra. The structure is linearized into a 1D array T following the 3D Morton order of the grid, with an auxiliary array U for bookkeeping.

$\mathbf{x}$ falls into. We then compute the barycentric coordinates λ_k of $\mathbf{x}$ with respect to the vertices $\hat{\mathbf{p}}_j$ of the deformed tetrahedron (Fig. 5b). Finally, we map back and query the NeRF in canonical space at a location computed from barycentric interpolation of the vertex positions $\mathbf{p}_j$ of the corresponding unmodified tetrahedron (Fig. 6a). To account for view-dependent changes in radiance due to deformation, we can use our local rotation estimates to transform the view direction input for the NeRF. Our approach enables stacking edits, i.e., further deform the outcome of a previous deformation. It is also possible to create multiple (deformed) copies of an object. This is achieved by backmapping recursively and testing for intersection with each edit's deformation cage in reverse order.

To render an image in deformed space, we use volume rendering with quadrature through Eq. 1 which can be rewritten as:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \alpha_i \mathbf{c}_i \text{ where } T_i = \prod_{j=1}^{i-1}(1-\alpha_i) \text{ and } \alpha_i = (1-\exp(-\sigma_i \delta_i)). \tag{6}$$

Samples that intersect a canonical cage can be included or skipped for copying or non-copying edits, respectively.

Rendering and editing as described above require determining which deformed tetrahedron encloses a given 3D query position for all samples along each view ray. For interactive performance, this needs to be performed in parallel for several rays, thus demanding an appropriate implementation. To this effect, ray tracing libraries (e.g., OptiX [Parker et al. 2010]) can be leveraged by tracing rays in arbitrary directions and testing which tetrahedra the intersected triangles belong to, as shown by previous work [Garbin et al. 2022; Wald et al. 2019]. However, our implementation is built on top of Instant-NGP for which we cannot trivially exploit hardware ray tracing pipelines, as it uses carefully designed software rendering routines. Enabling support for hierarchical acceleration structures would require additional per-ray states (e.g., traversal stack) to be maintained across multiple ray compaction steps.

As a consequence, we solve this problem using an adaptive lookup table, which is illustrated in Fig. 6b. Similar to our selection method (Sec. 3), we discretize our 3D domain into a regular voxel grid. Each voxel contains a list of the tetrahedra that intersect it, obtained through inclusion tests and cube-triangle intersections for each triangle of the tetrahedron. The evaluation of each voxel-tetrahedron pair can be done independently and list generation for voxels requires a simple prefix sum, which makes the construction perfectly parallel. The voxels are linearized into a 1D array T by following a Morton order [Morton 1966] to ensure high locality and reusability of

transferred data during ray traversal, regardless of the view direction. Since voxels contain a variable number of entries, we use an auxiliary array U to store the result of the prefix sum and to perform efficient lookups. We can now use these two arrays to quickly determine the candidate tetrahedra for a query position: We first map the query position to its enclosing grid cell and query U to obtain the candidate tetrahedra indices in T. With the dynamic lookup table, our two-level interpolation scheme enables efficient rendering when naïve $\Omega(N)$ MVC calculation would be intractable. Our experiments showed that the average number of tested tetrahedra per sample ranges from 2 for coarse cages to 7 for detailed ones (using $\approx$ 50/1k cage vertices, respectively). Therefore, we consider this a good trade-off between memory footprint and efficiency.

5 MEMBRANE-BASED CORRECTION AND DISTILLATION

In this section, we discuss a first attempt at correcting visual artifacts after editing, and a way to consolidate edits in a "distillation" process to convert the edited scene into an established NeRF format. These are two useful examples of how NeRFshop enables the recreation of operations we know from image editing, specifically, Poisson-style image fusion and editing layer collapse (image export). Many similar operations could be provided in our environment.

5.1 Membrane-based Correction

Moving an object in a NeRF can give rise to inconsistencies in appearance. For example, if the cage is large and includes density from a surface supporting the object, this density will incorrectly move with the displaced object, resulting in a "broken" color transition at the new location (Fig. 10). Similarly, in the case of disocclusions (e.g., due to the translation of an object), the NeRF may not have reliable appearance information and display unpredictable content.

We observe that these problems have strong analogies to "cut-and-paste" image editing operations, for which Poisson Image Editing [Pérez et al. 2003] is a gold-standard solution. However, the adaptation of Poisson editing to the NeRF setting is not straightforward, since we need to not only consider color as in the case of images but also volumetric density. To this end, we propose a first attempt at a volumetric membrane-based correction method for both color and density, by adapting Instant Image Cloning [Farbman et al. 2009]. Our preliminary approach approximates the costly Poisson solver with a color correction membrane using Mean Value Coordinates, which fits well with our tetrahedral MVC interpolation scheme, allowing efficient computation. In the following, we denote quantities 3D space at the vertices of the canonical and deformed cage using subscripts "in" and "out", respectively.

We proceed in three steps: The first two steps precompute necessary quantities from the cage and vertices of its tetrahedral mesh, respectively, while the third step uses these quantities during interactive rendering. Specifically, we consider densities σ and view-dependent colors $\mathbf{c}$, where the latter is baked into low-order spherical harmonics (SH), avoiding NeRF network evaluations during interactive rendering.

Step 1: Cage Vertex Preprocessing. At each initial cage vertex position $\mathbf{v}_k$, we query the NeRF network and store the corresponding density $\sigma_{\text{in}}^{(k)}$ and view-dependent color $\mathbf{c}_{\text{in}}^{(k)}$. As the user moves the cage vertices yielding new positions $\hat{\mathbf{v}}_k$, we compute corresponding new densities $\sigma_{\text{out}}^{(k)}$ and colors $\mathbf{c}_{\text{out}}^{(k)}$ at these locations, which act as the outside quantities to be matched. From these vertex quantities, we now compute density-weighted color residuals $\mathbf{c}_{\text{res}}^{(k)}$ for our correction membranes, encouraging propagation only when there is sufficient density:

$$\mathbf{c}_{\text{res}}^{(k)} = \mathbf{c}_{\text{out}}^{(k)} - \min\left(\frac{\alpha_{\text{in}}^{(k)}}{\alpha_{\text{out}}^{(k)}}, 1\right)\mathbf{c}_{\text{in}}^{(k)}, \tag{7}$$

with $\alpha_{\text{in}}^{(k)} = 1 - \exp\left(-\sigma_{\text{in}}^{(k)}\delta_c\right)$ and $\alpha_{\text{out}}^{(k)} = 1 - \exp\left(-\sigma_{\text{out}}^{(k)}\delta_c\right)$. Note that if $\alpha_{\text{in}}^{(k)}$ is similar to $\alpha_{\text{out}}^{(k)}$, the residual in Eq. 7 approaches the original Image Cloning formulation [Farbman et al. 2009].

Step 2: Propagation to Tetrahedral Mesh Vertices. We use MVC interpolation to propagate the residuals at the cage vertices to the vertices of the tetrahedral mesh. To avoid interpolating meaningless colors from vertices with no density, we again employ an α-weighted interpolation scheme:

$$\mathbf{c}_{\text{res}}(\mathbf{x}) = \frac{\sum_{k=1}^{n} w_k(\mathbf{x})\alpha_{\text{out}}^{(k)}\mathbf{c}_{res}^{(k)}}{\sum_{k=1}^{n}\alpha_{\text{out}}^{(k)}} \tag{8}$$

We evaluate this formulation for all tetrahedral mesh vertices, and store for each vertex $\mathbf{c}_{\text{res}}$, σ_{out}, and the residual density $\sigma_{\text{res}} = \sigma_{\text{out}} - \sigma_{\text{in}}$.

Step 3: Rendering. For each sample query during rendering, we interpolate the values of $\mathbf{c}_{\text{res}}$, σ_{out}, and σ_{res} using tetrahedral barycentric coordinates λ. Different from standard ray marching, these quantities need to be taken into account when computing final pixel values. Following notation from Eq. 6, we employ correction-aware alpha terms:

$$\alpha_i' = 1 - \exp\left(-(\sigma_i + \sigma_{\text{res}}\delta_i)\right). \tag{9}$$

We then obtain the final corrected color

$$\mathbf{c}_i' = \frac{\alpha_i\mathbf{c}_i + \alpha_{\text{out}}\mathbf{c}_{\text{res}}}{\alpha_i + \alpha_{\text{out}}}, \tag{10}$$

with α_{out} defined as in step 1. The volume rendering equation thus becomes:

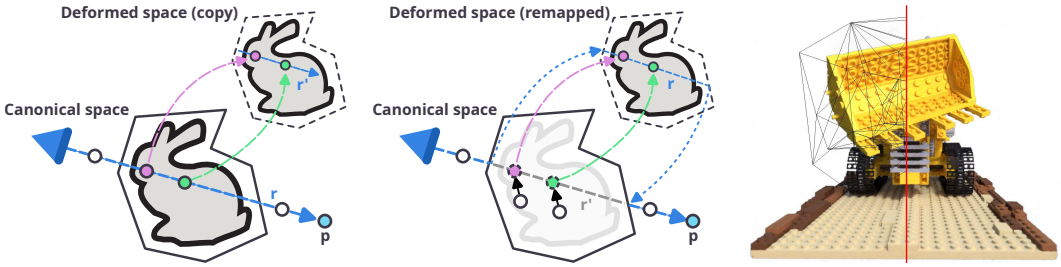
$$\hat{C}'(\mathbf{r}) = \sum_{i=1}^{N} T_i'\alpha_i'\mathbf{c}_i' \quad \text{where} \quad T_i' = \prod_{j=1}^{i-1}(1-\alpha_i'). \tag{11}$$

Due to the normalized nature of MVCs and the lack of a directional prior, this approach can result in density bleeding. To avoid this we cap the residual density so that adding it to the inside density does not exceed the initial outside density at this location.

5.2 NeRF Distillation

In image or 3D scene editing, the last steps of a project include finalizing edits (e.g., collapse layers or apply modifiers) and exporting the result to a standardized format. Publishing the data in this way allows it to be visualized, shared or converted by applications that can process the format. While NeRFs are a relatively new medium, the landscape of data formats and useful applications is quickly developing. Instant-NGP plays an important role in this: solutions for virtual reality and codeless rendering of Instant-NGP hashgrids are already available. Recent work laid the foundations for converting between various NeRF representations, including the Instant-NGP hashgrid [Fang et al. 2023].

In a similar spirit to image editing best-practice, we propose a *NeRF distillation* approach that "collapses" all edits performed by the user into the default hashgrid format of Instant-NGP. To achieve this, we re-train the scene, but additionally consider the user-provided edits to forward-map samples during optimization (Fig. 7). Importantly, this process enables us to perform distillation based on the original input images. For each sample s generated along rays through the scene, we check whether it is transformed by an edit i. If it is, this constitutes a new mapping $s \longrightarrow s_i$. Note



(a) Ray Copying (b) Ray Remapping (c) Distilling edit (left) to NeRF

Fig. 7. Scene edits are distilled into a regular NeRF representation by re-training and respecting deformations for aggregation and gradient propagation. Additional instances of an initial ray **r** are created for every transformative mapping, e.g., **r'** for a copied object in (a); both rays integrate samples and compute gradients according to the target image pixel **p**. Shared sample locations (e.g., outside of cages) are only updated by one instance. For non-copying edits (b), the initial ray **r** is remapped and **r'** clears the space inside the canonical-space cage.

that s_i may again be transformed by an edit j, thus mappings (e.g., $s \rightarrow s_{ij}$) can arise recursively. If a sample remains in canonical space, we say that it has only the "default mapping". During training, we keep track of all unique mappings along each traced ray.

In a second step, we re-trace each ray r with more than one mapping, as well as copies r' attempting to apply the identified mappings to each sample along r. Each mapping implies one or more deformations, which will copy or remap updates that would normally occur in canonical space. However, a mapping $s \rightarrow s_i$ may be "broken" if s_i lies inside the deformed cage of a later edit j, i.e., deforming space according to j overwrites the result of deformation by i. Note that "broken" mappings can also occur for the initial rays, i.e., the default mapping is no longer possible. We found that simply skipping "broken" mappings suffices to distill a wide range of edits, however, a more principled solution would be desirable to support arbitrary editing scenarios (Section 7).

Finally, we adapt the methods for color composition and gradient computation. A sample location may be shared (outside cages in Fig. 11a and b) or unique to a mapping. Shared sample locations only receive gradients from the first ray that references them. If an edit is non-copying, the contents at its source must be cleared. Failure to do so causes random density or color in the left-behind empty space. Samples that end up in the undeformed cage of a non-copying edit during their mapping are part of these clearing rays. They accumulate density at their respective locations and their integration target is empty space (zero density).

6 IMPLEMENTATION, RESULTS, AND EVALUATION

We first provide some details of our implementation, then present the results of our method of synthetic and real scenes. We also provide a comparison to previous work, demonstrating our clear advantage in the speed of updates.

6.1 Implementation

Our pipeline is built on top of the Instant-NGP [Müller et al. 2022] open-source codebase. We use the same cascaded hierarchy of 128^3 resolution grids, each spanning a larger domain $[-2^{k-1}+0.5, 2^{k-1}+0.5]^3$ centered around $(0.5, 0.5, 0.5)$ where $k \in \{1, \ldots, K\}$ and $K \in [1, 5]$. Please refer to the original paper for more details. Since the sampling strategy along each ray of Instant-NGP depends on the *occupancy grids* of the scene, we need to update them to take into account the density of the deformed objects after the edits. To do so, we uniformly sample $M = K \times 128^3$

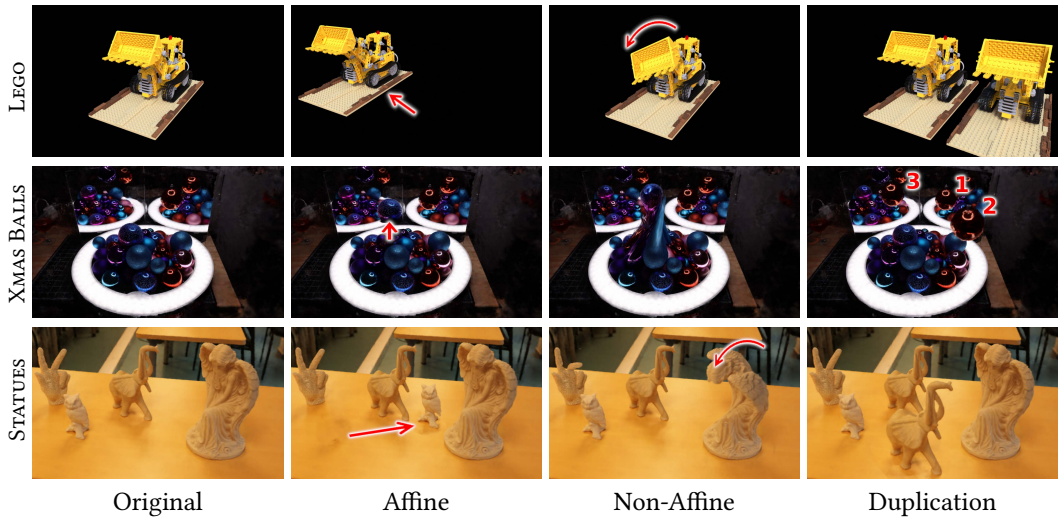


Fig. 8. We showcase three types of different edits (affine, non-affine, and duplication) while keeping the same viewpoint in three different scenes. Subtler edits are highlighted by red glyphs. All were applied interactively.

Fig. 9. Affine, non-affine, and duplicative edits with different viewpoints in the Hallway scene

cells and evaluate the density network of the NeRF at a random position within this cell. We then update the grid with exponential decay based on these samples. In a nutshell, the sampling along each ray is carried out in deformed space while the evaluation is performed in canonical space following the backward mapping scheme described in Sec. 4.

For specific geometry processing algorithms, we use the C++ library *libigl* [Jacobson et al. 2018] that provides direct bindings to *TetGen* [Si 2015] for the tetrahedralization step of cage-building, and we use its existing implementation of the "Progressive Hulls" algorithm for constrained mesh simplification introduced by Sander et al. [Sander et al. 2000]. The provided code reimplements the quadratic programming solver by Goldfarb and Idnani [Goldfarb and Idnani 1983] based on the dual method. NeRFshop offers an extensive user interface, allowing the user to perform a wide range of coarse and precision edits interactively. We use *Dear ImGui* to create interactive panes, tabs and gizmos for intuitive 3D manipulation. Auxiliary visual elements (wireframes, grids) are added through OpenGL for our custom selection methods.

6.2 Results

We used 5 scenes, Lego from NeRF Synthetic dataset, XmasBalls [1], Kitchen from [Prakash et al. 2022], as well as two real-world scenes (Hallway and Statues) that we captured ourselves. We apply different and diverse deformations and edits for all scenes, both synthetic and captured, focusing on affine and non-affine transformations and object duplication. We also use our tools to clean up and fix typical artifacts, i.e., fog or floaters, that often appear during the optimization. We show these results in Fig. 1 and Fig. 8, 9. The edit operations are best understood by watching the supplementary video, where we can see that edits can be easily performed with our direct-manipulation interface. NeRFshop allows intuitive, direct-manipulation editing of NeRF, allowing free-form deformations, and unleashing artistic creativity for captured scenes.

Without additional correction, superfluous density can be captured by a coarse cage and deformed along with the object of interest (e.g., the table surface stuck to the cup in the bottom left of Fig. 9). In Fig. 10, we illustrate the effect of the membrane correction; we see that distillation helps reduce artifacts incurred by the editing operation, even though some artifacts remain. Fig. 11 demonstrates our NeRF distillation and how it preserves the quality of the edited NeRF, allowing the edited result to be viewed in the original Instant-NGP codebase and its extensions (e.g., for virtual reality).

Fig. 10. Effect of our membrane correction step in two different cases. We observe that the artifacts due to "leftover matter" captured at the boundary of the undeformed cage are diminished.

Fig. 11. Distilling 3 edits in the Statues scene (left, compare with Fig. 8) into a self-contained NeRF (right).

6.3 Comparisons

We compare our method with the two most relevant and recent methods NeRF-Editing [Yuan et al. 2022] and Deforming-NeRF [Xu and Harada 2022]. We designed an experiment in which we

[1] Captured by Hugues Bruyère (@smallfly).

create a synthetic scene in Blender we call Hand that simulates a more realistic environment than the synthetic NeRF Blender dataset that consists of objects on a white background. Using Hand, we handcrafted a coarse cage which we deformed toward a target folded position (see first row of Fig. 12) and also derived intermediate positions (34% and 67% of the deformation). A custom Blender plugin was used for the ground truth deformation which follows closely the implementation of MVC-based deformations introduced by Ju et al. [Ju et al. 2005]. We then rendered images from several viewpoints with Blender's internal renderer, Cycles. We refer to these images as ground truth. Finally, we provided the same cages (original, intermediates, and target) to all pipelines and rendered the resulting deformations from the same camera poses. We used a resolution of 800×800 pixels and one sample per pixel in all cases. Ours and Deforming-NeRF were run on an Nvidia A6000, while NeRF-Editing was rendered on an Nvidia RTX 3090.

Deforming-NeRF [Xu and Harada 2022] relies on Plenoxels [Sara Fridovich-Keil and Alex Yu et al. 2022] as a backend for rendering. To achieve the best rendering performance possible, we trained the scene for 8 epochs, each with 12 800 iterations, and performed upsampling of the grid every 2 epochs with the following resolutions: $128^3 \rightarrow 256^3 \rightarrow 512^3 \rightarrow 640^3$. To produce comparable deformation, we used Mean Value Coordinates and a discretized grid of size 128^3 to cache these pre-computed coordinates, as it is the value recommended in the original publication. The background with the default hyper-parameters from the original method.

For NeRF-Editing [Yuan et al. 2022] we trained for 3 hours to get sufficiently good quality results. We then extracted a mesh for the target object and deformed it according to the given target cage. To propagate this edit we follow the instructions of the authors and convert the extracted mesh into a tetrahedral mesh before optimizing the vertex positions to match the target positions. To render the final results we use 64 coarse steps and 64 steps using importance sampling. We chose not to render the background as the extracted mesh included a part of the table, leading to artifacts when editing the hand.

Ours: We train the scene for 30k steps (3 min 32 s) before enforcing the edits by directly setting the cage vertices in canonical and deformed space. The hashgrid is parameterized with the default settings for Instant-NGP (16 levels, 2 features per level, 16 entries at base resolution).

Deforming-NeRF [Xu and Harada 2022] takes 0.954 s on average to render a frame before edits, NeRF-Editing [Yuan et al. 2022] takes 4.503 s, and ours 0.189 s. After edits, the times for each method are respectively 2.561 s, 35.696 s, and 0.243 s. We can see that Deforming-NeRF and Nerf-Editing incur a 2.68× and 7.92× overhead respectively for the same editing operation making them clearly unsuitable for interactive editing, while ours is only 1.28× slower and remains largely interactive. We found that our method introduces no significant artifacts due to deformation, even when the magnitude of the deformation is large. Competing methods have severe constraints regarding the scene setup, e.g., NeRF-Editing needs a clear separation between foreground and background. This is the reason why NeRF-Editing has no rendered background and why Deforming-NeRF has a lower quality for the background even without any deformation.

We address interactivity in NeRF editing with an intuitive, responsive and fully self-contained solution that provides screen-based selection, 3D scene manipulation scheme, compositing, and trivial portability of the results. To enable the latter two, we propose our membrane-based correction and distillation steps. Building on Instant-NGP, we achieve interactive editing and maintain real-time rendering of scenes due to careful design choices (MVC, two-level interpolation scheme, per-sample backmapping). Our dynamic lookup table ensures that coordinate-based mapping remains computationally tractable, even with detailed cages. NeRFshop leverages the GPU while minimizing the performance impact on Instant-NGP's carefully tuned routines. In contrast, interactive usability with previous NeRF editing methods faces several challenges. The cage-building processes of NeRF-Editing [Yuan et al. 2022] and Deforming-NeRF [Xu and Harada 2022] is automatic, but still require

additional refinements which have to be carried out manually in specialized software (e.g. Blender). The automated parts of their respective pipelines involve multiple, slow-running steps to carry out simple edits and provide no means to visualize intermediate results. Once edited, the results can only be visualized by their respective rendering solutions, using auxiliary data structures created by their processing pipeline. This severely limits portability of the edited scenes.

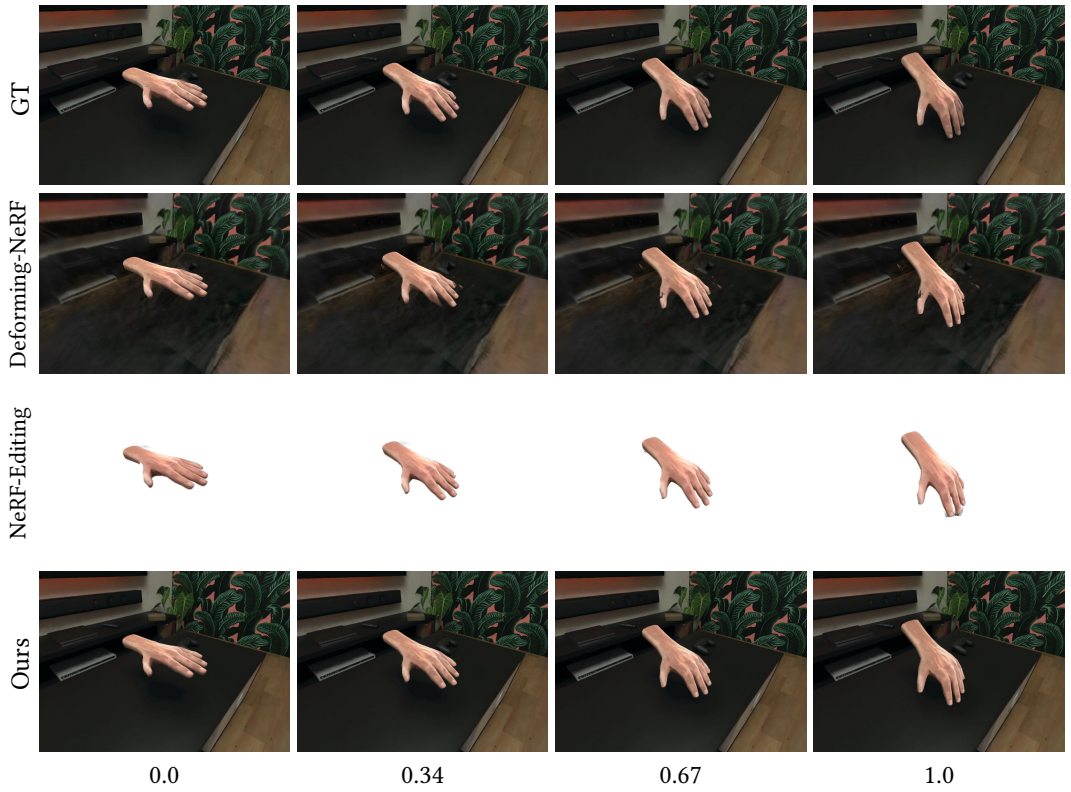


Fig. 12. In this figure, we show a comparison with other methods. From left to right we modulate the magnitude of the deformation starting with no deformation at all. On the top row, we see the ground truth rendered by Blender Cycles and deformed in Blender.

7 DISCUSSION AND FUTURE WORK

Our selection interface is rudimentary and the user interface could be redesigned to be easier to use. In addition, the actual selection algorithm itself could, e.g., use semantic scene analysis building on appearance features to improve and simplify the task.

Currently, we do not account for lighting inconsistencies due to edits. Changing the lighting conditions of a NeRF is still an open problem [Boss et al. 2021; Zhang et al. 2021]; correcting shadows, highlights, and lighting overall is left as an exciting avenue for future work.

The first attempt at membrane-based editing presented here can be improved. One limitation is that our tetrahedral volume is too coarse to allow high-quality Poisson-like smoothing; an adaptive refinement approach for the bounding cage could help. Making such a method sufficiently fast to maintain interactivity is an open challenge, however. We could also build on more advanced

methods such as Screened Poisson [Bhat et al. 2008; Morel et al. 2014] to take both the inside and outside densities into account.

The distillation approach we described suffices to create Instant-NGP NeRF representations of many editing scenarios, including all modifications shown in our supplemental video. However, it can fail in scenes with destructive edits, i.e., when large regions of non-empty space are overwritten. This effectively constitutes a "broken" mapping: for instance, if one object is moved to overlap with another in canonical space, the default mapping of the overwritten object onto itself is no longer feasible. In these cases, the training will attempt to consolidate information in the images with the edited scene by hallucinating superfluous density and color elsewhere in the scene. These artifacts could be addressed by a more sophisticated mechanism for handling "broken" sample mappings, e.g., by recovering or approximating missing density/color for integration along rays.

8 CONCLUSION

We proposed a novel end-to-end method allowing interactive direct manipulation of Neural Radiance Fields with free-form editing. We presented a fully self-contained pipeline composed of a scribble-based selection scheme followed by a user-supervised 3D region growing and carefully chosen morphological operations. Thanks to constrained mesh decimation, we obtain a cage that the user can interactively manipulate to perform free-form edits. To reduce visual artifacts due to these edits, we propose a preliminary membrane interpolation correction scheme inspired by Poisson image processing. We show our approach on several synthetic and real scenes, demonstrating interactive edit operations such as duplication, translation/scale/rotation, free-form non-rigid deformations, and floater removal as shown in the supplemental video. Overall, we hope that our solutions will empower new trends in neural scene manipulation and enable artists to express their creativity and apply their editing skills to NeRFs.

ACKNOWLEDGEMENTS

This research was funded by the ERC Advanced grant FUNGRAPH No 788065 (http://fungraph.inria.fr). The authors are grateful to the OPAL infrastructure from Université Côte d'Azur for providing resources and support. The authors would also like to thank Adobe for their generous research and software donations. Finally, the authors would like to thank Hugues Bruyère (@smallfly) for his scene XmasBalls.

A COMPACTNESS

The compactness $c(T)$ of a triangle T is given by its isoperimetric quotient (also named Polsby-Popper score in 2d) which is defined as:

$$c(T) = \frac{4\pi A(T)}{P(T)^2} \text{ where } A(T) \text{ and } P(T) \text{ are respectively its area and perimeter} \tag{12}$$

Intuitively, compactness describes how far from circular the triangle is or more precisely, how the area of the triangle differs from a circle with the same perimeter. In order to ensure collapsed edges do not decrease compactness too much we relatively bound the minimum compactness c_e of the triangles of the candidate edge by the value of minimum compactness c_v of the triangles resulting from the potential collapse. As suggested by Platis and Theoharis [Platis and Theoharis 2003], we use $r = 0.8$.

REFERENCES

Marco Attene, Marcel Campen, and Leif Kobbelt. 2013. Polygon Mesh Repairing: An Application Perspective. *ACM Comput. Surv.* 45, 2, Article 15 (mar 2013), 33 pages. https://doi.org/10.1145/2431211.2431214

Pravin Bhat, Brian Curless, Michael Cohen, and Larry Zitnick. 2008. *Fourier Analysis of the 2D Screened Poisson Equation for Gradient Domain Problems.*

Pravin Bhat, C. Lawrence Zitnick, Michael Cohen, and Brian Curless. 2010. GradientShop: A Gradient-Domain Optimization Framework for Image and Video Filtering. *ACM Trans. Graph.* 29, 2, Article 10 (apr 2010), 14 pages. https://doi.org/10.1145/1731047.1731048

Wang Bing, Lu Chen, and Bo Yang. 2022. DM-NeRF: 3D Scene Geometry Decomposition and Manipulation from 2D Images. *arXiv preprint arXiv:2208.07227* (2022).

Mark Boss, Raphael Braun, Varun Jampani, Jonathan T. Barron, Ce Liu, and Hendrik P.A. Lensch. 2021. NeRD: Neural Reflectance Decomposition from Image Collections. In *IEEE International Conference on Computer Vision (ICCV)*.

Stéphane Calderon and Tamy Boubekeur. 2017. Bounding Proxies for Shape Approximation. *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)* 36, 5, Article 57 (july 2017).

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. 2021. Emerging Properties in Self-Supervised Vision Transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. 2022. Efficient Geometry-aware 3D Generative Adversarial Networks. In *CVPR*.

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. TensoRF: Tensorial Radiance Fields. In *European Conference on Computer Vision (ECCV)*.

Zheng-Jie Deng, Xiao-Nan Luo, and Xiao-Ping Miao. 2011. Automatic Cage Building with Quadric Error Metrics. *Journal of Computer Science and Technology* 26, 3 (01 May 2011), 538–547. https://doi.org/10.1007/s11390-011-1153-4

Tamal K. Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry V. Nekhayev. 1998. Topology Preserving Edge Contraction. *Publ. Inst. Math. (Beograd) (N.S* 66 (1998), 23–45.

Shuangkang Fang, Weixin Xu, Heng Wang, Yi Yang, Yufeng Wang, and Shuchang Zhou. 2023. One is All: Bridging the Gap Between Neural Radiance Fields Architectures with Progressive Volume Distillation. *AAAI* (2023).

Zeev Farbman, Gil Hoffer, Yaron Lipman, Daniel Cohen-Or, and Dani Lischinski. 2009. Coordinates for Instant Image Cloning. *ACM Trans. Graph.* 28, 3, Article 67 (jul 2009), 9 pages. https://doi.org/10.1145/1531326.1531373

Michael S. Floater. 2015. Generalized barycentric coordinates and applications. *Acta Numerica* 24 (2015), 161–214. https://doi.org/10.1017/S0962492914000129

Xiao Fu, Shangzhan Zhang, Tianrun Chen, Yichong Lu, Lanyun Zhu, Xiaowei Zhou, Andreas Geiger, and Yiyi Liao. 2022. Panoptic NeRF: 3D-to-2D Label Transfer for Panoptic Urban Scene Segmentation. In *International Conference on 3D Vision (3DV)*.

Stephan J. Garbin, Marek Kowalski, Virginia Estellers, Stanislaw Szymanowicz, Shideh Rezaeifar, Jingjing Shen, Matthew Johnson, and Julien Valentin. 2022. VolTeMorph: Realtime, Controllable and Generalisable Animation of Volumetric Representations. https://doi.org/10.48550/ARXIV.2208.00949

Michael Garland and Paul S. Heckbert. 1997. Surface Simplification Using Quadric Error Metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 209–216. https://doi.org/10.1145/258734.258849

Donald Goldfarb and Ashok U. Idnani. 1983. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming* 27 (1983), 1–33.

Kai Hormann and Michael S. Floater. 2006. Mean Value Coordinates for Arbitrary Planar Polygons. *ACM Trans. Graph.* 25, 4 (oct 2006), 1424–1441. https://doi.org/10.1145/1183287.1183295

Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. https://libigl.github.io/.

Wonbong Jang and Lourdes Agapito. 2021. CodeNeRF: Disentangled Neural Radiance Fields for Object Categories. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 12929–12938. https://doi.org/10.1109/ICCV48922.2021.01271

Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean Value Coordinates for Closed Triangular Meshes. *ACM Trans. Graph.* 24, 3 (jul 2005), 561–566. https://doi.org/10.1145/1073204.1073229

Kacper Kania, Kwang Moo Yi, Marek Kowalski, Tomasz Trzciński, and Andrea Tagliasacchi. 2022. CoNeRF: Controllable Neural Radiance Fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. 2022. Decomposing NeRF for Editing via Feature Field Distillation. *arXiv* (2022).

Adam R Kosiorek, Heiko Strathmann, Daniel Zoran, Pol Moreno, Rosalia Schneider, Sona Mokra, and Danilo Jimenez Rezende. 2021. NeRF-VAE: A Geometry Aware 3D Scene Generative Model. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 5742–5752. https://proceedings.mlr.press/v139/kosiorek21a.html

Abhijit Kundu, Kyle Genova, Xiaoqi Yin, Alireza Fathi, Caroline Pantofaru, Leonidas J. Guibas, Andrea Tagliasacchi, Frank Dellaert, and Thomas Funkhouser. 2022. Panoptic Neural Fields: A Semantic Object-Aware Neural Scene Representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 12871–12881.

Verica Lazova, Vladimir Guzov, Kyle Olszewski, Sergey Tulyakov, and Gerard Pons-Moll. 2022. Control-NeRF: Editable Feature Volumes for Scene Rendering and Manipulation. https://doi.org/10.48550/ARXIV.2204.10850

Anat Levin, Assaf Zomet, Shmuel Peleg, and Yair Weiss. 2004. Seamless Image Stitching in the Gradient Domain. In *Computer Vision - ECCV 2004*, Tomás Pajdla and Jiří Matas (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 377–389.

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural Sparse Voxel Fields. *NeurIPS* (2020).

Steven Liu, Xiuming Zhang, Zhoutong Zhang, Richard Zhang, Jun-Yan Zhu, and Bryan Russell. 2021. Editing Conditional Radiance Fields. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM siggraph computer graphics* 21, 4 (1987), 163–169.

Haimin Luo, Teng Xu, Yuheng Jiang, Chenglin Zhou, QIwei Qiu, Yingliang Zhang, Wei Yang, Lan Xu, and Jingyi Yu. 2022. Artemis: Articulated Neural Pets with Appearance and Motion synthesis. *arXiv preprint arXiv:2202.05628* (2022).

Nelson Max. 1995. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics* 1, 2 (1995), 99–108.

James McCann and Nancy S. Pollard. 2008. Real-Time Gradient-Domain Painting. *ACM Transactions on Graphics (SIGGRAPH 2008)* 27, 3 (Aug. 2008).

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

Ashkan Mirzaei, Yash Kant, Jonathan Kelly, and Igor Gilitschenski. 2022. LaTeRF: Label and Text Driven Object Radiance Fields. https://doi.org/10.48550/ARXIV.2207.01583

Jean-Michel Morel, Ana-Belen Petro, and Catalina Sbert. 2014. Screened Poisson Equation for Image Contrast Enhancement. *Image Processing On Line* 4 (2014), 16–29. https://doi.org/10.5201/ipol.2014.84.

Guy M Morton. 1966. A computer oriented geodetic data base and a new technique in file sequencing. (1966).

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages. https://doi.org/10.1145/3528223.3530127

Jesús R. Nieto and Antonio Susín. 2013. *Cage Based Deformations: A Survey*. Springer Netherlands, Dordrecht, 75–99. https://doi.org/10.1007/978-94-007-5446-1_3

Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. 2021a. Nerfies: Deformable Neural Radiance Fields. *ICCV* (2021).

Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. 2021b. HyperNeRF: A Higher-Dimensional Representation for Topologically Varying Neural Radiance Fields. *ACM Trans. Graph.* 40, 6, Article 238 (dec 2021).

Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. 2010. OptiX: A General Purpose Ray Tracing Engine. *ACM Trans. Graph.* 29, 4, Article 66 (jul 2010), 13 pages. https://doi.org/10.1145/1778765.1778803

Yicong Peng, Yichao Yan, Shengqi Liu, Yuhao Cheng, Shanyan Guan, Bowen Pan, Guangtao Zhai, and Xiaokang Yang. [n. d.]. CageNeRF: Cage-based Neural Radiance Field for Generalized 3D Deformation and Animation. In *Advances in Neural Information Processing Systems*.

Patrick Pérez, Michel Gangnet, and Andrew Blake. 2003. Poisson Image Editing. *ACM Trans. Graph.* 22, 3 (jul 2003), 313–318. https://doi.org/10.1145/882262.882269

Nikos Platis and Theoharis Theoharis. 2003. Progressive Hulls for Intersection Applications. *Computer Graphics Forum* 22, 2 (2003), 107–116. https://doi.org/10.1111/1467-8659.00653 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00653

Siddhant Prakash, Gilles Rainer, Adrien Bousseau, and George Drettakis. 2022. Deep scene-scale material estimation from multi-view indoor captures. *Computers & Graphics* 109 (October 2022), 15–29. http://www-sop.inria.fr/reves/Basilic/2022/PRBD22

Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. 2020. D-NeRF: Neural Radiance Fields for Dynamic Scenes. *arXiv preprint arXiv:2011.13961* (2020).

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. https://doi.org/10.48550/ARXIV.2103.00020

Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. 2000. Silhouette Clipping. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley

Publishing Co., USA, 327–334. https://doi.org/10.1145/344779.344935

Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance Fields without Neural Networks. In *CVPR*.

Hang Si. 2015. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw.* 41, 2, Article 11 (feb 2015), 36 pages. https://doi.org/10.1145/2629697

Karl Stelzner, Kristian Kersting, and Adam R. Kosiorek. 2022. Decomposing 3D Scenes into Objects via Unsupervised Volume Segmentation. https://openreview.net/forum?id=rS9t6WH34p

Richard Szeliski. 2022. *Computer Vision - Algorithms and Applications, Second Edition.* Springer.

Towaki Takikawa, Alex Evans, Jonathan Tremblay, Thomas Müller, Morgan McGuire, Alec Jacobson, and Sanja Fidler. 2022. Variable Bitrate Neural Fields. https://doi.org/10.48550/ARXIV.2206.07707

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2021. Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes. (2021).

Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, et al. 2020. State of the art on neural rendering. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 701–727.

Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. 2021. Non-Rigid Neural Radiance Fields: Reconstruction and Novel View Synthesis of a Dynamic Scene From Monocular Video. In *IEEE International Conference on Computer Vision (ICCV)*. IEEE.

Wei-Cheng Tseng, Hung-Ju Liao, Lin Yen-Chen, and Min Sun. 2022. CLA-NeRF: Category-Level Articulated Neural Radiance Field. *arXiv preprint arXiv:2202.00181* (2022).

Ingo Wald, Will Usher, Nathan Morrical, Laura Lediaev, and Valerio Pascucci. 2019. RTX Beyond Ray Tracing: Exploring the Use of Hardware Ray Tracing Cores for Tet-Mesh Point Location. In *High-Performance Graphics - Short Papers*, Markus Steinberger and Tim Foley (Eds.). The Eurographics Association. https://doi.org/10.2312/hpg.20191189

Can Wang, Menglei Chai, Mingming He, Dongdong Chen, and Jing Liao. 2022. CLIP-NeRF: Text-and-Image Driven Manipulation of Neural Radiance Fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 3835–3844.

Chung-Yi Weng, Brian Curless, Pratul P. Srinivasan, Jonathan T. Barron, and Ira Kemelmacher-Shlizerman. 2022. HumanNeRF: Free-Viewpoint Rendering of Moving People From Monocular Video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 16210–16220.

Chuhua Xian, Hongwei Lin, and Shuming Gao. 2009. Automatic generation of coarse bounding cages from dense meshes. In *2009 IEEE International Conference on Shape Modeling and Applications*. 21–27. https://doi.org/10.1109/SMI.2009.5170159

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. 2022. Neural Fields in Visual Computing and Beyond. *Computer Graphics Forum* (2022). https://doi.org/10.1111/cgf.14505

Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. 2022. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5438–5448.

Tianhan Xu and Tatsuya Harada. 2022. Deforming Radiance Fields with Cages. In *ECCV*.

Bangbang Yang, Yinda Zhang, Yinghao Xu, Yijin Li, Han Zhou, Hujun Bao, Guofeng Zhang, and Zhaopeng Cui. 2021a. Learning Object-Compositional Neural Radiance Field for Editable Scene Rendering. In *International Conference on Computer Vision (ICCV)*.

Bangbang Yang, Yinda Zhang, Yinghao Xu, Yijin Li, Han Zhou, Hujun Bao, Guofeng Zhang, and Zhaopeng Cui. 2021b. Learning Object-Compositional Neural Radiance Field for Editable Scene Rendering. In *International Conference on Computer Vision (ICCV)*.

Yu-Jie Yuan, Yukun Lai, Tong Wu, Lin Gao, and Li-Gang Liu. 2021. A Revisit of Shape Editing Techniques: From the Geometric to the Neural Viewpoint. *Journal of Computer Science and Technology* 36, 3, Article 520 (2021), 34 pages. https://doi.org/10.1007/s11390-021-1414-9

Yu-Jie Yuan, Yang-Tian Sun, Yu-Kun Lai, Yuewen Ma, Rongfei Jia, and Lin Gao. 2022. NeRF-Editing: Geometry Editing of Neural Radiance Fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 18353–18364.

Xiuming Zhang, Pratul P Srinivasan, Boyang Deng, Paul Debevec, William T Freeman, and Jonathan T Barron. 2021. NeRFactor: Neural Factorization of Shape and Reflectance Under an Unknown Illumination. *https://arxiv.org/abs/2106.01970* (2021).

Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew Davison. 2021. In-Place Scene Labelling and Understanding with Implicit Scene Representation. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

Matthias Zwicker, Mark Pauly, Oliver Knoll, and Markus Gross. 2002. Pointshop 3D: An Interactive System for Point-Based Surface Editing. *ACM Trans. Graph.* 21, 3 (jul 2002), 322–329. https://doi.org/10.1145/566654.566584